John Danison

ECET 32900 – Lab 9

04/04/2025

**Goal:**

The goal of this lab was to design programmable logic to interface with a LM35 temperature sensor using the Nucleo 64 microcontroller and an LED.

**Conversion Equation:**

Per the LM35 datasheet, the output of the sensor has a correlation of 10mV/°C. To get the conversion equation figured out, the digital ADC value was converted to volts using the following equation.

**Figure 1 – ADC Reading to Voltage Conversion**

The temperature was then calculated using the 10mV/°C correlation shown below.

**Figure 2 – Voltage to Temperature Conversion**

**Electrical Schematic:**

A diagram of a diagram

AI-generated content may be incorrect.

**Flowchart:**

A diagram of a computer hardware system

AI-generated content may be incorrect.

**Source Code:**

/\* USER CODE BEGIN Header \*/

/\*\*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @file           : main.c

  \* @brief          : Main program body

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @attention

  \*

  \* Copyright (c) 2025 STMicroelectronics.

  \* All rights reserved.

  \*

  \* This software is licensed under terms that can be found in the LICENSE file

  \* in the root directory of this software component.

  \* If no LICENSE file comes with this software, it is provided AS-IS.

  \*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private variables ---------------------------------------------------------\*/

ADC\_HandleTypeDef hadc1;

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_ADC1\_Init(void);

/\* Define Constants \*/

#define LED\_PIN GPIO\_PIN\_1

#define THRESHOLD 25.0

/\* Function Prototypes \*/

void ledOn();

void ledOff();

/\*\*

  \* @brief  The application entry point.

  \* @retval int

  \*/

int main(void)

{

  /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

  HAL\_Init();

  /\* Configure the system clock \*/

  SystemClock\_Config();

  /\* Initialize all configured peripherals \*/

  MX\_GPIO\_Init();

  MX\_ADC1\_Init();

  /\* Define Local Variables \*/

  float temperature = 0.0;

  uint32\_t ADCReading;

  /\* Reset LED \*/

  ledOff();

  /\* Start ADC Conversion \*/

  HAL\_ADC\_Start(&hadc1);

  /\* Infinite loop \*/

  /\* USER CODE BEGIN WHILE \*/

  while (1)

  {

      /\* Poll ADC \*/

      HAL\_ADC\_PollForConversion(&hadc1, 100);   // Poll ADC

      ADCReading = HAL\_ADC\_GetValue(&hadc1);    // Record ADC Reading

      temperature = ((float) ADCReading) \* 3300.0 / 4095.0; // Converting to mV

      temperature = (temperature) / 10.0;   // Convert to degree C from mV (LM34 & LM35)

      /\* Check if breaks LED threshold \*/

      if (temperature > THRESHOLD) {

          ledOn();

      } else {

          ledOff();

      }

      /\* 10 mSecond Delay \*/

      HAL\_Delay(10);

  }

  /\* USER CODE END 3 \*/

}

/\*\*

  \* @brief System Clock Configuration

  \* @retval None

  \*/

void SystemClock\_Config(void)

{

  RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

  RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

  /\*\* Configure the main internal regulator output voltage

  \*/

  if (HAL\_PWREx\_ControlVoltageScaling(PWR\_REGULATOR\_VOLTAGE\_SCALE1) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Initializes the RCC Oscillators according to the specified parameters

  \* in the RCC\_OscInitTypeDef structure.

  \*/

  RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_MSI;

  RCC\_OscInitStruct.MSIState = RCC\_MSI\_ON;

  RCC\_OscInitStruct.MSICalibrationValue = 0;

  RCC\_OscInitStruct.MSIClockRange = RCC\_MSIRANGE\_6;

  RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

  if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Initializes the CPU, AHB and APB buses clocks

  \*/

  RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

                              |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

  RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_MSI;

  RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

  RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

  RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

  if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

  {

    Error\_Handler();

  }

}

/\*\*

  \* @brief ADC1 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_ADC1\_Init(void)

{

  /\* USER CODE BEGIN ADC1\_Init 0 \*/

  /\* USER CODE END ADC1\_Init 0 \*/

  ADC\_MultiModeTypeDef multimode = {0};

  ADC\_ChannelConfTypeDef sConfig = {0};

  /\* USER CODE BEGIN ADC1\_Init 1 \*/

  /\* USER CODE END ADC1\_Init 1 \*/

  /\*\* Common config

  \*/

  hadc1.Instance = ADC1;

  hadc1.Init.ClockPrescaler = ADC\_CLOCK\_ASYNC\_DIV1;

  hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

  hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

  hadc1.Init.ScanConvMode = ADC\_SCAN\_DISABLE;

  hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

  hadc1.Init.LowPowerAutoWait = DISABLE;

  hadc1.Init.ContinuousConvMode = ENABLE;

  hadc1.Init.NbrOfConversion = 1;

  hadc1.Init.DiscontinuousConvMode = DISABLE;

  hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

  hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

  hadc1.Init.DMAContinuousRequests = DISABLE;

  hadc1.Init.Overrun = ADC\_OVR\_DATA\_OVERWRITTEN;

  hadc1.Init.OversamplingMode = DISABLE;

  if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Configure the ADC multi-mode

  \*/

  multimode.Mode = ADC\_MODE\_INDEPENDENT;

  if (HAL\_ADCEx\_MultiModeConfigChannel(&hadc1, &multimode) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Configure Regular Channel

  \*/

  sConfig.Channel = ADC\_CHANNEL\_3;

  sConfig.Rank = ADC\_REGULAR\_RANK\_1;

  sConfig.SamplingTime = ADC\_SAMPLETIME\_2CYCLES\_5;

  sConfig.SingleDiff = ADC\_SINGLE\_ENDED;

  sConfig.OffsetNumber = ADC\_OFFSET\_NONE;

  sConfig.Offset = 0;

  if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN ADC1\_Init 2 \*/

  /\* USER CODE END ADC1\_Init 2 \*/

}

/\*\*

  \* @brief GPIO Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_GPIO\_Init(void)

{

  GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

  /\* GPIO Ports Clock Enable \*/

  \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

  /\*Configure GPIO pin Output Level \*/

  HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_1, GPIO\_PIN\_RESET);

  /\*Configure GPIO pin : PC1 \*/

  GPIO\_InitStruct.Pin = GPIO\_PIN\_1;

  GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

  GPIO\_InitStruct.Pull = GPIO\_NOPULL;

  GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

  HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

  \* @brief  This function is executed in case of error occurrence.

  \* @retval None

  \*/

void Error\_Handler(void)

{

  /\* USER CODE BEGIN Error\_Handler\_Debug \*/

  /\* User can add his own implementation to report the HAL error return state \*/

  \_\_disable\_irq();

  while (1)

  {

  }

  /\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef  USE\_FULL\_ASSERT

/\*\*

  \* @brief  Reports the name of the source file and the source line number

  \*         where the assert\_param error has occurred.

  \* @param  file: pointer to the source file name

  \* @param  line: assert\_param error line source number

  \* @retval None

  \*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

  /\* USER CODE BEGIN 6 \*/

  /\* User can add his own implementation to report the file name and line number,

     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

  /\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

/\* Custom Functions \*/

// Turn LED On

void ledOn() {

    HAL\_GPIO\_WritePin(GPIOC, LED\_PIN, GPIO\_PIN\_SET);

}

// Turn LED Off

void ledOff()

{

    HAL\_GPIO\_WritePin(GPIOC, LED\_PIN, GPIO\_PIN\_RESET);

}

**Conclusion:**

During this lab exercise, I learned how to set up and use the ADC on the Nucleo L476RG board using the STM32 Cube IDE environment. This process is significantly different than any board or environment that I have used in the past and was a new and fun challenge. I also learned how the LM35 temperature sensor works and now have some more knowledge about some of the current temperature sensors available on the market.

**Appendix:**

A diagram of a diagram

AI-generated content may be incorrect.

References

Purdue University. (2025). *ECET 32900 Lab 9 Instructional Documents*. Purdue University.

Texas Instruments. (2016, March). *LM35 precision centigrade temperature sensors* (Rev. J)

[Data sheet]. <https://www.ti.com/lit/ds/symlink/lm35.pdf>