# Implementation

## Configuring ADC and the sensor

Implementation of ADC involves first initializing the ADC configuration structures to enable the ADC peripherals and power for the bus APB2 that interfaces with the ADC1 peripheral. The embedded temperature sensor is internally connected to channel 16. The ADC features two clock schemes: ADCCLK is the clock for the analog circuitry that is common across all ADC peripherals – this is the clock of interest for this experiment. This clock is generated from the APB2 bus clock that is divided by a programmable pre-scaler, enabling the ADC peripheral to work at configurable frequencies. According to specification, the APB2 clock is set at 84 MHz, however the ADC peripherals supports maximum clock frequency of 36 MHz, hence ADC\_CLOCK\_SYNC\_PCLK\_DIV4 is specified to provide an operating frequency of 84 / 2 = 21 MHz. The resolution bits for the output data has been set to 12 bits in the data register – greater resolution translates to slower conversion, however for a system engaging only the temperature sensor, faster conversion rate is less crucial. Continuous conversion mode has been also been enabled instead of single-burst conversion, hence allowing continuous polling for data once conversion has completed (as evidenced by setting end-of-conversion flag). Since the embedded temperature sensor is the only peripheral of interest for this experiment, scan conversion mode has been disabled to avoid polling of multiple channels. Reader may reference system\_config.h and system\_config.c files to complement the above discussion.

## Configuring sampling frequency

The sampling frequency was setup by using ARM’s inherent HAL\_SYSTICK\_Config() method, which takes in frequency parameter based on the system’s core clock and the desired frequency to be configured. The system’s core clock operates at 168 MHz and the SysTick frequency provided in the specification is 100 Hz. Interrupt handler has also been defined by setting ticks parameter to 1. This gets checked in every iteration of the main loop before polling for end-of-conversion flag of ADC peripheral. The main.c and stm32f4xx\_it.c source files may be referred to for further implementation details.

The routines HAL\_ADC\_Start and HAL\_ADC\_PollForConversion starts the analog-to-digital conversion, and starts polling for end-of-conversion flag setting. After completion of digitizing each measurements the routine HAL\_ADC\_GetValue is called to extract the output value and reset the flag. Since ADC peripheral has been configured as continuous conversion mode, repeated start and stop of the sensor has been avoided.

## Signal filtering

The raw readings acquired from the ADC peripheral is very noisy and less representative of actual changes in the physical medium. In order to streamline and smoothen the digital signal a Kalman filter has been used. Kalman filter is a state-based adaptive estimator that is more efficient than fixed linear filters. The adaption is performed by a sequence of discrete operations whereby the filter parameters change based on the value observed in the physical medium and the current state of the filter. The difference value between the estimated output and the actual input has the statistical properties of white noise – a feature that has been leveraged during system testing and analysis phase.

Two routines are crucial to the implementation of Kalman filter – notably the Reset() routine that configures the initial state parameters, and the Kalmanfilter\_C() that recursively updates the state parameters following the input values. This routine takes a double-valued input which is the digitized output from the ADC peripheral, pointer to a double that stores the processed output of the filter and a pointer to the Kalman state structure that updates parameters. The function also checks for unbounded output value and returns 1 if the value is NaN, and 0 otherwise. A pseudocode for the Kalman filter structure and the calling routines have been included in the Appendix for ease of reference. Reader may refer to the Observations section that discusses how the initial values of the Kalman state has been decided and the filter calibrated.

## Data interpretation

The output of the Kalman filter maybe treated directly using mathematical equation to extract the corresponding temperature value in degrees Celsius. The reference manual and the datasheet has been extensively reference in this regards to extract the following equation:

Details of parameters in the equation are as follows:

*output* = output value of the Kalman filter (a double)

*V\_25* = 760 mV

*average\_slope* = 2.5

## Temperature Display

In order to display the temperature readings a 7-segment display has been integrated in the system. The 7-segment display interfaces with the GPIO port on the microprocessor board to set the individual segments as high or low. An external circuit has been assembled following the guidelines set forth in the specification – the circuit follows multiplexing design that conservatively uses the segment display pins for select lines and individual digit display.

The display implementation first involves initializing the GPIO configuration structures to enable the GPIO peripherals, and power for the bus AHB1 that interfaces with the peripherals. Since GPIO ports are only employed to set the display segments in this experiment, the ports have been configured as push-pull output without pull-up or pull-down activation. 3 GPIO ports have been configured in the final system design – 3 pins on PORTA for the display select lines, 8 pins on PORTE to set the unit segments to display digits and decimal point, and 4 pins on PORTD to toggle alarm LEDs (to be discussed in next section). The crux of the display logic is the Display() routine that lays out the required pin write configurations for the select lines (to choose digit display position – from most to least significant) and the digit display (between 0-9 and decimal point). Reader may consult the 7seg\_display.c source file for implementation details. The logic for display of unit digit employs case-switch structure to check which digit should be displayed and which corresponding segments should be set to accomplish the display.the datasheet has been referred to find out I/O pins available to be used. Caution has been exercised to not address the restricted I/O pins that are reserved for special debugging purposes.

The main.c source file refers to the Display() routine to incrementally display the digits that correspond to the interpreted temperature value (discussed in the previous section). However, prior to calling the Display() routine the individual digits must be extracted from the conversion value (a double). A simple iterative logic has been employed to accomplish this goal, a pseudocode has been included in the appendix for ease of reference for the readers.

As stated in specification, the system has been configured to sample temperature measurements at 100 Hz. However, this is beyond visually perceivable by the human eye. Besides an efficient implementation has been sought out that will not interrupt the temperature sampling in the process of imposing display delay. As a corrective measure, loop counter variables have been declared to impose the necessary delay, such that the temperature readings are easily perceivable, temperature sampling is uninterrupted and the display does not flicker. The counter variable sets the time delay for display of one temperature sample, and is configured to update after completion of 90 SysTick cycles. Hence, despite sampling temperature at every SysTick cycle, only the values sampled after every 90 cycles are chosen to be displayed. This design decision makes a compromise between the fast response time of temperature logging and display that can be visually perceivable.

Additionally counter2 imposes necessary delay in displaying a single digit. The 7-segment display has been implemented such that the select lines sweep across all the digits, displaying one character after another in a round-robin fashion (from most significant digit, decimal point to the least significant digit). Once it times out in displaying the least significant digit, the display wraps around to the most significant digit and continues to loop until 90 SysTick cycles have passed and a new temperature sample has been chosen. The delay for displaying single digit has been decided to be 1 SysTick cycle, after performing multiple tests to assess the display responsiveness and negation of flickering.

## Overheating alarm

Besides integrating the 7-segment display in the system to report temperature, an alarm mechanism has been devised that sets off when the temperature reading exceeds a certain threshold value. As discussed in the previous section, 4 pins on PORTD has been configured during GPIO peripheral configuration. These pins correspond to the green, orange, red and blue on-board LED lights in counter-clockwise order. The datasheet has been referenced to find out these pin mappings.

Similar to the implementation of 7-segment display, loop counter has been employed to impose the necessary delay between toggling of the LEDs. The delay for keeping each LED on has been set to 5 SysTick cycles. All the display logic has been implemented within the main loop structure, hence should the temperature exceed the specified threshold value, the LEDs will incrementally light up in a counter-clockwise manner to report the phenomena. For this experiment, maximum threshold has been set to 50 degrees Celsius, beyond which the overheating alarm is triggered.