Temperature Sensor Readout
Proposed Solution

Renato da Veiga Torres
renatoveigatorres@gmail.com

September 20, 2023

1 REQ01 - Analyze measuring circuit

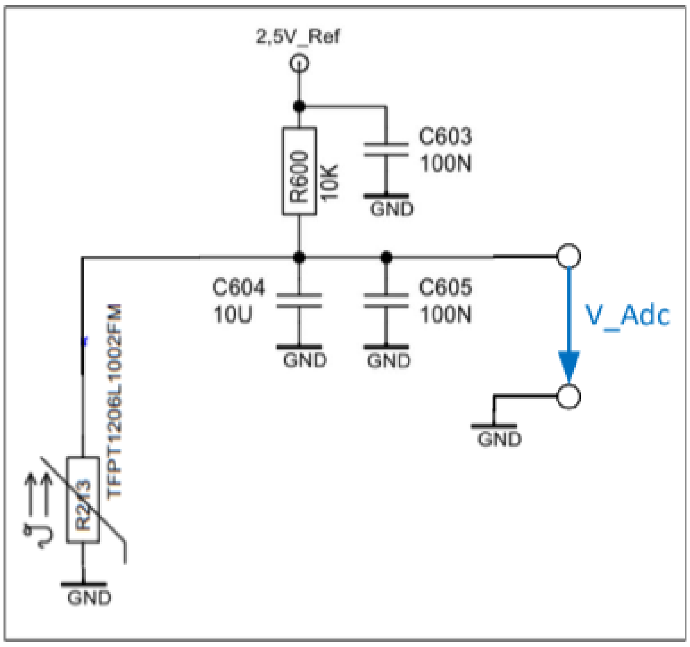

Figure 1: Temperature measurment circuit

For simplification, ADC input impedance, in this circuit (Figure 1), is considered high enough that is possible to neglect current leaking over ADC pin. In this way, the total current i on both resistors and V_{Adc} can be written as in (1):

$$i = \frac{V_{Ref}}{R_{600} + R_{213}^{RTC}} \tag{1}$$

As voltage divider, by construction, V_{Adc} can be calculated as described in (2):

$$V_{Adc} = \left[\frac{R_{213}^{RTC}}{R_{600} + R_{213}^{RTC}}\right].V_{Ref} \tag{2}$$

However, the purpose of the circuit is to 'read' V_{Adc} value and calculate R_{213}^{RTC}. As R_{213}^{RTC} is an RTC, its resistance/temperature are unequivocally correlated. Rewriting (2), R_{213}^{RTC} value and can be calculated as function of V_{Adc} as described in (3):

$$R_{213}^{RTC} = \left[\frac{V_{Adc}}{V_{Ref} - V_{Adc}}\right].R_{600} \tag{3}$$

As R_{600} is designed as same TFPT1206L1002FM (in 25° Celsius) - 10 kOhm, the value (R_{213}^{RTC}/R_{600}) is same as the average ratio (R_{Temp}/R_{25}) from datasheet (tfpt-223737.pdf - page 3):

$$ratio = \frac{R_{Temp}}{R_{25}} = \frac{R_{213}^{RTC}}{R_{600}} = \frac{V_{Adc}}{V_{Ref} - V_{Adc}} \tag{4}$$

Obtaining this ratio (R_{Temp}/R_{25}) - using (4) is possible to locate (as lookup table) the current temperature of the resistor in the datasheet (tfpt-223737.pdf - page 3).

2 REQ02 - Implement software that calculates temperature of NTC

```c
#include "platform_types.h"
#include "RTC_TFPT1206L1002FM.h"

/**
* @brief Binary search to find the temperature based on datasheet values.
*
* This function performs a binary search on the TFPT_Vishay_Values array to find the
* temperature corresponding to a target resistance ratio.
*
* @param targetRatio The target resistance ratio for which to find the temperature.
* @return The temperature in degrees Celsius if a match or approximate match is
    found,
*         or -273.0f (absolute zero) if the ratio is out of range.
*/
static float getTemp_Datasheet(float targetRatio) {
    int low = 0;
    int high = sizeof(TFPT_Vishay_Values) / sizeof(TFPT_Vishay_Values[0]) - 1;
    int resultIndex = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (TFPT_Vishay_Values[mid].fRatio_R25 == targetRatio) {
            resultIndex = mid;
            break;
        }

        if (TFPT_Vishay_Values[mid].fRatio_R25 < targetRatio) {
            low = mid + 1;
        } else {
            resultIndex = mid;
            high = mid - 1;
        }
    }

    if (resultIndex != -1) {
        // Found the exact match or the next lower fRatio
        if (resultIndex > 0) {
            // Use linearization between 2 datasheet points to optimize accuracy
            // fTx = fT1 + ((fRx - fR1)/(fR2 - fR1))*(T2 - T1);
            // (T2 - T1) is always 1 oC
            float fT1 = TFPT_Vishay_Values[resultIndex - 1].ftemperature;
            float fT2 = TFPT_Vishay_Values[resultIndex].ftemperature;
            float fR1 = TFPT_Vishay_Values[resultIndex - 1].fRatio_R25;
            float fR2 = TFPT_Vishay_Values[resultIndex].fRatio_R25;
            float fTx = fT1 + ((targetRatio - fR1) / (fR2 - fR1));
            return fTx;
        } else {
            return TFPT_Vishay_Values[resultIndex].ftemperature;
        }
    } else {
        // Return 'absolute zero' in case of ratio out of range
        return -273.0f;
    }
}
```

```c

    /**
    * @brief Calculate the resistance ratio from an ADC value.
    *
    * This function calculates the resistance ratio based on a 12-bit ADC value and
    * the characteristics of the TFPT1206L1002FM sensor.
    *
    * @param uiADCVal The ADC value obtained from the sensor.
    * @return The resistance ratio.
    */
    static float getRatioFromADCValue(uint16_t uiADCVal) {
        // 12-bit ADC => 2^12 = 4096
        // Range:
        // 0V   - 0
        // 2.5V - 4095

        // R213 = [(uiADCVal) / (VREF - uiADCVal)] * R600
        // However, R600 is 10K and R213 (TFPT1206L1002FM) is 10K on 25 oC.
        // R213/R600 = datasheet_ratio itself

        uint16_t VREF = 4095;
        float fratio = (uiADCVal) / (VREF - uiADCVal);

        return fratio;
    }

    /**
    * @brief Get the temperature in degrees Celsius from an ADC value.
    *
    * This function calculates the temperature in degrees Celsius based on the ADC
    * value obtained from the sensor and the datasheet values.
    *
    * @param uiADCVal The ADC value obtained from the sensor.
    * @return The temperature in degrees Celsius if within the valid range,
    *         or -273.0f (absolute zero) if out of range.
    */
    float getTemperature(uint16_t uiADCVal) {
        return getTemp_Datasheet(getRatioFromADCValue(uiADCVal));
    }
```

Listing 1: C Code for Temperature Sensor

3 REQ03 - Implement test-bench for software

For this requirement, testbench provided will be used for applying ADC values and confirm/assert temperatures according implemented code.

Using (4) is possible to calculate expected value of the ADC V_{Adc}^{Temp} for each ratio (R_{Temp}/R_{25}).

$$V_{Adc}^{Temp} = \frac{(\frac{R_{Temp}}{R_{25}}).4095}{1 + (\frac{R_{Temp}}{R_{25}})} \tag{5}$$

With (5) is possible to generate a lookup table with expected ADC values for whole range of temperatures covered in the datasheet.

Same test framework of magic square was used to implement some sample test benches and terminal application for test was implemented.

```

#include <stdio.h>
#include "unittests.h"
#include "platform_types.h"
#include "RTC_API.h"

#include <math.h>

GROUP_SETUP(RTC_API_TempVsADC)
{

}

GROUP_TEARDOWN(RTC_API_TempVsADC)
{

}

DEFINE_TEST_CASE(RTC_API_TempVsADC, TEMP_0)
{
    ASSERT_TRUE_TEXT( (fabs(getTemperature(1940) - 0.0f) < 0.5f)  , "Expected␣Temperature␣
        for␣0␣degrees");
}

DEFINE_TEST_CASE(RTC_API_TempVsADC, TEMP_45)
{
    ASSERT_TRUE_TEXT( ( fabs(getTemperature(2130) - 56.0f ) < 0.5f), "Expected␣Temperature␣
        for␣45␣degrees");
}

DEFINE_TEST_CASE(RTC_API_TempVsADC, RANGE_0_45)
{
    for(int i = 0; i<46; i++)
    {
        uint16_t adcVal = getADCfromTemp( i );
        ASSERT_TRUE_TEXT( ( fabs(getTemperature(adcVal) - ((float)i) ) < 0.5f), "Expected␣
            Temperature␣for␣CORRECT␣for␣0-45␣degrees");
    }
}



TEST_GROUP_RUNNER(RTC_API_TempVsADC)
{
    INIT_TEST_GROUP(RTC_API_TempVsADC);
    RUN_TEST_CASE(RTC_API_TempVsADC, TEMP_0);
    RUN_TEST_CASE(RTC_API_TempVsADC, TEMP_45);
}
```

Listing 2: Unittests example