Digital backend for a resistor-based Wien Bridge temperature sensor

By

Miloš Grubor

Supervisors:

Prof. dr. K.A.A. Makinwa

Dr. S. Pan

In partial fulfillment of the requirements for the degree of

Master of Science

In Electrical Engineering,

At Delft University of Technology,

Faculty of Electrical Engineering, Mathematics and Computer Science

To be defended on January 21st, 2022

Student Number:	5007062
Thesis Committee:	Prof. dr. K.A.A. Makinwa
	Dr. Ir. N.P. van der Meijs

Abstract

This thesis describes the design of a digital backend for a resistor-based temperature sensor realized in a 180nm CMOS process. The sensor's analog frontend outputs a bitstream at 500kHz, which is then decimated and linearized with the help of 3 different polynomials, whose coefficients are obtained after a calibration process. A polynomial engine, with storage registers for the coefficients, a decimation filter, and an SPI communication module for off-chip communication of results/coefficients were implemented. The polynomial engine was implemented in fixed-point representation using two evaluation methods: Horner's rule and scaling method. The implemented digital backend does not degrade the resolution, accuracy, and energy-efficiency performance of the sensor. It has an area of 0.068mm^2 (compared to the 0.12mm^2 of the sensor). It consumes 28µW (compared to the 66µW of the sensor). With the digital backend, the sensor achieves a resolution of 501µK at 25℃, which is only slightly worse than with off-chip (floating point) calculations (476 µK). The sensor's inaccuracy, determined from 32 samples, also maintained its original order of magnitude: 0.05℃ (3σ) over the military temperature range (-55℃ to 125℃).

Contents

1 Introduction

Temperature sensors are critical in many applications. They are used in medical devices, for weather forecasts, in other sensors that need temperature compensation, and in control systems. Driven by technological advances in manufacturing integrated circuits, the need for small, accurate, and energy-efficient CMOS temperature sensors has risen over the years. Compared to discrete sensors, CMOS sensors have become popular due to their low production cost, and ease of integration [1][2]. One of the main uses of high-resolution integrated sensors is in the temperature compensation of on-chip frequency references [3].

Based on their sensing principles, CMOS temperature sensors can be categorized into several types: BJT-, MOSFET-, resistor-, and thermal-diffusivity-based sensors [2]. Because of their large temperature sensitivity, resistor-based temperature sensors can achieve superb resolution and the best energy efficiency among all the sensor types [4]. To suppress quantization noise, such sensors are typically read-out by ΔΣ modulators. However, the resulting bitstream output still requires substantial digital processing.

First, decimation filters are required to down-sample their bitstream (BS) outputs and remove out-of-band quantization noise. Second, the result needs to be trimmed and linearized to compensate for the process spread and non-linear temperature dependence of resistors. So far, these operations have been typically done off-chip, with the help of Matlab or Labview code running on PCs [5].

Depending on the choice of reference resistor, there are two main types of resistor-based temperature sensors: dual-R and RC. Dual-R sensors use one type of resistor as the sensor and another type of resistor as the reference. By choosing a resistor whose temperature coefficient (TC) is opposite to that of the sensing resistor, a larger signal and thus a higher energy efficiency can be obtained. As in [6], such reference resistors are typically made from polysilicon, and thus exhibit significant $1/f$ noise. Also, the TC spread of both the sensing and reference resistors contributes to the sensor's inaccuracy. On the other hand, RC-based sensors employ the impedance of a capacitor as a reference. They are typically less energy-efficient and require a well-defined frequency reference [7]. However, since metal-insulator-metal CMOS capacitors are much more stable than resistors, RC-based sensors can achieve better accuracy. At a fixed frequency, the temperature in an RC-based sensor is dependent on the filter's phase shift. Due to their 2nd order phase response, Wien-bridge filters are often used in RC sensors.

The goal of this thesis is to design a digital backend for a state-of-the-art resistor-based temperature sensor based on a Wien-bridge filter [8]. For ease of use, BS decimation, trimming, and nonlinearity compensation should all be realized on-chip, while minimizing the added chip area and power consumption. Also, communication with the sensor, e.g. reading and writing registers, should be done via standard microprocessor-compatible protocols such as SPI or I^{2}C.

1.1 Wien-bridge sensor prototype

1.1.1 Front-end and ADC

The circuit diagram of a Wien-bridge (WB) filter is shown in Figure 1.1a. It employs silicided polysilicon resistors with a large temperature coefficient (TC) ~$0.3\%/℃$ and metal-insulator-metal capacitors $(\mathrm{MIM}, \mathrm{TC}{\sim}50\mathrm{ppm}/℃)$[9]. The circuit is a second-order bandpass filter, whose phase shift from input (V_{drive}) to output (V_{WB}) is represented by Equation (1.1):

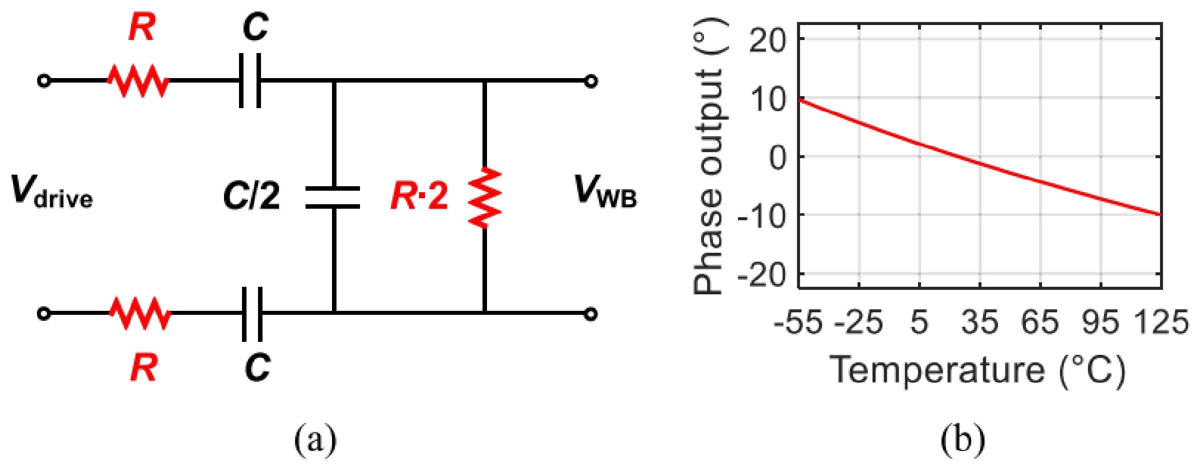

Figure 1.1 Wien-bridge circuit (a) and its calculated phase response (b) [8]

The circuit is a second-order bandpass filter, whose phase shift from input (V_{drive}) to output (V_{WB}) is represented by Equation (1.1):

$$\varphi_{WB}(\omega) = -tan^{-1}(\frac{R^2C^2\omega^2 - 1}{3RC\omega}) \tag{1.1}$$

When driven at a fixed input frequency, chosen to ensure zero phase shift at room temperature (25°C), the WB phase shift will then vary by about 20° over the military temperature range (-55°C to 125°C), as shown in Figure 1.1b. However, the resulting characteristic is noticeably non-linear and thus requires linearization.

In Figure 1.2, a simplified single-ended implementation of a Wien-bridge (WB) sensor is shown. For easy implementation, the input signal is a square wave of fixed frequency. The output current of the WB is denoted as I_{WB} and is a phase-shifted version of the input signal. The demodulator is realized by a chopper that controls the direction of the current (I_{demod}) flowing into the integration capacitor Cint so that the demodulation signal is thus a square-wave of the same frequency.

The WB output current is digitized by a Phase-domain Delta-Sigma Modulator (PDΔΣM). This is a negative feedback loop, in which a comparator detects the polarity of the integrator output, and then controls the phase of the demodulation signal such that, on average, I_{demod} is forced to zero. As a result, the BS output of the comparator is a representation of the WB phase and thus temperature.

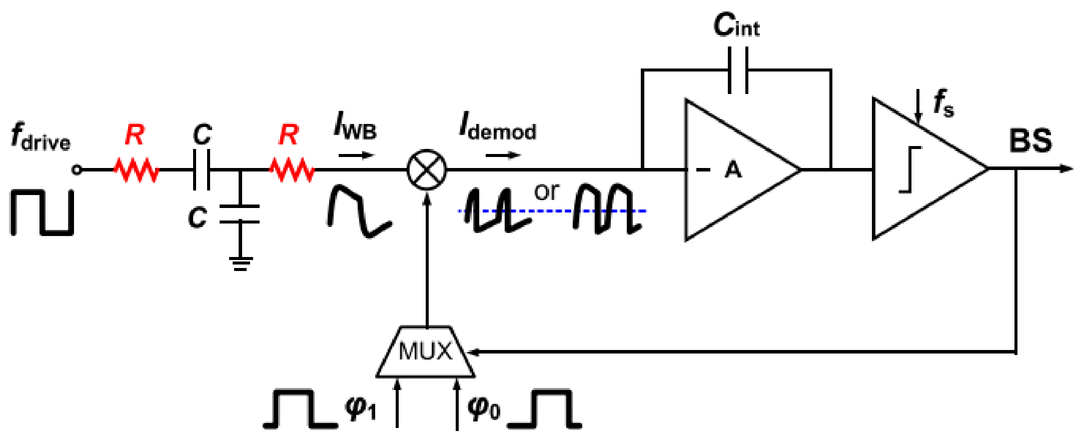

Figure 1.2 Simplified single-ended Wien-bridge circuit diagram and its read-out by PDΔΣM [8]

In the actual implementation, the first amplifier stage of the loop filter is also chopped to reduce the 1/*f* noise of the amplifier used to realize the integrator (see Figure 1.3). To minimize the required control logic and errors due to charge injections, the input chopper and the demodulation chopper are combined. Moreover, the ΔΣM employs a switched capacitor (SC) second stage to provide more loop gain, and, thus, more effectively suppress quantization noise.

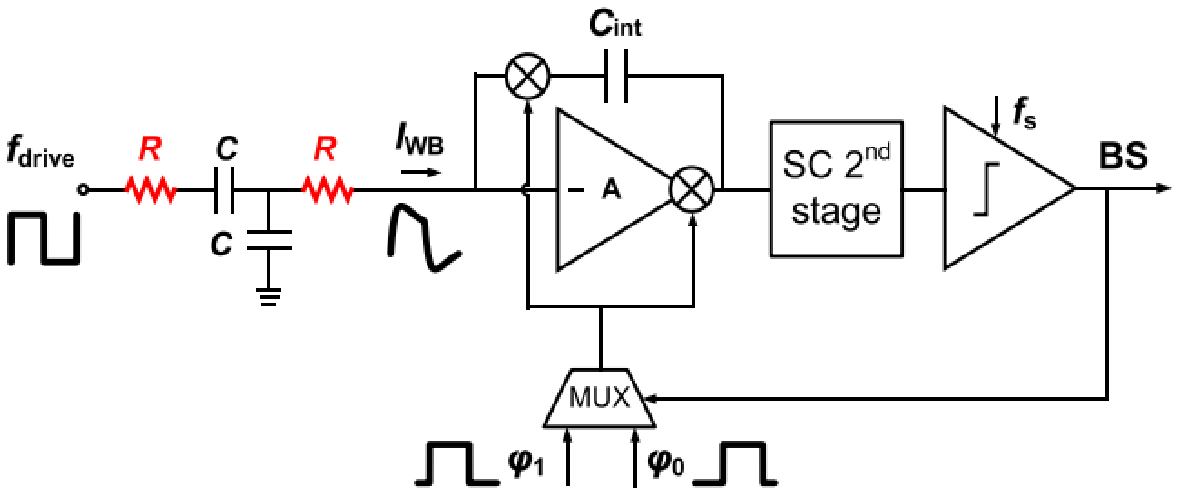

Figure 1.3 Circuit after adding the second stage in the forward path and merging chopping with demodulation [8]

The prototype sensor occupies an active chip area of 0.12mm^2 and consumes 66µW power from a 1.8V supply. It achieves a thermal-noise limited resolution of 0.45mK in a 10ms conversion time. This translates to a resolution FoM (Figure of Merit) [10] of 0.13pJ·K^2.

The sensor's non-linear phase-temperature characteristic is caused by three main mechanisms: the non-linear dependence of resistance on temperature, the nonlinear dependence of WB phase on resistance and the nonlinearity introduced by the PDΔΣM due to the use of square-wave excitation and demodulation signals. A higher-order polynomial can be used to remove these nonlinearities, a process that will be referred to as µ-R translation (µ representing the BS average). To account for the spread of nominal resistance and TC, each sensor must be trimmed to achieve high accuracy. After trimming, a systematic non-linearity (SNL) is observed, which is due to higher-order TC coefficients. This, in turn, can be removed by an SNL-removal polynomial.

The choice of the order of each polynomial, and the order in which they are applied, will be explained in the following section.

1.1.2 Off-chip digital processing

The output of the existing sensor is a bitstream, which requires trimming and digital processing to generate high-accuracy temperature information [10]. The processing flow is shown in Figure 1.4.

First, the DSM BS output is decimated and down-sampled using a sinc^2 decimation filter. After truncation, the filtered output is then sent to a 6th-order polynomial engine, which translates the filtered BS to a resistance value (µ-R translation). After that, two-point trimming is performed at -35 and 105°C, referenced to a pre-calibrated Pt-100 thermistor. Finally, the residual nonlinearity caused by the high-order TC of the sensing resistor is removed with a 4th order polynomial.

Overall, there is one decimation filter and three polynomials: 6th order, 1st order, and 4th order (see Figure 1.4). Details of the signal processing will be introduced in Chapter 3.

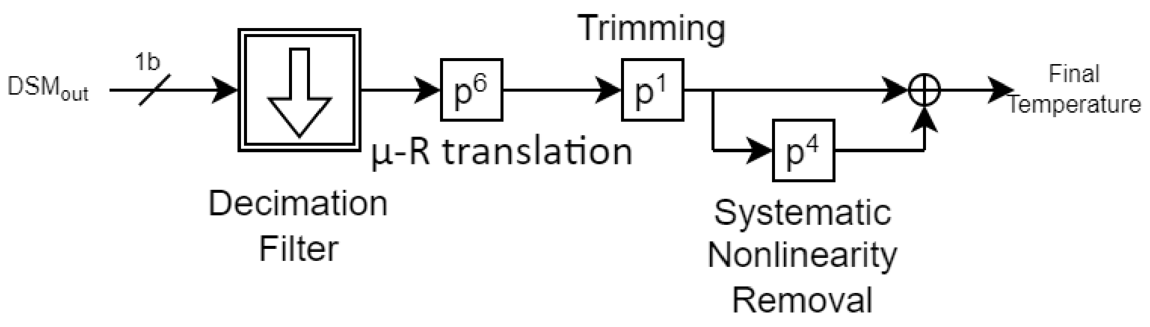

Figure 1.4 Digital signal processing flow

1.2 Targeted specifications

The aim of the project is to design a digital backend that will perform decimation filtering, polynomial processing, store internal coefficients and intermediate results, and implement a protocol for off-chip communication, for receiving said coefficients and sending the results to a microprocessor.

The area of this digital backend should be less than the area of the implemented sensor (0.12mm^2), and its power consumption should be lower as well (60μW). Using the rule of three, the extra quantization noise it introduces should be around 100μK, to ensure that the sensor's resolution is limited by its own thermal noise floor (450μK). Furthermore, the added inaccuracy due to fixed-length arithmetic should be less than 10% of the sensor's inaccuracy, i.e. less than 3mK.

The main requirements of the sensor are summarized in Table 1.1 List of requirements:

Performance metric	Prototype sensor without digital circuits	Digital circuit
Area	0.12mm^2	<0.12mm^2
Power	66μW	<60μW
Temperature range	-55°C to 125°C	-55°C to 125°C
Resolution	450μK (Thermal-noise limited)	~100μK (added quantization noise)
3σ Inaccuracy	0.03°C	<3mK (added inaccuracy)

Table 1.1 List of requirements

1.3 Thesis Organization

The rest of the thesis is organized as follows: the reasoning behind the chosen polynomial processing is presented in Chapter 2, followed by a detailed overview of the digital backend. Chapter 3 deals with the design architecture of each subpart and the digital processing backend as a whole. Chapter 4 discusses the circuit design and its functionality verification along with the simulation results. Chapter 5 presents the measurement setup and results. Finally, Chapter 6 concludes this thesis and discusses of future work.

2 Off-chip polynomial processing currently in use

As described in [8], the average BS produced by 40 sensors over the military temperature range is shown in Figure 2.1. It can be seen that they exhibit a systematic nonlinearity (SNL), together with some random spread due to process variations.

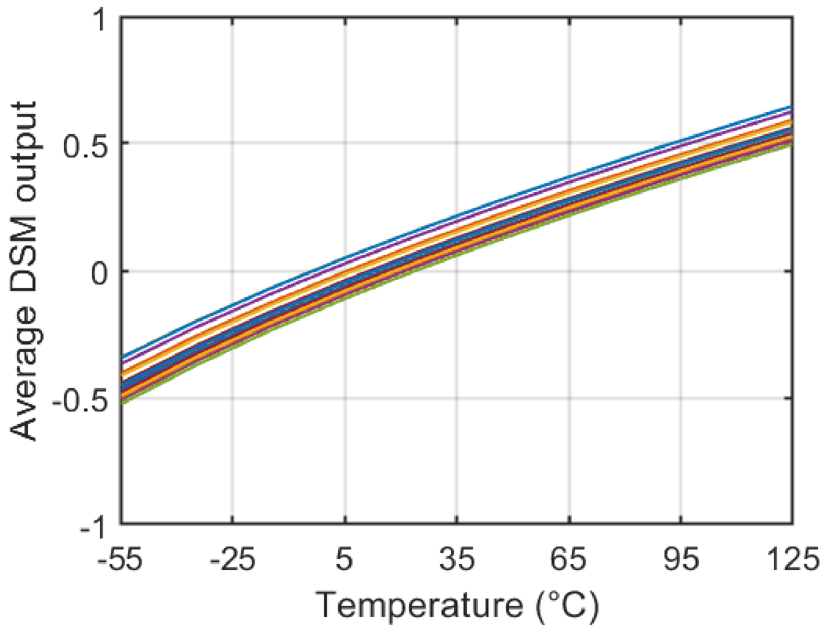

Figure 2.1 Average BS for 40 sensors from [8] over the military temperature range

Trimming the BS average without removing the nonlinearity first results in lower accuracy. This is illustrated in Figure 2.2. Due to process spread, the nominal resistance R_0 will also spread resulting in a different nonlinearity in the output D_{out} of each sensor, as seen in the upper plot. Trimming the sensors first will then result in different error profiles, as seen in the lower two plots. Thus, before trimming, the sensor's output should be linearized to achieve better accuracy.

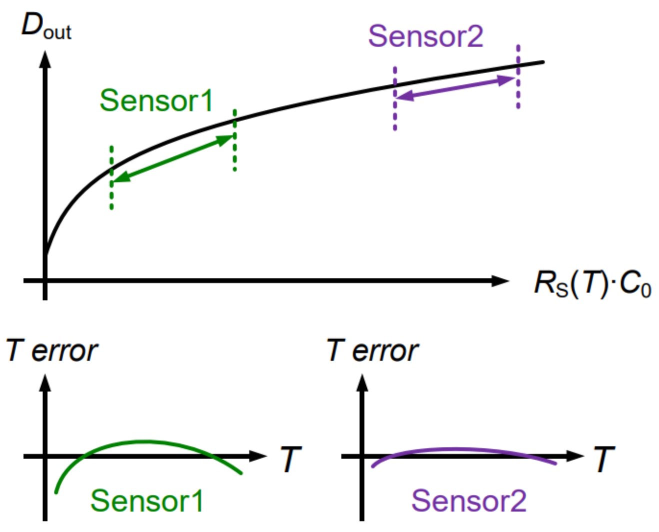

Figure 2.2 Nonlinear dependence of BS average on resistance and influence of trimming before removing said nonlinearities [8]

To remove SNL, a µ-R translation polynomial is needed, which can be obtained by simulating the sensor's output as a function of WB resistance. This is explained in more detail in Section 3.2.1.1. Sufficient accuracy, e.g. less than 1℃ error, can be achieved with a 6th order polynomial. After this SNL removal, the resulting resistance value versus temperature is shown in Figure 2.3.

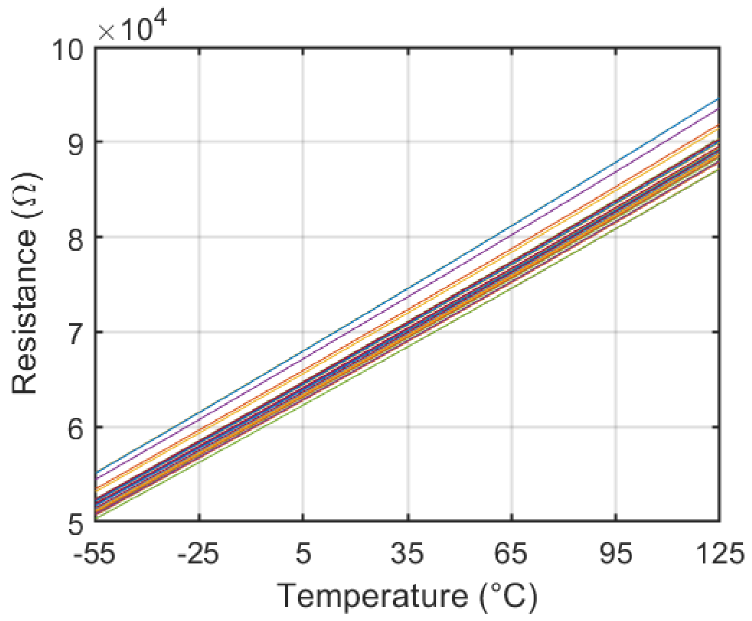

Figure 2.3 Obtained resistance versus the corresponding temperature

The remaining spread in TC and nominal resistance can then be eliminated by performing a 1st order fit based on measurements at two temperatures: -35 and 105℃. The resulting measurement error is shown in Figure 2.4. It appears to be systematic and nonlinear, with significantly less spread.

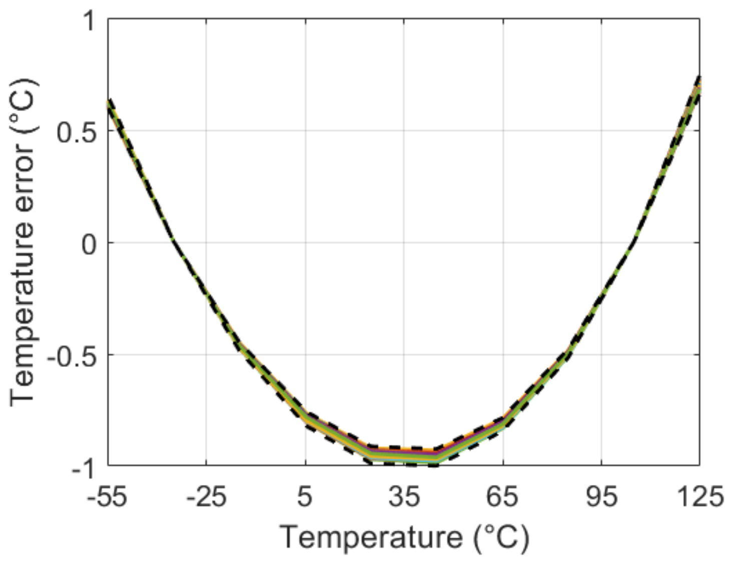

Figure 2.4 Temperature error after 2-point trim

This systematic nonlinearity is attributed to the higher-order temperature dependence of the sensing resistors. This error can also be removed by a fixed polynomial. To determine the required order, the final inaccuracy obtained with 2nd, 3rd, and 4th order polynomials is shown in Figure 2.5.

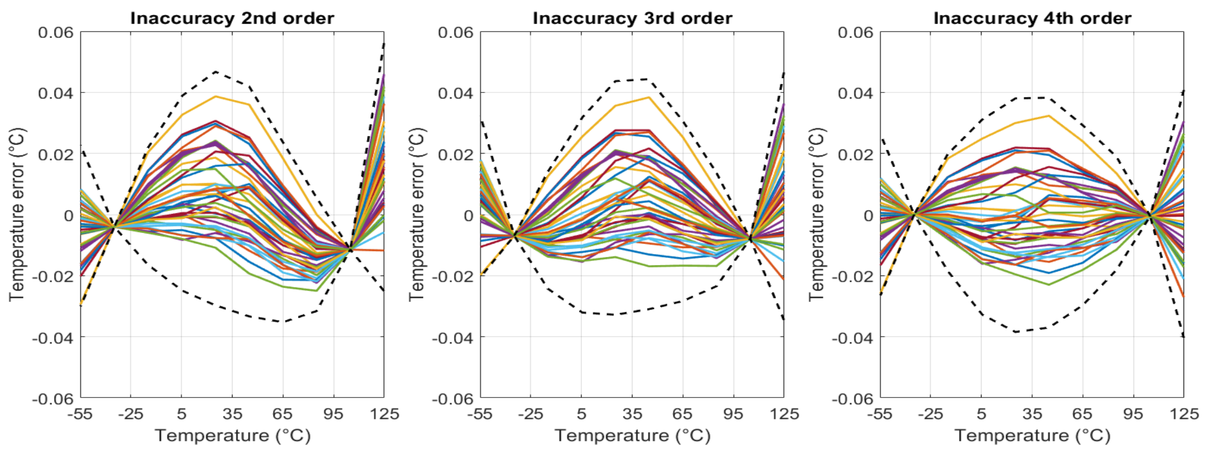

Figure 2.5 Final inaccuracy for 3 different orders of systematic error removal polynomial

The 4th order polynomial results in the lowest residual error, which is also quite symmetric, whereas the residual error achieved with lower-order polynomials is quite skewed. Increasing the polynomial order does not improve accuracy, and actually introduces a risk of over-fitting.

To conclude, the chosen polynomial processing for the lowest inaccuracy assumes that the decimated BS output is first passed through a 6th order µ-R translation polynomial. The result is then passed through a 1st order polynomial that is unique to each sensor (two-point trim). Finally, the result is applied to a fixed 4th order polynomial, whose output is then added to the result of the two-point trim to obtain the final measured temperature.

The 1st order polynomial is unique to each sensor, while the 4th order polynomial is fixed for a batch of sensors. Their coefficients are not obtained in the same manner, as explained in Section 3.2.1, and for that reason, they cannot be combined into one polynomial. Further benefits of separating these polynomials in terms of calculation accuracy are described in Section 3.2.2.

3 System-level overview of the digital backend

This chapter introduces the various building blocks of the digital backend, as well as their function in the overall system. The architecture of the digital backend is shown in Figure 3.1. It consists of a decimation filter, followed by a polynomial engine. First, the choice of decimation filter is explained, along with the required number of bits to achieve the required resolution. After that, the core of this project- “the polynomial engine”- is described in detail. Finally, the chapter ends with a discussion of the communication protocol needed to do temperature read-out and to set the polynomial coefficients in the internal on-chip register memory.

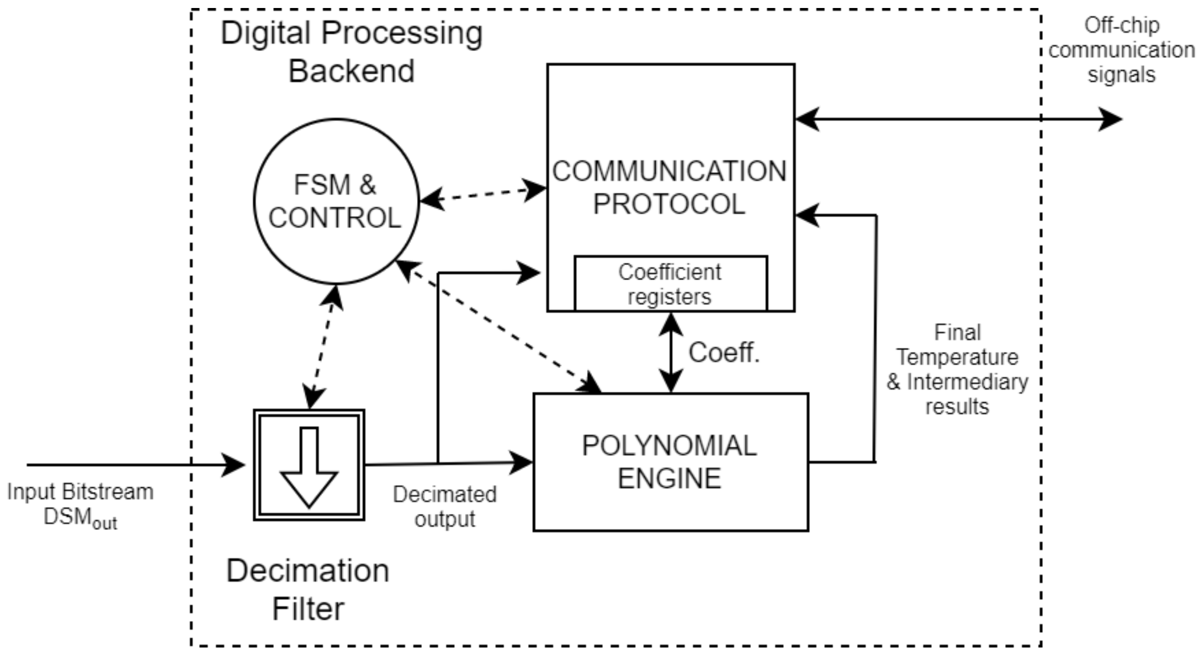
Digital Processing Backend
Off-chip communication signals
FSM & CONTROL
COMMUNICATION PROTOCOL
Coefficient registers
Coeff.
Final Temperature & Intermediary results
POLYNOMIAL ENGINE
Input Bitstream DSM$_{out}$
Decimated output
Decimation Filter

Figure 3.1 System-level overview diagram

3.1 Decimation filter

As discussed in chapter 1, a Wien-bridge filter is readout by a Phase-domain Delta-Sigma Modulator (PDΔΣM) that outputs a 500kHz bitstream. This then needs to be filtered and down-sampled before polynomial processing. This operation is carried out by the decimation filter.

3.1.1 Requirements & input resolution

As discussed in Section 1.2, the WB sensor’s resolution is limited to about 450μK by thermal noise. So the quantization noise should be below 100μK over the entire temperature range.

The relation between the decimated output of the PDΔΣM and the final temperature is shown in Figure 3.2. Within the chosen range of -55 to 125°C, the decimated output varies between -0.5 and 0.6. The sensor’s temperature sensitivity is lowest at 125°C, and thus the decimation errors at this temperature should be minimized. When transferred to the range of the DSM (from -1 to 1), a 100μK temperature change corresponds to a DSM output change of $4.43 \cdot 10^{-7}$, which corresponds to a resolution of $\sim 1/2^{21}$.

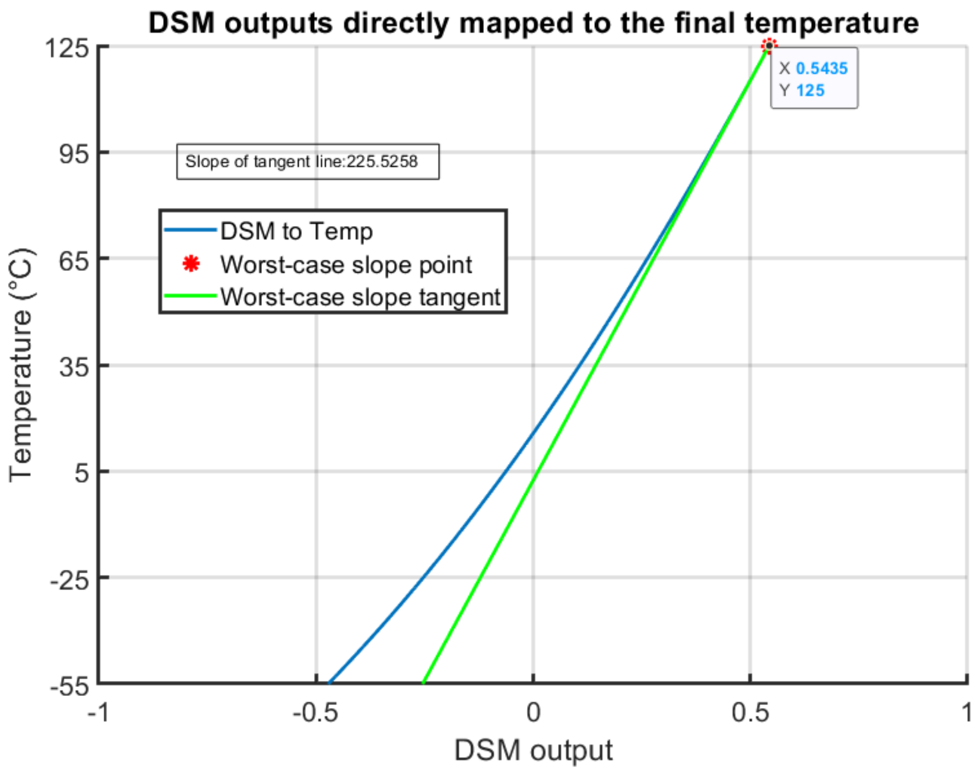

Figure 3.2 Decimated filter output vs. Final temperature with worst-case slope

3.1.2 Type of filter

As per [11], a sinc^2 filter is chosen to achieve a good balance between high resolution and low complexity as shown in Figure 3.3. To simplify its implementation, the coefficients are realized as integers, and the final output is then scaled. If the largest coefficient M (= 2048 in Figure 3.3) is a power of 2, the scaling factor is also a power of 2, and scaling is done by shifting, instead of division.

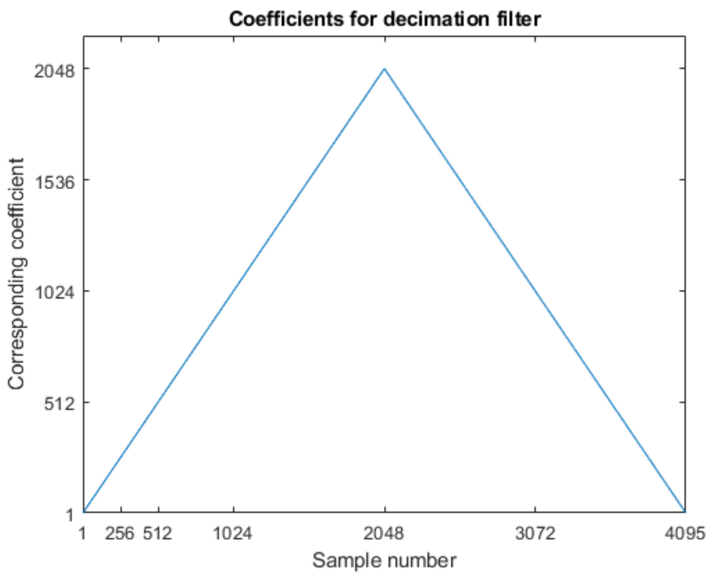

Figure 3.3 Decimation filter coefficients

The larger M is, the longer the decimation cycle is and thus the higher the resolution is. The relation between the largest coefficient, M, and the number of significant bits (representing resolution), N, following the derivation in [12] is:

$$N = 2 \cdot \log_2(M) \tag{3.1}$$

The lowest value of M (only powers of 2 were examined for implementation simplicity) that meets the required DSM resolution is 2048. That means that the mentioned decimation filter only needs 4095 samples to achieve this resolution. Bitstreams were generated for random numbers between -0.7 and 0.7 to represent realistic DSM output, as shown in Figure 2.1. These random values are denoted as input numbers. The quantization error for these input numbers in the decimation filter output was simulated. The output quantization error of this filter is shown in Figure 3.4, and the achieved resolution is $4.14 \cdot 10^{-7}$.

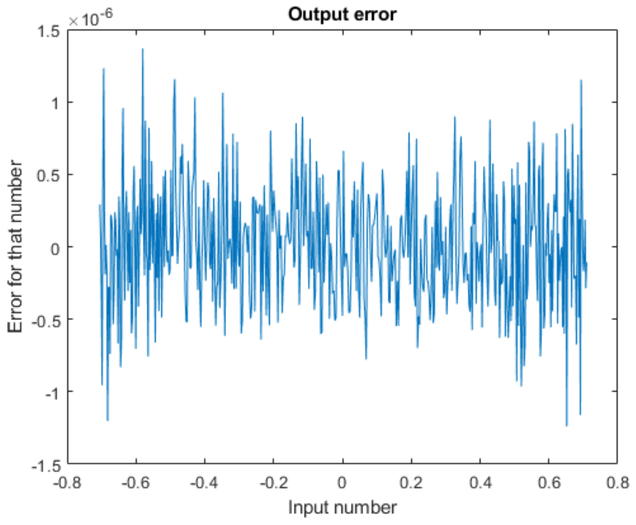

Figure 3.4 Quantization error after decimation filter

The total conversion time would be $t_{conv} = 4095 \cdot \frac{1}{500kHz} \approx 8.2ms$, which is in line with the requirement of being below 10ms.

The decimation filter can be analytically described as $dec_{out} = \sum_{i=1}^{4095} BS_i \cdot f_i$, where BS_i are bits of corresponding bitstream output of DSM and f_i are coefficients of the proposed decimation filter:

$$f_i = \begin{cases} i, & 1 \leq i \leq 2048 \\ 4096 - i, & i > 2049 \end{cases} \tag{3.2}$$

3.1.3 Number of bits

Effective Number of Bits (ENOB) for the required resolution is $\log_2\left(\frac{|0.6-(-0.5)|}{4.43 \cdot 10^{-7}}\right) = 21.24.$ After rounding it up to 22, this result is in line with the number obtained from Equation (3.1), while substituting M as 2048. One extra bit is added for robustness, and the final number of 23 bits is obtained.

In the rest of the paper, to limit the area usage of registers for coefficients, intermediate results and of multiplies, adders, etc., 23 bits are set as the maximum length of all the coefficients and intermediate results of the polynomial engine (see Figure 3.5).

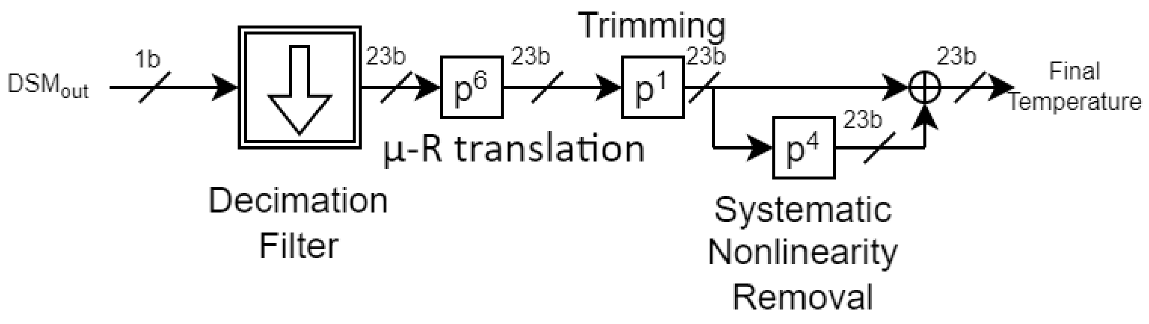

Figure 3.5 Digital signal processing flow with NOB

3.2 Polynomial engine

In this subsection, the three polynomials used in polynomial engine (see Figure 3.5) are explained in detail: what their function is and how they can be modified and combined efficiently.

3.2.1 Full-precision polynomial prototype

3.2.1.1 µ-R translation

The first polynomial in the cascade is µ-R transition as explained in Section 1.1. Removing the nonlinearity introduced by the WB and the PDΔΣM facilitates further trimming directly on the resistance value (or strictly speaking, a normalized RC value) [5].

The µ-R translation is a 6th order polynomial that is obtained through the circuit simulation of the average currents as the temperature varies from -55℃ to 125℃ and the resistances in the WB range from 40 to 115kΩ. With two phase references (-22.5° and 22.5°) corresponding to BS=0 and BS=1, the average WB output current has been simulated as $I_{ave,0}$ and $I_{ave,1}$. Since the average current flowing into the ΔΣM is zero, the BS average µ must satisfy :

$$(1-\mu)\cdot I_{ave,0} + \mu \cdot I_{ave,1} = 0 \tag{3.3}$$

$$\mu = {}^{I_{ave,0}}\!\big/_{(I_{ave,0} - I_{ave,1})} \tag{3.4}$$

The average bitstream value is scaled to fit in the range (-1,1), and then the polynomial coefficients are obtained by fitting a 6th order relation between the balanced bitstream average (µ) value and resistance range.

The resulting dependence is shown in Figure 3.6:

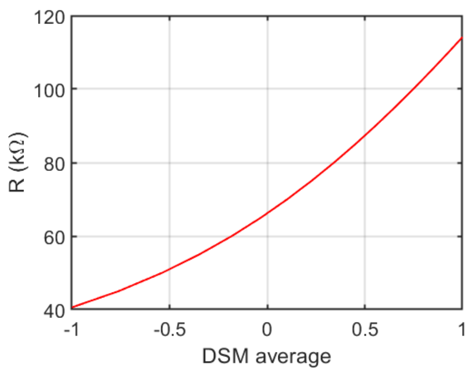

Figure 3.6 µ-R translation

3.2.1.2 2-point trim

As explained in Chapter 2, after µ-R translation, the obtained resistance is converted to temperature. To account for process variations and spread of both the nominal resistance (R_0) and TC, two-point trimming (-35 and 105°C, referenced by a Pt-100 thermistor) is done on each sensor to determine its R_0 and TC

The remaining error is shown in Figure 3.7. As it can be seen, the error graph crosses the 0 mark at the above-mentioned temperatures of -35 and 105℃.

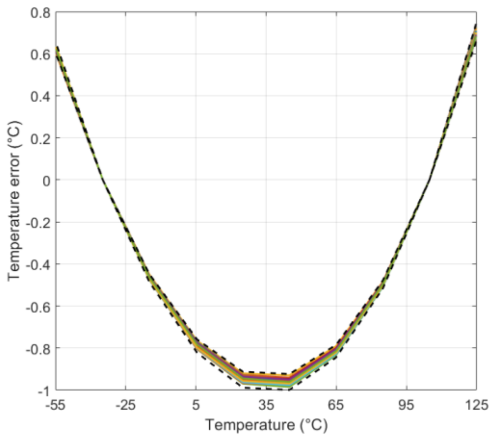

Figure 3.7 Residual temperature error after 2-pt trim

3.2.1.3 Systematic error removal

The error after the two-point calibration is systematic over a batch and can be corrected by a fixed polynomial. As shown in Figure 3.7, the error is mostly quadratic but also contains higher-order components. After a 4th order polynomial correction (Figure 3.8), the residual error is below 0.03℃ (Figure 3.9). Further increasing the polynomial order results in insignificant improvements.

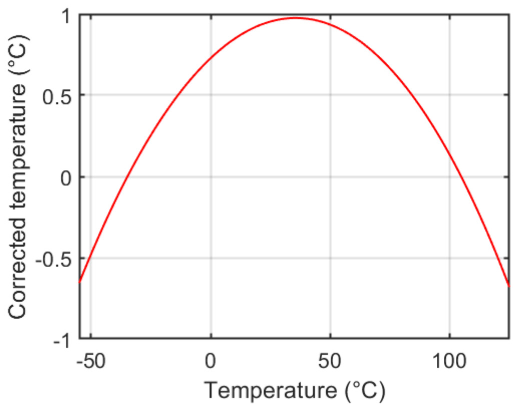

Figure 3.8 Systematic error removal polynomial

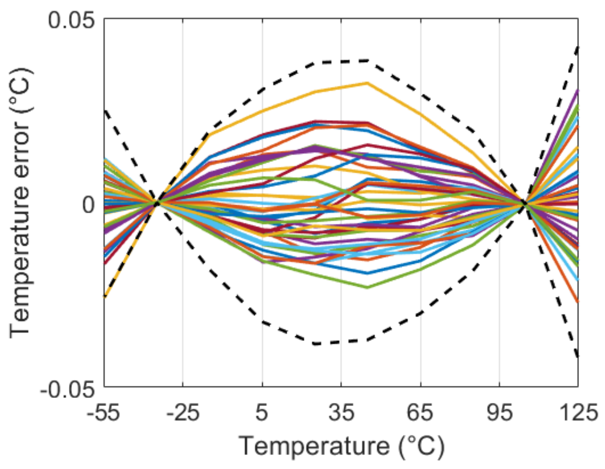

Figure 3.9 Final residual temperature error

3.2.2 Customized polynomial engine

3.2.2.1 Fixed-point vs. floating-point

To implement standard 32-bit floating-point polynomial evaluation, a multiply-add unit is needed. After scaling the area reported in [13], this would occupy about 0.45mm^2 in 180nm CMOS, which is about four times higher than the sensor's area and the set requirement.

To achieve a simpler and more energy-efficient hardware, fixed-point calculations are preferred to floating-point. This means that all the coefficients should ideally have one representation: with a fixed number of integer bits and a fixed number of fractional (non-integer part) bits.

3.2.2.2 Horner's rule and scaling method

Horner's rule is one of the most common and most widely used algorithms for polynomial evaluation, due to its recursive structure and simplicity of implementation [14]. The standard polynomial representation of a fourth-order polynomial is:

$$p(x) = a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 \tag{3.5}$$

By applying Horner's rule, this can be transformed into a more regular structure of 1st order polynomials $a_n \cdot x + a_{n-1}$, as shown below:

$$p(x) = (((a_4 \cdot x + a_3) \cdot x + a_2) \cdot x + a_1) \cdot x + a_0 \tag{3.6}$$

A single multiply-add unit can then be used to evaluate polynomials of any order. Using Horner's rule, an nth order polynomial can be evaluated with n additions and n multiplications, which has been proven mathematically to be the minimum number of multiplications needed [15].

Take the 4th order SNL removal polynomial, for example:

$$p(x) = a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 \tag{3.7}$$

Horner's rule makes the evaluation simple, with just using consecutive multiplications and additions, without the evaluation of powers of x. Ideally, the coefficients, the intermediate as well as the final results should have the same fixed-point representation.

Typical SNL removal polynomial coefficients are shown in Table 3.1:

Coefficient	Coefficient value	NOB needed for the fractional part[1]
a_0	0.8342	3
a_1	1.0157	0
a_2	$-2.2626 \cdot 10^{-4}$	12
a_3	$1.2380 \cdot 10^{-7}$	23
a_4	$-1.0958 \cdot 10^{-9}$	29

Table 3.1 Coefficients of systematic error removal polynomial

With a fixed-point representation, at least 29 fractional bits are needed to ensure one significant decimal digit for a_4. This exceeds the limit of 23 bits for both decimation results and coefficients.

Because the magnitude of the coefficients gradually decreases, the polynomial can be expressed as shown below, in which each coefficient is scaled by its adjacent neighbor. Coefficients that were added after the multiplication in Horner's rule are now scaled and placed after the parenthesis as a multiplication factor, and the adding coefficients that remain after scaling are 1:

$$p(x) = \left(\left(\left(\left(\frac{a_4}{a_3}x+1\right)\frac{a_3}{a_2}x+1\right)\frac{a_2}{a_1}x+1\right)\frac{a_1}{a_0}x+1\right)a_0 \tag{3.8}$$

When the coefficients a$_x$ from Equation (3.7) are substituted by new coefficients s$_x$, the following relation is obtained:

$$p(x) = \left(\left(\left((s_4x+1)xs_3+1\right)xs_2+1\right)xs_1+1\right)s_0, \tag{3.9}$$
$$where \quad s_i = \frac{a_i}{a_{i-1}}, \;\; s_0 = \; a_0$$

The table of the newly obtained coefficients and corresponding Number of Bits (NOBs) is shown below:

Coefficient	Coefficient value	NOB needed for the fractional part
s_0	0.8342	3
s_1	0.0188	6
s_2	-0.0144	6
s_3	$-5.4714 \cdot 10^{-4}$	14
s_4	-0.0089	10

Table 3.2 Coefficients of scaled systematic error removal polynomial

After scaling, the worst-case number of bits (NOB) becomes 14 bits for the fractional part, which allows a 23-bit representation to be used with a margin of 9 more bits, for extra precision.

Another way to analyze the difference between these two methods is to compare the dynamic range of the coefficients, which is the ratio of the absolute values of the largest and smallest coefficient. In the case of Horner's rule, it is 181.4dB. If this ratio is converted to an equivalent number of bits by the formula for signal-to-noise ratio $DR = 6.02 \cdot N + 1.76dB$, it requires a 29.85 bit word length to

[1] This signifies the least Number of Bits (NOB) needed for fractional part to represent correct at least the first significant digit of the coefficient. For example for the a_1 which is higher than 1, no additional fractional bits are needed as the first significant digit can be correctly represented with integer bits. On the other hand, for a_2, for example, 12 bits after the binary/decimal point are needed to represent -0.0002.

represent both the largest and the smallest coefficients. For the coefficients of the scaling method this is only 77.1dB or 12.5 bits (these NOBs are in line with the values presented in Tables 3.1 and 3.2).

The drawback of this scaling method is increased complexity: it requires 9 multiplications and 4 additions, compared to the original 4 multiplications and 4 additions. However, the added computation time remains well below ~8.2ms, which is the duration of one cycle of decimation. The scaling method requires the same computational units as Horner's method (multiplier and adder) while requiring a lower number of bits for the coefficients. The cost of the added calculation time is acceptable.

3.2.2.3 Combining coefficients

The other two polynomials: µ-R translation (6th order) and 2-point trim (1st order), do not suffer from this worst-case NOB problem, and the standard Horner's method can be applied. To further reduce the implementation complexity, the two polynomials can be combined into a single 6th order polynomial as shown in Equations (3.10)-(3.13).

$$p_{\mu-R}(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \tag{3.10}$$

$$p_{2pt}(x) = b_1x + b_0 \tag{3.11}$$

$$\begin{aligned} p_{comb}(x) &= p_{2pt}\left(p_{\mu-R}(x)\right) \\ &= b_1(a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0) + b_0 \end{aligned} \tag{3.12}$$

$$p_{comb}(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \tag{3.13}$$

The final polynomial flow is shown in Figure 3.10:

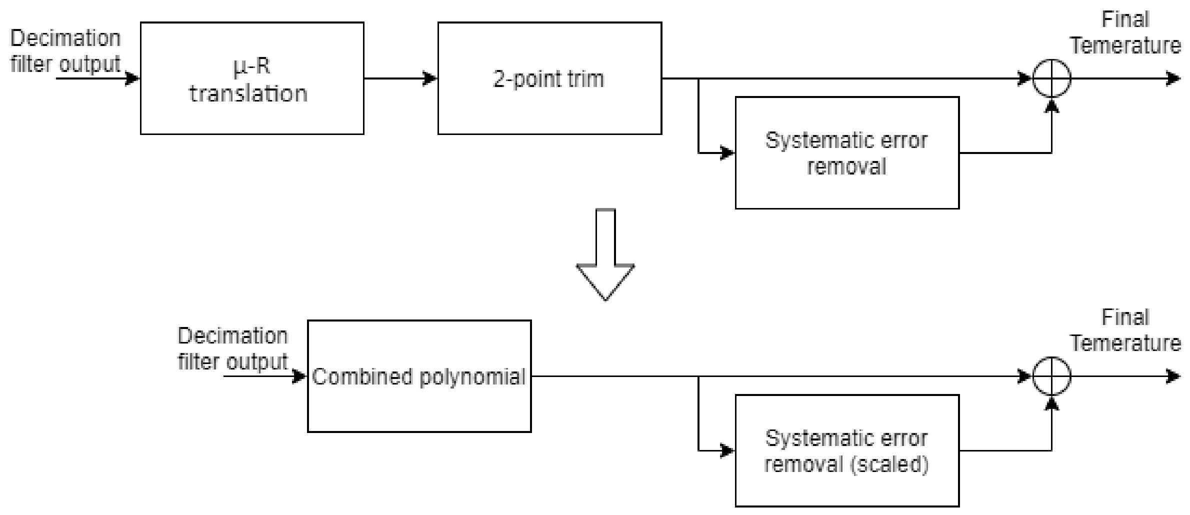

Figure 3.10 Final polynomial evaluation flow

3.3 Communication protocol and coefficient storage

For ease of use, the sensor input/output should be compatible with standard microprocessor protocols, e.g., UART, RS-485, I2C, or SPI. UART can only connect two devices but not an array of sensors; RS-485 is complicated, resulting in a power-hungry design. Both I2C and SPI can allow multiple devices to be connected to the same bus signals, but SPI is preferred for its simplicity [16].

In this subsection, the detailed protocol for off-chip communication with SPI is explained. Also, the structure of storage elements for the coefficients is shown.

The SPI module requires a different clock frequency than that used for the sensor. The challenge of ensuring synchronization between the two different clock domains will be explained in this chapter.

3.3.1 SPI interface

A standard SPI system consists of 2 parties: a master and a slave. The master initiates communication and specifies whether it wants to read from or write data to the slave. The protocol uses 4 wires, as can be seen in Figure 3.11:

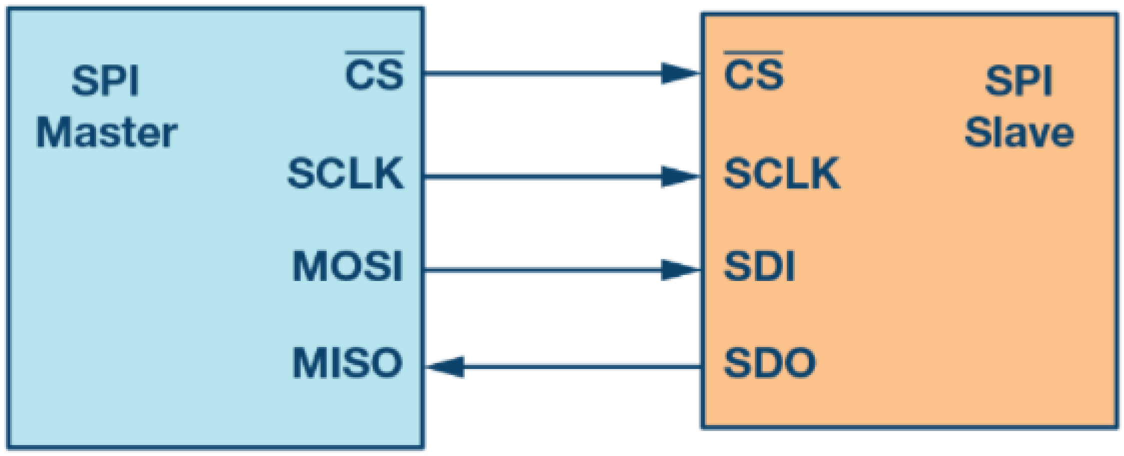

Figure 3.11 SPI master and slave configuration [17]

The CS (Chip Select) signal is high in the idle state and drops low when communication is initiated by the master. SCLK is a clock signal for the SPI. Through the SDI (Serial Data In), the SPI master sends synchronized data (or commands) to its slave/sensor. In this project, the commands include writing the polynomial coefficients and reading the results from the sensor, which is done through the SDO (Serial Data Out) pin [17].

3.3.2 Coefficient registers as storage memory

There are two options to write coefficients: the first involves serially shifting in the desired coefficient, storing it to a temporary register, and then, in a second step, transferring it to a designated storage register (Figure 3.12a). The second involves directly shifting the coefficient into a designated storage register (Figure 3.12b). Option (a) involves routing 23 wires per one coefficient, whereas option (b) would involve only one, with one extra, enable wire for each signal. Each extra wire requires an area for routing and switching of the signal on the wire contributes to power consumption. Due to area and power constraints, the serially shifted-in coefficient registers (b) option is preferred.

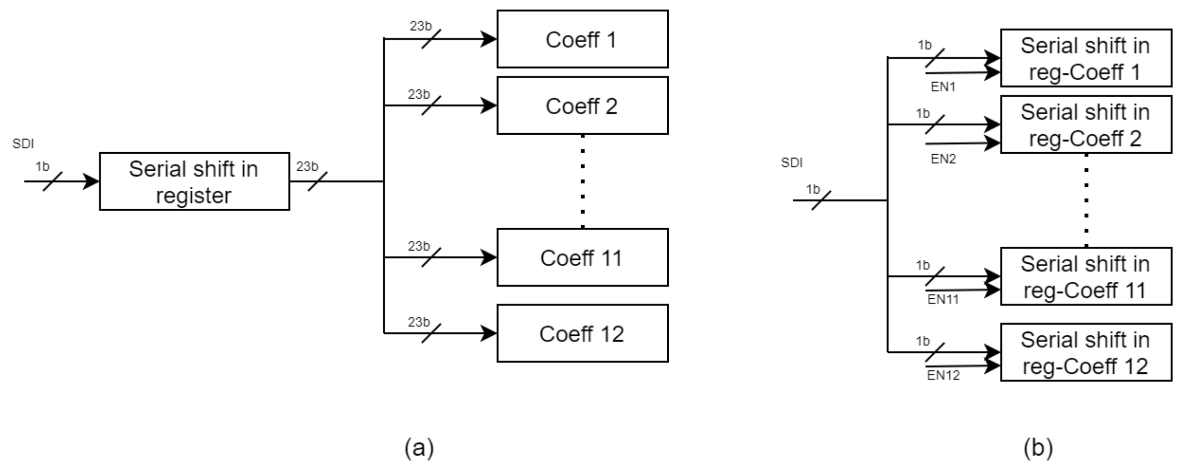
SDI
1b
Serial shift in register
23b
Coeff 1
Coeff 2
Coeff 11
Coeff 12
(a)
SDI
1b
EN1
EN2
EN11
EN12
Serial shift in reg-Coeff 1
Serial shift in reg-Coeff 2
Serial shift in reg-Coeff 11
Serial shift in reg-Coeff 12
(b)

Figure 3.12 Two options for coefficients storage in registers - (a) as regular registers that will be updated once SPI communication is done, (b) serially shifted in registers, updated every cycle of SPI communication

In this design, 12 coefficients are stored: 7 for the 6th order combined polynomial and 5 for the 4th order systematic error removal polynomial. Apart from those registers, 3 more shift-out registers are designed to store intermediate and final results: one for the output of the decimation filter, one for the output after the combined polynomial, and one for the final temperature.

3.3.3 Detailed SPI protocol

To be compatible with the standard SPI protocol, both read and write operations in this work are done with integer multiples of bytes. First, when the CS signal drops low, the master sends an 8-bit (first byte) command through SDI, during the 8 SPI clock (SCLK) cycles. Depending on the function encoded by those 8 bits, the next 24 SCLK cycles will be responsible for the writing of sensor coefficients (via SDI) or the reading of sensor status (via SDO), as shown in Figure 3.13.

To encode 15 different functions (writing 12 shift registers and reading 3 results), 4 address bits are sufficient. For simplicity, it was chosen to discard the first 4 shifted-in bits in the 8-bit command. As shown in Figure 3.13, adr7 to adr4 (in orange) will be disregarded, whereas the adr3 to adr0 will be stored to decode the following function.

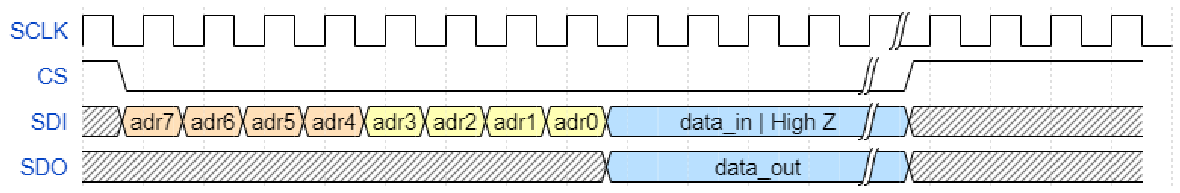

Figure 3.13 an example of one SPI communication

The function encoding table is shown below:

ADDR (adr3 to adr0 in Figure 3.13)	R/W (read or write the type of function)	Description
0..6	W	Writing coefficients for the combined polynomial where ADDR 0 corresponds to a_6, ADDR 1 to a_5, … and ADDR 6 to a_0
7..B	W	Writing coefficients for the systematic error removal polynomial where ADDR 7

		corresponds to s_4, ADDR 8 to s_3, … ADDR B to s_0
C	W	This ADDR is not used, but it is reserved for communication with the analog part of the sensor should there be a need to
D	R	This function shifts out the output of the decimation filter
E	R	This function shifts out the output of the combined polynomial
F	R	This function shifts out the final temperature

Table 3.3 SPI address function encoding and their meaning

Shifting out the value is done from the most significant bit (MSB) to the least significant one. As 23-bit results are shifted out, to comply with the standard SPI protocol, 24 bits must be shifted out. This is done by first shifting out an extra bit, which will be the repeated MSB. After that repeated bit, the results are shifted out as explained. Shifting in the value is also done from MSB to the LSB, the first bit to be shifted in is discarded, while the following 23 are stored in the desired register.

3.3.4 Synchronization

The SPI module works on its clock signal- SCLK, which is independent of the bitstream clock on which the rest of the chip works. As a result, synchronization between the two clock domains is necessary applied to ensure proper functioning.

Two situations need to be taken care of:

- The first case is when writing the coefficient values via the SPI module. While the coefficients are being changed by the master, the polynomial engine should not use it for calculation. In this design, this is ensured by resetting the rest of the chip (apart from the SPI module) while updating coefficients.
- The second case is when the polynomial engine and decimation filter update their outputs, and the SPI module has to update the shift-out registers for sending the results out to the master. This is solved by using a MUX-based synchronizer, a clock-domain crossing technique which will be explained in detail in the following section.

4 Design Architecture

This chapter describes the detailed architectural design of the three blocks of the digital backend: decimation filter, polynomial engine, and SPI, along with that of the global Finite State Machine (FSM) and its controlling signals.

4.1 Decimation filter

As mentioned in Section 2.1.2, a sinc2 filter is used, whose basic architecture is shown in Figure 4.1. Since the BS is a single-bit input, the multiplication operation can be achieved using a multiplexer: with the input being zero or one, a zero or the filter coefficient will be accumulated. The filter coefficient is generated by a counter, which counts from 1 to 2048 and then down from 2048 to 1.

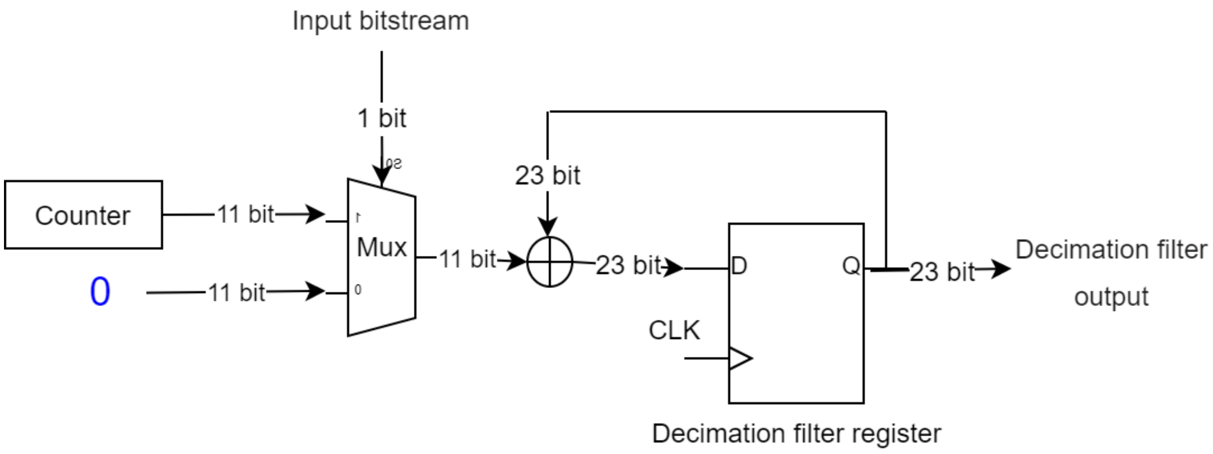

Figure 4.1 Architecture of the decimation filter

Moreover, the multiplexer can be replaced with a simple AND gate, which passes the coefficient only when the input BS is one.

4.1.1 Adder

The design in this project has to operate with a frequency of several hundred kHz, and the tolerable delay is thus more than 1 µs. Since this is much longer than the logic delay in the chosen technology (TSMC 180nm), the adder should be designed for the optimal power and chip area.

Among the common adder architectures such as Carry Look-Ahead (CLA), Carry Save Adders (CSA), Ripple Carry Adders (RCA), etc. an RCA occupies the smallest area and has the lowest power consumption [18]. The schematic of an RCA is shown in Figure 4.2.

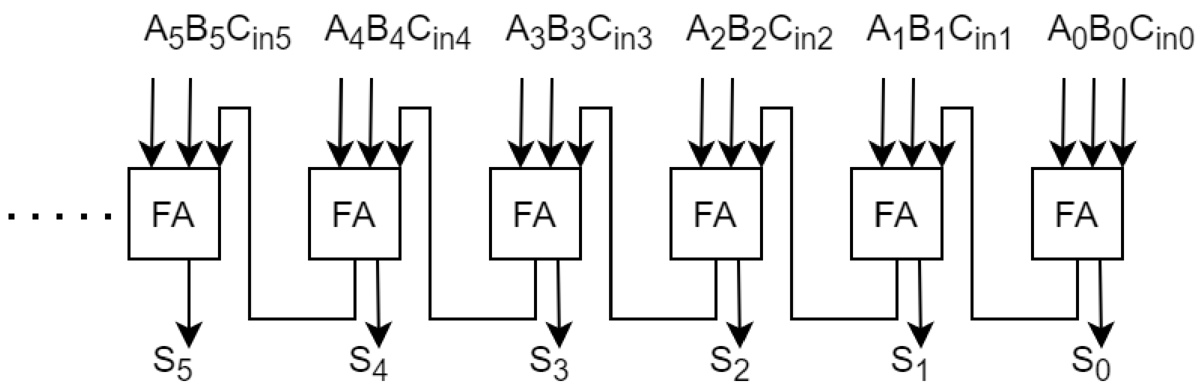

Figure 4.2 Ripple Carry Adder

4.1.2 Coefficient & FSM

Coefficients are generated with an 11-bit up-down counter which has a control signal to count up or down. Therefore, the counting range is 0 to $2^{11} - 1 = 2047$. To adapt the range to 1 – 2048 without implementing another counter bit, an input carry bit is used. Whenever there is an addition of coefficient (input bit is 1) then the carry-in signal of the adder is 1, otherwise, it is 0. This functionality is implemented using a multiplexer shown in Figure 4.1.

The final summed value ranges from 0 to 1 and is scaled to the range from -1 to 1 using: $DEC_{fin} = (DEC_{init} - 0.5) \cdot 2$, where DEC_{init} is the decimated value in range 0-1, and DEC_{fin} is the decimated value in the range from -1 to 1. The decimation filter's result is accumulated during the first 4095 cycles, and in the subsequent cycle, it is being scaled. It is done by adding 1 to the MSB (subtracted by 0.5) and shifting one bit to the left (multiplied by 2).

The finite state machine of the counter has 4 states, IDLE, UP, DOWN, and SCALE, which can be encoded using 2 bits. The state diagram is shown in Figure 4.3. While the circuit is in reset, the state machine is in IDLE state, as reset drops low, FSM goes to UP and starts counting up. FSM goes to the state DOWN at the maximum counter value and then the counter starts counting backwards until 0. After this, FSM moves to the state SCALE for one clock cycle to perform the scaling and immediately goes back to IDLE. During the switching from SCALE to IDLE, the FSM produces a *decim_done* signal. In case *reset*=1, FSM goes to IDLE state irrespective of its current state.

Together with the decim_done signal, the current value of the decimation filter register shown in Figure 4.1 is written onto the output register. None of the internal signals apart from the value of this output register is available outside of the decimation filter module.

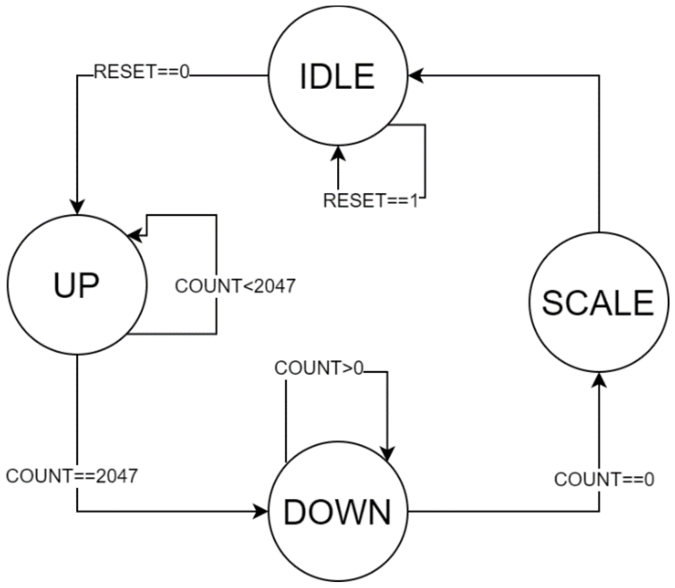

Figure 4.3 State diagram for decimation filter's FSM

4.2 Polynomial Engine

A polynomial engine consists of adders and multipliers. The design of the lowest-area adder was already covered in Section 4.1.1. In this subsection, the design architecture of the multiplier is presented, including the design of FSM and internal registers.

4.2.1 Multiplier

Since both the decimation filter output and the polynomial coefficients have a bit length of 23, a signed 23x23 bit multiplier is chosen, which outputs a 46bit result. Which 23 bits of this 46 are stored in the final result, depends on the state of FSM, which is explained in Section 4.2.3.

The multiplication of fixed-point numbers is similar to that of integers, with the only difference on the place on binary point position. Figure 4.4 shows an example of a 4*4 bit multiplier.

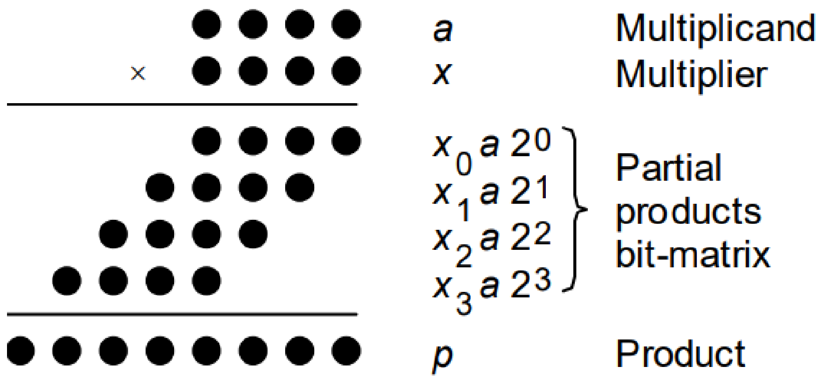

Figure 4.4 Dot notation of multiplication of two 4-bit binary numbers [18]

The differences among multiplier implementations are the realizations of partial products addition, and one of the simplest and most area-efficient ways uses a Shift/Add algorithm [19]. After each clock cycle, one new partial product is added to the intermediate result. And after each addition, the intermediate result is shifted by 1 bit. The order in which the partial sums are added determines whether to shift the intermediate result right or left. The bottom-top approach in the case of adding the last partial sum would need a 46bit adder to account for a carry propagation to the MSB. The top-bottom approach shifts the partial sums to the right and only a 23bit adder is needed. This is a preferred choice, the hardware implementation is shown in Figure 4.5.

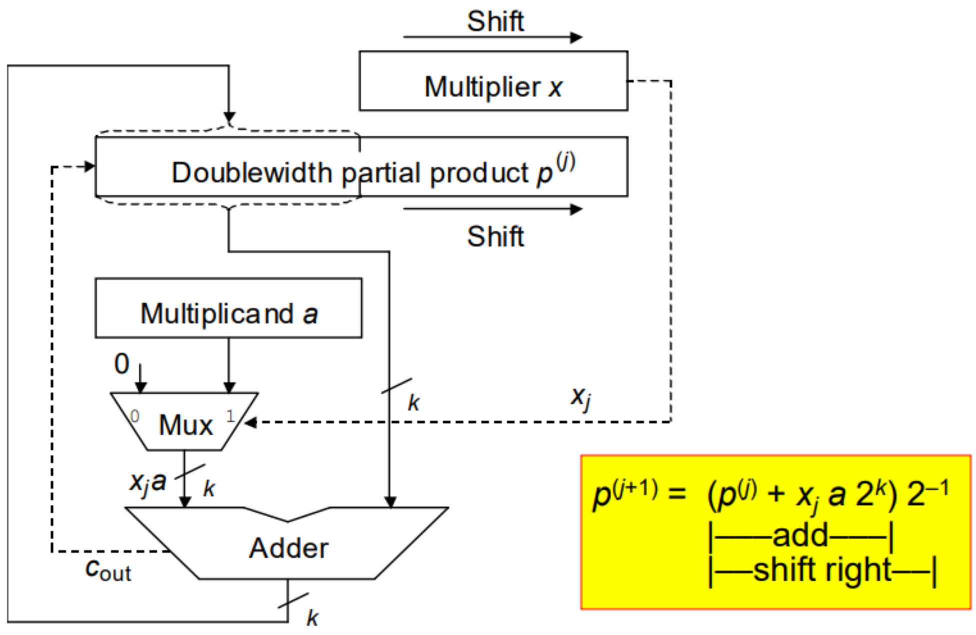

Figure 4.5 Hardware implementation of shift-right/add sequential multiplier [18]

The equation in the yellow box in Figure 4.5 describes the multiplication algorithm, where the starting condition is $p^{(0)} = 0$. This implementation is sequential and takes 23 cycles to finish the multiplication. As the polynomial engine is running in parallel to the decimation filter, which takes ~8.2ms to finish one decimation cycle, a time margin of 23 cycles per multiplication does not pose a problem. This implementation only requires one 23-bit adder(as previously explained- RCA) and a 46-bit shift register.

Both multiplier and multiplicand are stored in their corresponding register inside the multiplier. A multiplier is stored into a 23-bit shift register as shown in Figure 4.5. To make this a signed multiplier, the sign of the result is stored which will be used as a sign extension, and if the multiplier is a negative number, the complementary value of both multiplier and multiplicand are stored in the above-mentioned registers. As the numbers were assigned, multiplication is done with sign extension instead of simply with 0, which is the case in unsigned multiplication.

The simple FSM of the multiplier is shown in Figure 4.6. The PREPARE state lasts only one cycle, in which the sign bit has been set, and if needed, the complement values of inputs are stored into internal registers. A simple 5-bit counter is implemented as well to control the number of cycles needed to be in the MULTIPLY state, after which the FSM goes to the IDLE state, raising signal *MULT_DONE*, indicating that multiplication has finished.

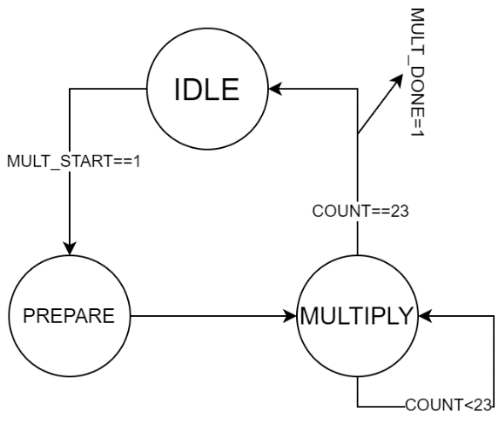

Figure 4.6 FSM of the multiplier

4.2.2 Internal registers & Multiplexers

As it was explained in Section 3.2.2.3 and shown in Figure 3.9, polynomial processing consists of evaluating two polynomials consecutively and the final result is obtained by adding the final result of both polynomials.

Processing in the polynomial engine is done on the output of the decimation filter, which is the input to the engine. Three other internal registers are also used for storing intermediate and final results: *COMB_OUT*- register to store the intermediate results of the first polynomial in the evaluation flow(combined polynomial); *SYS_OUT* register to store the intermediate results of the second polynomial(systematic error removal polynomial); and *FINAL_OUT* register to store the final value that is obtained by addition of values of both previously mentioned registers when the polynomial evaluation is over.

Two of these registers (*COMB_OUT* and *FINAL_OUT*) have separate output buffer registers whose output is connected to the SPI module. Their value is updated only when the polynomial evaluation is done, to make sure that the value that is read by the master is not an intermediate result of the evaluation.

All three internal registers have enable signals to control when their values are supposed to be updated, and the outputs of these registers serve as input to both multiplier and adder for the next evaluation step. The simplified architecture is shown in Figure 4.7. For simpler design and smaller area usage, the same multiplexers are used to generate only one set of input signals that will both go to the adder and multiplier. Control of the multiplexers is presented in detail in the following subsection, where the polynomial engine's FSM is discussed.

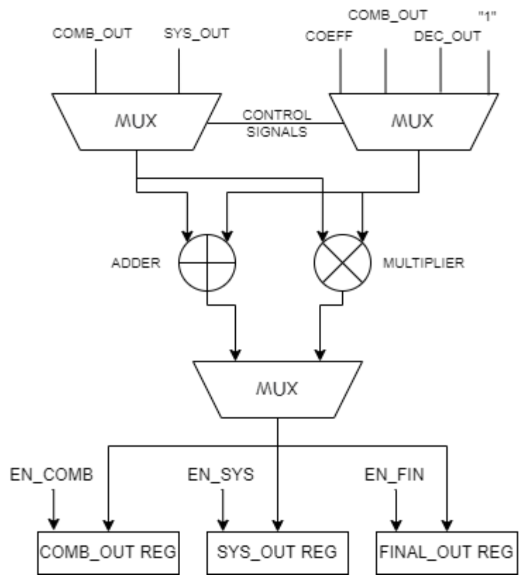

Figure 4.7 Simplified architecture of the polynomial engine

Simulation of the combined polynomial with assuming fixed representations for all the operands (intermediate results and coefficients) resulted in insufficient accuracy. As mentioned in Section 3.2.2, changes of representations should be allowed, as a compromise between simplicity of fixed-point and accuracy of floating-point representation. The number of those changes should be minimized, and based on simulation results, the introduction of only one more representation is needed.

For the evaluation of higher powers of the combined polynomial, which are evaluated at the beginning using Horner's rule, 5 more bits in the fractional part are needed. Therefore, for the transition to the final representation, the first 18 MSBs of the stored result are used and 5 sign extension(SE) bits are added at the beginning. To use the proper representation, one additional multiplexer is implemented between the operand one and the first input of the adder as shown in Figure 4.8.

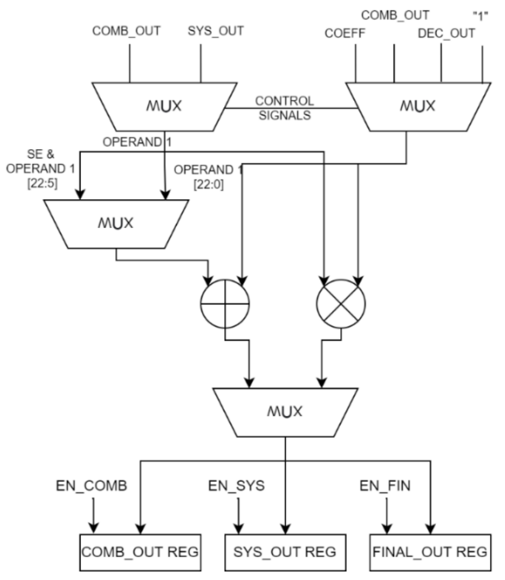

Figure 4.8 Architecture after adding one more representation

The multiplier outputs a 46-bit result and by choosing which of its 46 bits to store in a 23-bit register, the necessity of multiplexing inputs to the multiplier is avoided. In the evaluation of systematic error removal, there are two different types of multiplications of the intermediate result, one with the coefficient and the other with the output of the combined polynomial. In that case, two different choices of the set of bits are needed for storing the final result: *MULT[44:22]* and *MULT[36:14]*, respectively. The modified architecture is shown in Figure 4.9.

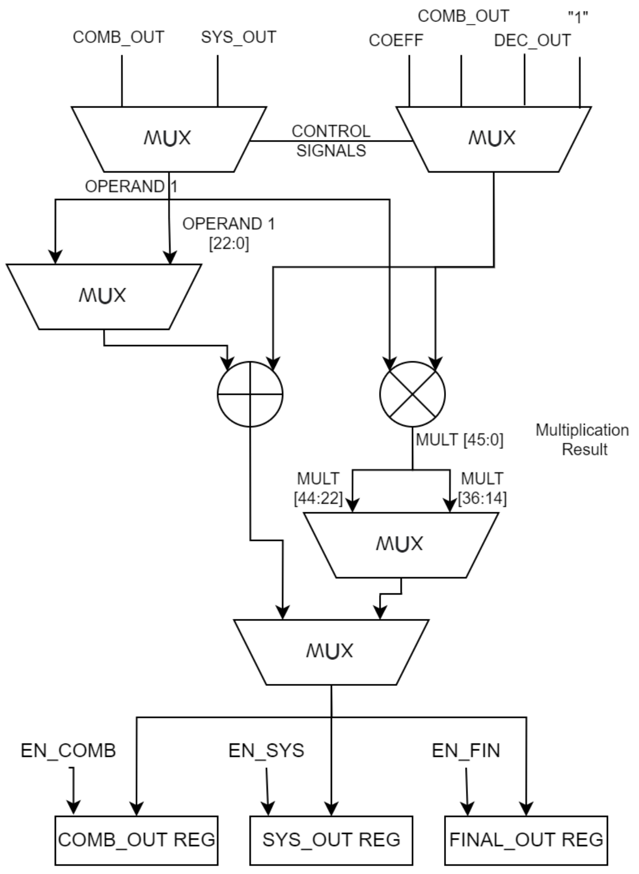

Figure 4.9 Final architecture

Coincidentally, the set of bits after multiplication for the combined polynomial to be stored is also *MULT[44:22]*, which saves the area of another multiplexer. The *FINAL_OUT* register is only updated with the value of the sum of *COMB_OUT* and *SYS_OUT*. Once the final value is stored in the register, the polynomial processing is finished, and the values of the output registers going to the SPI module are updated.

4.2.3 FSM

Apart from the IDLE state, all other states signify either addition or multiplication being currently done. All of the "addition" states last only for one clock cycle, whereas, once the FSM enters any of the multiplication states it stays in that state until the multiplier raises the signal *MULT_DONE*. During the change of each state, only one enable signal is high, while the states are not changing, none of the enable signals are active. Before the final state diagram and a table containing representations and control signals are presented, obtaining the coefficients' value and part of the FSM corresponding to each polynomial are explained.

A 4-bit counter is implemented that counts the current coefficient that needs to be used. Each time a coefficient is used in one of the arithmetic operations, the counter counts one up. In the IDLE state, the value of the counter is reset to 0. The 4-bit counter value also represents the address of the corresponding coefficient and it is as such passed directly to the SPI module, which in this situation behaves as Read-Only Memory.

The first polynomial, as described in Section 3.2, is the combined polynomial of 6th order (Equation (3.12)). Its evaluation using Horner's rule can be represented with the following equation:

$$p_{comb}(x) = (((((a_6 \cdot x + a_5) \cdot x + a_4) \cdot x + a_3) \cdot x + a_2) \cdot x + a_1) \cdot x + a_0 \qquad (4.1)$$

The state machine that describes this process is shown in Figure 4.10.

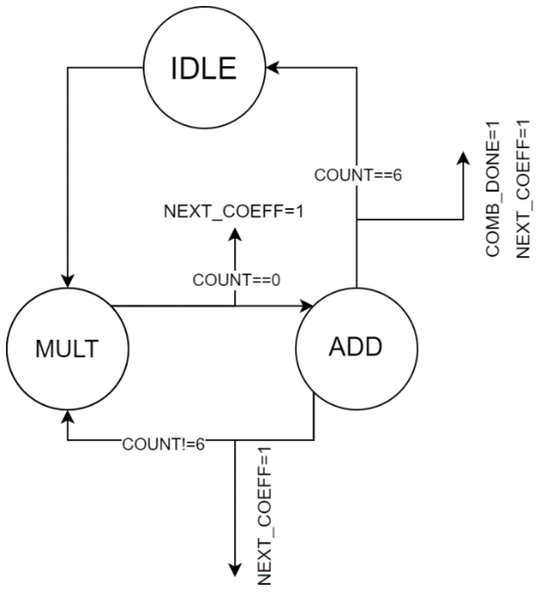

Figure 4.10 FSM for the combined polynomial

When the evaluation is started, FSM moves to the state MULT from the IDLE state, where the term $a_6 \cdot x$ is evaluated, as it is the first expression, it immediately increases the counter by *NEXT_COEFF=1*, which enables the counter. In the ADD state the coefficient a_5 is added, and the FSM returns to the state MULT, with again updating the coefficient. This process is iterated until finally the last coefficient a_0 is added and the output of the counter (*COUNT*) reaches 6, then FSM produces *COMB_DONE* signal and updates the counter value again for the latter polynomial evaluation.

There are three possible modes of operation, and they are shown in Figure 4.11. The first mode is adding a coefficient (a), second is multiplying the current value with *DEC_OUT* (b). The third mode represents the shifting between two different fixed-point representations (c). Shifting of the representations happens when *COUNT* is equal to 4, or when a_2 is added. Before said addition, the fixed-point representation of *COMB_OUT* has 5 more bits in the fractional part, and after that, the extra precision is removed. The configuration shown in (c) is only used once in the ADD state when *COUNT* is 4, in the rest of the evaluation, configuration (a) is used.

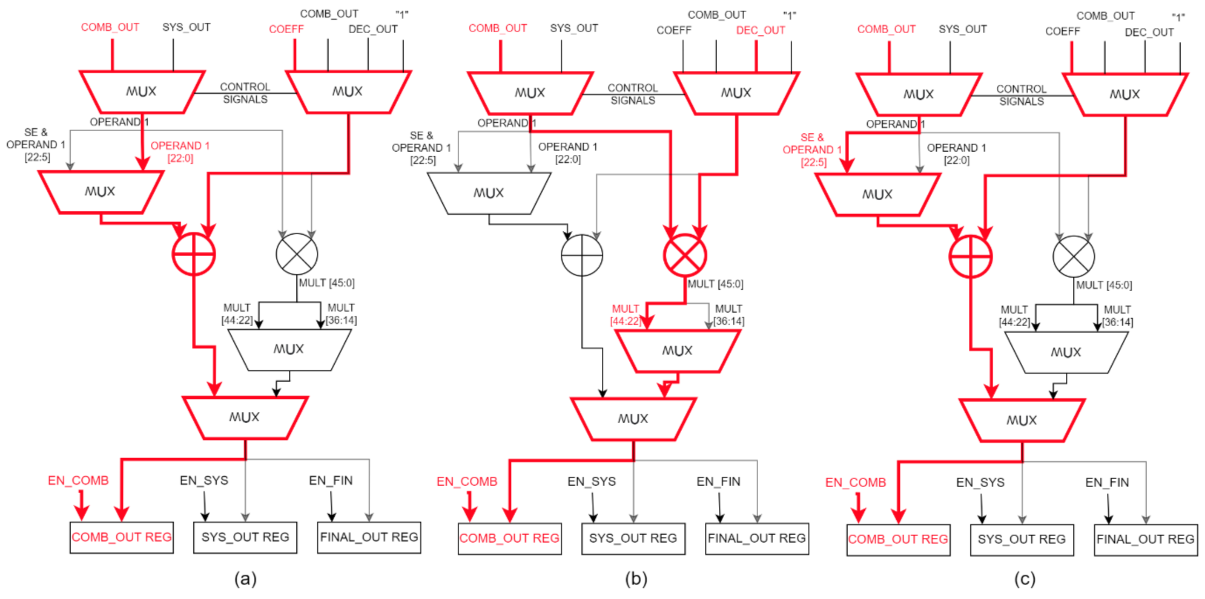

Figure 4.11 Three possible modes of operation for the evaluation of the combined polynomial

The second polynomial, as described in Section 3.2 is the systematic error removal polynomial of 4th order (Equation (3.8)). Its evaluation using the scaling method can be represented with the following equation:

$$p_{sys}(x) = \left(\left(\left((s_4x + 1)xs_3 + 1\right)xs_2 + 1\right)xs_1 + 1\right)s_0, \qquad (4.2)$$

$$where \quad s_i = \frac{a_i}{a_{i-1}}, \quad s_0 = a_0$$

The state machine diagram for the corresponding FSM is shown in Figure 4.12:

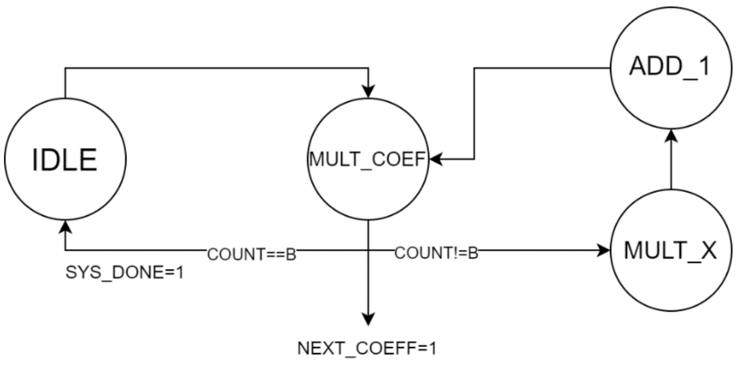

Figure 4.12 FSM for the systematic error removal polynomial

The evaluation starts when the state machine goes from the IDLE state to the MULT_COEFF state. The initial value of *SYS_OUT* is a fixed-point representation of number 1. Multiplying with coefficient produces a value of s_4, further entering the state MULT_X, produces $s_4 x$. After the multiplication, a fixed-point representation of the number 1 is added to the result. The cycle continues until the *COUNT* value does not reach 11 (hexadecimal B), after which the state machine returns to the IDLE state and raises the *SYS_DONE* signal.

There are three possible modes of operations and their configuration in the proposed architecture is shown in Figure 4.13. The first mode (a) is the MULT_COEFF state, where the value of the coefficient is multiplied with the intermediate result in the *SYS_OUT* register, the second mode (b) is the MULT_X state where the intermediate result is multiplied with *COMB_OUT*, and the third mode (c) is the addition of the number 1.

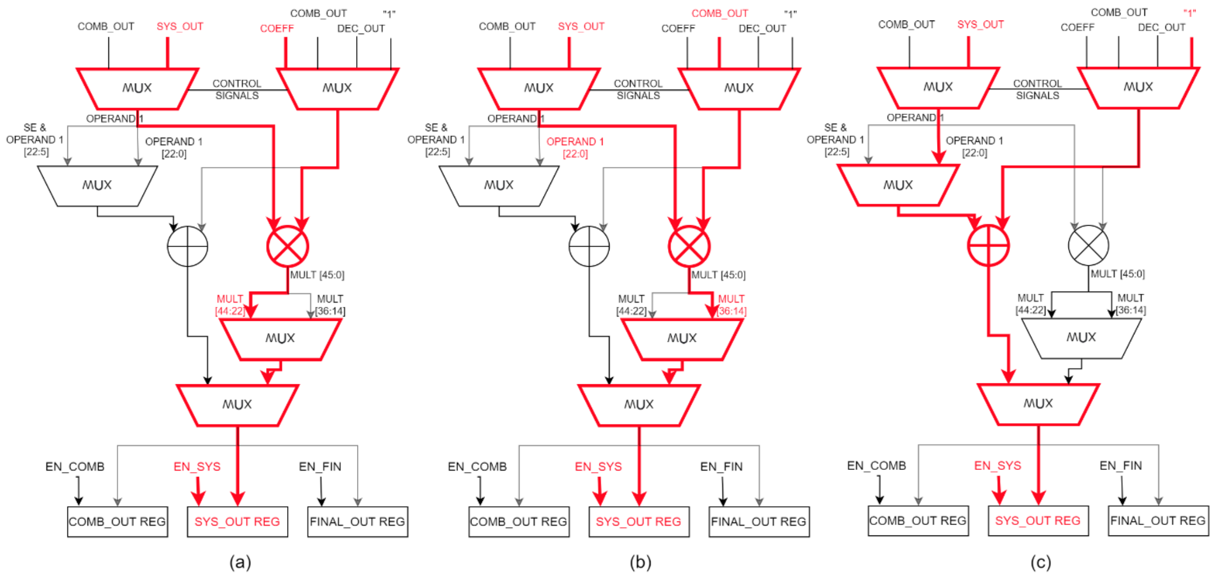

Figure 4.13 Three possible modes of operation for the evaluation of the systematic error removal polynomial

The final step in polynomial evaluation is the addition of the outputs of *SYS_OUT* and *COMB_OUT* registers. That configuration mode of the proposed architecture is shown in Figure 4.14.

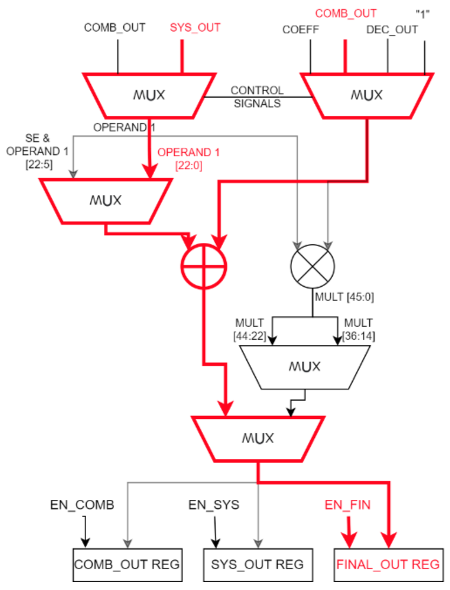

Figure 4.14 Proposed architecture's configuration of the final addition

To simplify control, and save area with registers for FSM states, above mentioned FSMs can easily be combined. The final state diagram is shown in Figure 4.15. If there is a reset at any moment of evaluation, it is automatically returned to the IDLE state. The process only starts when global FSM raises the *POLY_START* signal, once the polynomial evaluation is done, the *POLY_DONE* signal is raised high.

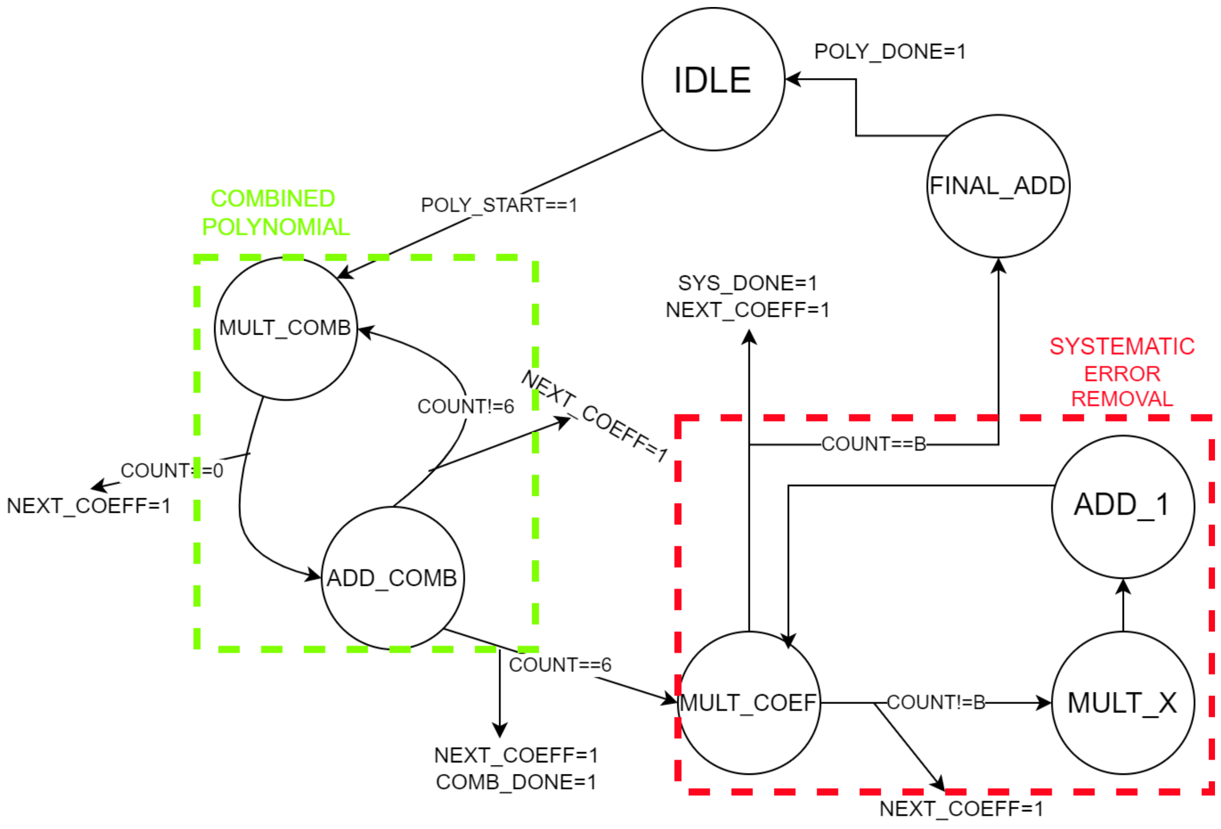

Figure 4.15 Final FSM state diagram

Summary of all the fixed-point representations of the intermediate signals and type of operands that are chosen in every state can be seen in Table .

Intermediate result register	Fixed-point representation (NOB integer part, NOB fractional part)[2]	FSM state	Operand 1	Operand 2
COMB_OUT	*COUNT*<4 =>(3,19)	MULT_COMB	*COMB_OUT*	*DEC_OUT*
	COUNT>=4 =>(8,14)	ADD_COMB	*COMB_OUT*	*COEFF*
SYS_OUT	(2,20)	MULT_COEFF	*SYS_OUT*	*COEFF*
		MULT_X	*SYS_OUT*	*COMB_OUT*
		ADD_1	*SYS_OUT*	*"1"*
FINAL_OUT	(8,14)	FINAL_ADD	*SYS_OUT*	*COMB_OUT*

[2] It can be noticed that sum of NOB for integer and fractional part is equal to 22, the one extra bit in 23 bits is reserved for the sign

Table 4.1 FSM and control signal summary

4.3 SPI

As explained in Section 3.3, the SPI module provides three functionalities: the ability to send out the processing results, the ability to read and store the desired coefficient values, and the ability to provide the polynomial engine with their value.

In Section 3.3, a basic overview is given, here in the following subsection, a detailed explanation of the organization of the internal registers and their control signals is given. At the end of this subsection, the design solution for the clock-domain crossing problem is given.

4.3.1 Internal registers & Multiplexer

The first 8 bits represent the coded desired functionality (see Figure 4.16), and as described, only 4 bits are needed (adr3-adr0 in yellow), therefore there is no need to store all 8. This could easily be implemented by having a 4-bit shift-in register and a 3-bit counter that counts 8 cycles of shifting in. As the address bits are shifted in with MSB first, the adr7 to adr4 will be shifted out and lost, and the remaining 4 bits in the register will be the desired adr3-adr0.

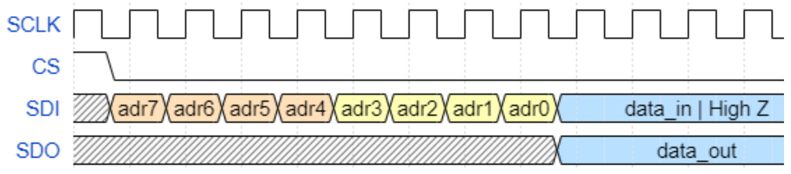

Figure 4.16 SPI protocol: reading the desired function/address

As described previously in Section 3.3, the coefficients and results will be stored in the shift registers. The basic architecture is shown in Figure 4.17. On the left side, *ADDR* represents a 4-bit shift register storing the desired function and its value is decoded to 16-bit one hotkey, which represents shift enable signal. On the right side, the same 4-bit shift register is used to multiplex LSB of only one register. Even though the values of *COEFF* shift registers are write-only from the perspective of the off-chip user, shifting out via *SDO* of their old value(while their new is being shifted in through *SDI*) is allowed for debugging purposes.

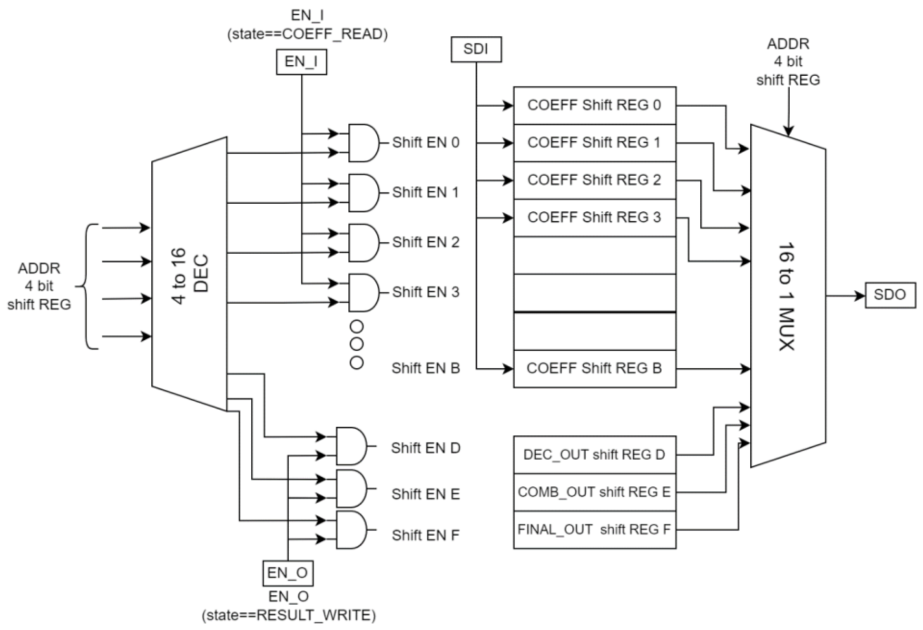

Figure 4.17 SPI module's basic architecture

To protect from shifting of the undesired registers, while the first 4 discarded bits are being shifted in and out, two enable signals are added *EN_O* and *EN_I*. They are only active if the 8 bit has been shifted in, and the state machine goes to one of the mentioned states.

The first functionality: sending the result off-chip. An example of the final temperature result being written out is shown in Figure 4.18. This example illustrates the activated signals after the input address/function has been shifted. Once this functionality is started, even if a new value of the result arrived from the polynomial engine, the shift-out register is not updated until the communication cycle finishes.

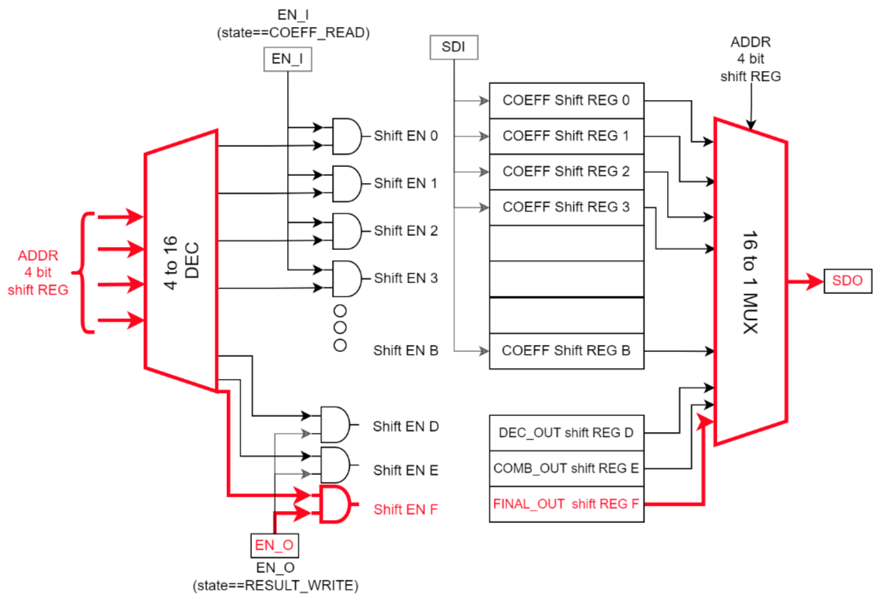

Figure 4.18 An example of sending the final result via SDO

The second functionality: setting the coefficients' value. An example of reading the coefficient 2 register's value is shown in Figure 4.19. Bolded in red are activated signals after the first 8 bits have been shifted in. Bolded in blue are activated signals that are added for the debugging purposes.

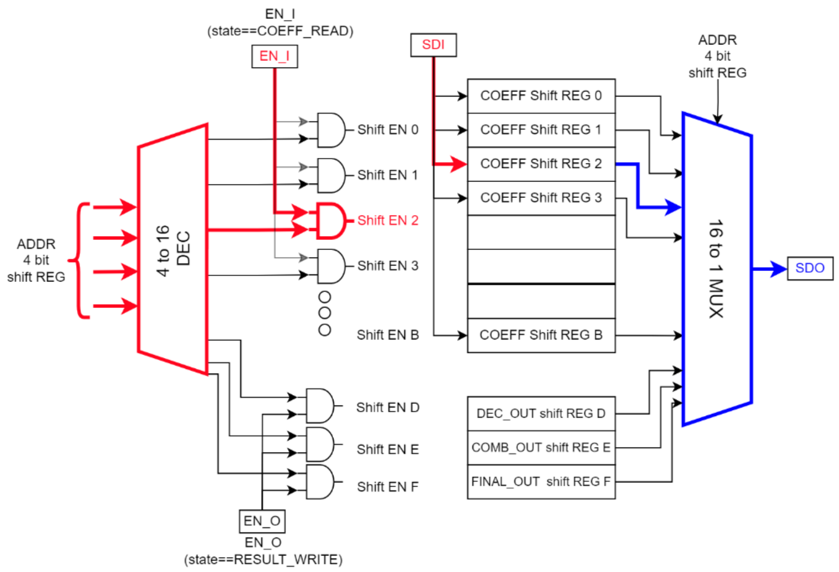

Figure 4.19 An example of setting a coefficient's value, activated signals for the functionality (red), and added debug signals (blue)

Above mentioned architecture has omitted one multiplexer that is used for the third functionality. The third functionality is providing the coefficients' value to the polynomial engine. For this functionality, there is no need for control signals and it is done asynchronously. The design is shown in Figure 4.20. It is seen as an internal flash memory.

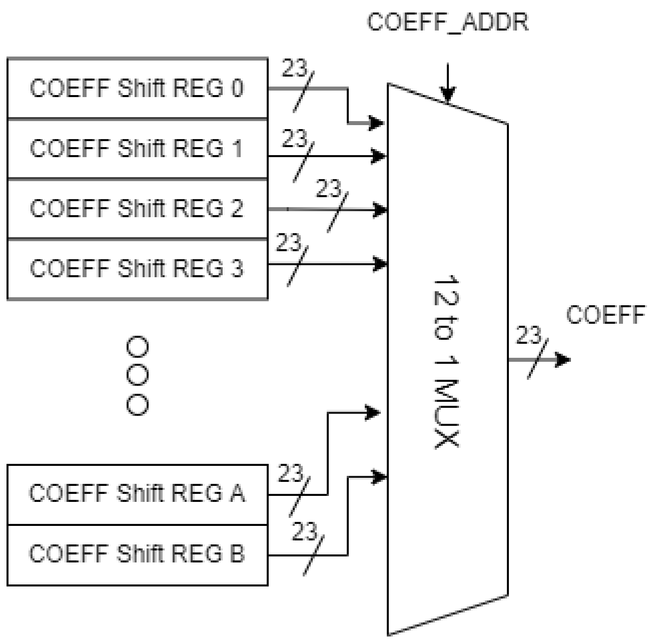

Figure 4.20 Functionality of providing the polynomial engine with coefficients

Coefficients are ordered by the time of their use for convenient design. If the *COEFF_ADDR* has an address higher than hexadecimal B, the output *COEFF* is 0. After reset, coefficients' values are not set to 0, but to a statistical average obtained from a chip batch, for providing functionality (although with degraded performance) after reset without the need for setting the coefficients.

The potential problem of the polynomial engine reading the value during the COEFF_READ state while the same value is updated is solved by the global control module that produces a reset to all of the modules apart from the SPI, while any of the coefficients are being read. If the input function/address is hexadecimal C, the module also goes into the state of COEFF_READ and thus resetting the polynomial engine and the decimation filter.

4.3.2 FSM

There are two counters implemented, one to count to 8, for the address/function readout and another counter to count until 23, for controlling the other two states before returning to the IDLE. SPI module stays in the IDLE state as long as *reset* or *CS* is high. The communication cycle starts when *CS* drops low. If the *CS* signal, at any time, goes high before the communication cycle is finished, the FSM immediately returns to the IDLE state. Once done, the communication cycle finishes in the IDLE state, and the FSM stays in it until there is again a falling edge of the CS.

The state diagram is shown in Figure 4.21. It is worth noting that the *POLY_DEC_RESET* goes high only in the COEFF_READ state, which goes to the global control signal to produce reset for both the other modules in the digital backend.

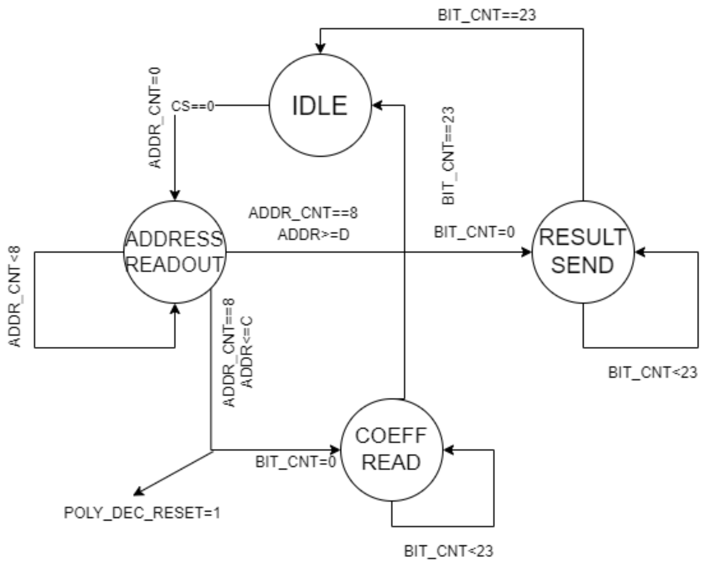

Figure 4.21 State diagram for the SPI module

4.3.3 Synchronization module

Sharing signals and data between two clock domains can cause data incoherency if no specific technique is applied. The data exchange in two different clock domains in the digital backend happens in two different cases.

The first case is when the coefficient's address and value are loaded. This is assumed to be no data change from one side (the coefficient's value is fixed). Since there is no change on one side, it is assumed that the SPI module behaves as asynchronous memory. During the loading of the value of the new coefficient, the SPI module resets the rest of the design, making sure the value of the coefficients is not used. In this way, the usage of wrong coefficient values is averted.

The second case is when the results values from the polynomial engine are sent to the SPI module, so they could be stored in a shift-out register and shifted out if necessary. Here, there is a need for a synchronization technique.

The problem with just accepting data/signals from another clock domain with clocked inputs is illustrated in Figure 4.22. The metastability problem of data on the receiving side could occur if data is sampled while changing. The indefinite value (neither high nor low) can cause metastability in the receiving flip flop, and it might happen that the value does not stabilize during the duration of the clock period.

The simplest solution to this problem would be adding another flip-flop on the receiving side, and thus making a pipeline of two stages of receiving signal. This would highly reduce the risk of metastability. The solution with the signal waveforms is shown in Figure 4.23.

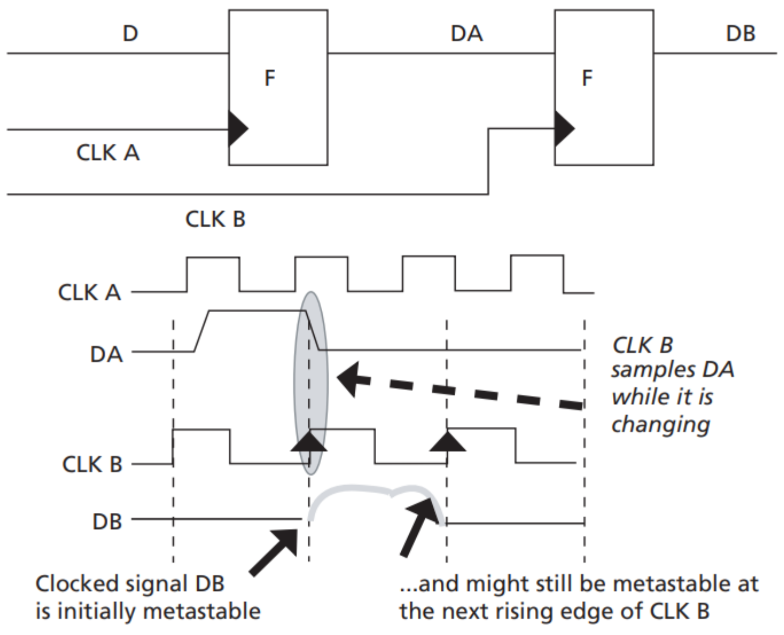
D
DA
DB
F
F
CLK A
CLK B
CLK A
DA
CLK B
DB
CLK B samples DA while it is changing
Clocked signal DB is initially metastable
...and might still be metastable at the next rising edge of CLK B

Figure 4.22 Metastability problem of clock domain crossing [20]

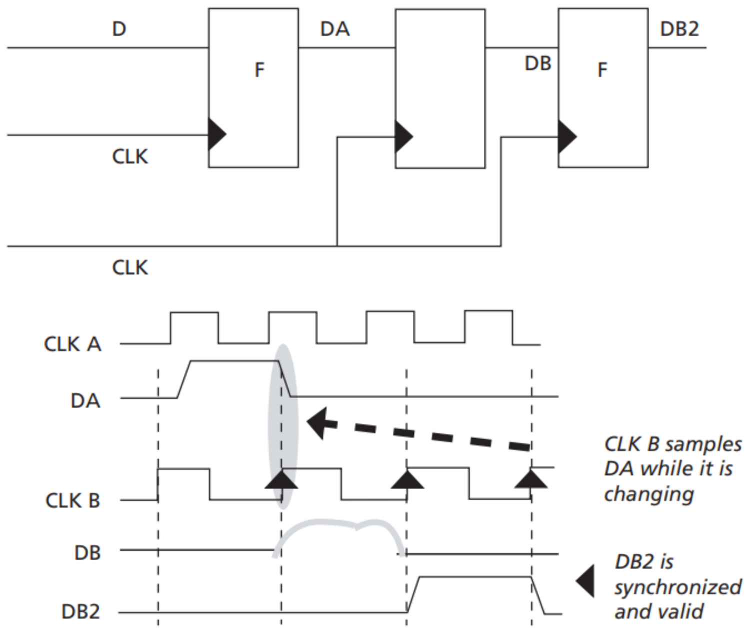
D
DA
DB2
F
DB
F
CLK
CLK
CLK A
DA
CLK B
DB
DB2
CLK B samples DA while it is changing
DB2 is synchronized and valid

Figure 4.23 Synchronized solution with two flip-flops [20]

In the case of transferring multi-bit data (23-bit results), it can happen that not all the bits are changing at the same time, and it increases the chance of metastability. Even though the metastability of each bit would be solved with two flip-flops, some of the arriving bits might be delayed for one clock cycle. Delaying only few incoming bits can change the received data.

The simple solution for this problem is that with every data intended to send, there is one control signal that becomes high when data changes. This signal is transferred to the other domain with two flip-flops and controls a multiplexer that chooses which value to be stored in a register: the previously stored one or the one arriving from the other clock domain. After the control signal has passed two stages of flip-flops it is assumed that the 23-bit register has stabilized its output.

The implemented solution is shown in Figure 4.24. This solution is called MUX-based synchronization. Once signal *FINAL_RECEIVED* becomes high, it stores the value from the other clock domain. As part of the handshake protocol, *FINAL_RECEIVED* goes back to the polynomial engine's clock domain and clears the *FINAL_DONE* control signal. Then in the next 2 SPI clock cycles, *FINAL_RECEIVED* will also be cleared, and the final temperature's shift register will be latched, and protected from future changes in the other clock domain. As temperature measurement result registers change with the frequency of order 100Hz, this handshake protocol's time overhead is insignificant.

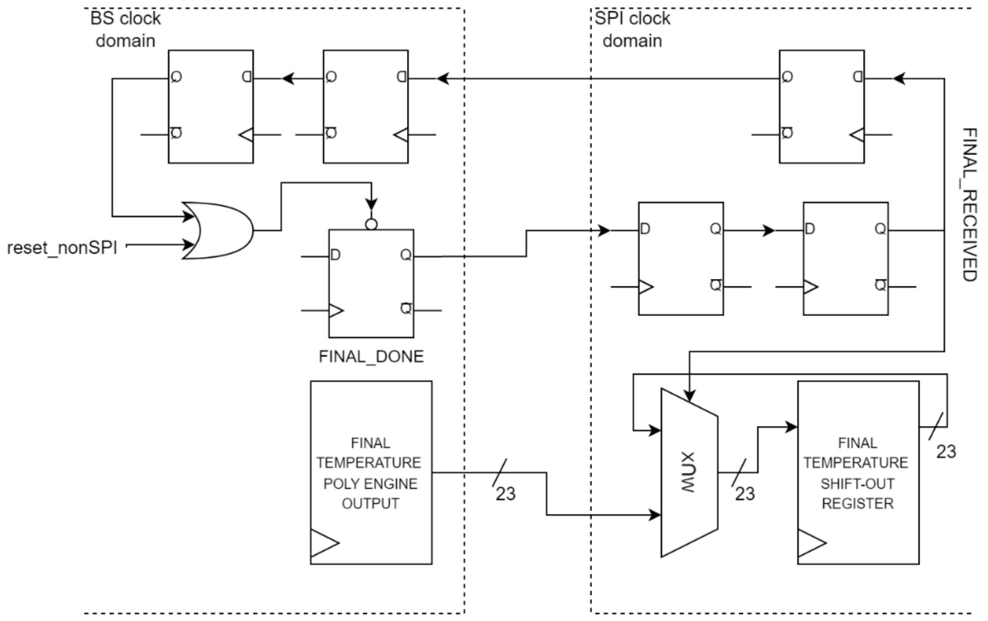

Figure 4.24 MUX-based implementation, example for final temperature result

This solution has been implemented for all three results that are being sent from the polynomial engine to the SPI module: the output of the decimation filter, the output of the combined polynomial, and the final temperature value.

4.4 Global FSM & Control

The whole system has an input signal *reset*, that signal directly goes to the SPI module, and it resets also the coefficient value. That *reset* in combination with the signal produced by the SPI module- *POLY_DEC_RESET* combined make a reset signal for the rest of the design- *reset_nonSPI*. The whole design with a focus on reset signals is shown in Figure 4.25.

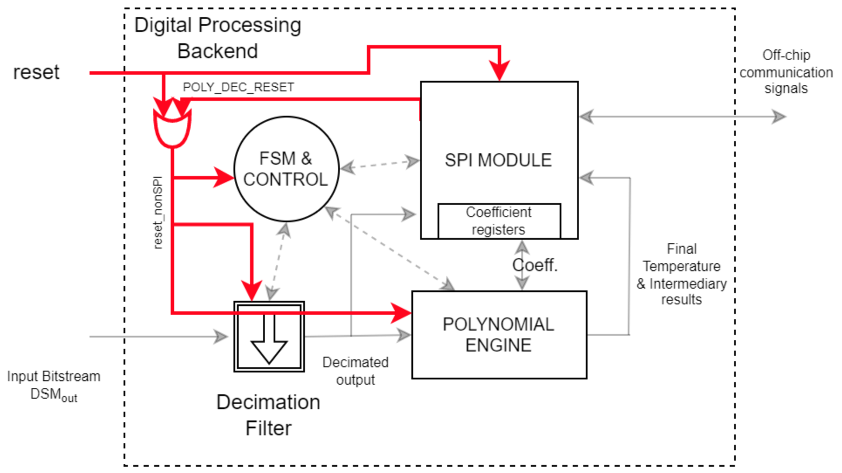

Figure 4.25 Overall architecture with a focus on reset signal

The overall state diagram of the global state machine is shown in Figure 4.26. As mentioned in Section 4.1.3, the decimation filter does not require control signals from the global FSM, it runs continuously. Therefore, there are only two states. Once one decimation cycle is done, the decimation filter raises the *DECIM_DONE* signal. Global FSM uses this signal to start the polynomial evaluation by raising the *POLY_START* signal. Once the polynomial evaluation is done and the polynomial engine returns the *POLY_DONE* high. The FSM waits in the IDLE state until the next decimation cycle finishes.

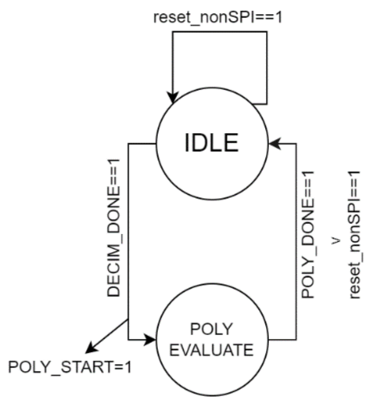

Figure 4.26 State diagram for the global FSM

Global control signals are summarized in the following table:

Module	Input control signal	Output (control) signal
Decimation filter	/	*DECIM_DONE*
Polynomial engine	*POLY_START*	*POLY_DONE*
SPI	/	*POLY_DEC_RESET*

5 Design and verification

This section focuses on the verifications which are used to guarantee a successful tape-out. The step-by-step design and verification flow are presented in Figure 5.1.

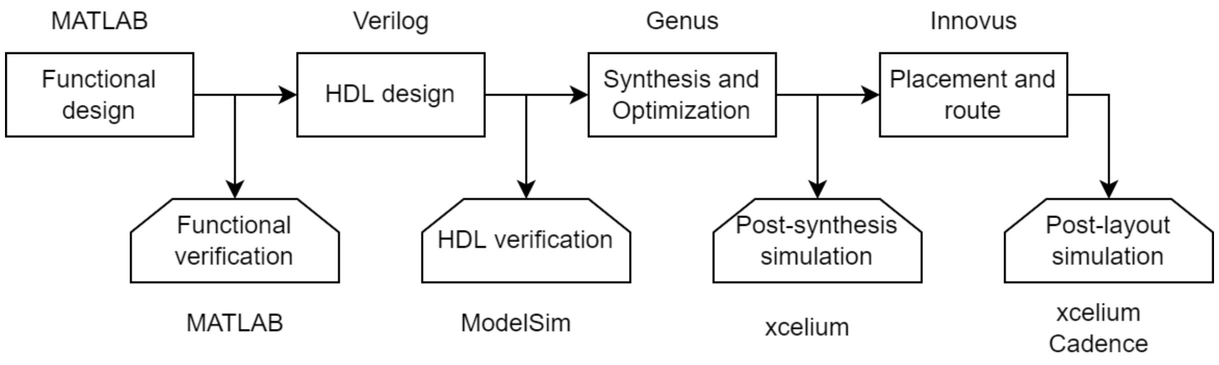

Figure 5.1 Design flow for the digital processing backend

5.1 Functional design & verification

The functional design, including that of the decimation filter and the polynomial engine, is done in MATLAB with the help of the fixed-point toolbox '*fi*'. The verification of the decimation filter has been presented in Section 3.1.2.

As for the polynomial engine, its functionality is verified using measurement results from 40 sensors presented in [ref]. As 10 temperature points (ranging from -55 to 125℃) were measured for each sensor, 400 polynomial evaluations are done. Compared to the temperature derived from double-precision calculations, the fixed-point polynomial will generate a small error, whose distribution is shown in Figure 5.2. The quantization noise added by the polynomial engine is ~145μK.

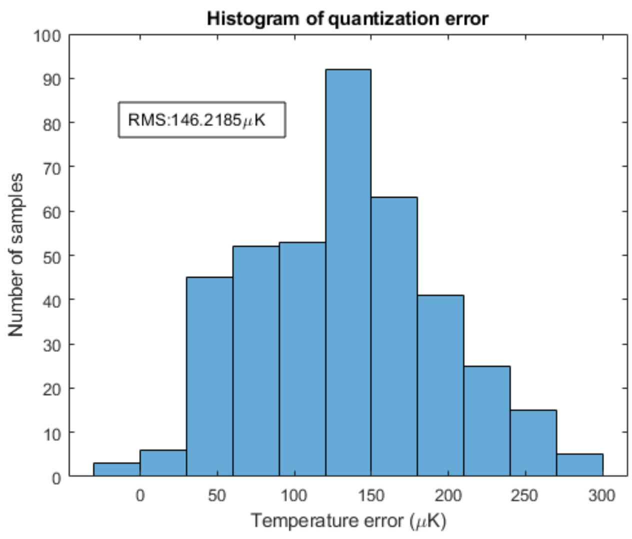

Figure 5.2 Quantization noise introduced by the polynomial engine

As a sanity check, 40 new virtual "sensors" were created using randomly tweaked coefficients (±10%). With the same 10 temperature points, 400 more points were used to evaluate the polynomial engine, and the characteristics remain the same.

5.2 HDL implementation & verification

After the functional designs shown in Figure 5.1, the decimation filter, the polynomial engine, the SPI module, and their integration are implemented using Verilog, a hardware description language. After so, the verifications are done using Modelsim.

For the decimation filter and the polynomial engine, Modelsim simulation achieves exactly the same result as that from Matlab simulations.

The SPI module is checked in two aspects. First, after writing a newer set of coefficients, the previous set is verified to be correctly shifted out. Also, the SDO output is checked to be correct.

Extreme care should be taken when integrating synchronous and asynchronous circuits. For this design, the reset asynchronous signal should be thoroughly checked. For all three modules, simulations have been done with the *reset* signal going high during all possible state transitions in their respective FSMs. After the *reset* was dropped to 0, the functionality of all three components remained correct.

5.3 Synthesis

The Genus software solution is used for synthesis. Afterward, Xcelium is used to verify the functionality of the synthesized design.

The estimated area of the synthesized design is presented in Table 5.1:

Module/ design part	Total cell area (μm^2)
Decimation filter	5 407
Polynomial engine	17 922
SPI module	30 775
Whole digital processing block	52 744

Table 5.1 Total cell area of the synthesized design block-wise

5.4 Placement & route

The Innovus tool is used for placement and routing of the synthesized design with the standard digital library cells in 180nm TSMC technology. The layout was generated for separate blocks for verification purposes and as an integrated digital backend. The tool outputs: the interconnection delay between the standard cells, which can be imported into Cadence for verifications.

After obtaining the interconnection delay information, post-synthesis simulations are performed using Xcelium. The designed blocks have been verified to be robust to different delay corners, and are free from setup and hold-time violations.

The area of the implemented design, area density without the fillers, and time delay of the critical data path are shown in the following Table 5.2. The critical data path is the longest signal path from the output of one register to the input to another register. The critical path determines the highest possible operating frequency.

Module	Area(μm^2)	Area density (%)	Time delay of critical data path (ns)
Decimation filter	6 889	85.8	-
Digital processing backend (with dec. Filter)	67 600	82.2	24.454

Table 5.2 Area and delay results of the placement and route

With a 24.454ns maximum delay, the digital block can operate at a maximumly 40MHz, which is well above the designed frequency of 500KHz.

The layout is imported into Cadence, and parasitic resistors and capacitances were extracted. The mixed-signal simulations are run with the same test bench on both the decimation filter and the digital backend to verify the functionality and the power consumption.

The circuit's power consumption is simulated in Cadence after post-layout extraction of parasitic resistors and capacitors. At 500kHz, the power of different blocks are shown in Table 5.3:

Module	Simulated power consumption (µW)
Decimation filter	4.16
Digital processing backend	20.41

Table 5.3 Simulated values of power consumption

The final area and simulated power consumption comparison to the requirements set for this project in Section 1 are shown in the following table:

Digital processing backend	Area (mm^2)	Power (µW)
Implemented and simulated	0.068	20.41
Requirement	<0.12	<60

Table 5.4 Simulated power and implementation area compared to the project requirements

Figure 5.3 shows the signal waveforms related to the decimation filter. After a negative edge of the reset signal, the decimation filter output will be ready in 8.2ms, and the output will be synchronized by a decim_done signal.

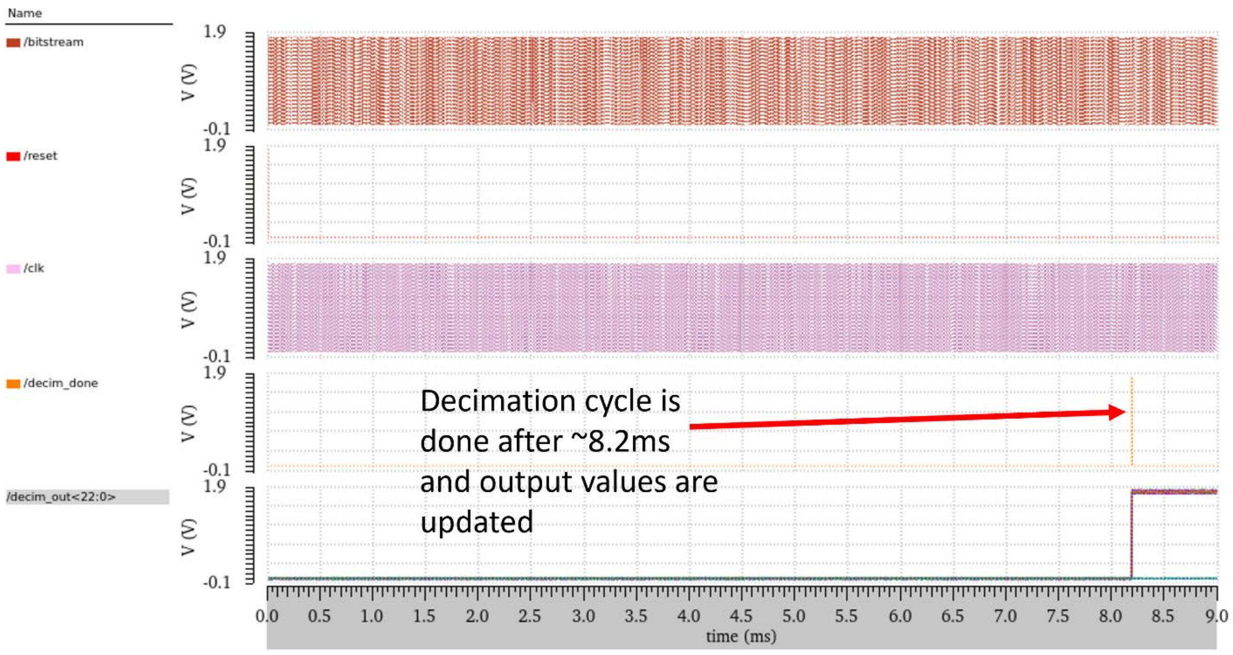

Figure 5.3 Post-layout simulation of one decimation cycle

6 Measurement Setup & Evaluation

6.1 Chip layout

The chip was fabricated in a TSMC 180nm CMOS technology. The chip layout is shown in Figure 6.1. Two sensors are placed on one chip to facilitate differential resolution measurements. Each sensor occupies an active chip area of 0.19mm^2 (0.12mm^2 analog frontend and 0.068mm^2 digital)

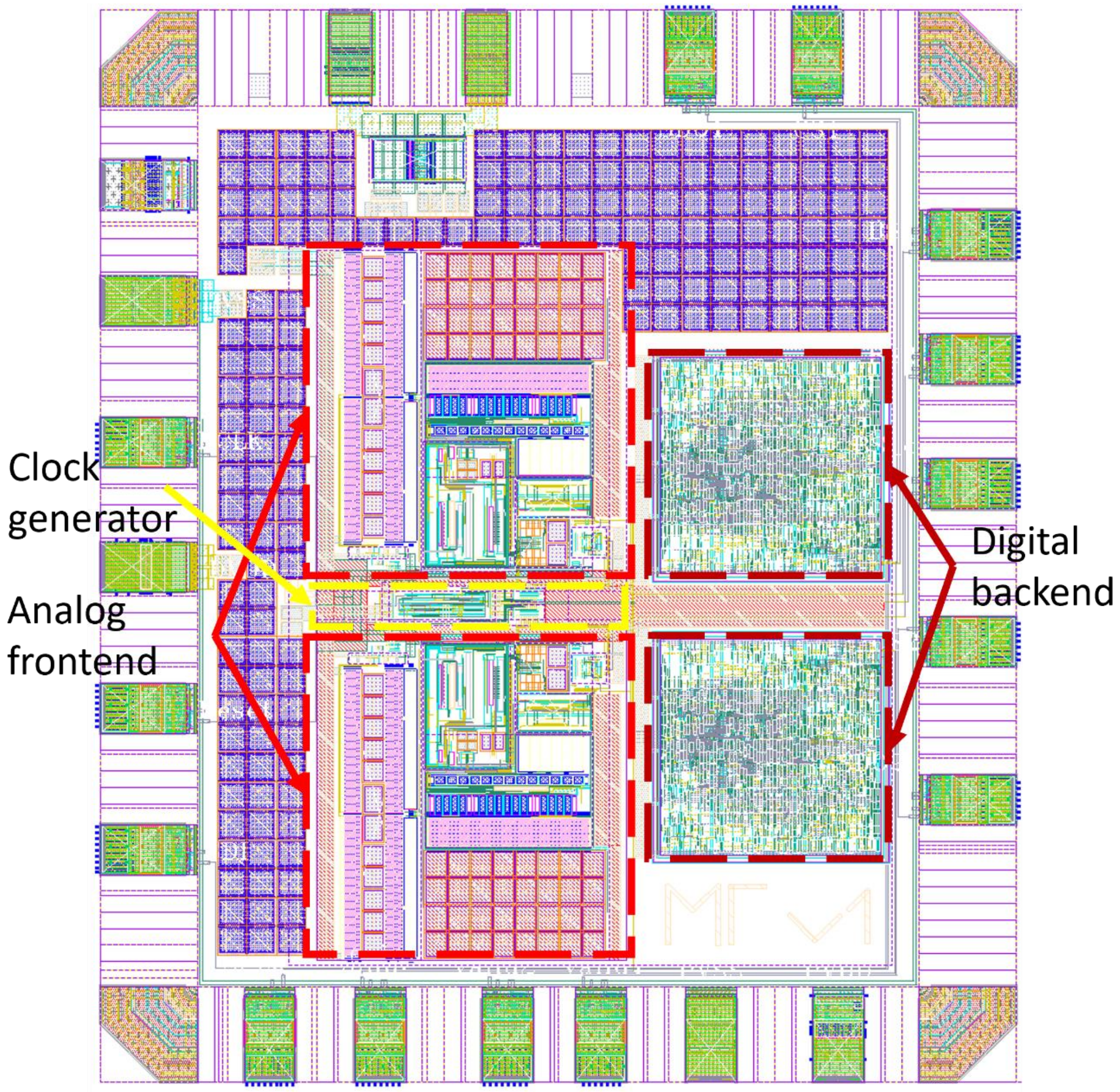

Figure 6.1 Final chip layout

6.2 Measurement setup

The measurement setup is shown in Figure 6.2. To achieve high sensing resolution, the frequency reference is provided by a low-noise function generator (Keysight 33500B), whose jitter is lower than 1ps. An NI DAQ card is used to send control signals to the PCB and SPI inputs for the designed chips. It also receives BS and SPI output from chips, such that the digital circuit's performance can be verified by comparing the temperature outputs generated on/off-chip. As in [8], a calibrated Pt-100 is used as the temperature reference, which resistance is readout by a Keithley 2002 Multimeter. To achieve good sensing inaccuracy, both the sensors (with ceramic packages) and the Pt-100 are mounted in good thermal contact with a large aluminum block inside a temperature-controlled oven.

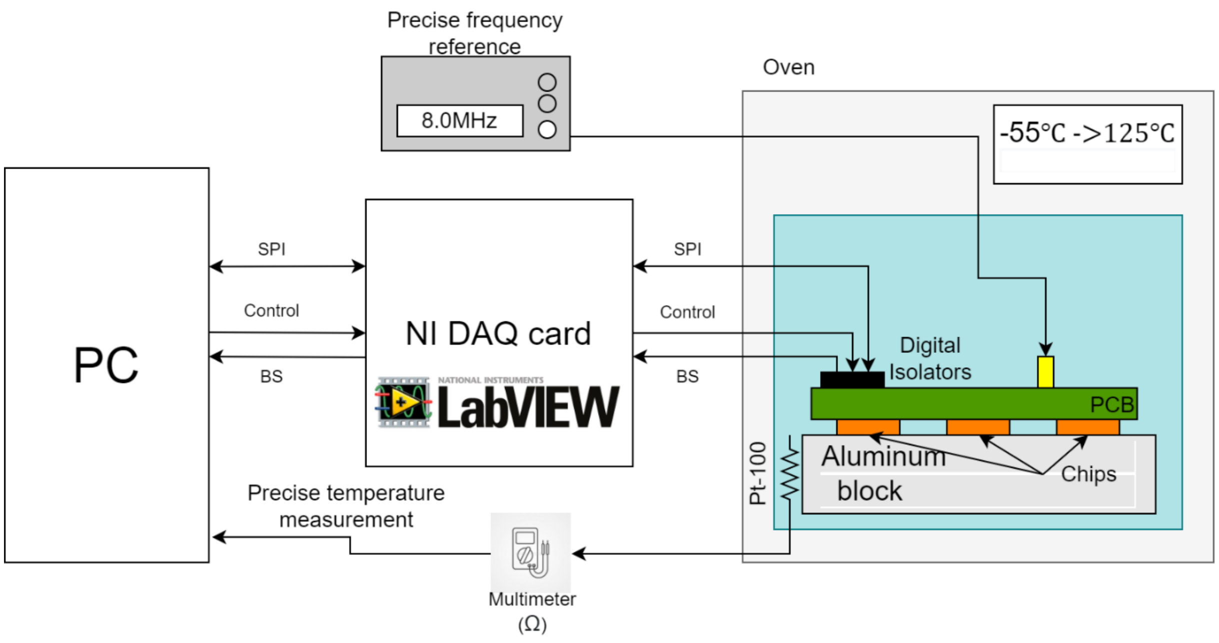

Figure 6.2 Measurement setup

6.3 Measurement results

Power consumption is obtained by measuring the current consumption in 2 cases: with and without active SPI communication. The power performance of the digital backend is shown in the following table:

Power (µW)	Simulated	Only decimation filter & polynomial engine (SPI module inactive)	Full functionality when SPI sends all 3 results per decimation cycle
Digital backend	20.4	28	40.6

Table 6.1 Simulated and measured power of the digital backend

6.3.1 Functionality

Firstly, the functionality of the SPI was tested: writing the coefficients onto the sensor. In Figure 6.3, writing the coefficient c_2 of the combined polynomial is shown. The signal CS, SDI, and SDO were captured by the oscilloscope. When CS goes low, the first 8 bits shifted in via SDI represent the functionality of the initiated communication(yellow and red in the upper part). To be precise, only the last 4 bits (red- "0100") are representing a code (hexadecimal 0x4) that 5th coefficient is being written.

Shifting in "0100" as the last four bits means that the next 24 bits are replacing the previous coefficient value c_2 of the combined polynomial (coefficient with address 0x4 in the SPI module). The following 24 bits in green represent the new coefficient value. Simultaneously, the old coefficient value is being shifted out via SDO (lower part), the bits in green (lower waveform) represent the previous coefficient value. The output value of the SDO is valid after the first 8 cycles after CS drops low (right of the dotted blue line).

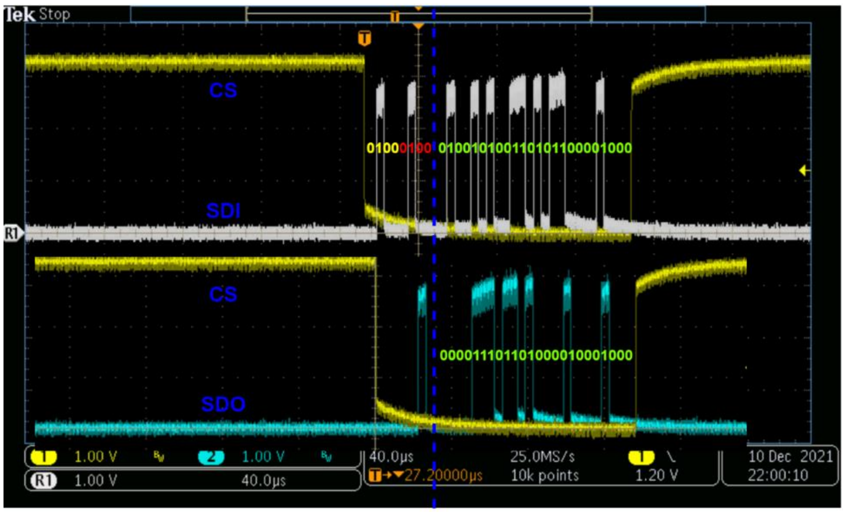

Figure 6.3 Observed and measured example of the SPI communication

Setting all the coefficients in the sensor is tested, and writing different sets of coefficients onto two different sensors on one chip as well. The newly written coefficients onto the sensor produce the same result that is produced by the MATLAB model of the polynomial engine using the new coefficients. With this, the functionality of writing the coefficients is verified.

To verify the functionality of the polynomial engine the testing scheme shown in Figure 6.4 is used. The decimated value, the intermediate and final results of the polynomial engine are read out via SPI. The decimated value is passed to a MATLAB fixed-point model of the polynomial engine. The results of the MATLAB model are then compared to the results read via SPI. They produce the exact match, both the final temperature and the combined polynomial's result.

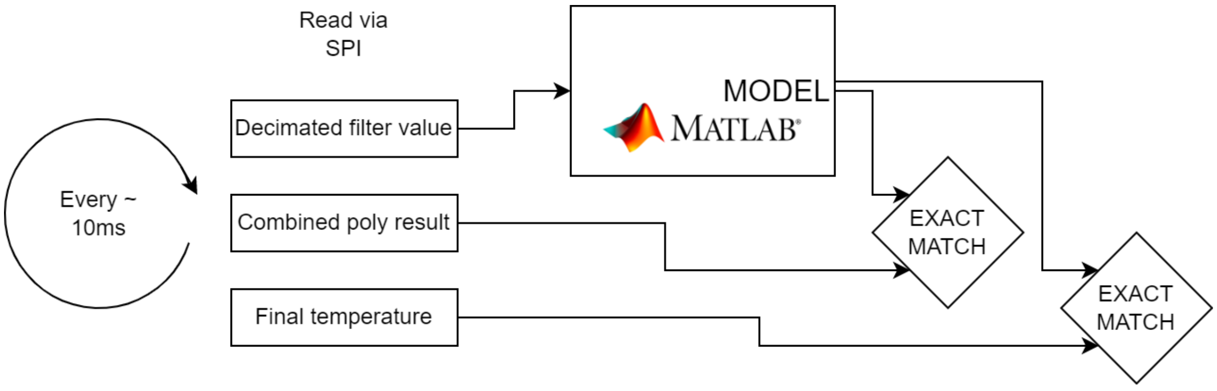

Figure 6.4 Test strategy for the polynomial engine (functionality)

Synchronization of the clock domains of the SPI module and the BS clock is verified to be working on the following frequencies of SPI (at fixed 500kHz BS clock frequency): 1MHz, 800kHz, 500kHz, 250kHz, 125kHz, and 50kHz.

6.3.2 Resolution

To calculate the resolution of the polynomial engine the testing scheme in Figure 6.5 is used. The same strategy is used as described in the previous subsection, only the MATLAB calculation is done using double-precision instead of the polynomial engine fixed-point model. By using the decimated value read out via SPI and not the one obtained by the bitstream, the influence of the decimation filter on the results is bypassed.

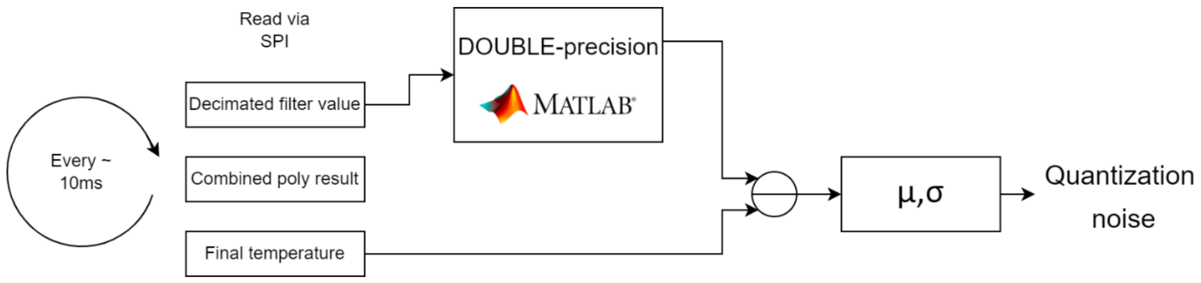

Figure 6.5 Test strategy for the polynomial engine (resolution)

The quantization error of the final temperature (polynomial engine) at around room temperature is shown in Figure 6.6, and it is ~130μK. These values are, as expected, similar to the ones obtained for the whole range from the simulation described in Section 5.1.

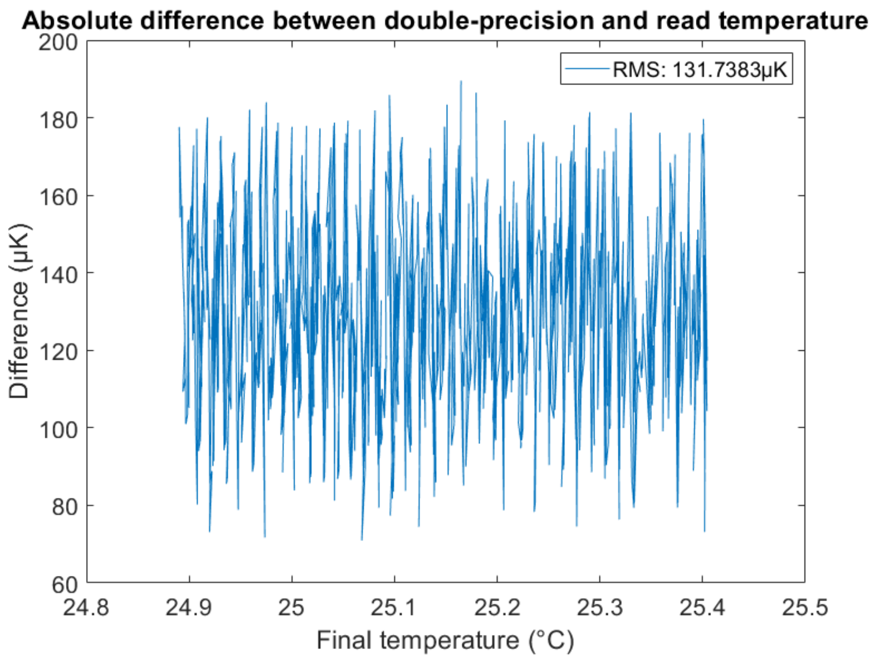

Figure 6.6 Difference between polynomial engine read results and ones obtained by double-precision MATLAB calculation

The resolution of the bitstream that can be achieved depending on a certain conversion time is shown in Figure 6.7. This figure was obtained from sampling bitstream for ~10 seconds at constant room temperature (27℃). The recorded bitstream of each sensor is divided into 10 intervals of 1 second to reduce the effect of ambient temperature drift as described in [21], to further reduce the effect of the drift, the difference between two sensors' outputs on one die is used in the same 1-second intervals [22]. For the conversion time of 8.2ms, which is the duration of the decimation cycle at 500kHz, the average (for 16 sensors) thermal noise floor limited resolution at 27℃ is 490μK.

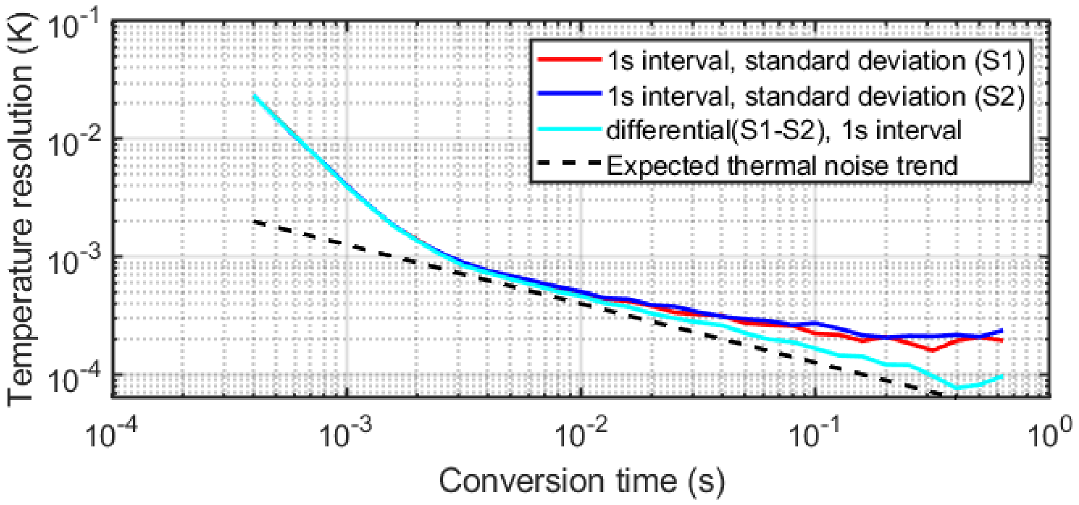

Figure 6.7 Temperature resolution dependence versus conversion time (averaging done on 1-second intervals)

To measure the total resolution of the digital backend, 500 consecutive samples of final temperature are read via SPI at constant temperatures over the military range as shown in Figure 6.8. The standard deviation of the difference of two consecutive measurements is determined at each temperature point for each sensor. The standard deviation of each individual measurement is $\sqrt{2}$ times smaller than that of the difference.

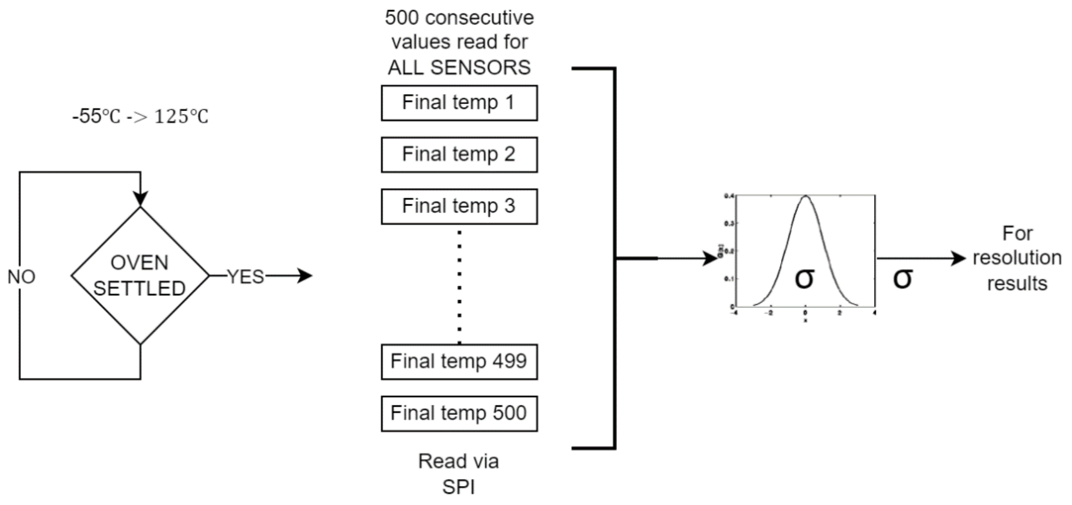

Figure 6.8 Measurement setup for final resolution

Final temperature measurement resolution over the temperature range (-55℃->125℃) is shown in Figure 6.9. The average resolution at room temperature is 501µK.

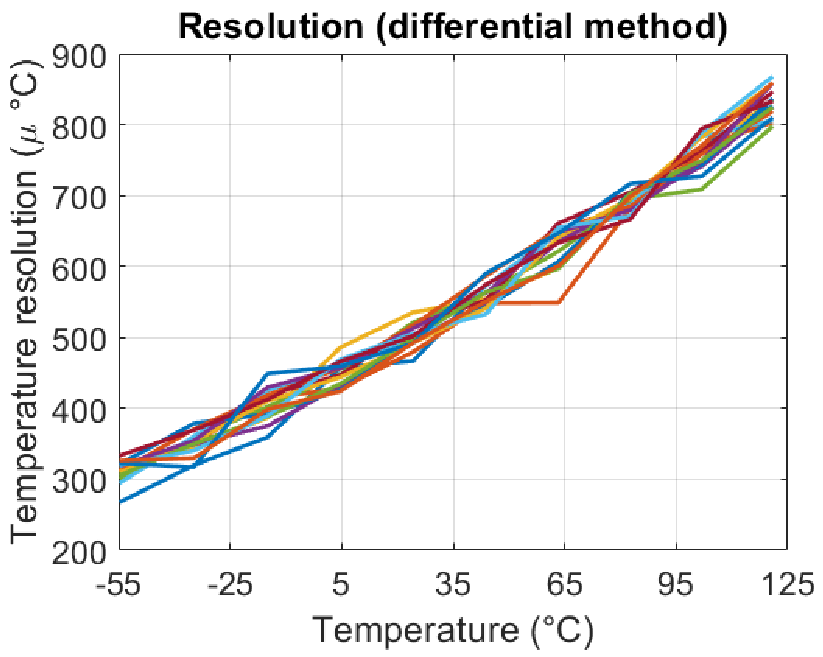

Figure 6.9 Final temperature resolution for 16 sensors

6.3.3 Monotonicity & Accuracy

The sensor's responsiveness to the quick temperature changes was measured by going from the room temperature to the -45℃, then to the 90℃, and back to room temperature, within 3 minutes. The decimated value of both sensors is matched with the decimated value obtained by obtaining BS and processing it in MATLAB.

The decimated value of both sensors on the chip was read simultaneously via SPI, every 100ms. The monotonicity of the decimated value from both sensors is observed (see Figure 6.10).

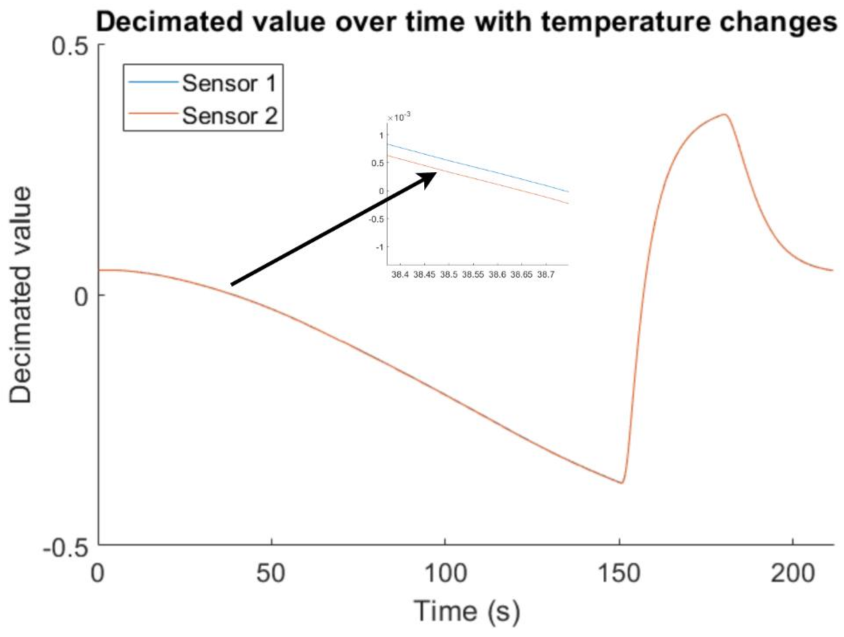

Figure 6.10 Decimated filter change over the time during temperature sweep

Along with the decimation filter's result, the final temperature computed by both sensors' digital backend was also read via SPI and shown in Figure 6.11.

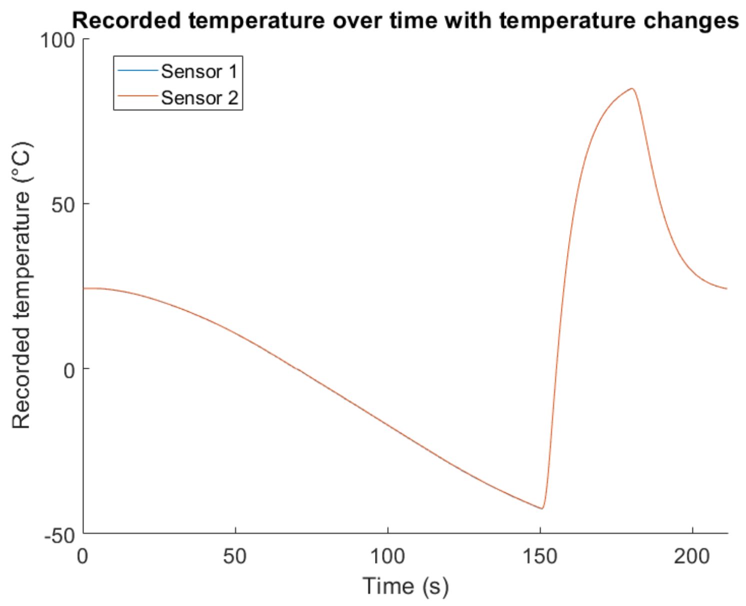

Figure 6.11 Obtained final temperature measurements over time during temperature sweep

After trimming at -35 and 105°C, the inaccuracy of the sensor is shown in Figure 6.12. It can be noticed that there is a systematic nonlinearity. The average inaccuracy of the measurements obtained by the sensors in [8] is also shown. It could be noticed that the SNL in this batch is different than the one observed in [8], thus new SNL removal polynomial coefficients are obtained.

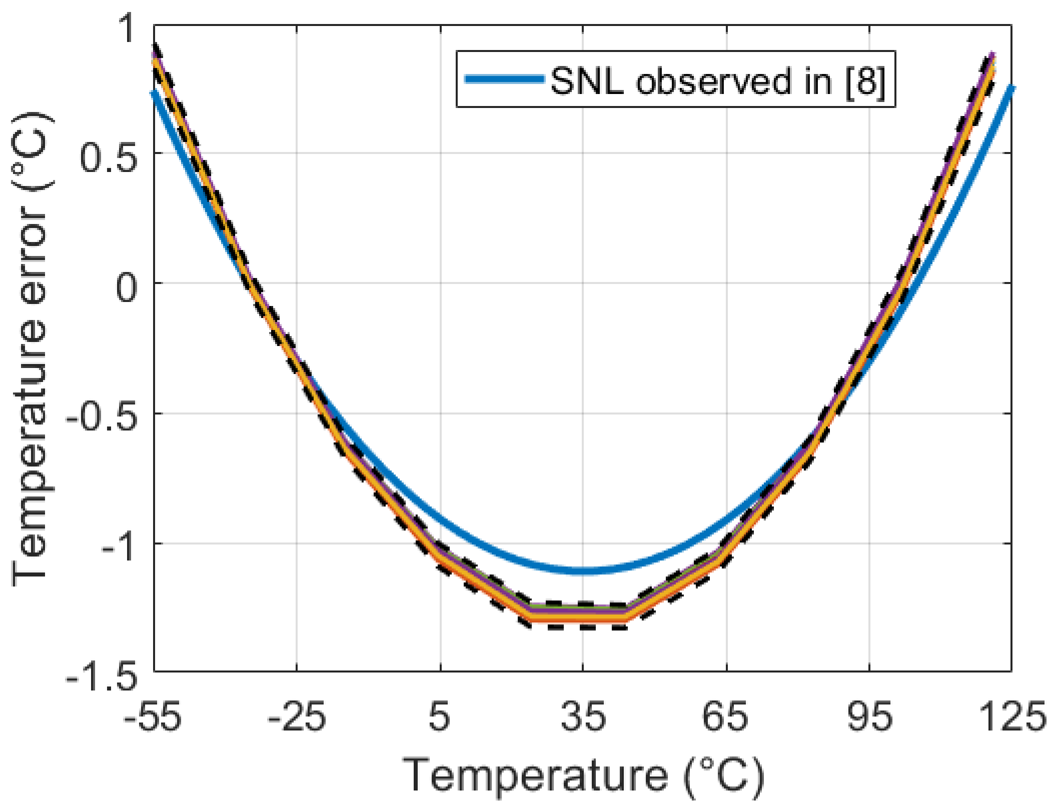

Figure 6.12 Inaccuracy after trimming 16 sensors

After writing the coefficients of the new SNL removal polynomial, the final measurement error on the batch of 32 sensors is shown in Figure 6.13.

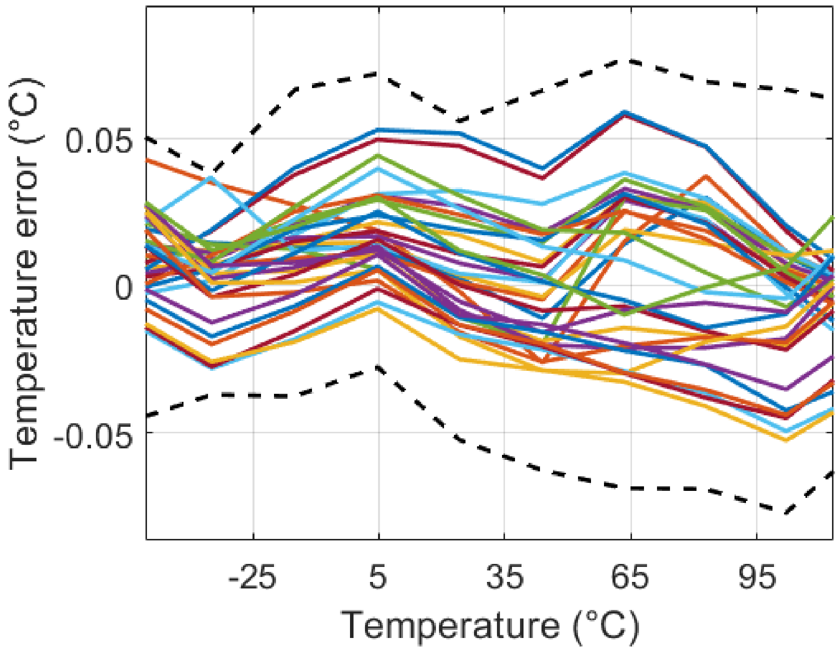

Figure 6.13 Inaccuracy of the final temperature measurement

6.3.4 Result overview

The summary of the obtained results is shown in the following table:

Performance metric	Prototype sensor without digital circuits	Digital circuit requirements	Implemented digital backend	Full design
Area	0.12mm^2	<0.12mm^2	0.068 mm^2	0.19mm^2
Power	66μW	<60μW	28 μW (40 μW fully active SPI)	106μW
Temperature range	-55°C to 125°C	-55°C to 125°C	-55°C to 125°C (can support wider range)	-55°C to 125°C
Resolution	450μK (Thermal-noise limited)	~100μK (added quantization noise)	131 μK (polynomial engine)	501 μK
3σ Inaccuracy	0.03°C	<3mK (added inaccuracy)	-	0.05°C

Table 6.2 Summary of the obtained results

7 Conclusion & Future Work

An energy-efficient, low-area digital backend for resistor-based Wien bridge temperature sensor has been designed. The final achieved area of the digital backend is 0.068mm^2, which is roughly half of the analog frontend (0.12mm^2). The power consumption of the digital backend (28µW) is also lower than that of the analog (66µW). After integration, the sensor's resolution and inaccuracy performance is still dominated by the analog part. This work opens the possibility of integrating digital backends of state-of-the-art temperature sensors with little power and area overhead.

7.1 Future work

7.1.1 Increasing the range of the polynomial processing.

The existing combined polynomial output, with its current fixed-point representation, can support a temperature range of more than ±200°C without overflowing. However, the systematic error removal polynomial limits the final temperature range to -90 to 160°C, beyond which the systematic error becomes larger than ±4°C, which is not supported by the existing fixed-point representation. As a result, the straightforward fix would be simply adding more bits to the systematic error removal polynomial.

7.1.2 Reusing the existing design for different temperature sensors

This design strategy could be applied to the other types of temperature sensors, e.g., Wheatstone bridge sensors [23], in which the digital processing flow is shown in Figure 7.1. Digital processing also includes decimation, trimming, and systematic nonlinearity removal. Compared to that of the Wien-bridge sensor, the systematic polynomial order of the Wheatstone bridge sensor is larger, but no µ-R translation polynomial is required.

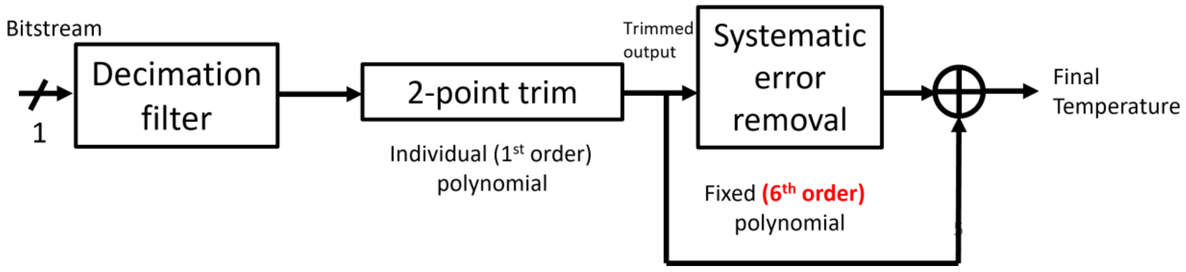

Figure 7.1 Data processing flow of the Wheatstone bridge temperature sensor

For the measurement data obtained in [23], the possible achieved accuracy for only two consecutive polynomial evaluations is shown in Figure 7.2.

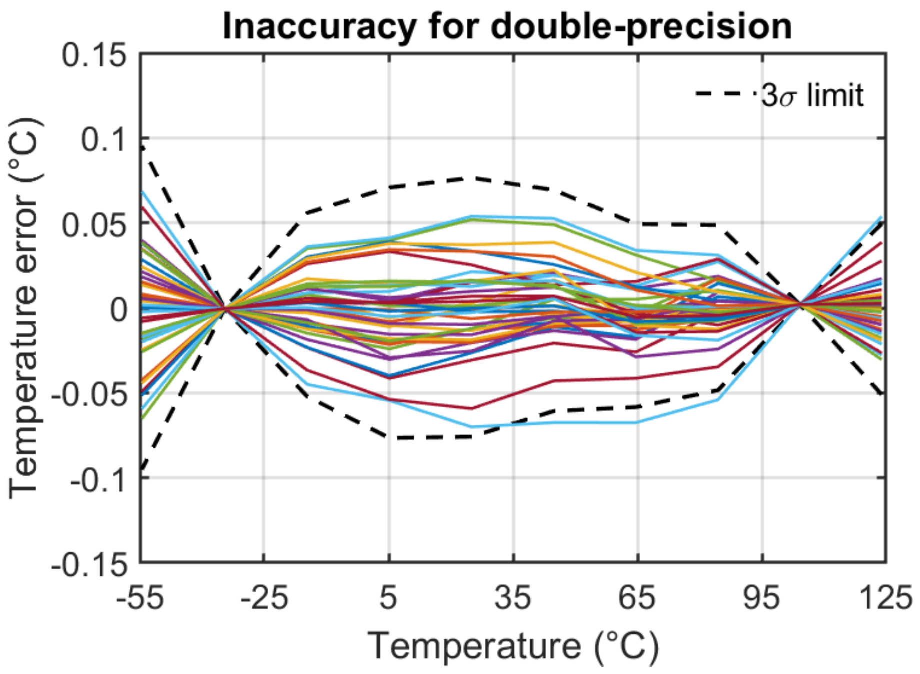

Figure 7.2 Accuracy of Wheatstone bridge resistor-based temperature sensor implemented in [21]

7.1.3 Improving the usability of the sensor

In this design, the polynomial coefficients need to be programmed every time after powering up. To avoid this and improve the usability of the sensor, the coefficient shift registers can be replaced by programmable non-volatile memory cells.

Also, as the digital backend is tested to be fully functional, the number of pins could be reduced to allow a smaller package. In principle, 7 pins should be sufficient: 2 for power supply, 4 for SPI communication, and 1 for the precise external clock signal. In the case of sensors that have relaxed requirements on the jitter and accuracy of the reference clock, e.g., the Wheatstone bridge sensors, a simple RC-based oscillator can be implemented on-chip, and the total number of pins could be further reduced to 6.

8 References

[1] K. Makinwa, "Smart temperature sensors in standard CMOS", *Procedia Engineering*, vol. 5, pp. 930-939, 2010. Available: 10.1016/j.proeng.2010.09.262

[2] S. Pan, "A High-Resolution Energy-Efficient Resistor-Based Temperature Sensor", MSc, TU Delft, 2016.

[3] S. Pan and K. Makinwa, "Energy-Efficient High-Resolution Resistor-Based Temperature Sensors", *Hybrid ADCs, Smart Sensors for the IoT, and Sub-1V & Advanced Node Analog Circuit Design*, pp. 183-200, 2017. Available: 10.1007/978-3-319-61285-0_10

[4] S. Pan, J. Angevare and K. Makinwa, "5.4 A Hybrid Thermal-Diffusivity/Resistor-Based Temperature Sensor with a Self-Calibrated Inaccuracy of ±0.25° C(3 Σ) from -55°C to 125°C", *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, 2021. Available: 10.1109/isscc42613.2021.9366032

[5] C. Gurleyuk, L. Pedala, S. Pan, F. Sebastiano and K. Makinwa, "A CMOS Dual-*RC* Frequency Reference With ±200-ppm Inaccuracy From −45 °C to 85 °C", *IEEE Journal of Solid-State Circuits*, vol. 53, no. 12, pp. 3386-3395, 2018. Available: 10.1109/jssc.2018.2869083

[6] S. Pan and K. Makinwa, "A 10 fJ·K2 Wheatstone Bridge Temperature Sensor With a Tail-Resistor-Linearized OTA", *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 501-510, 2021. Available: 10.1109/jssc.2020.3018164

[7] S. Pan, *Resistor-based Temperature Sensors in CMOS Technology*, 1st ed. Delft, 2021.

[8] S. Pan, J. Angevare and K. Makinwa, "A Self-Calibrated Hybrid Thermal-Diffusivity/Resistor-Based Temperature Sensor", *IEEE Journal of Solid-State Circuits*, pp. 1-1, 2021. Available: 10.1109/jssc.2021.3094166

[9] S. Pan, C. Gurleyuk, M. Pimenta and K. Makinwa, "10.3 A 0.12mm2 Wien-Bridge Temperature Sensor with 0.1°C (3σ) Inaccuracy from -40°C to 180°C", *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019. Available: 10.1109/isscc.2019.8662457

[10] K.A.A. Makinwa. “Temperature sensor performance survey,”. [Online]. Available:http: //ei.ewi.tudelft.nl/docs/TSensor_survey.xls

[11] Markus, J. Silva and G. Temes, "Theory and Applications of Incremental ΔΣ Converters", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 4, pp. 678-690, 2004. Available: 10.1109/tcsi.2004.826202

[12] Y. Chae et al., "A 2.1 M Pixels, 120 Frame/s CMOS Image Sensor With Column-Parallel Delta-Sigma ADC Architecture", *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 236-247, 2011. Available: 10.1109/jssc.2010.2085910

[13] S. Galal and M. Horowitz, "Energy-Efficient Floating-Point Unit Design", *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 913-922, 2011.

[14] S. Xu, S. Fahmy and I. McLoughlin, "Square-rich fixed point polynomial evaluation on FPGAs", *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, 2014. Available: 10.1145/2554688.2554779

[15] S. Winograd, "ON THE NUMBER OF MULTIPLICATIONS REQUIRED TO COMPUTE CERTAIN FUNCTIONS", *Proceedings of the National Academy of Sciences*, vol. 58, no. 5, pp. 1840-1842, 1967. Available: 10.1073/pnas.58.5.1840

[16] M. Harris, "Comparing All Serial Communications Protocols", *Altium*, 2021. [Online]. Available: https://resources.altium.com/p/comparing-all-serial-communications-protocols.

[17] P. Dhaker, "Introduction to SPI Interface | Analog Devices", *Analog.com*, 2021. [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

[18] S. Ganesan, "Area, delay and power comparison of adder topologies" , *Hdl.handle.net*, 2021. [Online]. Available: http://hdl.handle.net/2152/35303

[19] B. Parhami, "*Computer Arithmetic: Algorithms and Hardware Designs",* 2nd edition, Oxford University Press, New York, 2010.

[20] P. Girish, "Clock domain crossing- Closing the loop on clock domain functional implementation problems", Cadecnce Design Systems, 2004

[21] S. Pan and K. Makinwa, "A 0.25 mm2-Resistor-Based Temperature Sensor With an Inaccuracy of 0.12 °C (3<inline-formula> <tex-math notation="LaTeX">σ </tex-math> </inline-formula>) From −55 °C to 125 °C", *IEEE Journal of Solid-State Circuits*, vol. 53, no. 12, pp. 3347-3355, 2018. Available: 10.1109/jssc.2018.2869595

[22] S. Pan and K. Makinwa, "10.4 A Wheatstone Bridge Temperature Sensor with a Resolution FoM of 20fJ.K2", *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019. Available: 10.1109/isscc.2019.8662337 [Accessed 17 January 2022].

[23] S. Pan and K. Makinwa, "3.6 A CMOS Resistor-Based Temperature Sensor with a 10fJ·K2 Resolution FoM and 0.4°C (30) Inaccuracy From −55°C to 125°C After a 1-point Trim", *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020. Available: 10.1109/isscc19947.2020.9063064