Lab 7: STM32 Nucleo ADC and LDRs

EEE 174 / CpE 185

Lab Section (02)

Eric Telles

Tuesday 6:30-
9:10pm

Edgar Granados

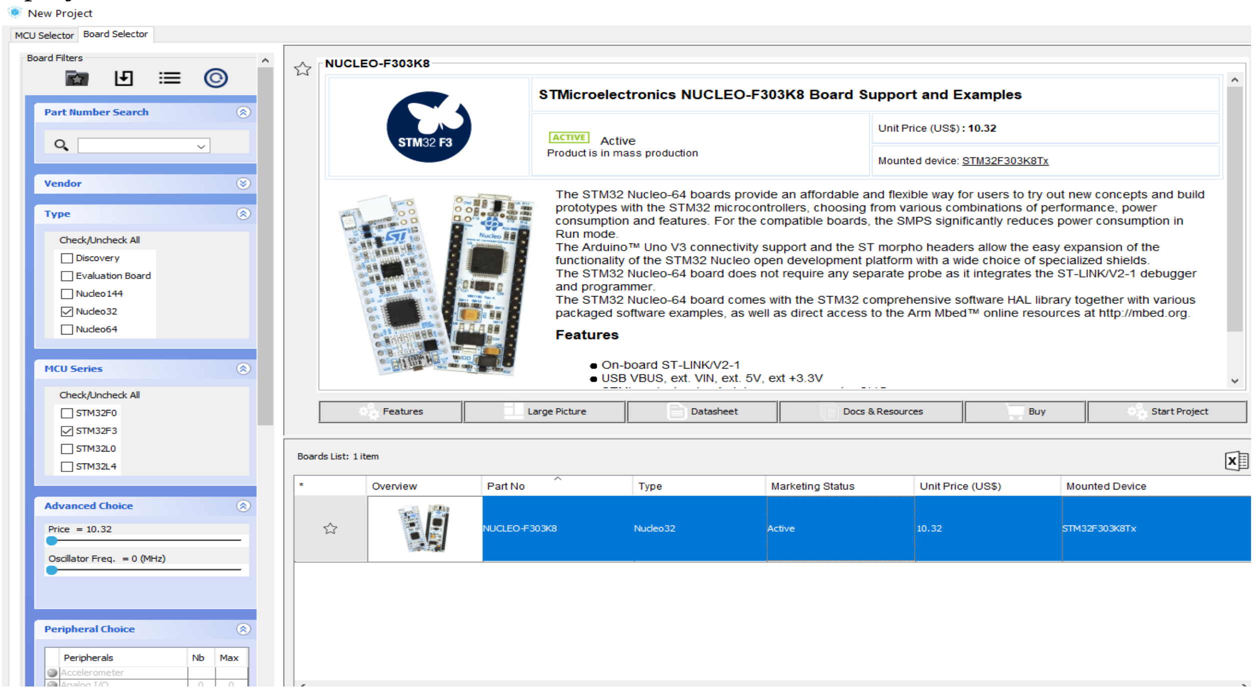
Lab Overview:

In this lab, we will take a look into setting up ADC conversion on the STM32 Nucleo board; specifically, the STM32F303K8. ADC stands for Analog to Digital Converter an it is used to convert an analog signal, to a digital value. In other words, it is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. Typically, the digital output is a two's complement binary number that is proportional to the input, but there are other possibilities. The way the converter accomplishes this by sampling the Analog signal and taking in small discrete samples values from the analog signal. The higher sampling rate a converter has, the more accurate and precise it is when outputting its value. Moreover, ADC also have another distinct feature, and that is its Resolution. To be more specific, a ADC's resolution is the range of numbers it can output to represent an analog signal. The resolution of an ADC is determined by the reference input and by the word width. The resolution defines the smallest voltage change that can be measured by the ADC. If we have an 8-bit resolution ADC, this means that it can output any value between 0 and 255.

Objective: The main objective of this lab will be to gain experience using the Analog to Digital Converters built in on the STM32 board. We will learn how to assign the pins needed by using the application STM32CUBEMX. In addition, we will also gain experience using the Attolic IDE to program our board to read in values from the LDRS and output a digital value.

Part I: Setting Up the ADC Parameters on STM32CUBEMX

The first step of this lab was to launch the application and create a new project. We needed to select the correct board for this, so on the Board Selector Tab we had to "check mark" the proper parameters so that the correct nucleo board was chosen. Figure 7.1 demonstrates the boxes that were checked. Once, the board is selected, we can now click the "Start Project" button to initiate the project.



Type: ☐ Discovery ☐ Evaluation Board ☐ Nucleo144 ☑ Nucleo32 ☐ Nucleo64

MCU Series: ☐ STM32F0 ☑ STM32F3 ☐ STM32L0 ☐ STM32L4

Advanced Choice: Price = 10.32; Oscillator Freq. = 0 (MHz)

Peripheral Choice:

Peripherals	Nb	Max
Accelerometer		
Analog I/O	0	0

NUCLEO-F303K8

STMicroelectronics NUCLEO-F303K8 Board Support and Examples

ACTIVE Active
Product is in mass production

Unit Price (US$) : **10.32**

Mounted device: STM32F303K8Tx

The STM32 Nucleo-64 boards provide an affordable and flexible way for users to try out new concepts and build prototypes with the STM32 microcontrollers, choosing from various combinations of performance, power consumption and features. For the compatible boards, the SMPS significantly reduces power consumption in Run mode.
The Arduino™ Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.
The STM32 Nucleo-64 board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger and programmer.
The STM32 Nucleo-64 board comes with the STM32 comprehensive software HAL library together with various packaged software examples, as well as direct access to the Arm Mbed™ online resources at http://mbed.org.

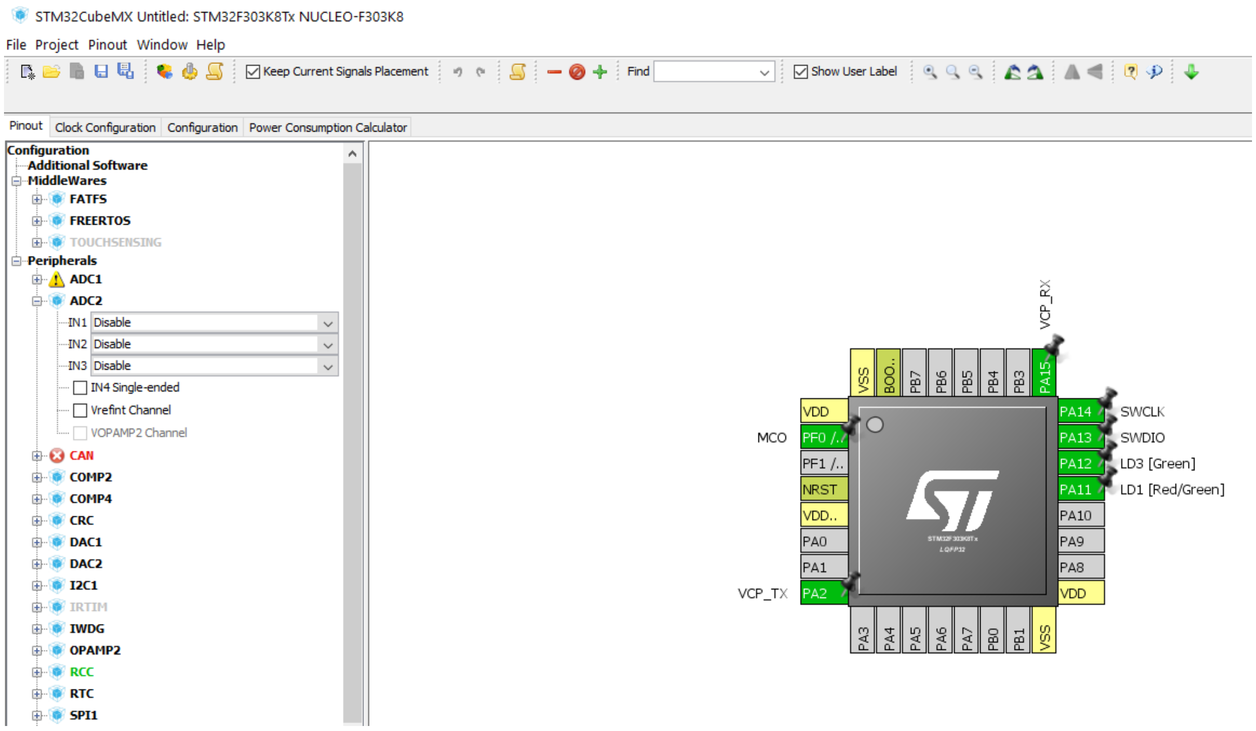
Features

- On-board ST-LINK/V2-1
- USB VBUS, ext. VIN, ext. 5V, ext +3.3V

Boards List: 1 item

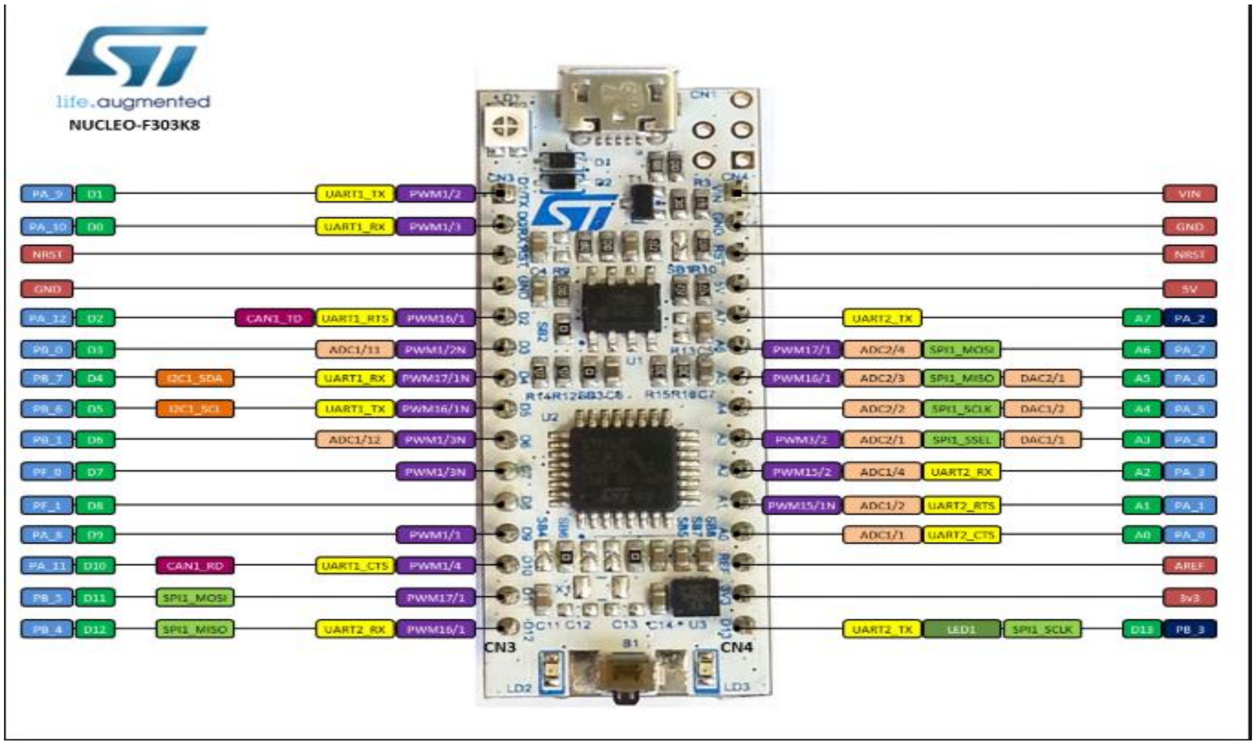
*	Overview	Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		NUCLEO-F303K8	Nucleo32	Active	10.32	STM32F303K8Tx

Once the project has launched, we will be prompted with this windows (Figure 7.2). On this specific board, there are two built-in ADC systems that we can utilize to our advantage. By systems, we mean that two independent entities can assign certain pins to read out digital values from devices. In total, there are about 9 pins that can utilize Analog to Digital Conversion.

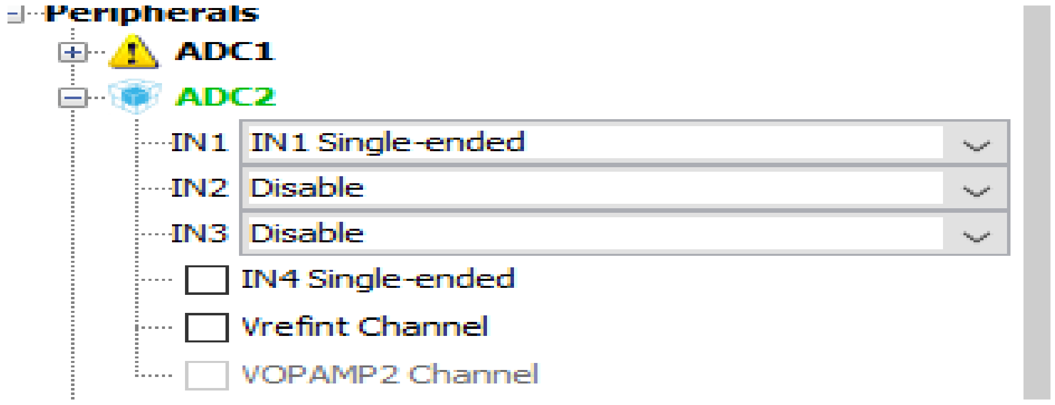


IN1 Disable
IN2 Disable
IN3 Disable
☐ IN4 Single-ended
☐ Vrefint Channel
☐ VOPAMP2 Channel

Figure 7.2: Prompted window after project has been created


life.augmented
NUCLEO-F303K8
PA_9 D1 UART1_TX PWM1/2
PA_10 D0 UART1_RX PWM1/3
NRST
GND
PA_12 D2 CAN1_TD UART1_RTS PWM16/1
PB_0 D3 ADC1/11 PWM1/2N
PB_7 D4 I2C1_SDA UART1_RX PWM17/1N
PB_6 D5 I2C1_SCL UART1_TX PWM16/1N
PB_1 D6 ADC1/12 PWM1/3N
PF_0 D7 PWM1/3N
PF_1 D8
PA_8 D9 PWM1/1
PA_11 D10 CAN1_RD UART1_CTS PWM1/4
PB_5 D11 SPI1_MOSI PWM17/1
PB_4 D12 SPI1_MISO UART2_RX PWM16/1
CN3
VIN
GND
NRST
5V
UART2_TX A7 PA_2
PWM17/1 ADC2/4 SPI1_MOSI A6 PA_7
PWM16/1 ADC2/3 SPI1_MISO DAC2/1 A5 PA_6
ADC2/2 SPI1_SCLK DAC1/2 A4 PA_5
PWM3/2 ADC2/1 SPI1_SSEL DAC1/1 A3 PA_4
PWM15/2 ADC1/4 UART2_RX A2 PA_3
PWM15/1N ADC1/2 UART2_RTS A1 PA_1
ADC1/1 UART2_CTS A0 PA_0
AREF
3v3
UART2_TX LED1 SPI1_SCLK D13 PB_3
CN4

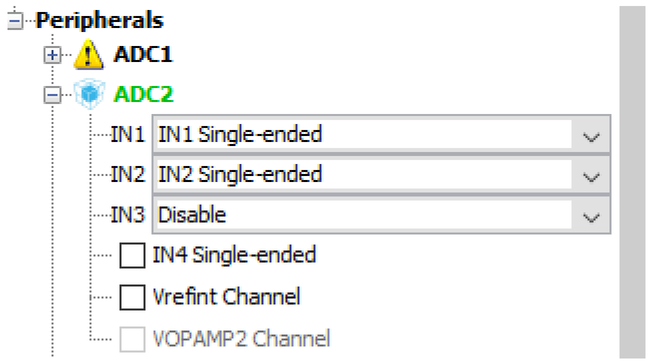
The next step will be to assign the pins we want to read out the data. To do this, we move our cursor to the left-hand side and choose ADC1 or ADC2. Once hovering over one of these, we click the option to expand it, and we will be prompted with four options 5 to 6 options. Figure 7.4 demonstrates the options shown after ADC2 was expanded.



Peripherals
ADC1
ADC2
IN1 IN1 Single-ended
IN2 Disable
IN3 Disable
☐ IN4 Single-ended
☐ Vrefint Channel
☐ VOPAMP2 Channel

Figure 7.4: Expanded options for ADC2

All of the options that read "INx" are pins that can be used to convert analog signals to digital values. For this lab, we will only need two of these pins so for IN1 we will select the option " Single Ended" and for IN2 the same. Differential Option is a dynamic option, that reads input from two pins and outputs the difference of those two pins. For this lab, this is not required but it may be useful to know for future labs. Also, take note of the pins that are assigned ADC on the right side of the chip diagram. Figure 7.5 and 7.6 demonstrate these changes.



Peripherals
ADC1
ADC2
IN1 IN1 Single-ended
IN2 IN2 Single-ended
IN3 Disable
☐ IN4 Single-ended
☐ Vrefint Channel
☐ VOPAMP2 Channel

Figure 7.5: Choosing Single-ended options for ADC pins

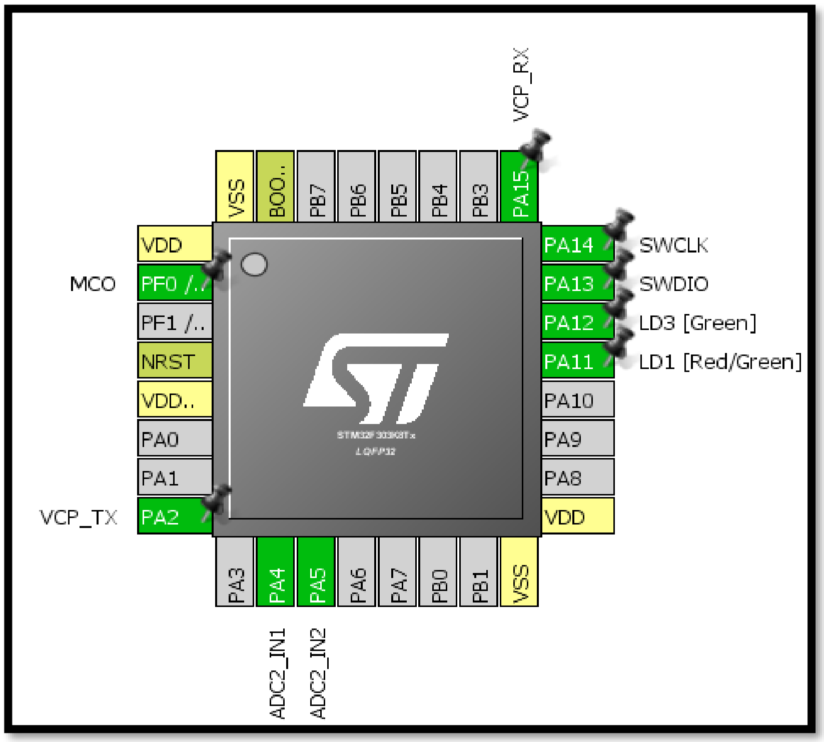
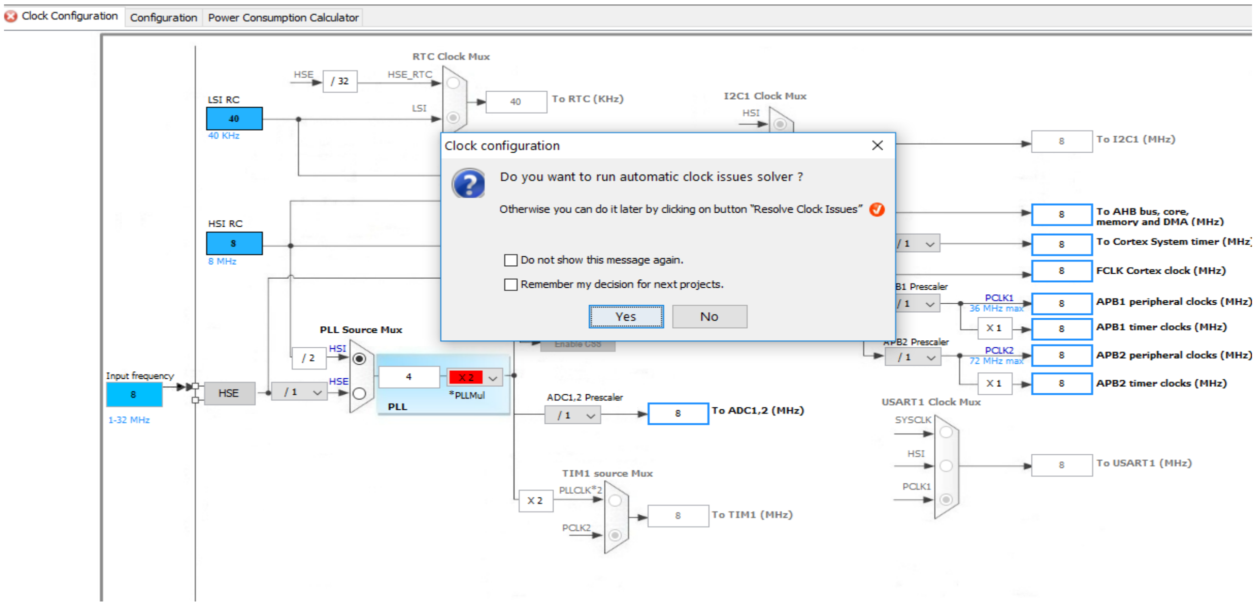


Figure 7.6: Chip Diagram of after ADC pins are assigned to the MCU

Part II: Setting up the ADC parameters

Once the pin configuration is complete, we will need to adjust the internal clock frequency of the pins. To do this, we must hover over to the clock configuration tab and select it. Once here we may be prompted with an error message, saying that there is an error in the clock configuration and if we want to automatically resolve it. Click “Yes” to solve the problem. On the STM32 board, ADC can function up to 16 MHz, therefore we can leave it running at the frequency, or lower it. In my case, I decided to lower the frequency to about 4MHz as the higher frequency was not necessarily needed for this lab.



Clock configuration

Do you want to run automatic clock issues solver ?

Otherwise you can do it later by clicking on button "Resolve Clock Issues"

☐ Do not show this message again.

☐ Remember my decision for next projects.

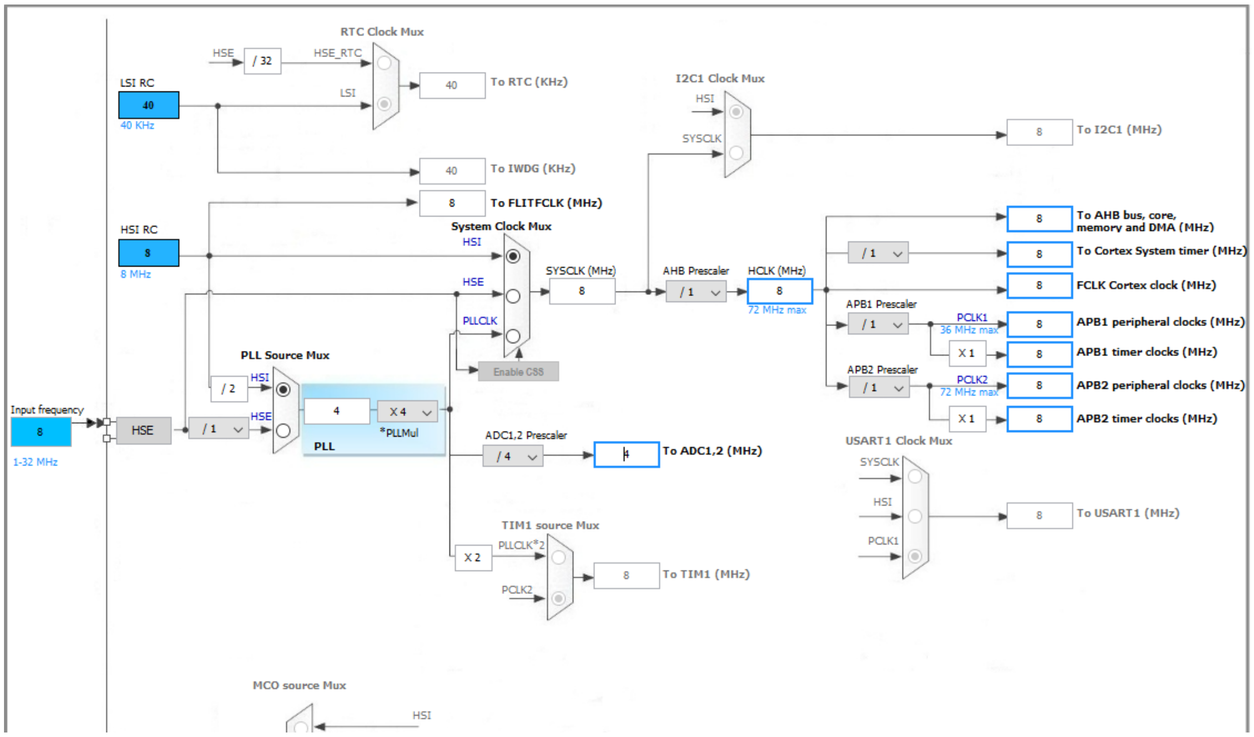
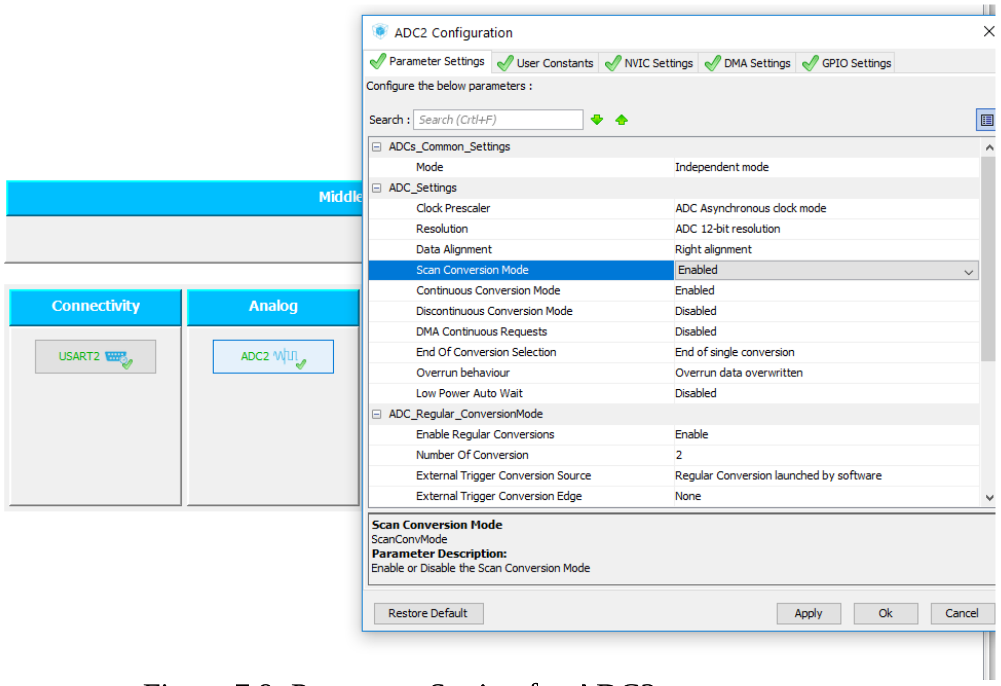


Figure 7.8: Clock Configuration changed to 4 MHz

The final step for this part will be to set the proper reading parameters for the ADCs. To do this we will now hover over to the Configuration tab and in it select the ADC2 button listed under the ADC category. Once here we will need to change a couple of options to have the MCU properly set up to read values from two LDRs.



Configure the below parameters :

ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	ADC Asynchronous clock mode
Resolution	ADC 12-bit resolution
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	End of single conversion
Overrun behaviour	Overrun data overwritten
Low Power Auto Wait	Disabled
ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable
Number Of Conversion	2
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None

Scan Conversion Mode
ScanConvMode
Parameter Description:
Enable or Disable the Scan Conversion Mode

Figure 7.9: Parameter Setting for ADC2

Once everything is set we can now generate the code and open up the generated code using Attollic IDE studio.

Part III: Setting Up LDR circuit and reading values

The final step of this lab will be to wire everything up and read the values coming out of the LDRs. For this circuit, we will need two LDRs, two 100k Ohm resistors, and a couple of jumper wires. The following figure shows how an LDR circuit is wired to an Arduino board. Using the same principle, we will wire the LDR the same way with an input voltage applied to one side of the terminal and on the other side an ADC pin in between the LDR and resistor; and finally a wire connecting back to ground.

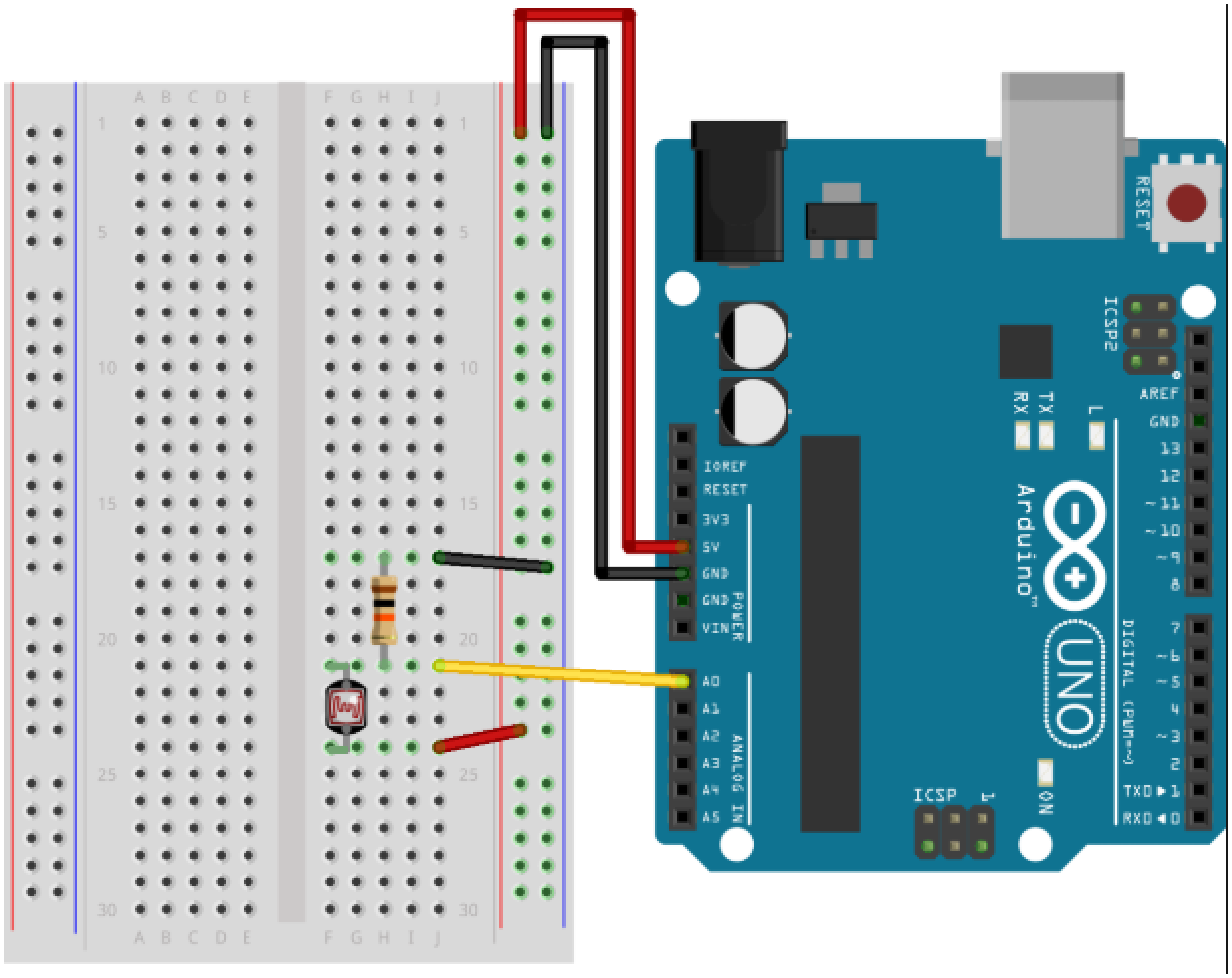
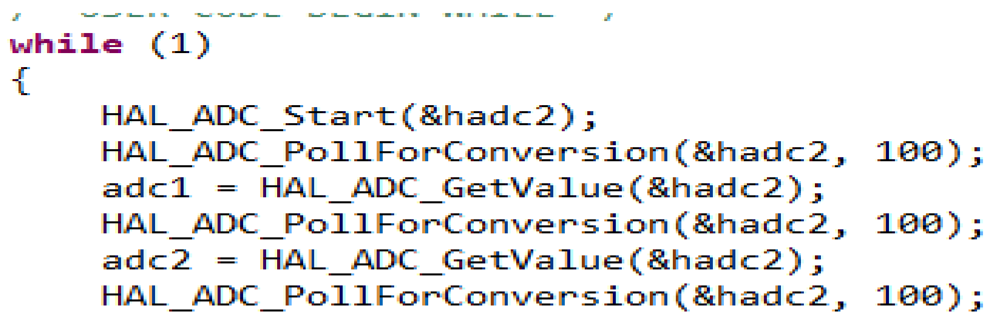


Figure 7.10: Example Circuit of How LDR is wired to Arduino Board

When everything is wired correctly we should have two LDRs powered with a 3.3 V input and an ADC pin wired in between the LDR and 100k Ohm resistor.

Now that the circuit is built, we will need to connect the ADC jumper wire to the correct pin on the nucleo board. In our case, since we had chosen IN1 and IN2 on the ADC2 category, these pins should correspond to pins A3 and A4 on the STM32 Nucleo board. The final part will be to code the commands within the generated code to read the values from the LDRs. To do this will use the function HAL_ADC_PollForConversion(&hadc2, 100). This function reads the value from the ADC pin and waits 100 ms to make sure that the function has enough time to read. We will need to use this function twice in order to get both readings of the LDRs. The following figure displays the code that was written to read the data from the LDR sensors.



```
while (1)
{
    HAL_ADC_Start(&hadc2);
    HAL_ADC_PollForConversion(&hadc2, 100);
    adc1 = HAL_ADC_GetValue(&hadc2);
    HAL_ADC_PollForConversion(&hadc2, 100);
    adc2 = HAL_ADC_GetValue(&hadc2);
    HAL_ADC_PollForConversion(&hadc2, 100);
```

Figure 7.11: Code used to Read the values from the LDRs

The variables adc1 and adc2 store the digital value for each of the LDRs. To confirm that they have a value stored in them we can debug the code and determine whether a value was properly read. Once completed, we have successfully set up and read values from the LDRs can converted them to digital values.

Conclusion:

The objective for this lab was to create a project that would utilize Analog to Digital Conversion. We were successful in setting up the proper parameters to accomplish this task. Once the code was generated, we needed to tell the MCU to read the values being outputted by the LDRs. Through some small lines of code, we were able to successfully store the outputted digital values onto a variable. Once we had enough practice using ADC, we could implement this knowledge into our final project, so that our program could read 4 LDRs readings instead of just two; and then send these readings using UART, back to the Raspberry Pi. Moreover, once the Raspberry Pi received this information, it would display the readings onto the Web Page created to control and monitor the Solar Tracker. In conclusion, I was able to set up Analog to Digital Conversion on the STM32 Nucleo Board to read out the data from LDR sensors.