NTNU

Kunnskap for en bedre verden

Department of Electronic Systems

TFE 4188 - Advanced Integrated Circuits

Assignment M3 - Group 5

Authors:
Markus Pedersen
Andrea Al Muktash
Trond Christiansen

Date: 22.03.2024

Table of Contents

List of Figures

1 Introduction

For this milestone, the goal is to create a circuit design that implements the functionality of converting a time domain value to a digital value. The design utilizes the Skywater Open Source PDK and its SKY130 process node.

This report serves as preliminary documentation of the current progress of the project.

2 Key Parameters

The following table presents the key parameters for the design.

Table 1: Key parameters

Name	Description	Min	Typ	Max	Unit
Width	Maximum width	-	-	160	μm
Height	Maximum height	-	-	100	μm
VDD	Input supply	1.62	1.80	1.98	V
Temperature	Temperature range to measure	-40	-	125	$^{\circ}C$
Current	Input current	1	10	100	μA

3 Specification

The following table presents the specifications for the design.

Table 2: Specifications

Name	Description	Min	Typ	Max	Unit	Notes
Accuracy	One-point accuracy	-5	0	+5	$^{\circ}C$	-
Resolution	Temperature resolution	-	1	-	$^{\circ}C$	-
Stability	Phase margin	-	74	-	$^{\circ}$	-
Stability	Gain margin	-	-23	-	dB	-
Noise	Noise of the OTA	-	-	-	$V/\sqrt{HzS}$	TBD
Offset	Offset of the OTA	-	-	-	V	TBD
CMRR	CMRR of the OTA	-	-	-	dB	TBD
Gain	Gain of the OTA	-	26	-	dB	-

4 Verification Plan

4.1 Linearity

Simulations will be performed to determine the linearity of the design. Data analysis in the form of regression analysis will performed on the raw output of the time-to-digital circuit. The produced regression models can then be analyzed to determine the R-Squared values and SSE values.

4.2 Process Corners

The design will be simulated in the typical, slow, fast, and mismatch corners to determine the effects of process variations.

4.3 Compensation and Temperature Error

The digital design implements a compensation block that reduces the errors caused by the non-linearity of the analog circuit. The error between the actual temperature and the measurement produced by the sensor will be analyzed with regards to maximum positive error, maximum negative error, and average absolute error.

5 Architecture

5.1 Top Design

The following block diagram presents a simplified top-level design of the digital circuit. Each block is explained in the subsequent subsections.

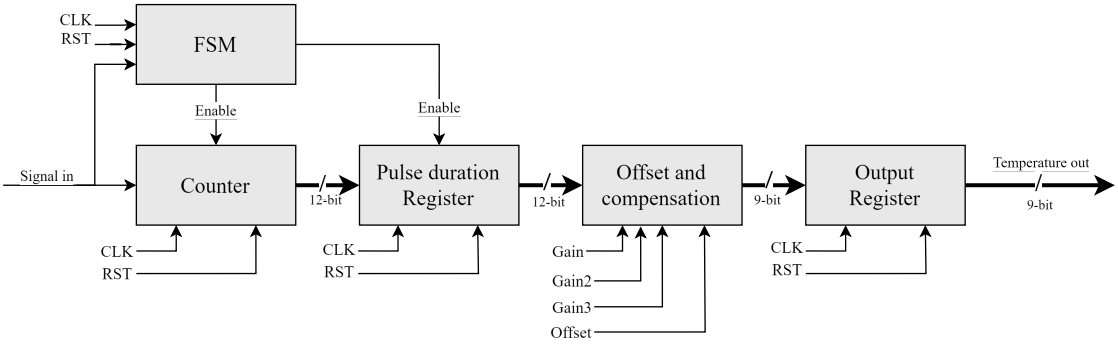

Figure 1: Top level block diagram of the digital circuit

The circuit accepts an input signal (Signal in), which is connected to the reset signal for the integrator in the "current-to-time" part of the analog design. Additionally, a clock signal and a reset signal are required inputs. The circuit outputs the final temperature as a parallel 9-bit signed integer value, making it possible to represent temperatures ranging from -256 °C to 255 °C.

5.2 FSM

The finite state machine takes "Signal in" as input and outputs enable signals for the counter and the pulse duration register. The default state is IDLE, where the FSM waits for the input signal to go low. Once a low input signal is detected, the FSM transitions to the COUNT state, where the counter is reset and enabled. The counter increments once per clock cycle as long as the input signal is low. When the input signal goes high, the counter is disabled and the FSM returns to IDLE to await a new measurement.

The following state diagram demonstrates the aforementioned state machine.

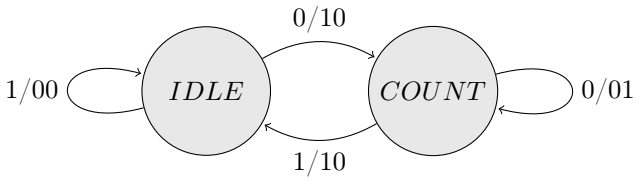

Figure 2: FSM state diagram

5.3 Counter

The counter converts the input signal from the time domain to a digital representation by counting the number of clock pulses that elapse while the input signal is low. With a clock of 40 MHz, it was found that the lowest and highest measurable pulses within the temperature range of -40 °C and 125 °C for all corners were 1076 and 1901 clock pulses, respectively. A 12-bit counter was, therefore, implemented to cover all measurable values with some safety margin.

5.4 Offset and Compensation

The Offset and compensation block implements calibration compensation and converts the raw value from the counter to a signed temperature value.
The compensation is performed by using the conversion function presented in the next chapter. The raw value is exponentiated and multiplied with gain constants and the product of this calculation is added to a offset. The current design utilizes a third-order compensation function, which is computationally demanding. The design can easily be simplified by setting one or more of the gain constants to zero in the Verilog code.
Finally, the compensated 12-bit value is converted to a signed integer value represented using two's complement, and truncated to a 9-bit output to account for the valid temperature range.

The following figure presents a block diagram of the contents of the offset and compensation block:

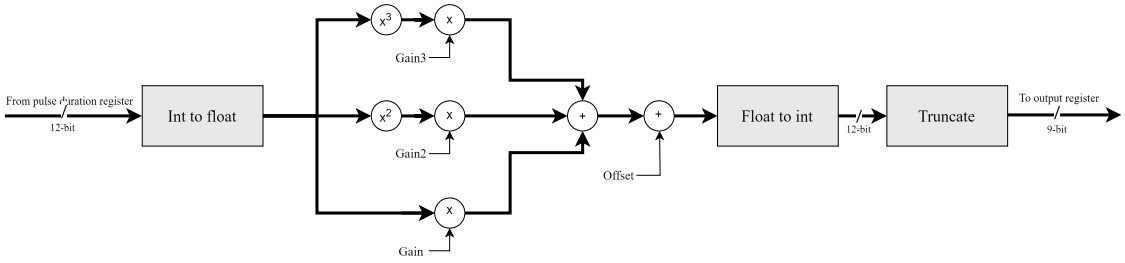

Figure 3: Block diagram of offset and compensation block

5.5 Verilog Code

Listing 1: Verilog code for the digital design

```
/*  Verilog code for converting an input pulse signal to a temperature value.
*
*   The code works by counting the number of clock cycles between each input pulse.
*   The actual temperature is then calculated using a 3rd order conversion function.
*   The temperature is finally output as a 9-bit signed value on a parallel bus.
*/


module PulseDurationMeasurement(
    input wire clk,                                 // Clock input
    input wire reset_n,                             // Reset input (active low)
    input wire signal_in,                           // Input signal to measure
    output reg signed [8:0] temperature_output = 0  // Signed temperature value
    output

);

// Internal variables
reg [1:0] state;
reg [11:0] count;
reg [11:0] pulse_duration = 0;
reg signed [31:0] temp_value;
reg signed [8:0] temp_value1;

// Constants declaration
parameter IDLE = 2'b00;
parameter COUNT = 2'b01;

// Constants for temperature conversion
parameter GAIN3 = -0.0000004731;
parameter GAIN2 = 0.0020044419;
parameter GAIN = -2.4734734627;
```

```
parameter OFFSET = 847.1700446582;


// State machine for the counter
always_ff @(posedge clk) begin
    if (~reset_n) begin
        $display("\tReset");
        state = IDLE;
        count = 12'h0;
        pulse_duration = 12'b0;
    end else begin
        case(state)
            IDLE: begin
                count = 12'b000000000001;
                if (signal_in == 0) begin
                    state = COUNT;
                end
            end

            COUNT: begin
                if (signal_in == 0) begin
                    count = count + 1;
                end else begin
                    pulse_duration = count;
                    state = IDLE;
                end
            end
            default:
                state = IDLE;
        endcase
    end
end

// Calculate the temperature value based on the raw pulse duration value
// Convert from real to signed int and truncate to 9 bits
always_comb begin
    temp_value = $rtoi(GAIN3*pulse_duration*pulse_duration*pulse_duration + GAIN2*
    pulse_duration*pulse_duration + GAIN*pulse_duration + OFFSET);
    temp_value1 = temp_value[8:0];
end

// Update output register
always_ff @(posedge clk) begin
    if (~reset_n) begin
        temperature_output = 9'b0;
    end else begin
    temperature_output = temp_value1;
    end
end;

endmodule
```

6 SPICE Simulations

All simulations presented in this chapter were performed by configuring a temperature step from -40 °C to 125 °C in 5 °C increments using the following SPICE commands:

Listing 2: Code for the temperature step simulation test bench

```
optran 1 1 1 100p 2n 0

write

foreach vtemp -40 -35 -30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30 35 40 45 50 55 60
    65 70 75 80 85 90 95 100 105 110 115 120 125
  option temp=$vtemp
  tran 10p 1000n 10p
  write {cicname}_($vtemp).raw
  end

quit
```

6.1 Co-Simulation Using Verilog and Ngspice

To simplify the design process of the digital circuit, the group chose to describe it in Verilog and co-simulate it with the analog circuit in ngspice. This was done by compiling the Verilog code for ngspice with Verilator and including it in the spice simulation. The compilation of the Verilog code for ngspice is done with the command "ngspice vlnggen <verilog_file>" which creates a shared object library that can be included in the spice file.
To insert the digital part into the spice code one has to set up the port connections using the following code:

Listing 3: General setup of DUT

```
adut [<inputs>] [<outputs>] [<inouts>] <name>
```

and link it to the shared object file with:

Listing 4: General setup of model

```
.model <name> d_cosim simulation="<filepath>"
```

For this design, this was implemented in the following manner:

Listing 5: Setup of co-simulation for the specific design

```
adut [VCLK VPWR_UP VB] [~D8 ~D7 ~D6 ~D5 ~D4 ~D3 ~D2 ~D1 ~D0] null vdut
.model vdut d_cosim simulation="./../verilog_include_file.so"
```

All the IO ports here are spice IO-pins to make linking from the .spi file easier. Even though the output pins are connected to the digital design, they also had to be connected in the schematic to be simulated. To solve this they got connected to a pull-down resistor.

6.2 Simulation of Output Pulse Durations

This simulation was performed to verify the output of the pulse duration register. For every temperature defined in the temperature step simulation, the value stored in the pulse duration register was extracted. Table 3 presents the results of this simulation. A graphical representation of the results is given in Figure 4.

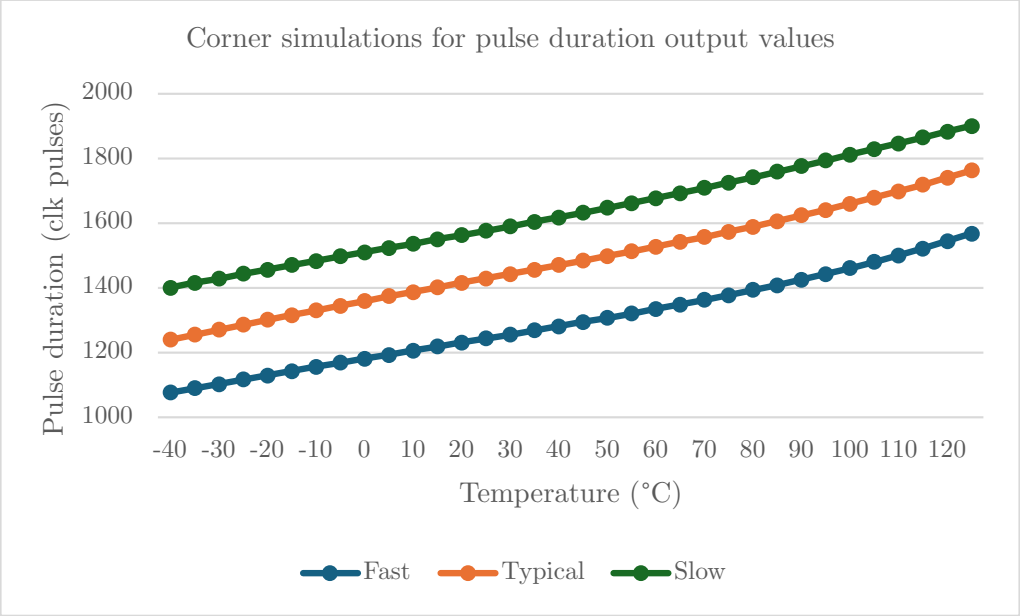

Figure 4: Corner simulations for pulse duration output values

Table 3: Pulse duration values for the slow, typical, and fast corners

Actual temperature	**Fast**	**Typical**	**Slow**
-40	1076	1239	1400
-35	1089	1255	1415
-30	1102	1270	1428
-25	1116	1285	1443
-20	1129	1301	1456
-15	1142	1315	1470
-10	1155	1330	1483
-5	1168	1344	1497
0	1181	1359	1510
5	1192	1374	1523
10	1205	1387	1536
15	1218	1401	1550
20	1231	1415	1563
25	1243	1429	1576
30	1255	1442	1590
35	1268	1456	1604
40	1281	1470	1618
45	1294	1484	1632
50	1307	1498	1647
55	1320	1513	1662
60	1334	1527	1677
65	1348	1542	1693
70	1362	1557	1709
75	1377	1573	1725
80	1393	1589	1742
85	1408	1606	1759
90	1425	1624	1776
95	1442	1641	1794
100	1461	1660	1812
105	1480	1679	1829
110	1500	1698	1847
115	1521	1719	1865
120	1544	1741	1883
125	1568	1763	1901

6.3 Regression Analysis

Regression analysis was performed on the data set presented in Table 3. The intention is to generate an optimal mathematical function for the "Offset and compensation" block and compare the predicted mathematical function for each corner.

6.3.1 Linear Regression Model

A two-variable regression analysis using a linear regression model was performed using Geogebra. The analysis produced the following linear functions where the input argument x is the pulse duration and the output is the estimated temperature:

Fast corner:
$f1(x) = 0.3503544961x - 412.2910495065$

Typical corner:
$t1(x) = 0.3272187657x - 443.3813706806$

Slow corner:
$s1(x) = 0.3336094988x - 503.1968315978$

The gains of the functions are relatively similar, while the difference in offset appears significant.

6.3.2 Second Degree Polynomial Regression Model

Using the same method as described in Section 6.3.1, a regression analysis was performed using a second order polynomial regression model. The analysis produced the following polynomial functions:

Fast corner:
$f2(x) = -0.0002120635x + 0.9073789214x - 773.8972677646$

Typical corner:
$t2(x) = -0.0001214015x + 0.6903351217x - 712.1728254667$

Slow corner:
$s2(x) = -0.0001445061x + 0.8096694042x - 892.1467608853$

6.3.3 Third Degree Polynomial Regression Model

Likewise, a third order polynomial regression model was also used, and produced the following functions:

Fast corner:
$f3(x) = -0.0000005535x + 0.0019750605x - 1.9523770492x + 463.324028103$

Typical corner:
$t3(x) = -0.0000004731x + 0.0020044419x - 2.4734734627x + 847.1700446582$

Slow corner:
$s3(x) = -0.0000001333x + 0.0005156684x - 0.2763157895x - 298.4833512352$

6.3.4 Comparison Of Regression Models

The following set of tables shows the R-Squared (coefficient of determination) and SSE (Sum of Squares Error) for the regression models.
The R-Squared parameter ranges from 0 to 1 where 1 indicates that the model explains all the variability of the response data around its mean. Essentially, it measures how well the independent variables explain the variability in the dependent variable. Higher R-squared values indicate a better fit.
SSE is a measure of the discrepancy between the observed values of the dependent variable and the values predicted by the model. It is calculated by summing the squares of the differences between the actual and predicted values for each data point. SSE measures the overall level of variation or "error" in the model. Lower SSE values indicate better fit.

Table 4: R-Squared and SSE (Sum of Squares Error) for the different regression models

	Linear		
Parameter	**f1(x)**	**t1(x)**	**s1(x)**
R-Squared	0.9926728565	0.9963848911	0.9964857111
SSE	599.4519290872	295.761098638	287.5127608068

	Second order		
Parameter	**f2(x)**	**t2(x)**	**s2(x)**
R-Squared	0.999206562	0.9991640995	0.9998409351
SSE	64.9131450699	68.3871088808	13.0134942544

	Third order		
Parameter	**f3(x)**	**t3(x)**	**s3(x)**
R-Squared	0.9999461648	0.9999534458	0.9998307775
SSE	4.404391486	3.8087161203	13.8445195964

One can observe the increase in the R-Squared values and decrease in the SSE value when going from linear to second order, and second order to third order.

The following figure is added to more clearly demonstrate the effects of the different regression models:

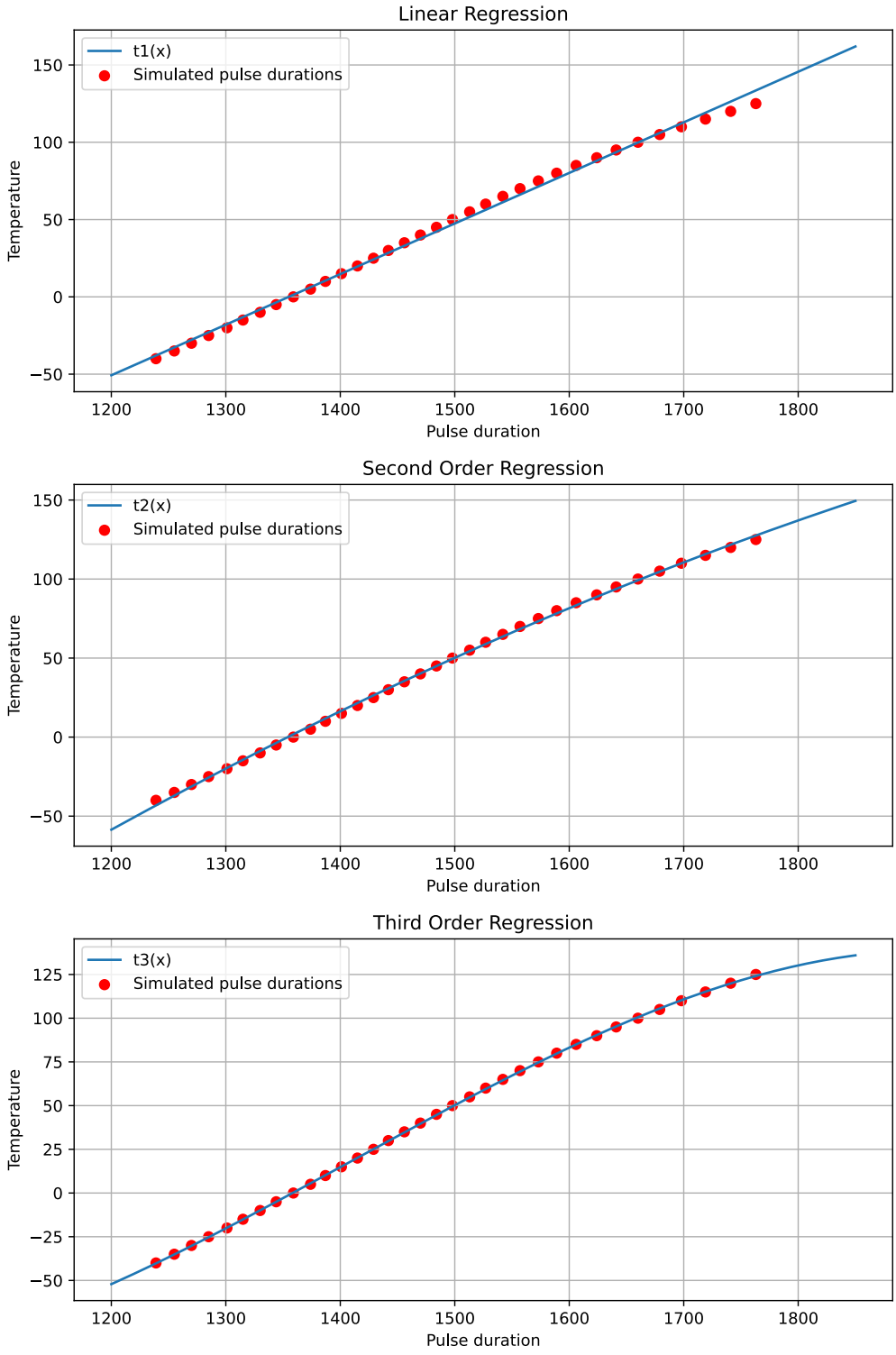

Figure 5: Plots of the regression models for the typical corner

One can observe how the higher-order regression models better represent the plotted points of the simulated pulse durations.

6.4 Simulation of Output Temperature Error

This simulation was performed to verify the output of the complete time-to-digital circuit. For every temperature defined in the temperature step simulation, the value stored in the output register was extracted. These values are the result of the "Offset and compensation" block. The error was calculated by finding the difference between the simulated temperature and the output temperature of the design. Statistics for all the error simulations are presented in Table 5.

6.4.1 Simulation of Output Temperature Error With Linear Compensation

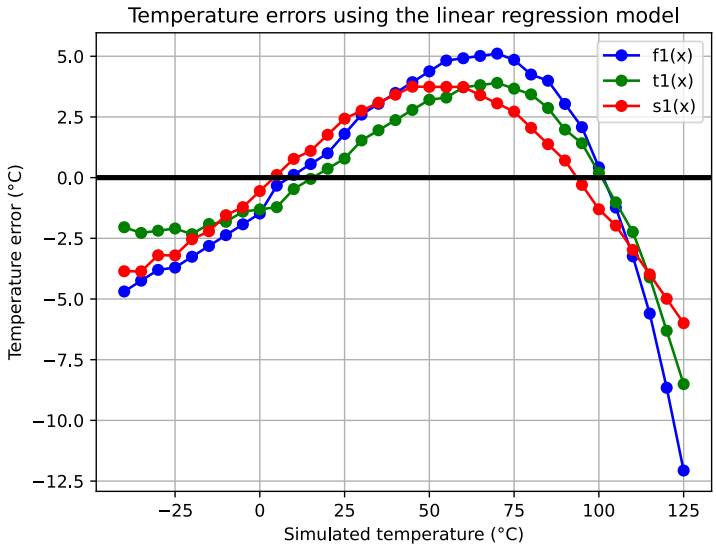

Figure 6: Temperature errors using linear regression for the fast, typical, and slow corners

6.4.2 Simulation of Output Temperature Error With Second Order Compensation

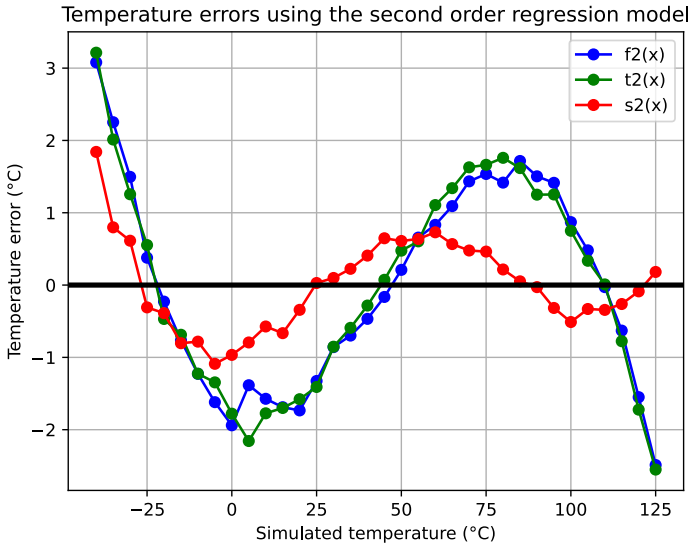

Figure 7: Temperature errors using second order regression for the fast, typical, and slow corners

6.4.3 Simulation of Output Temperature Error With Third Order Compensation

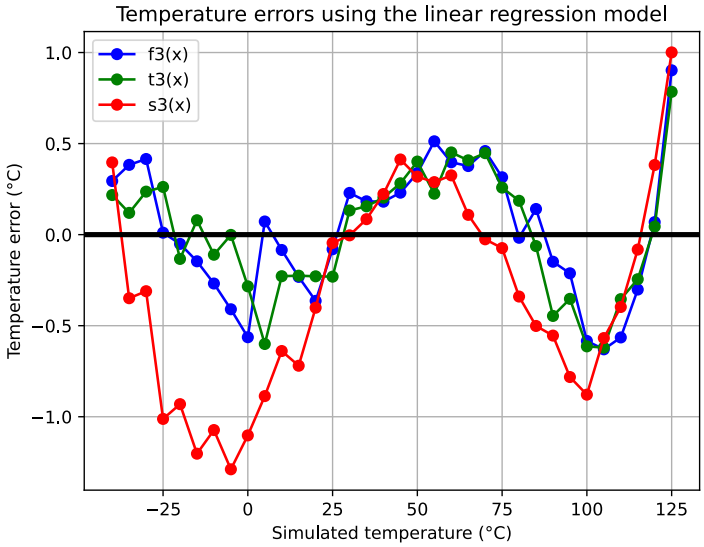

Figure 8: Temperature errors using third order regression for the fast, typical, and slow corners

6.4.4 Error statistics

Table 5: Error statistics for the regression models in the fast, typical, and slow corners

	Linear		
Parameter	**f1(x)**	**t1(x)**	**s1(x)**
Max positive error (°C)	5.108	3.902	3.746
Max negative error (°C)	-12.065	-8.505	-5.995
Average absolute error (°C)	3.495	2.429	2.571

	Second order		
Parameter	**f2(x)**	**t2(x)**	**s2(x)**
Max positive error (°C)	3.080	3.214	1.841
Max negative error (°C)	-2.489	-2.552	-1.089
Average absolute error (°C)	1.198	1.230	0.505

	Third order		
Parameter	**f3(x)**	**t3(x)**	**s3(x)**
Max positive error (°C)	0.902	0.784	1.001
Max negative error (°C)	-0.630	-0.620	-1.289
Average absolute error (°C)	0.299	0.283	0.521

6.5 Mismatch Simulation

The following mismatch simulation was performed by running a Monte Carlo simulation with 30 points for a temperature of 27 °C. Figure 9 presents the results of this simulation. The x-axis represents the output pulse duration for each simulation. The y-axis represents the number of simulations that were within the corresponding x-value +/- 50.

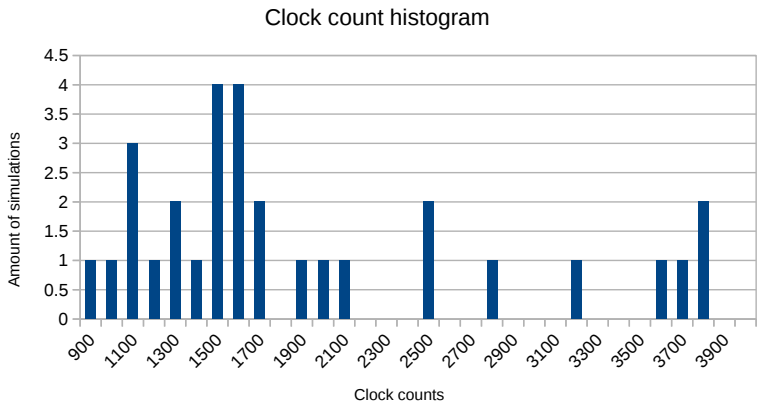

Figure 9: Monte Carlo simulation of the pulse width duration for 30 points at 27 °C

One can observe that the pulse duration output differs greatly with regard to mismatch. Ideally, to have a design immune to mismatch, all pulse durations should appear in the same pillar.

The reason for the bad performance is the analog design's high degree of offset for a varying mismatch. This can be observed in Figure 10, which shows a Monte Carlo simulation of the PTAT current for 30 points.

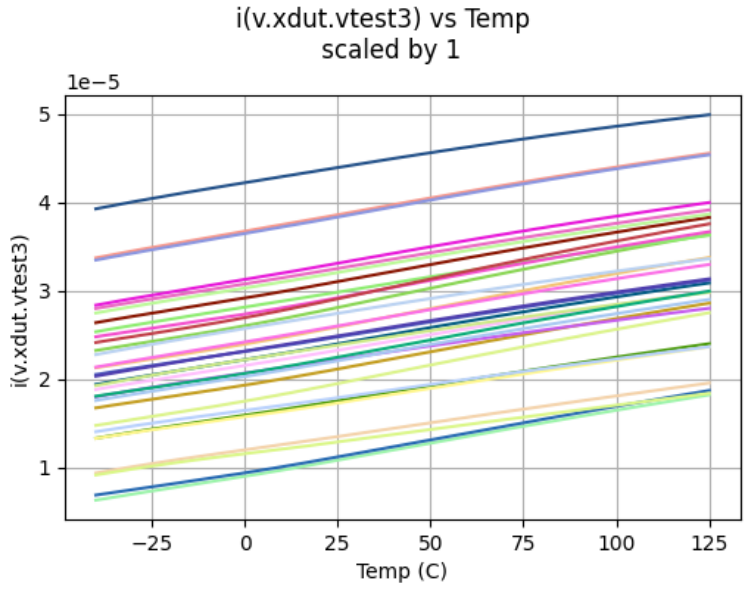

Figure 10: Monte Carlo simulation of the PTAT current

One can observe a relatively constant gain and a highly corner-dependent offset between the PTAT currents.

7 Discussion

7.1 Discussion of Simulation Results

As shown, the offset of the design is highly dependent on the process variations. Because of this, offset calibration is required for the temperature sensor to be usable. The group will attempt to improve the offset before the submission of the final design. With a linear compensation, the temperature error is +/- 5 °C for the temperature range of -24 °C to 120 °C, which fulfills the specifications of the design. With polynomial compensation, this error is significantly reduced.

7.2 Theoretical End Product

The current design implements a third-order compensation. The idea is that the registers "Offset", "Gain", "Gain2", and "Gain3" can be set externally. An interface for this is not yet implemented, as the group considers it beyond the scope of this project. The 12-bit pulse duration value could also be made available as an output for the temperature sensor, so the end user could perform calibration and post-processing externally, for example using a microcontroller.

To make use of the third-order compensation, one would have to deploy multi-point calibration. This will be far too expensive to do during production, so it would be up to the end user to decide how much effort they want to put in. A positive aspect of the design is that the end user can select either single-point, linear, second-order, or third-order compensation themselves. If the user, for example, only requires linear compensation, the registers "Gain2" and "Gain3" can simply be set to zero.

7.3 Synthesizing the Verilog Code

The Verilog code, in its current state, might not be synthesizable with OpenLane. One example is the *$rtoi* used in line 68, which is used to convert from a real value to an integer value. Another example is the implicit definitions of the combinatorial circuit, also in line 68. The group is, therefore, prepared to alter the Verilog code during the layout process of the next milestone.

7.4 Reflections Regarding the Implemented Compensation

Employing this level of compensation is somewhat cheating because one can correct a bad analog design by throwing a complex combinatorial digital design at it. The main reason for doing this was to research and experiment with a more advanced RTL implementation, as the group considers this a great learning opportunity.