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.

$$Voltage = ADCReading * \frac{3300(mVRef)}{4095}$$

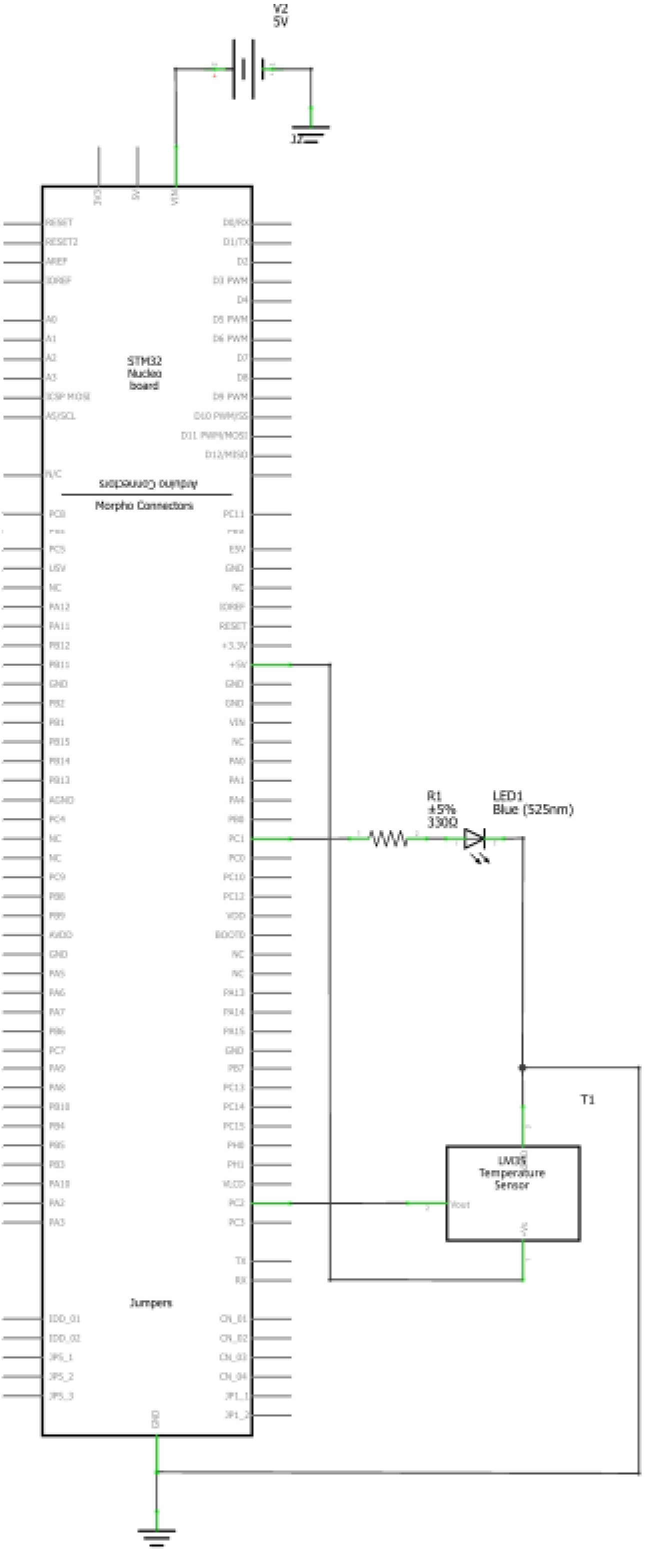
Figure 1 – ADC Reading to Voltage Conversion

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

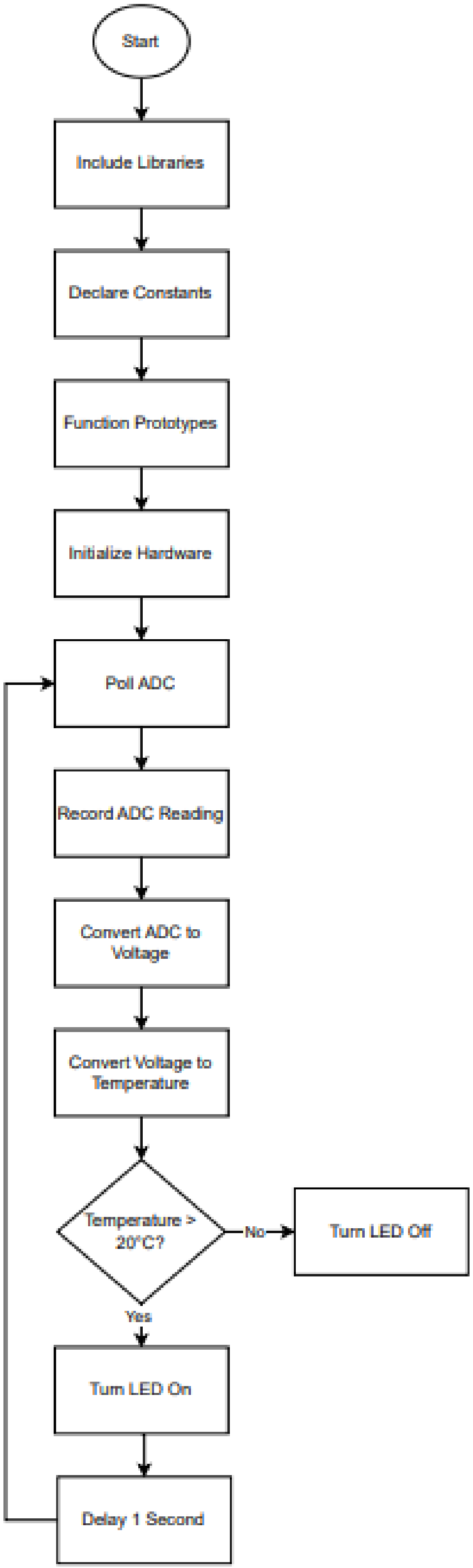
$$Temperature\ (°C) = \frac{Voltage\ (V)}{10\left(\frac{mV}{°C}\right)}$$

Figure 2 – Voltage to Temperature Conversion

Electrical Schematic:



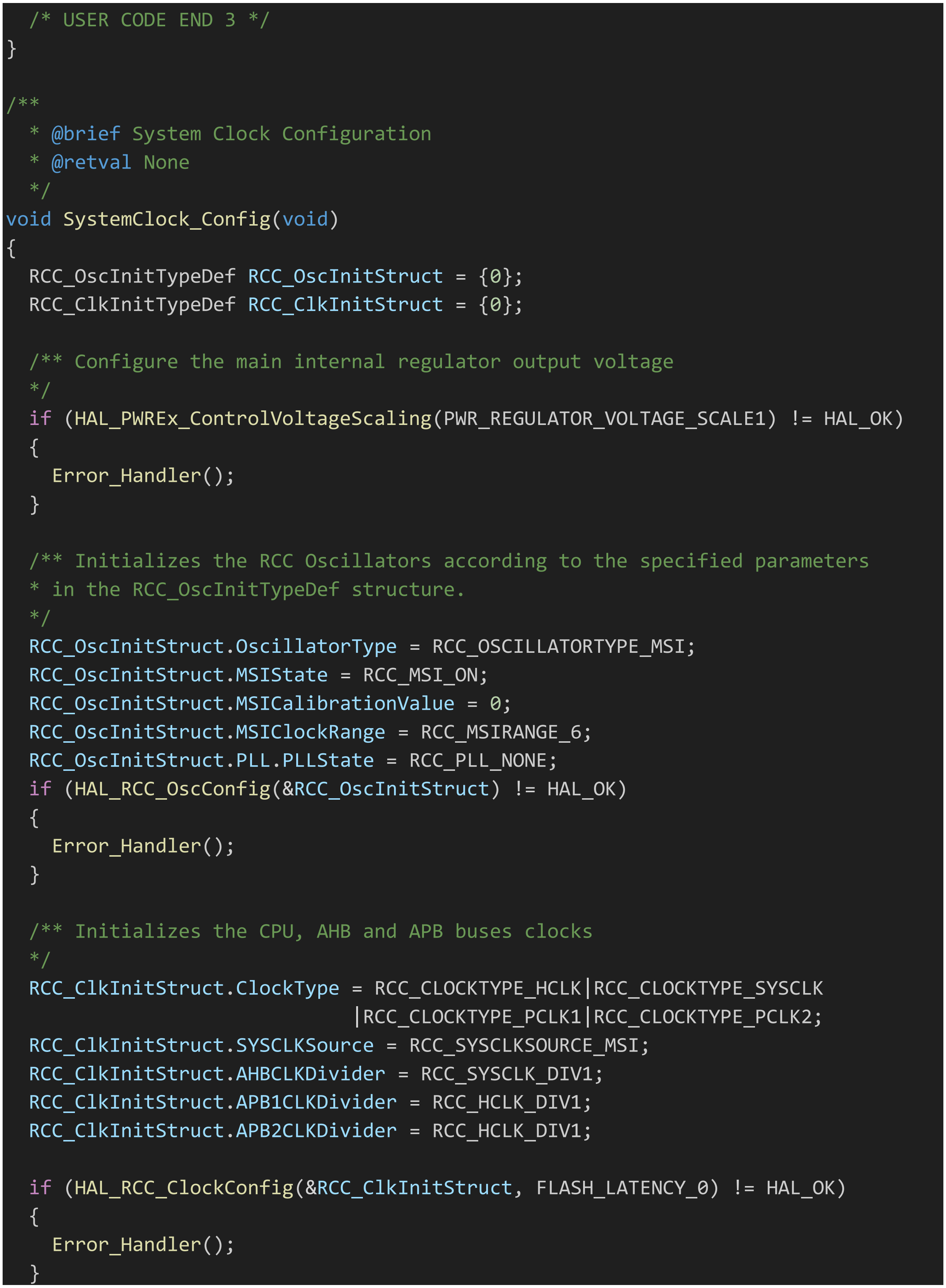
Flowchart:


Start
Include Libraries
Declare Constants
Function Prototypes
Initialize Hardware
Poll ADC
Record ADC Reading
Convert ADC to Voltage
Convert Voltage to Temperature
Temperature > 20°C?
No
Turn LED Off
Yes
Turn LED On
Delay 1 Second

Source Code:

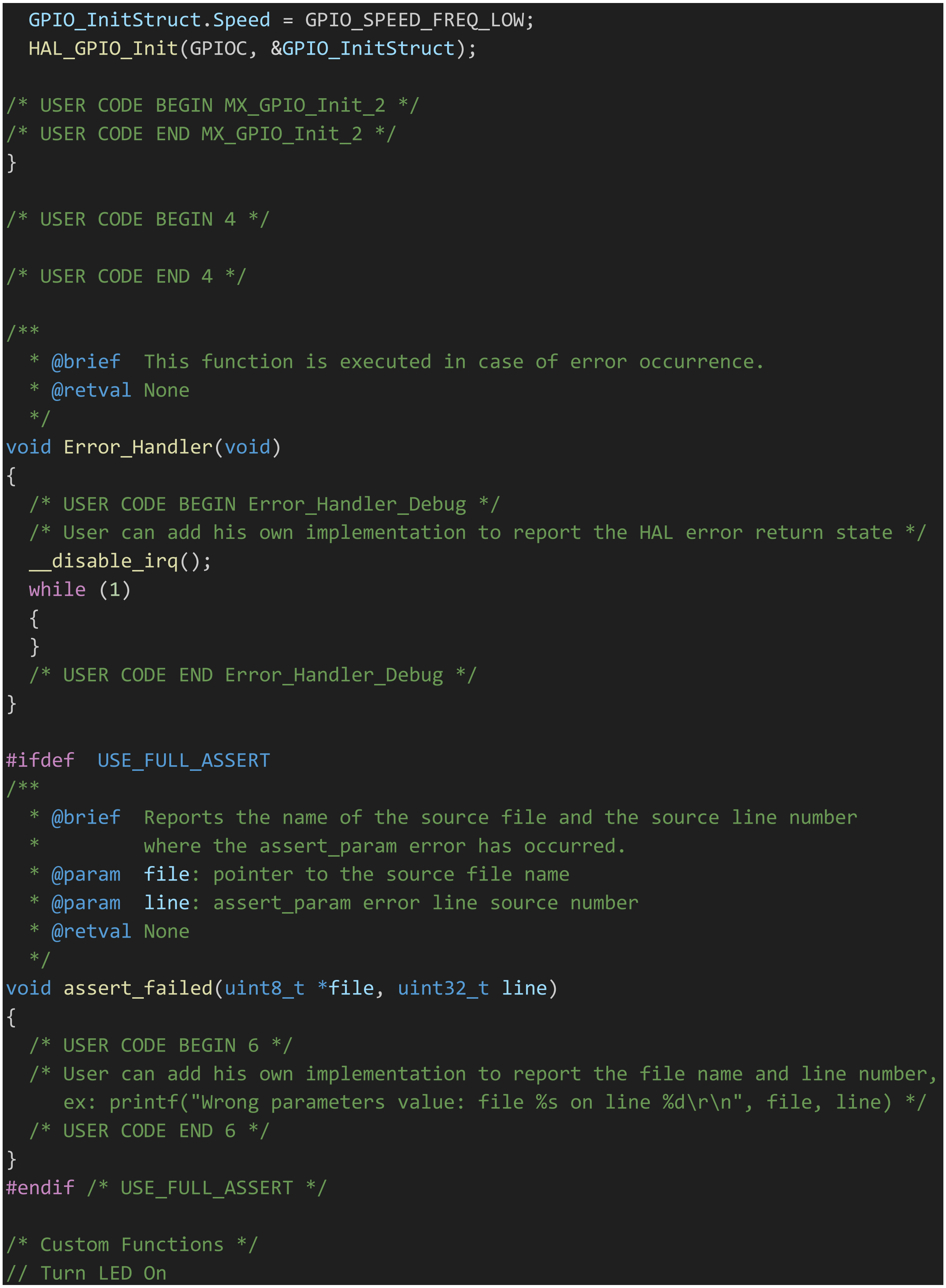
```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (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);
  }
```



```c
  /* 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;
```



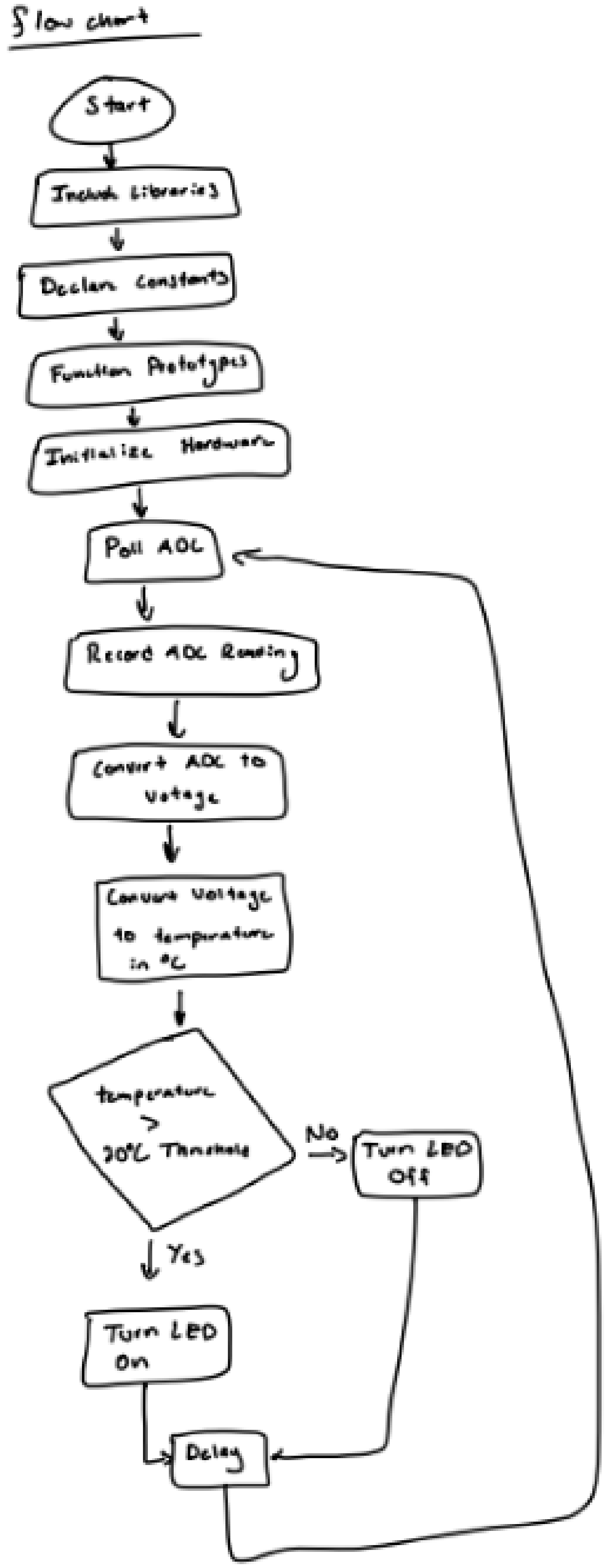
```c
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/* 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
```

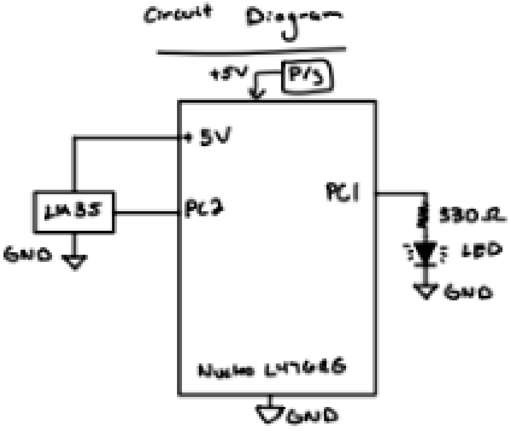
```
void ledOn() {
    HAL_GPIO_WritePin(GPIOC, LED_PIN, GPIO_PIN_SET);
}

// 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:





LAB9

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