

## **LAB-3**

# **Understanding Interrupts and Different ADC Modes**

### **CEG 7360 Embedded Systems**

**Name : - Keerthi Patil**

**UID : - U01110229**

**Email : - [patil.115@wright.edu](mailto:patil.115@wright.edu)**

**Name : - Dhamini Vootkuri**

**UID : - U01111434**

**Email : - [vootkuri.3@wright.edu](mailto:vootkuri.3@wright.edu)**

## Lab 3

# STM32 Configuration and Programming

### (Understanding Interrupts and Different ADC Modes)

#### **Introduction:**

This report outlines a series of experiments aimed at enhancing the understanding of interrupts and Analog-to-Digital Converter (ADC) configurations using the STM32 microcontroller.

The first experiment involved setting up an external interrupt on pin PC13, renamed pwBtn , which toggled an LED (LD2) when the button was pressed. This application illustrated how external events can prompt specific actions in the microcontroller, demonstrating the effectiveness of interrupt-driven programming compared to traditional polling techniques.

The following experiments delved into various ADC functionalities. The second experiment configured multiple ADC channels (A0, A1, A2) in scan mode to facilitate simultaneous readings, with results sent via UART. The third experiment incorporated Direct Memory Access (DMA) for improved data handling, where an interrupt was triggered once the ADC buffer was filled. Lastly, the fourth experiment explored continuous mode with DMA, enabling continuous data acquisition and highlighting the microcontroller's real-time capabilities. Collectively, these experiments provided valuable insights into interrupt management and ADC applications in embedded systems.

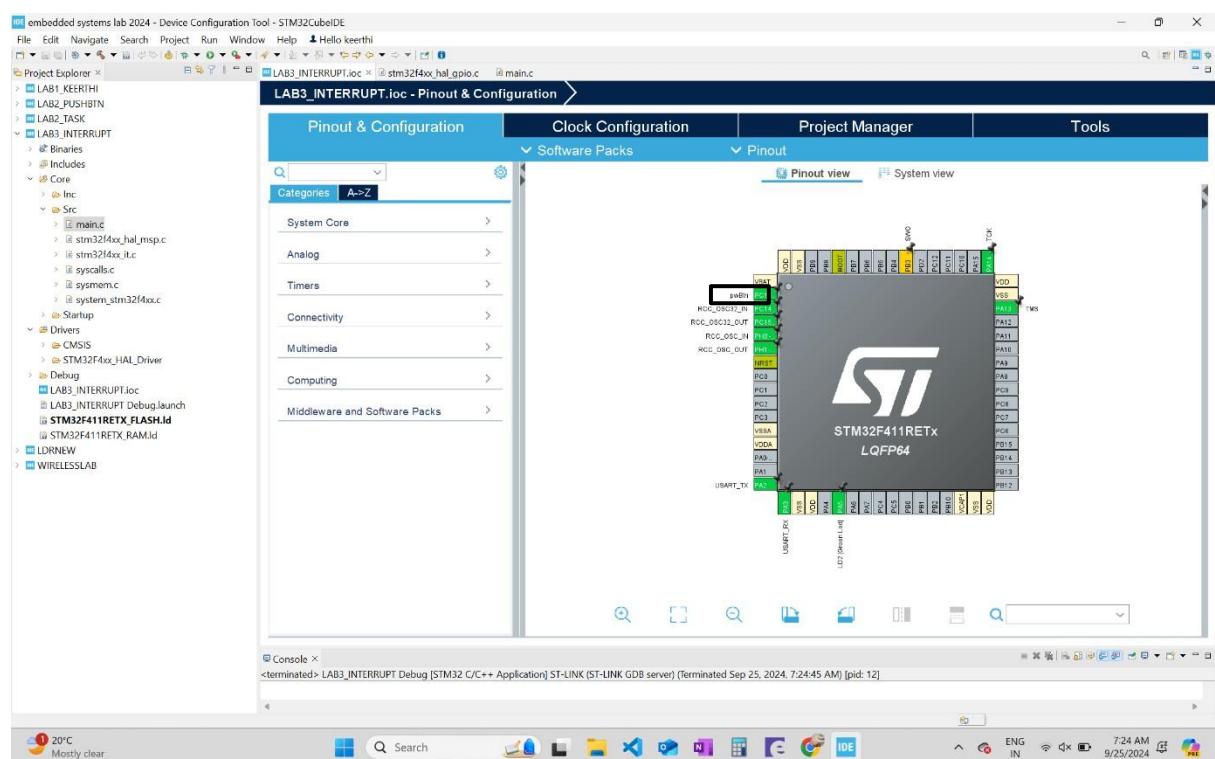
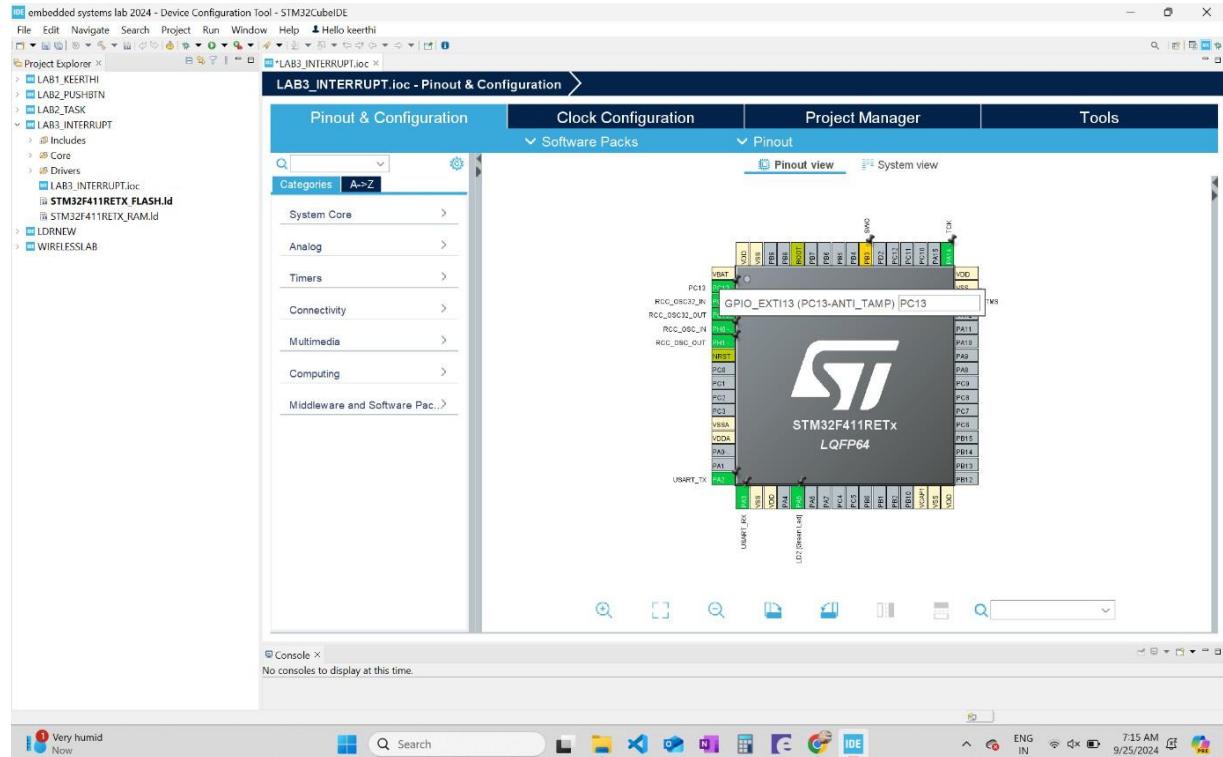
#### **Materials:**

- Software tools: STM32CubeIDE
- Hardware tools: STM32 Nucleo Board, 3 resistors, 3 LDR's (Light-dependent resistor) , Breadboard, jumper wires, USB Cable.

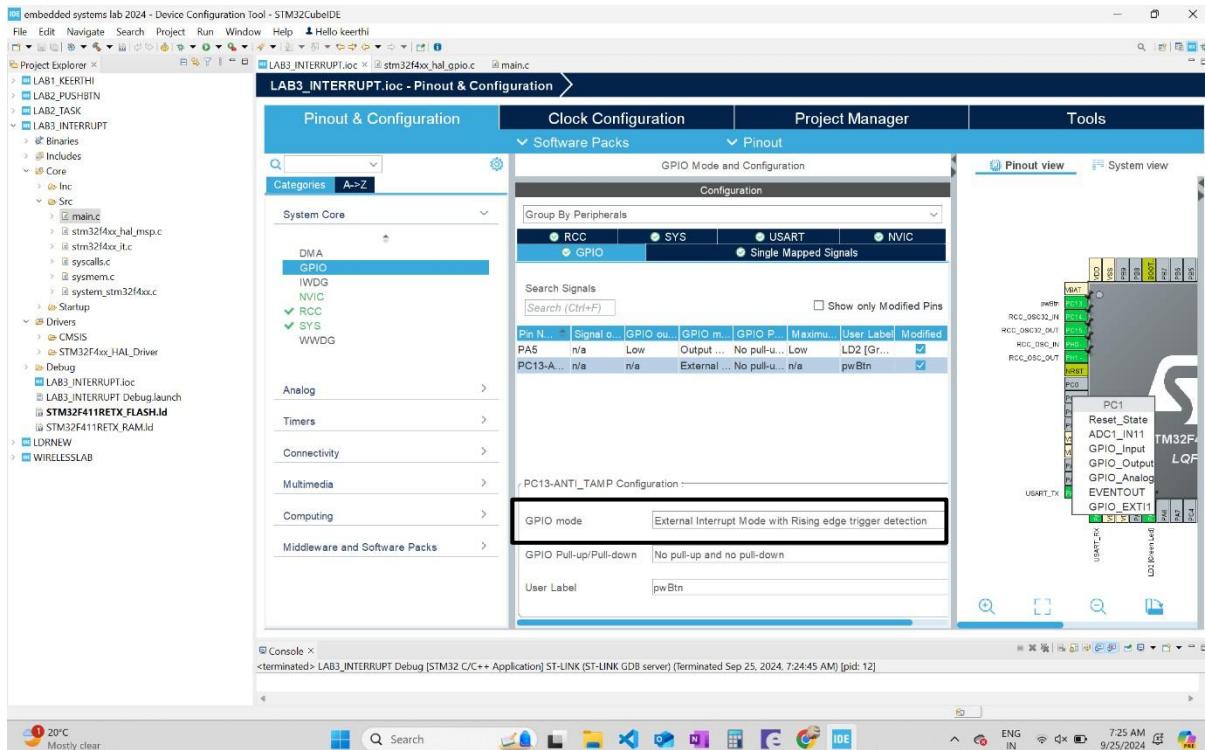
## Procedure:

### Part 1. GPIO External Interrupt (EXTI) Configuration

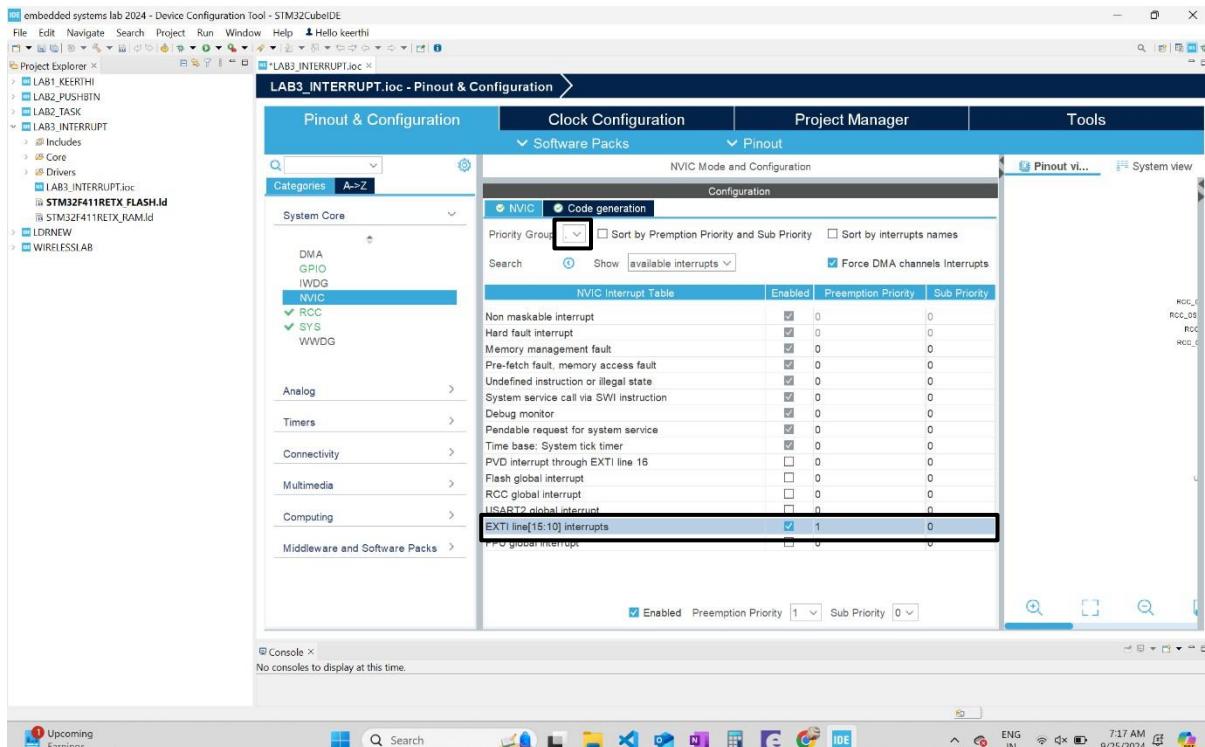
- To configure the GPIO External Interrupt, I have set the PC13 pin to **GPIO\_EXTI13** and named it as **pwBtn**.



- In System Core, Under GPIO I have set the Pin PC13 mode to be **Rising edge trigger detection**.



- In System Core, Under NVIC Set the **priority group** as **4-bit**.
- After selecting EXTI line[15:10] interrupts, I have Set the Preemption priority to 1.



- After saving, Under Drivers in STM32F4xx\_HAL\_Driver in **stm32f4xx\_hal\_gpio.c** I have copied the code from the line 507-514 as highlighted below to program the interrupt.

```

489 * @brief This function handles EXTI interrupt request.
490 * @param GPIO_Pin Specifies the pins connected EXTI line
491 * @retval None
492 void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
493 {
494     /* EXTI line interrupt detected */
495     if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
496     {
497         __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
498         HAL_GPIO_EXTI_Callback(GPIO_Pin);
499     }
500 }
501 /*@*/
502 /**
503  * @brief EXTI line detection callbacks.
504  * @param GPIO_Pin Specifies the pins connected EXTI line
505  * @retval None
506  */
507 weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
508 {
509     /* Prevent unused argument(s) compilation warning */
510     UNUSED(GPIO_Pin);
511     /* NOTE: This function Should not be modified, when the callback is needed,
512      the HAL_GPIO_EXTI_Callback could be implemented in the user file
513 */
514 }
515 /**
516  * @brief EXTI line detection callbacks.
517  * @param GPIO_Pin
518  */
519 /**
520  * @brief EXTI line detection callbacks.
521  * @param GPIO_Pin
522  */
523 /**
524 */
525 #endif /* HAL_GPIO_MODULE_ENABLED */
526 /**
527  */
528 /**
529  */
530 /**
531  */
532 /**
533 */
534

```

- The copied code from **stm32f4xx\_hal\_gpio.c** is pasted in **main.c** in the line 228 under **USER CODE BEGIN 4** and I have made few changes and added a code in while loop.

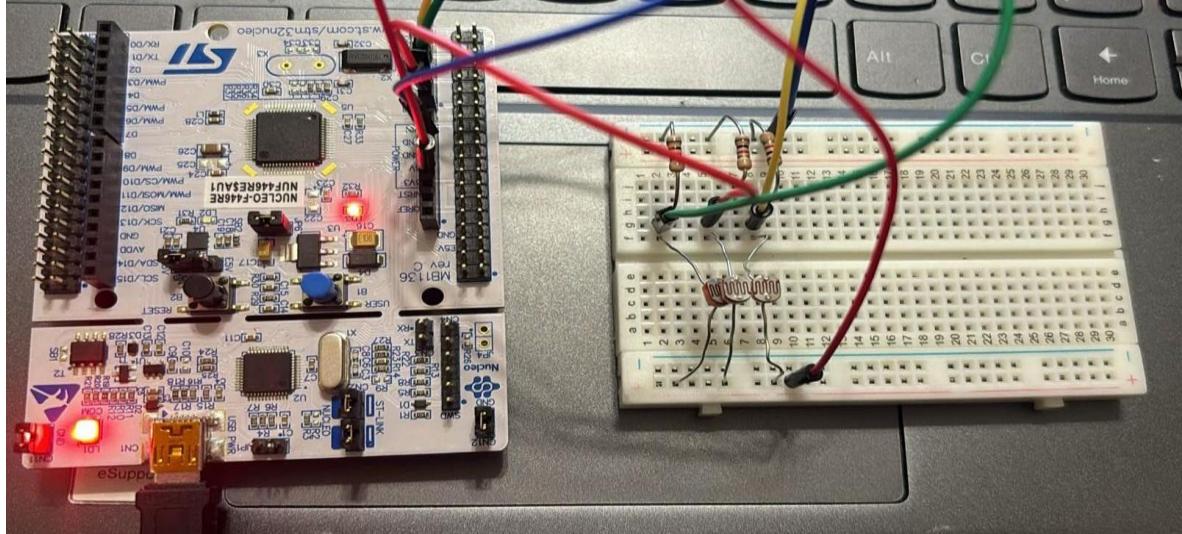
```

191 static void MX_GPIO_Init(void)
192 {
193     GPIO_InitTypeDef GPIO_InitStruct = {0};
194     /* USER CODE BEGIN MX_GPIO_Init_1 */
195     /* USER CODE END MX_GPIO_Init_1 */
196
197     /* GPIO Ports Clock Enable */
198     __HAL_RCC_GPIOC_CLK_ENABLE();
199     __HAL_RCC_GPIOH_CLK_ENABLE();
200     __HAL_RCC_GPIOA_CLK_ENABLE();
201     __HAL_RCC_GPIOB_CLK_ENABLE();
202
203     /*Configure GPIO pin Output Level */
204     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
205
206     /*Configure GPIO pin : pwBtn_Pin */
207     GPIO_InitStruct.Pin = pwBtn_Pin;
208     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
209     GPIO_InitStruct.Pull = GPIO_NOPULL;
210     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
211     HAL_GPIO_Init(pwBtn_GPIO_Port, &GPIO_InitStruct);
212
213     /*Configure GPIO pin : LD2_Pin */
214     GPIO_InitStruct.Pin = LD2_Pin;
215     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
216     GPIO_InitStruct.Pull = GPIO_NOPULL;
217     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
218     HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
219
220     /* EXTI interrupt init*/
221     HAL_NVIC_SetPriority(EXTI15_10_IRQn, 1, 0);
222     HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
223
224     /* USER CODE BEGIN MX_GPIO_Init_2 */
225     /* USER CODE END MX_GPIO_Init_2 */
226
227     /* USER CODE BEGIN 4 */
228 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
229 {
230     if(GPIO_Pin == pwBtn_Pin)
231         HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
232 }
233
234
235 /* USER CODE END 4 */
236
237 /**
238  * @brief This function is executed in case of error occurrence.
239  * @retval None
240 */

```

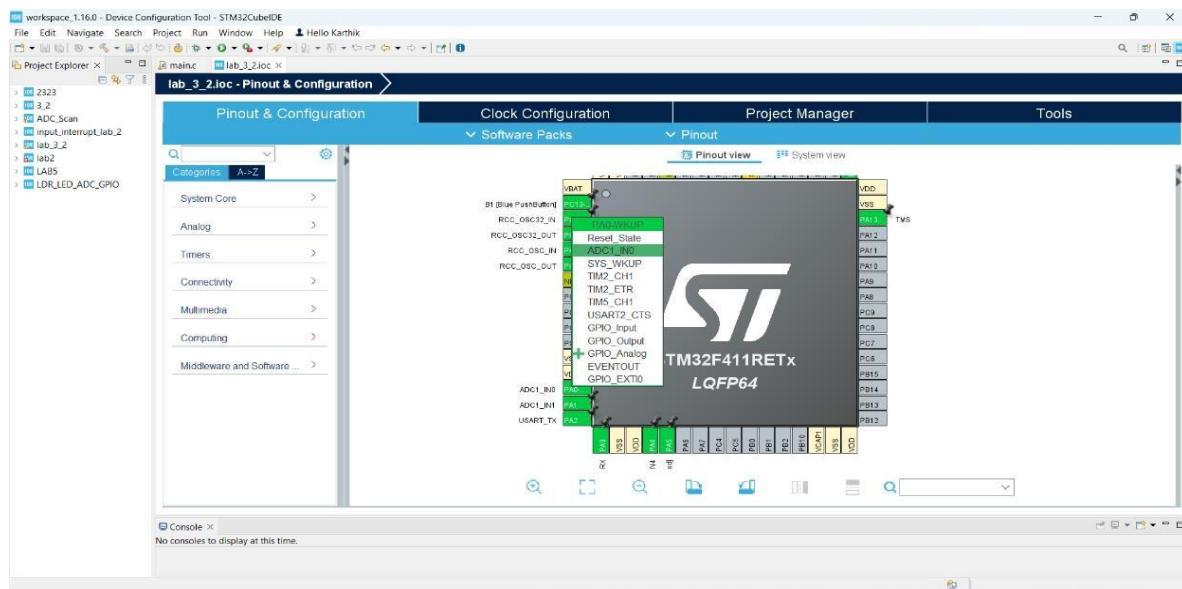
## Experimental Setup:

- Here the board is directly connected to the system using a USB cable and the blue switch on the board acts as push button.

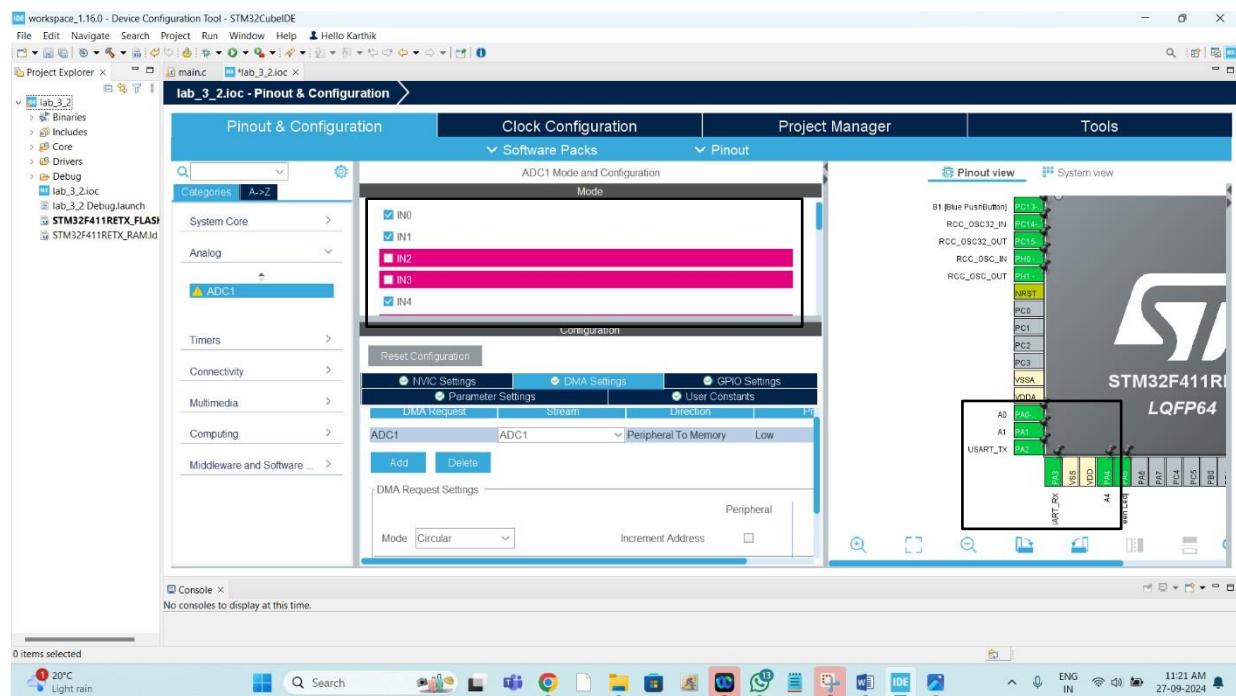


## Part 2. ADC Scan Mode Configuration

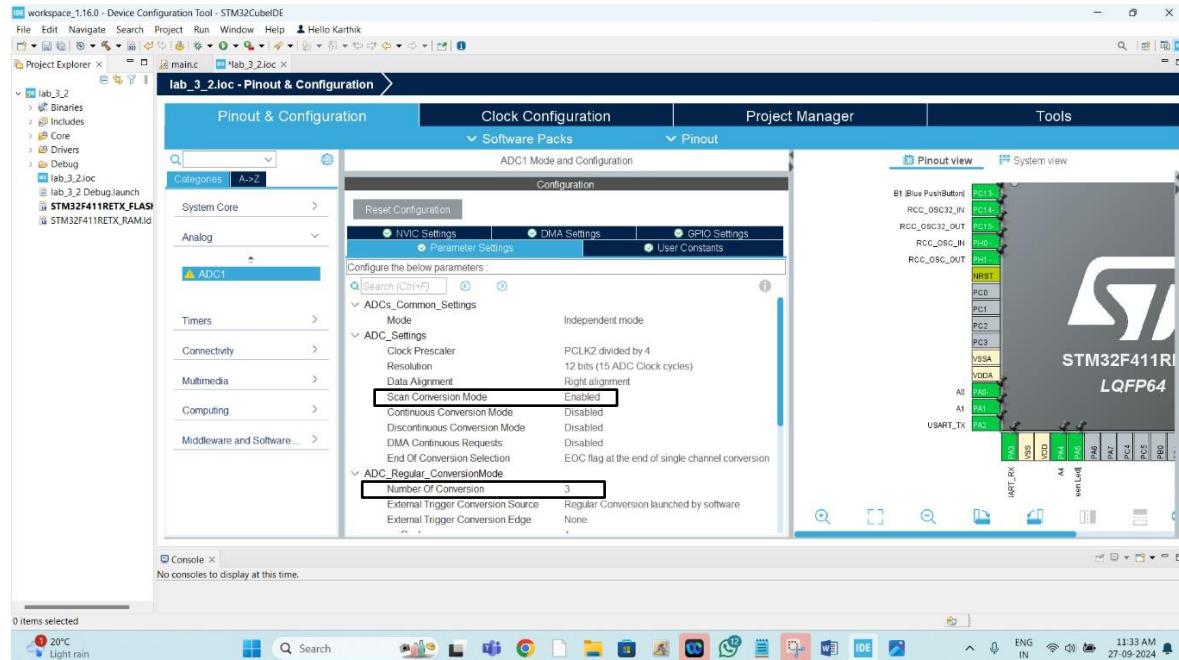
- The aim of the part 2 is to create 3 channels which can read the analog data and send it to the micro controller
- For this operation we have selected **PA0, PA1 and PA4** as **ADC1\_IN0, ADC1\_IN1 and ADC1\_IN2**



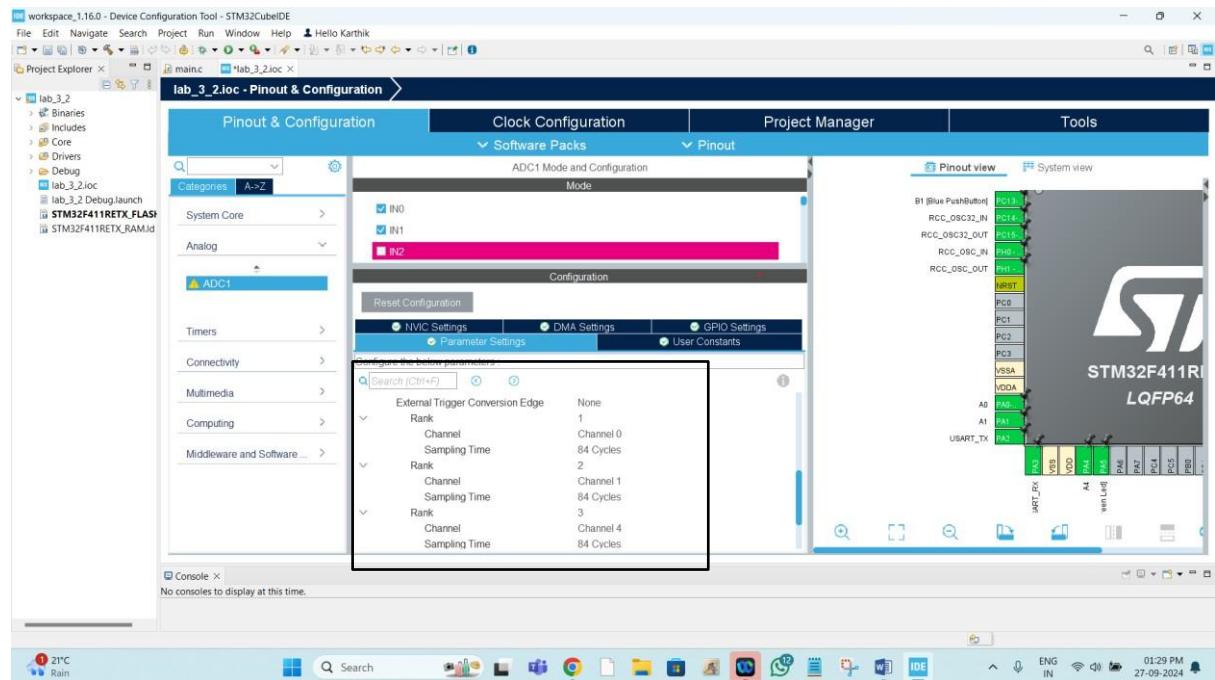
- Rename each ADC converter pin as A0, A1 and A4
- Select ADC1 mode and configuration as IN0, IN1 and IN4



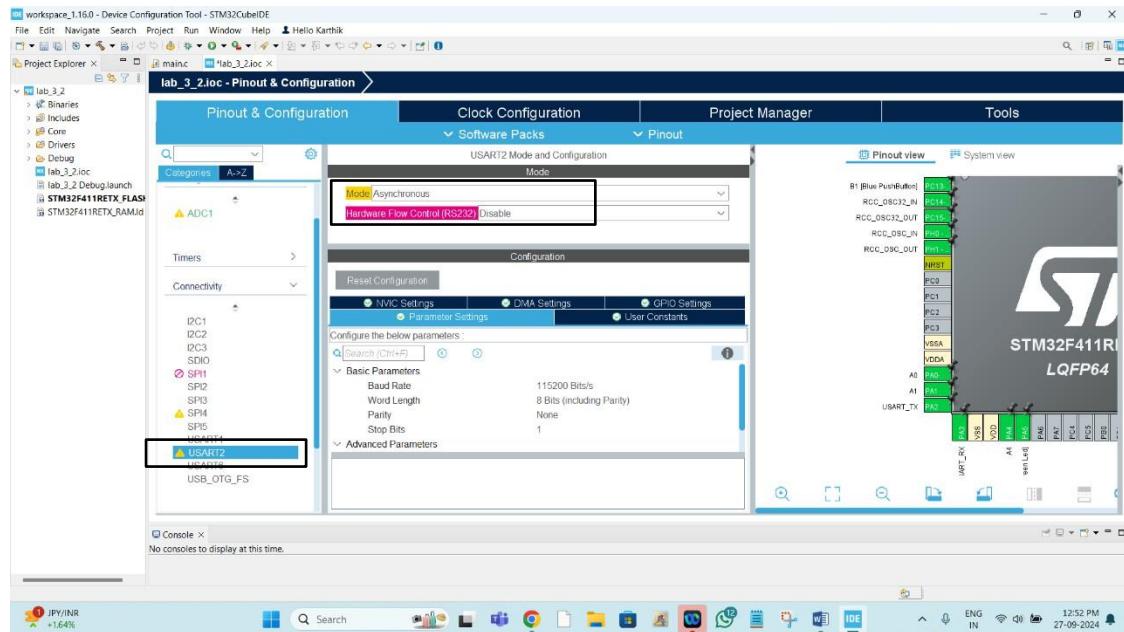
In system core under Analog in ADC1 , I have Enabled the scan conversion mode and increased the Number of conversions to 3



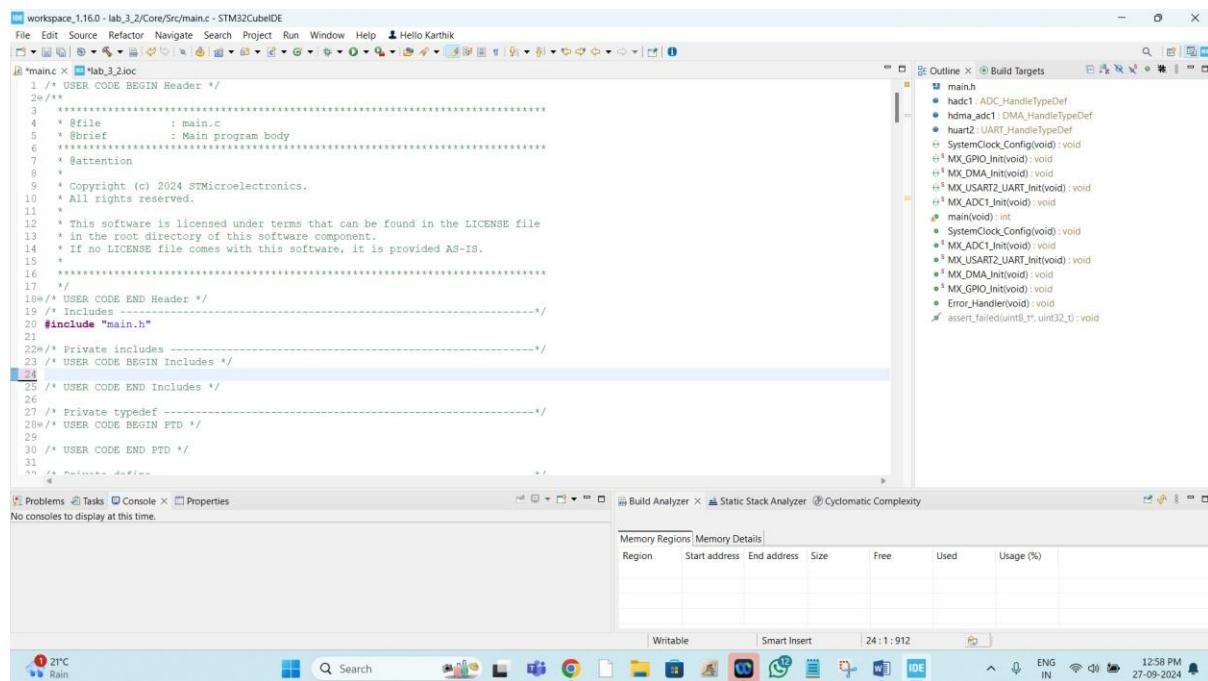
- I have made channel 0, channel 1 and channel 4 to 84 Cycles.



- I have made sure that **USART2** is in **Asynchronous mode**.



- After saving configuration the code is generated



- I made some changes to the code adding the library in main.c

The screenshot shows the STM32CubeIDE interface. The main window displays a C source file named 'main.c' with syntax highlighting for comments, strings, and code blocks. The code includes standard library headers like stdio.h and string.h, as well as driver initialization functions for ADC, DMA, GPIO, and USART. A copyright notice from STMicroelectronics is present. The sidebar on the right lists the build targets, which include main, stdio, string, and various peripheral drivers. The bottom of the screen features a taskbar with icons for Problems, Tasks, Console, Properties, and several system status indicators.

```
/* USER CODE BEGIN Header */
2/*
3 * ***** : main.c
4 * #brief : Main program body
5 * ****
6 * @attention
7 *
8 * Copyright (c) 2024 STMicroelectronics.
9 * All rights reserved.
10 *
11 * This software is licensed under terms that can be found in the LICENSE file
12 * in the root directory of this software component.
13 * If no LICENSE file comes with this software, it is provided AS-IS.
14 *
15 *
16 * ****
17 */
18/* USER CODE END Header */
19/* Includes - Please put in正确的头文件位置 */
20#include "main.h"
21
22/* Private includes - Please put in正确的头文件位置 */
23#include <string.h>
24#include <stdio.h>
25/* USER CODE END includes */
26
27/* Private typedef */
28/* USER CODE BEGIN PTD */
29
30/* USER CODE END PTD */
31
32/*
33 */
34
35/* USER CODE BEGIN Includes */
36#include <string.h>
37/* USER CODE END Includes */
38
39/* Private functions */
40/* USER CODE BEGIN Functions */
41
42/* USER CODE END Functions */
43
44/* USER CODE BEGIN Private Functions */
45
46/* USER CODE END Private Functions */

```

- Added the code which prints the digital value of the resistance(interruption) after processing it in the micro controller

The screenshot shows the STM32CubeIDE interface with the following details:

- Title Bar:** workspace\_1,1.6.0 - lab\_3\_2/Core/Src/main.c - STM32CubeIDE
- Main Editor:** The main.c file is open, showing code related to system initialization and ADC/DMA operations. A blue selection highlights the HAL\_UART\_Transmit call.

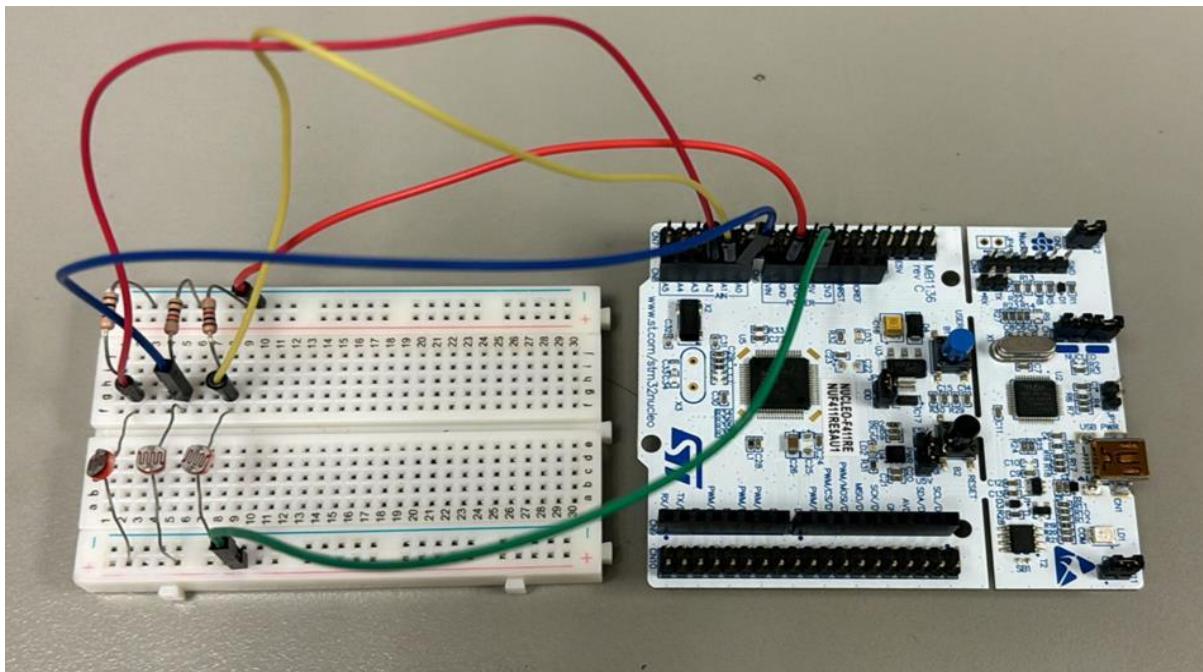
```
main.c
91  /* USER CODE BEGIN SysInit */
92  /* USER CODE END SysInit */
93
94  /* Initialize all configured peripherals */
95  MX_GPIO_Init();
96  MX_DMA_Init();
97  MX_USART2_UART_Init();
98  MX_ADC1_Init();
99
100 /* USER CODE BEGIN 2 */
101 volatile uint16_t adcDMA[3];
102 char txStr[100];
103 /* USER CODE END 2 */
104
105 /* Infinite loop */
106 /* USER CODE BEGIN WHILE */
107 while (1)
108 {
109     /* USER CODE END WHILE */
110
111     /* USER CODE BEGIN 3 */
112     HAL_UART_Transmit(txHandle, (uint8_t*) &txStr, 3);
113     sprintf(txStr, "Tx = %u\\tCm2 = %u\\tCm3 = %u\\r\\n", adcDMA[0], adcDMA[1], adcDMA[2]);
114     HAL_UART_Transmit(txHandle, (uint8_t*) txStr, strlen(txStr), HAL_MAX_DELAY);
115     HAL_Delay(50);
116 }
117 /* USER CODE END 3 */
118 }
```
- Outline View:** Shows the project structure with files like main.h, stdio.h, string.h, and various HAL initialization functions.
- Build Analysis:** Shows the following results:
  - Memory Regions: Memory Details
  - Regions: Region, Start address, End address, Size, Free, Used, Usage (%)
  - Build Status: Build Analyzer, Static Stack Analyzer, Cyclomatic Complexity
- Bottom Status Bar:** Shows the date (27-09-2024), time (01:03 PM), and language (ENG IN).

## Experimental Setup:

The below image depicts the hardware setup for the experiment to perform. The connections are made as follows

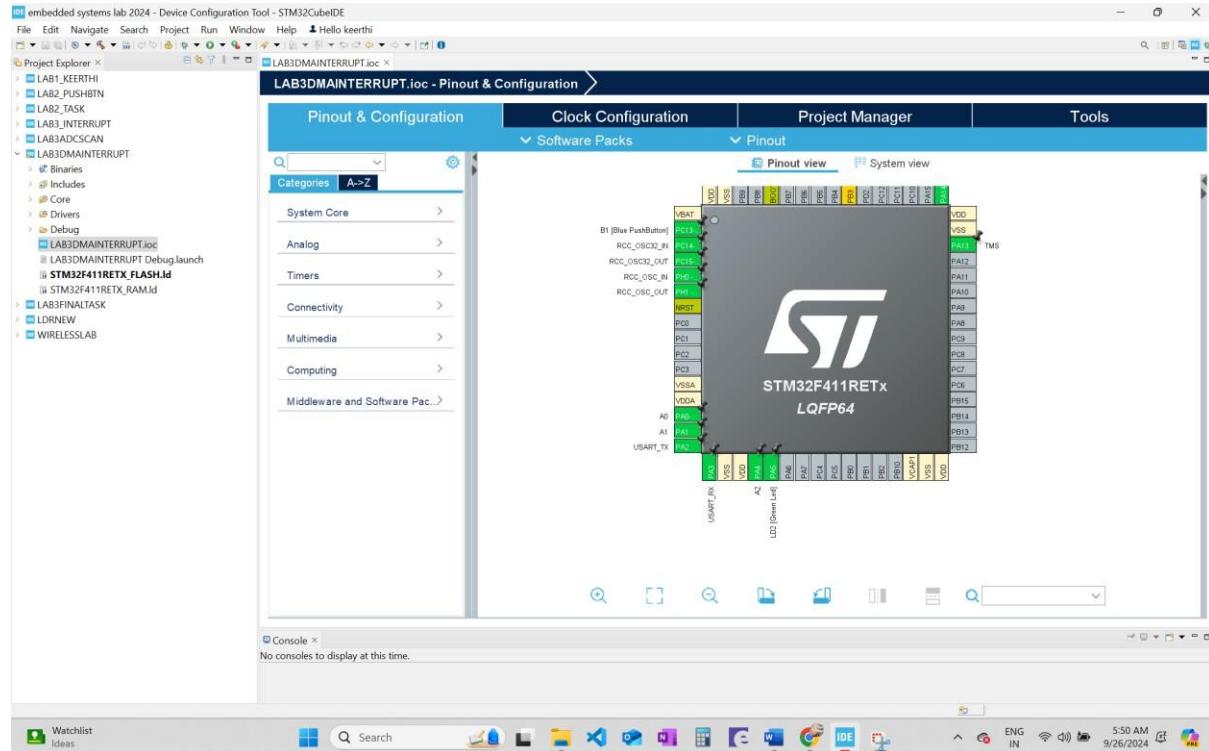
**1. LDR:** I have used 3 LDR's to perform the experiment, in which One of the terminal of all the 3 LDR's is connected to 5v on the board through a wire and the other terminal of first LDR is connected to A0 (because we configured the PA0 pin to ADC1\_IN0) and second LDR is connected to A1 and similarly the third LDR is connected to A2.

**2. Resistor:** I have used 3 resistors to perform the experiment in which three of the resistor terminals are connected in parallel to the LDR terminals that is connected to A0, A1, A2 pin. Other terminal of all the resistors is connected to the ground using a wire to the board.

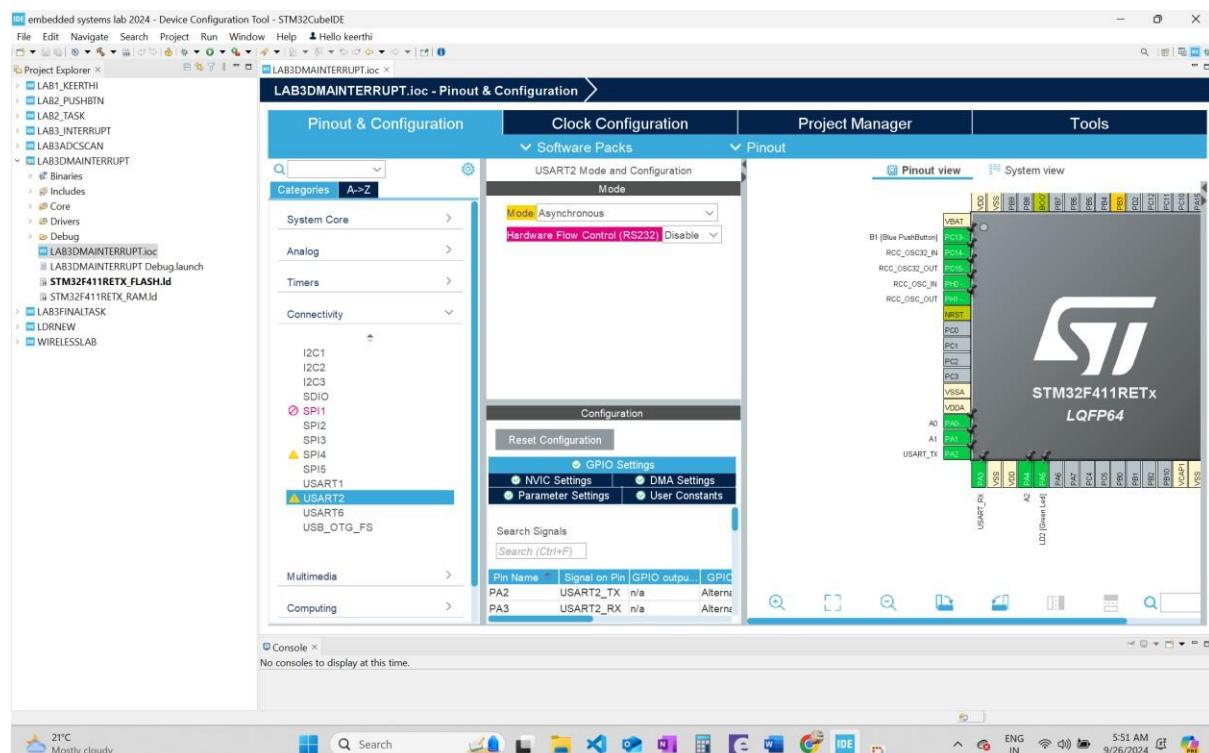


## Part 3. ADC Scan Mode with DMA (& Interrupt)

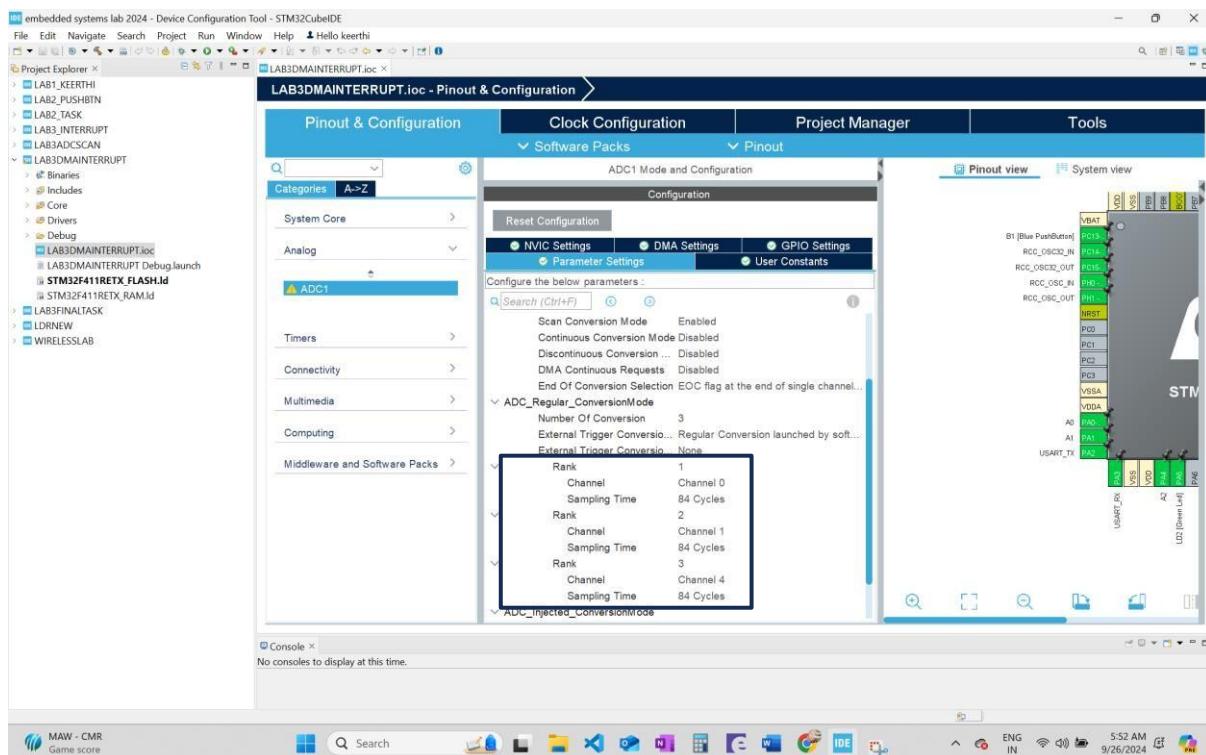
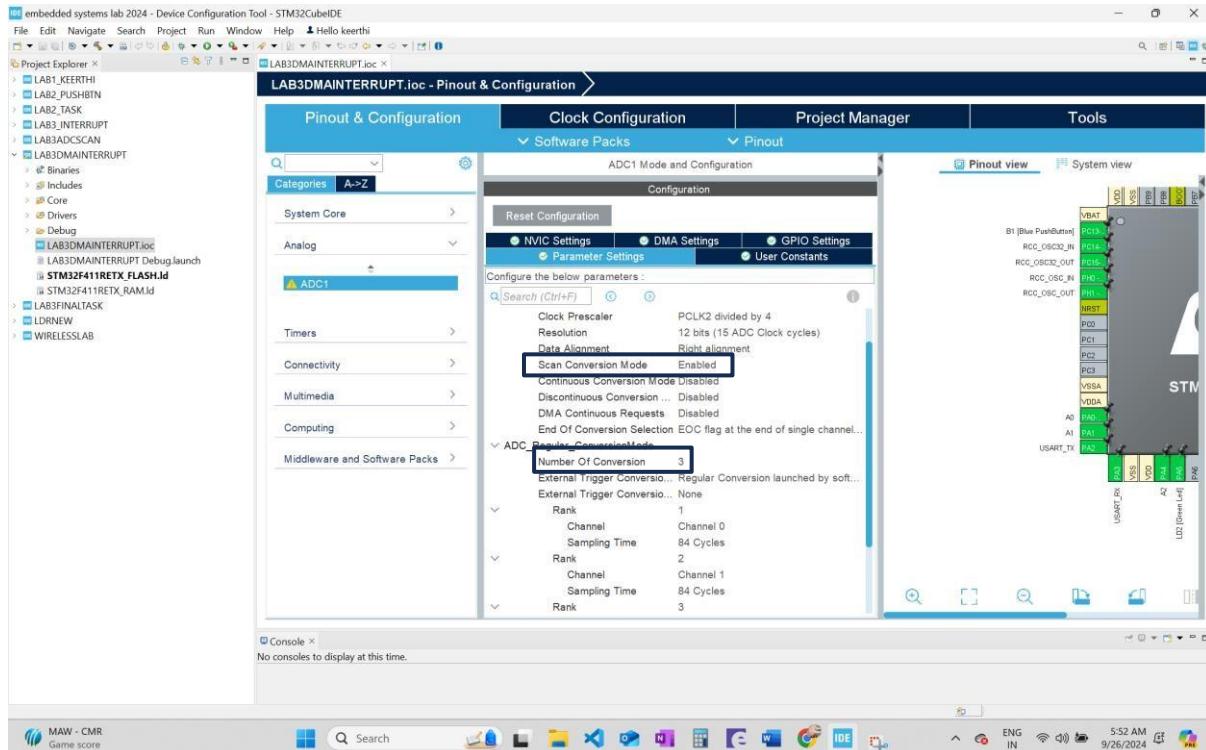
- I have selected the pin **PA0**, **PA1** and **PA4** as **ADC1 inputs** and renamed as **A0**, **A1** and **A2**.



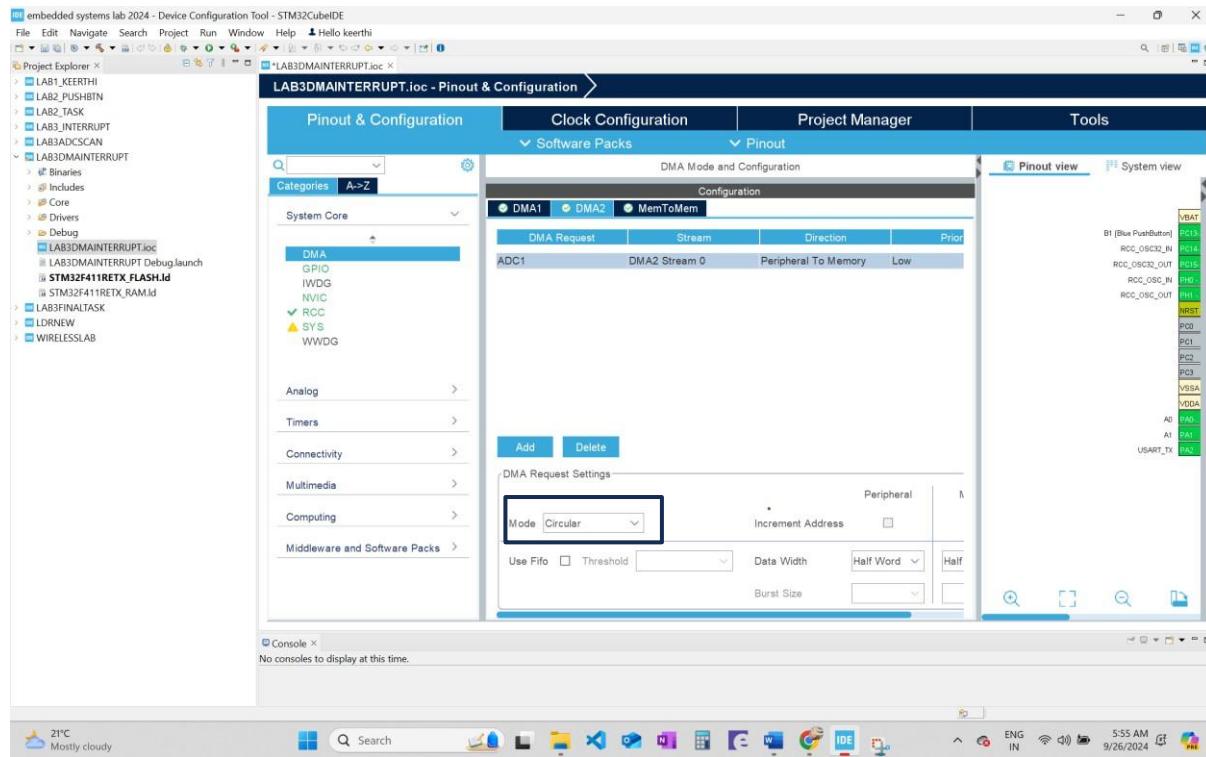
- I have made sure that **USART2** is in **Asynchronous mode**.



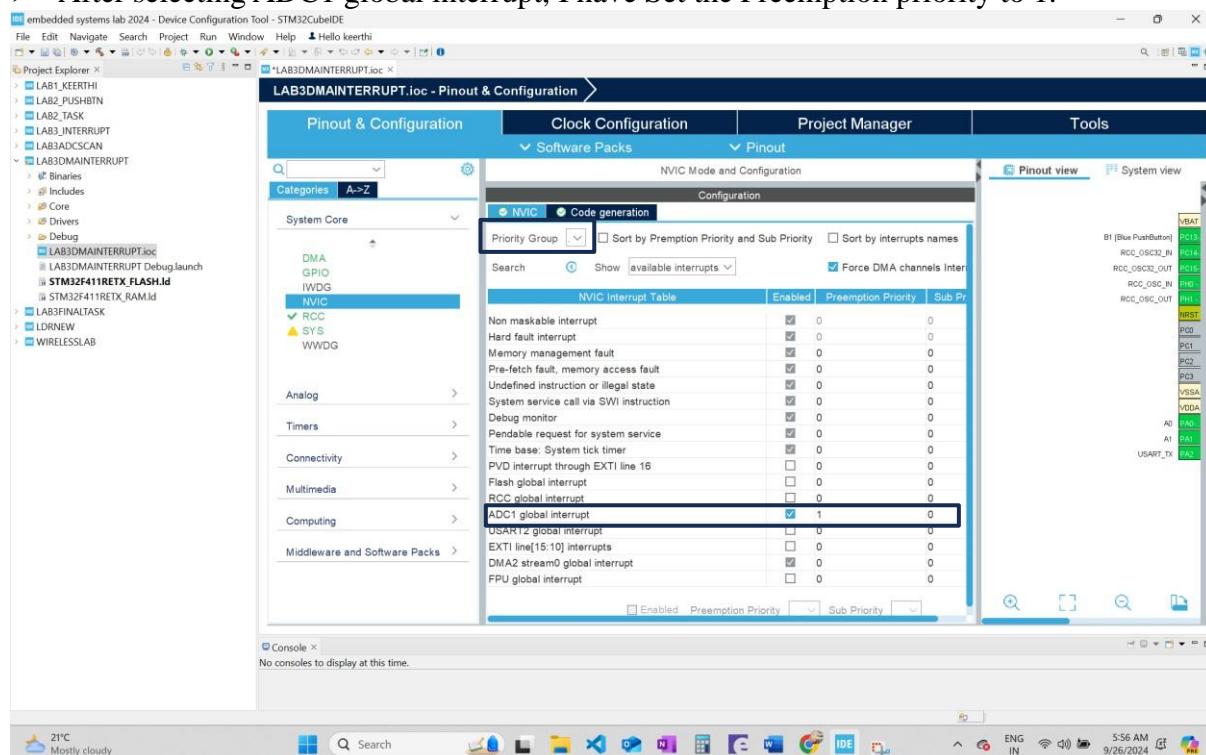
- In system core under Analog in ADC1 , I have **Enabled** the **scan conversion mode** and increased the **Number of conversions** to **3** and I have made channel 0, channel 1 and channel 4 to **84 Cycles**.



- In System Core, Under DMA I have added the ADC1 and made the mode to be Circular.



- In System Core, Under NVIC Set the **priority group** as **4-bit**.
  - After selecting ADC1 global interrupt, I have Set the Preemption priority to 1.



- After saving the file, I have added the code in the main.c.

```

17 /* USER CODE END Header */
18 /* Includes ---*/
19 #include "main.h"
20
21 /* Private includes ---*/
22 #include <stdio.h>
23 #include <string.h>
24
25 /* USER CODE BEGIN Includes */
26
27 /* Private typedef ---*/
28
29 /* USER CODE BEGIN PTD */
30
31 /* USER CODE END PTD */
32
33 /* Private define ---*/
34
35 /* USER CODE BEGIN PD */
36
37
38 /* Private macro ---*/
39
40 /* USER CODE BEGIN PM */
41
42
43 /* Private variables ---*/
44 ADC_HandleTypeDef hadc1;
45 DMA_HandleTypeDef hdma_adcl;
46
47 UART_HandleTypeDef huart2;
48
49 /* USER CODE BEGIN PV */
50 int ConvCplt_0;
51 volatile uint16_t adcDMA [3];
52 char txStr[100];
53 /* USER CODE END PV */
54
55 /* Private function prototypes ---*/
56 void SystemClock_Config(void);
57 static void MX_GPIO_Init(void);
58 static void MX_DMA_Init(void);
59 static void MX_USART2_UART_Init(void);
60 static void MX_ADC1_Init(void);
61 /* USER CODE BEGIN PFP */
62
63 /* USER CODE END PFP */
64
65 /* Private user code ---*/
66 /* USER CODE BEGIN 0 */
67

```

```

71 /**
72 * @brief The application entry point.
73 */
74 int main(void)
75 {
76
77     /* USER CODE BEGIN 1 */
78
79     /* USER CODE END 1 */
80
81     /* MCU Configuration-- */
82
83     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
84     HAL_Init();
85
86     /* USER CODE BEGIN Init */
87
88     /* USER CODE END Init */
89
90     /* Configure the system clock */
91     SystemClock_Config();
92
93     /* USER CODE BEGIN SysInit */
94
95     /* USER CODE END SysInit */
96
97     /* Initialize all configured peripherals */
98     MX_GPIO_Init();
99     MX_DMA_Init();
100    MX_USART2_UART_Init();
101    MX_ADC1_Init();
102
103    /* USER CODE BEGIN 2 */
104
105
106    /* Infinite loop */
107
108    /* USER CODE BEGIN WHILE */
109    HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3);
110    while (1)
111    {
112        /* USER CODE END WHILE */
113
114        /* USER CODE BEGIN 3 */
115
116    }
117
118 /**
119 * @brief System Clock Configuration
120 */
121

```

embedded systems lab 2024 - LAB3DMAINTERRUPT/Core/Src/main.c - STM32CubeIDE

```

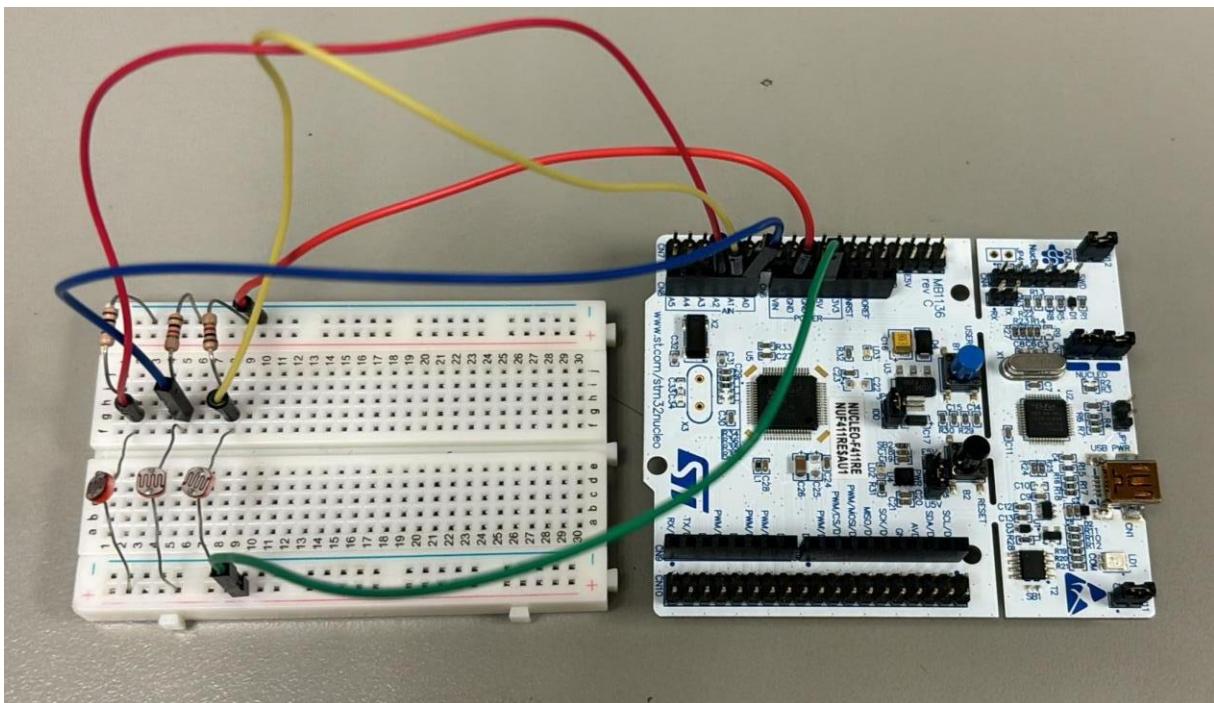
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
Project Explorer X
LAB1_KEERTHI
LAB2_PUSHBTN
LAB2_TASK
LAB3_INTERRUPT
LAB3ADCSCALAR
LAB3DMAINTERRUPT
Binaries
Includes
Core
Drivers
Debug
LAB3DMAINTERRUPT.ioc
STM32F411RETX_FLASH.JLINK
STM32F411RETX_RAM.JLINK
LAB3FINALTASK
LDNEW
WIRELESSLAB
LAB3DMAINTERRUPT.Debug.JLINK

308 /*Configure GPIO pin : LD2_Pin */
309 GPIO_InitStruct.Pin = LD2_Pin;
310 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
311 GPIO_InitStruct.Pull = GPIO_NOPULL;
312 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
313 HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
314
315 /* USER CODE BEGIN MX_GPIO_Init_2 */
316 /* USER CODE END MX_GPIO_Init_2 */
317
318 }
319
320 /* USER CODE BEGIN 4 */
321 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
322 {
323 ConvCplt = 1;
324 HAL_UART_Transmit(&huart2, (uint8_t *) "Interrupt Fired: DMA Buffer filled.\r\n",
325 "DMA interrupt Fired: DMA Buffer filled.\r\n", HAL_MAX_DELAY);
326 sprintf(txStr, "CH1 = %u\r\nCH2 = %u\r\nCH3 = %u\r\n", adcdMA[0], adcdMA[1],
327 adcdMA[2]);
328 HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);
329 HAL_ADC_Start_DMA(shadcl, (uint32_t*)adcdMA, 3);
330 // Restart the ADC DMA to keep reading values
331 }
332 /* USER CODE END 4 */
333
334 /**
335 * @brief This function is executed in case of error occurrence.
336 * @retval None
337 */
338 void Error_Handler(void)
339 {
340 /* USER CODE BEGIN Error_Handler_Debug */
341 /* User can add his own implementation to report the HAL error return state */
342 __disable_irq();
343 while (1)
344 {
345 }
346 /* USER CODE END Error_Handler_Debug */
347 }
348
349 #ifdef USE_FULL_ASSERT
350 /**
351 * @brief Reports the name of the source file and the source line number
352 * where the assert_param error has occurred.
353 * @param file: pointer to the source file name
354 * @param line: assert_param error line source number
355 * @retval None
356 */
357 void assert_failed(uint8_t *file, uint32_t line)
358 {

```

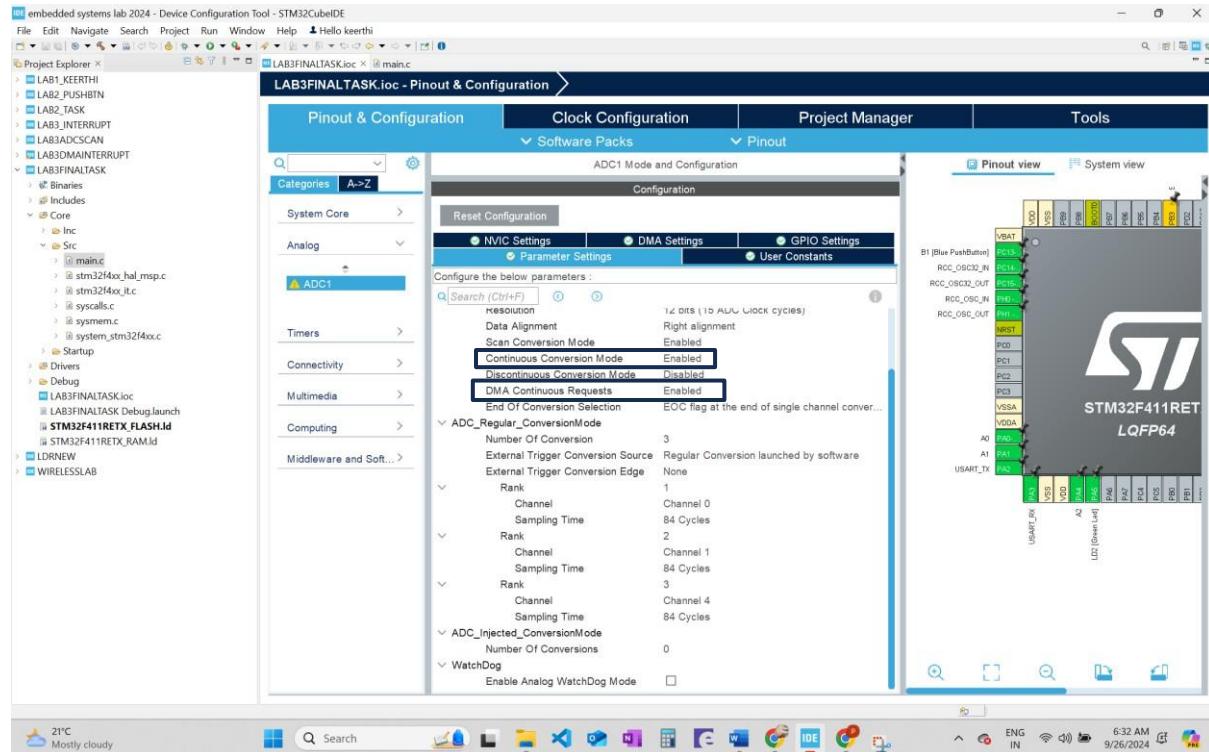
## Experimental Setup:

- This experiment is conducted using the same setup as the ADC Scan Mode Configuration experiment.



## Part 4. ADC Continuous Mode with DMA (& Interrupt)

- This procedure follows exactly the same steps as the ADC Scan Mode with DMA (& Interrupt) software configuration.
- Additionally in this experiment the **Continuous Conversion Mode** and **DMA Continuous Mode is Enabled**.



- After saving the file, I have added the code in the main.c.

```

17 */
18 /* USER CODE END Header */
19 /* Includes -- */
20 #include "main.h"
21
22 /* Private includes -- */
23 #include <stdio.h>
24 #include <string.h>
25
26 /* -- */
27 /* USER CODE BEGIN Includes */
28
29 /* Private typedef -- */
30 /* USER CODE BEGIN PTD */
31
32 /* USER CODE END PTD */
33
34 /* Private define -- */
35 /* USER CODE BEGIN PD */
36
37 /* USER CODE END PD */
38
39 /* Private macro -- */
40 /* USER CODE BEGIN PM */
41
42 /* USER CODE END PM */
43
44 /* Private variables -- */
45 ADC_HandleTypeDef hadc1;
46 DMA_HandleTypeDef hdma_adc1;
47
48 UART_HandleTypeDef huart2;
49
50 /* USER CODE BEGIN PV */
51 int ConvCplt = 0;
52 volatile uint32_t adciMA [3];
53 char txStr[100];
54 /* USER CODE END PV */
55
56 /* Private function prototypes -- */
57 void SystemClock_Config(void);
58 static void MX_GPIO_Init(void);
59 static void MX_DMA_Init(void);
60 static void MX_USART2_UART_Init(void);
61 static void MX_ADC1_Init(void);
62 /* USER CODE BEGIN FPF */
63
64 /* USER CODE END FPF */
65
66 /* Private user code -- */
67

```

The screenshot shows the IAR Embedded Workbench IDE interface with the following details:

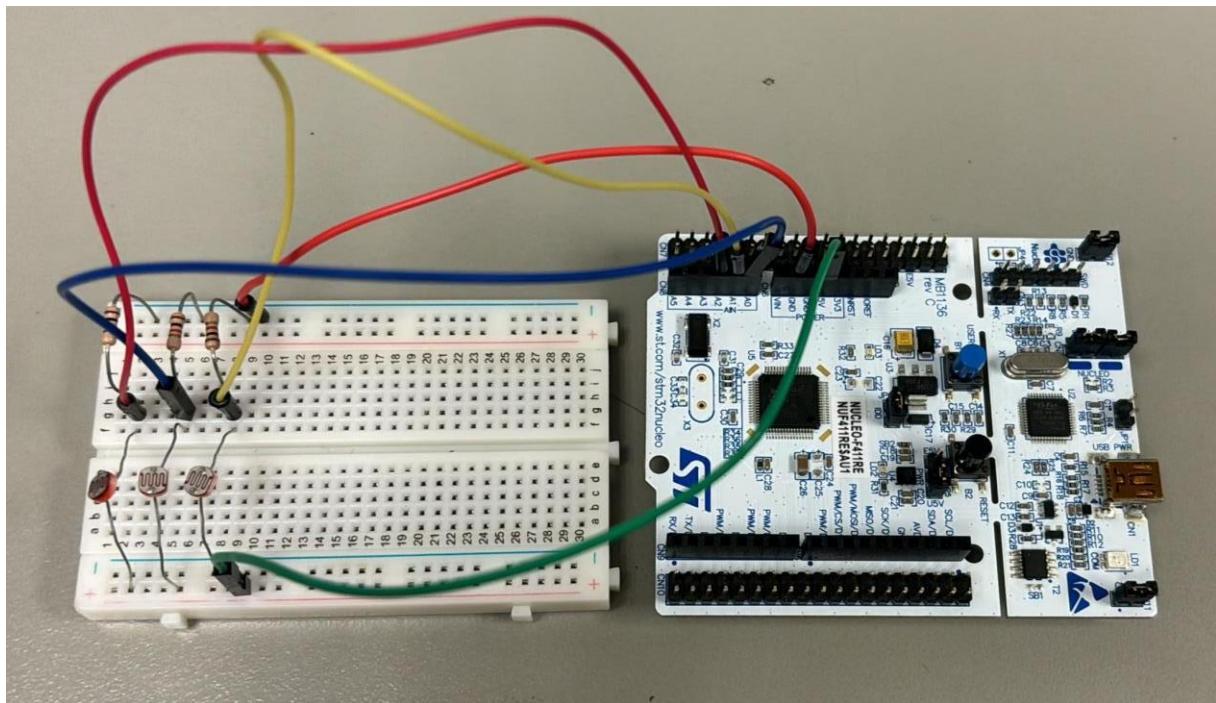
- Project Explorer:** Shows the project structure with files like main.c, HAL1\_KEERTHI.c, HAL2\_PUSHBTN.c, HAL2\_TASK.c, HAL3\_INTERRUPT.c, HAL3ADCSCAN.c, HAL3DMAINTERRUPT.c, and LAB3FINALTASK.c.
- Code Editor:** Displays the main.c file content. The code initializes peripherals, performs an infinite loop with DMA transmission, and configures the RCC oscillator.

```
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi

Project Explorer X LAB3FINALTASK.c @ main.c X
LAB1_KEERTHI
LAB2_PUSHBTN
HAL2_TASK
HAL3_INTERRUPT
HAL3ADCSCAN
HAL3DMAINTERRUPT
HAL3FINALTASK
main.c
HAL2_USART2_UART_Init();
HAL_ADC_Start_DMA(hadc1, (uint16_t *) adcDMA, 3);
while(1)
{
    HAL_Delay(500);
    HAL_UART_Transmit(&huart2, (uint8_t *) "Interrupt Fired: DMA Buffer filled.\r\n", strlen("Interrupt Fired: DMA Buffer filled.\r\n"), HAL_MAX_DELAY);
    adcDMA[1], adcDMA[2];
    HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr),
    HAL_MAX_DELAY);
}
/* USER CODE BEGIN 3 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLN = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
}
```

## **Experimental Setup:**

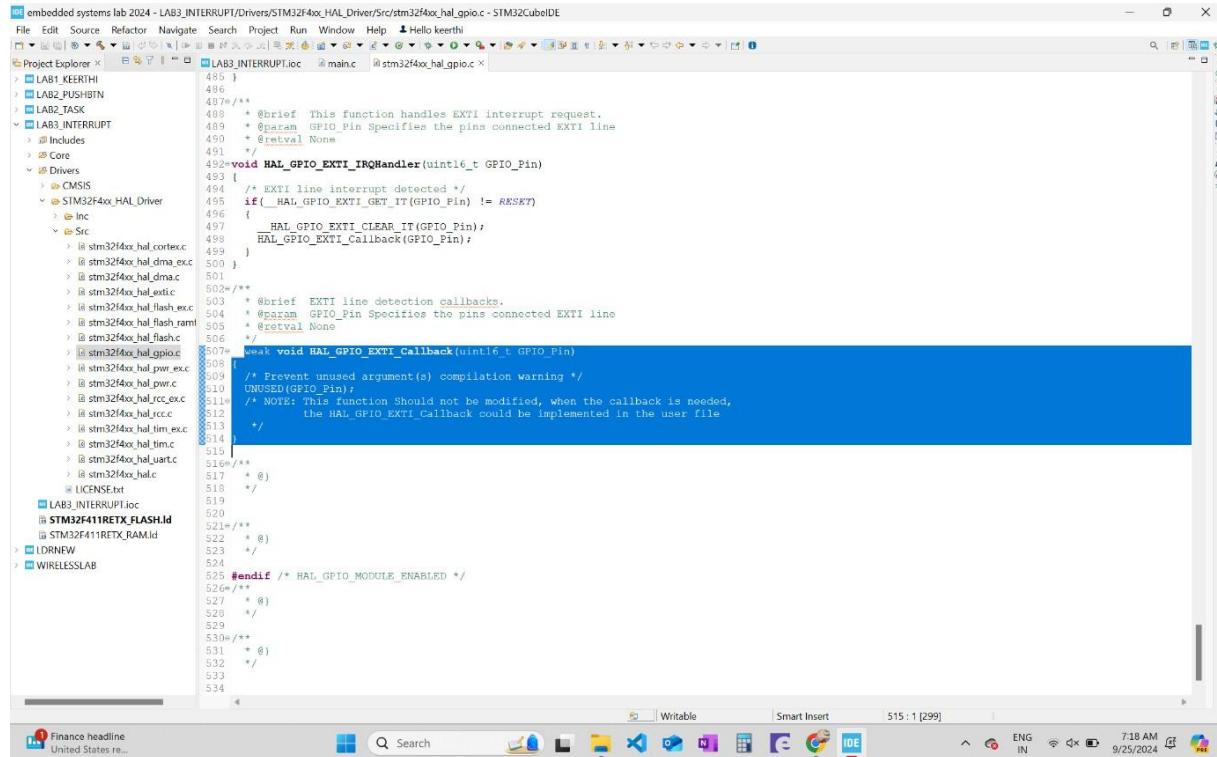
- This experiment is conducted using the same setup as the ADC Scan Mode Configuration experiment.



## Code Review:

### Part 1. GPIO External Interrupt (EXTI) Configuration

- This code executes to flip the LED (LD2) when the button (pwBtn) is pressed, enabling a prompt response to user input without requiring the main program loop to continuously monitor the button state. As a result, the system is more effective and event-responsive.



```

embedded systems lab 2024 - LAB3_INTERRUPT/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help ▲ Hello keerthi
Project Explorer X
LAB1_KERTHI
LAB2_PUSHBTN
LAB2_TASK
LAB3_INTERRUPT
  Includes
  Core
  Drivers
    CMSIS
    STM32F4xx_HAL_Driver
      Inc
      Src
        stm32f4xx_hal_cortex.c
        stm32f4xx_hal_dma_ex.c
        stm32f4xx_hal_dma.c
        stm32f4xx_hal_exti.c
        stm32f4xx_hal_flash_ex.c
        stm32f4xx_hal_flash_raml.c
        stm32f4xx_hal_flashc.c
        stm32f4xx_hal_gpio.c
        stm32f4xx_hal_pwr_ex.c
        stm32f4xx_hal_rcc_ex.c
        stm32f4xx_hal_rcc_ex.c
        stm32f4xx_hal_rcc.c
        stm32f4xx_hal_tim_ex.c
        stm32f4xx_hal_tmc.c
        stm32f4xx_hal_uart.c
        stm32f4xx_hal.c
LICENSE.txt
LAB3_INTERRUPT.ioc
STM32F411RETX_FLASH.ld
STM32F411RETX.RAM.ID
LDRNEW
WIRELESSLAB

HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function should not be modified, when the callback is needed,
   * the HAL_GPIO_EXTI_Callback could be implemented in the user file
  */
}

/* USER CODE BEGIN MX_GPIO_Init_1 */
MX_GPIO_Init(void)
{
  /* GPIO Ports Clock Enable */
  HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_RCC_GPIOC_CLK_ENABLE();
  HAL_RCC_SPIOB_CLK_ENABLE();

  /* Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : pwBtn_Pin */
  GPIO_InitStruct.Pin = pwBtn_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(pwBtn_GPIO_Port, &GPIO_InitStruct);

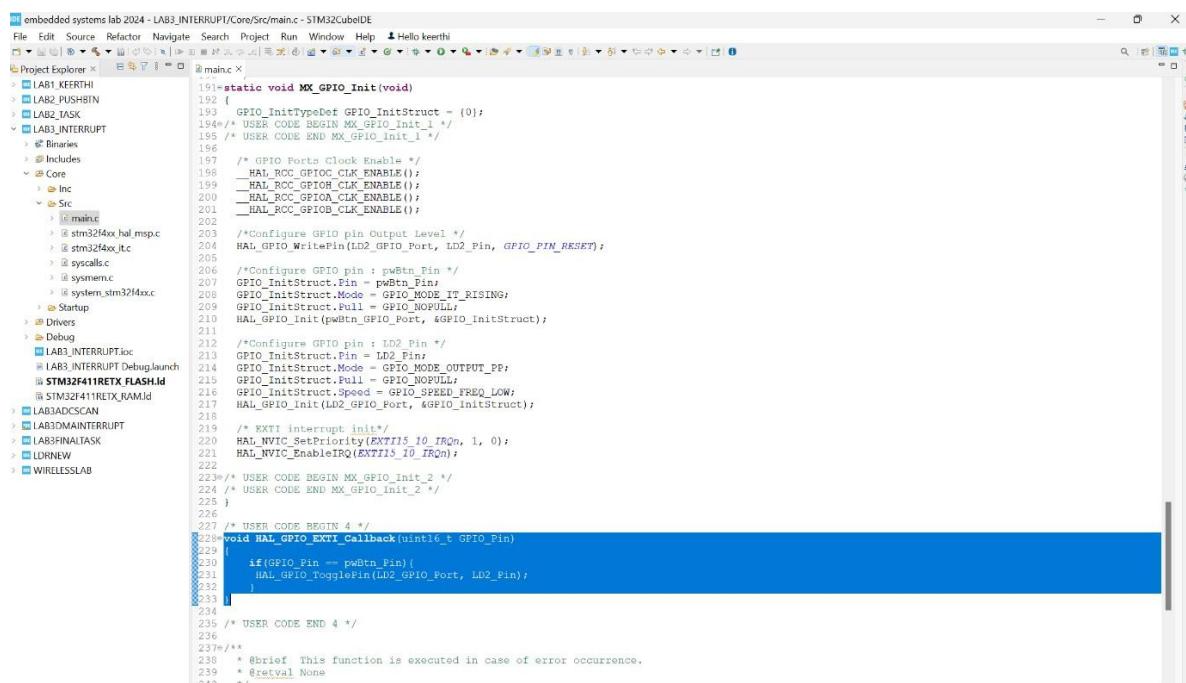
  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

  /* USER CODE BEGIN MX_GPIO_Init_2 */
  /* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == pwBtn_Pin)
  {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
}
/* USER CODE END 4 */

/* Brief This function is executed in case of error occurrence.
 */

```



```

embedded systems lab 2024 - LAB3_INTERRUPT/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help ▲ Hello keerthi
Project Explorer X
LAB1_KERTHI
LAB2_PUSHBTN
LAB2_TASK
LAB3_INTERRUPT
  Binaries
  Includes
  Core
    Inc
    Src
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      system_stm32f4xx.c
    Startup
  Drivers
  Debug
  LAB3_INTERRUPT.ioc
  LAB3_INTERRUPT_Debug.JLINK
  STM32F411RETX_FLASH.ld
  STM32F411RETX.RAM.ID
  LABADCSCAN
  LABDMAINTERRUPT
  LAB3FINALTASK
  LDRNEW
  WIRELESSLAB

static void MX_GPIO_Init(void)
{
  /* GPIO Ports Clock Enable */
  HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_RCC_GPIOC_CLK_ENABLE();
  HAL_RCC_SPIOB_CLK_ENABLE();

  /* Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : pwBtn_Pin */
  GPIO_InitStruct.Pin = pwBtn_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(pwBtn_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

  /* USER CODE BEGIN MX_GPIO_Init_2 */
  /* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == pwBtn_Pin)
  {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
}
/* USER CODE END 4 */

/* Brief This function is executed in case of error occurrence.
 */

```

## Code:

```
/* USER CODE BEGIN 4 */  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {  
    if(GPIO_Pin == pwBtn_Pin){  
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);  
    }  
}  
/* USER CODE END 4 */
```

This function is a callback that the STM32 HAL library calls when an external interrupt occurs. The GPIO\_Pin parameter tells you which pin triggered the interrupt.

This conditional statement checks if the GPIO pin that triggered the interrupt is the same as the pwBtn\_Pin. If it matches, it means the push button connected to pwBtn\_Pin has been pressed or released.

If the condition in step 2 is true, this line of code toggles the state of the LED connected to the LD2\_Pin. This means that if the LED was on, it will be turned off, and vice versa.

## Part 2. ADC Scan Mode Configuration

- After generating the code we made some changes in the main.c

The screenshot shows the STM32CubeIDE interface. The left pane displays the main.c file with several code snippets highlighted in blue, indicating they have been modified. The right pane shows the 'Outline' view, which lists various header files and functions defined in the project. The status bar at the bottom right shows the date and time as 27-09-2024 02:15 PM.

```
1 // Main program header
2 /*
3 * @brief      : main.c
4 * @brief      : Main program body
5 */
6
7 * @attention
8 *
9 * Copyright (c) 2024 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 */
17
18 /* USER CODE END Header */
19 /* Includes -----*/
20 #include "main.h"
21
22 /* Private includes -----*/
23 /* USER CODE BEGIN Includes */
24 #include <stdio.h>
25 #include <string.h>
26 /* USER CODE END Includes */
27
28 /* Private typedef -----*/
29 /* USER CODE BEGIN PTD */
30
31 /* USER CODE END PTD */
32
```

Outline View:

- main.h
- stdio.h
- string.h
- hadc1 : ADC\_HandleTypeDefDef
- hdma1 : DMA\_HandleTypeDefDef
- huart2 : UART\_HandleTypeDefDef
- SystemClock\_Config(void)
- MX\_GPIO\_Init(void)
- MX\_DMA\_Init(void)
- MX\_USART2\_UART\_Init(void)
- MX\_ADC1\_Init(void)
- main(void)
- SystemClock\_Config(void)
- MX\_ADC1\_Init(void)
- MX\_USART2\_UART\_Init(void)
- MX\_DMA\_Init(void)
- MX\_GPIO\_Init(void)
- Error\_Handler(void)
- assert\_failed(uint8\_t\*, uint32\_t):void

## Code

```
/* USER CODE BEGIN Includes */
```

```
#include <stdio.h>
```

```
#include <string.h>
```

}

These headers provide functions for standard input/output operations and string manipulation.

```
/* USER CODE END Includes */
```

The screenshot shows the STM32CubeIDE interface. The main window displays the `main.c` file with the following code snippet:

```

74
75 /* USER CODE BEGIN 1 */
76
77 /* USER CODE END 1 */
78
79 /* MCU Configuration-- */
80
81 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
82 HAL_Init();
83
84 /* USER CODE BEGIN Init */
85
86 /* USER CODE END Init */
87
88 /* Configure the system clock */
89 SystemClock_Config();
90
91 /* USER CODE BEGIN SysInit */
92
93 /* USER CODE END SysInit */
94
95 /* Initialize all configured peripherals */
96 MX_GPIO_Init();
97 MX_DMA_Init();
98 MX_USART2_UART_Init();
99 MX_ADC1_Init();
100 /* USER CODE BEGIN 2 */
101 volatile uint16_t adcDMA [3];
102 char txStr[100];
103 /* USER CODE END 2 */

```

The code defines global variables `adcDMA` (an array of 3 `uint16_t`) and `txStr` (a character array of size 100). The `txStr` variable is highlighted with a blue selection bar.

The right side of the interface shows the `Outline` and `Build Targets` panes. The `Outline` pane lists various source files and their symbols. The `Build Targets` pane shows the build configuration.

At the bottom, the taskbar shows the system tray with weather information (21°C Rain), search, and other application icons. The date and time are also displayed.

- `adcDMA[3]`: An array of 3 `uint16_t` elements to store the ADC conversion results.
- `txStr[100]`: A character array to store the formatted string for UART transmission.

The screenshot shows the STM32CubeIDE interface with the main.c file open. The code initializes peripherals (MX\_GPIO\_Init(), MX\_DMA\_Init()), configures an ADC (HAL\_ADC\_Start\_DMA), formats a string with three ADC values (sprintf(txStr, "CH1 = %lu\r\nCH2 = %lu\r\nCH3 = %lu\r\n\r\n", adcDMA[0], adcDMA[1], adcDMA[2])), and transmits it over UART (HAL\_UART\_Transmit). A callout box highlights the DMA start function.

```

91  /* USER CODE BEGIN SysInit */
92
93  /* USER CODE END SysInit */
94
95  /* Initialize all configured peripherals */
96  MX_GPIO_Init();
97  MX_DMA_Init();
98  MX_USART2_UART_Init();
99  MX_ADC1_Init();
100 /* USER CODE BEGIN 2 */
101 volatile uint16_t adcDMA [3];
102 char txStr[100];
103 /* USER CODE END 2 */
104
105 /* Infinite loop */
106 /* USER CODE BEGIN WHILE */
107 while (1)
108 {
109     /* USER CODE END WHILE */
110
111     /* USER CODE BEGIN 3 */
112     HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3);
113     sprintf(txStr, "CH1 = %lu\r\nCH2 = %lu\r\nCH3 = %lu\r\n\r\n", adcDMA[0], adcDMA[1], adcDMA[2]);
114     HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);
115     HAL_Delay(50);
116
117     /* USER CODE END 3 */
118 }
119
120 /**
121 * @brief System Clock Configuration
122 */

```

## Code explanation

```

/* USER CODE BEGIN 2 */

volatile uint16_t adcDMA [3];

char txStr[100];

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3); → This function starts the ADC to read three channels and store the results in the adc DMA array using DMA (Direct Memory Access).

    sprintf(txStr, "CH1 = %lu\r\nCH2 = %lu\r\nCH3 = %lu\r\n\r\n", adcDMA[0], adcDMA[1], adcDMA[2]); → Formats the ADC conversion results into a string for transmission.

    HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);

    HAL_Delay(50);
}

/* USER CODE END 3 */

```

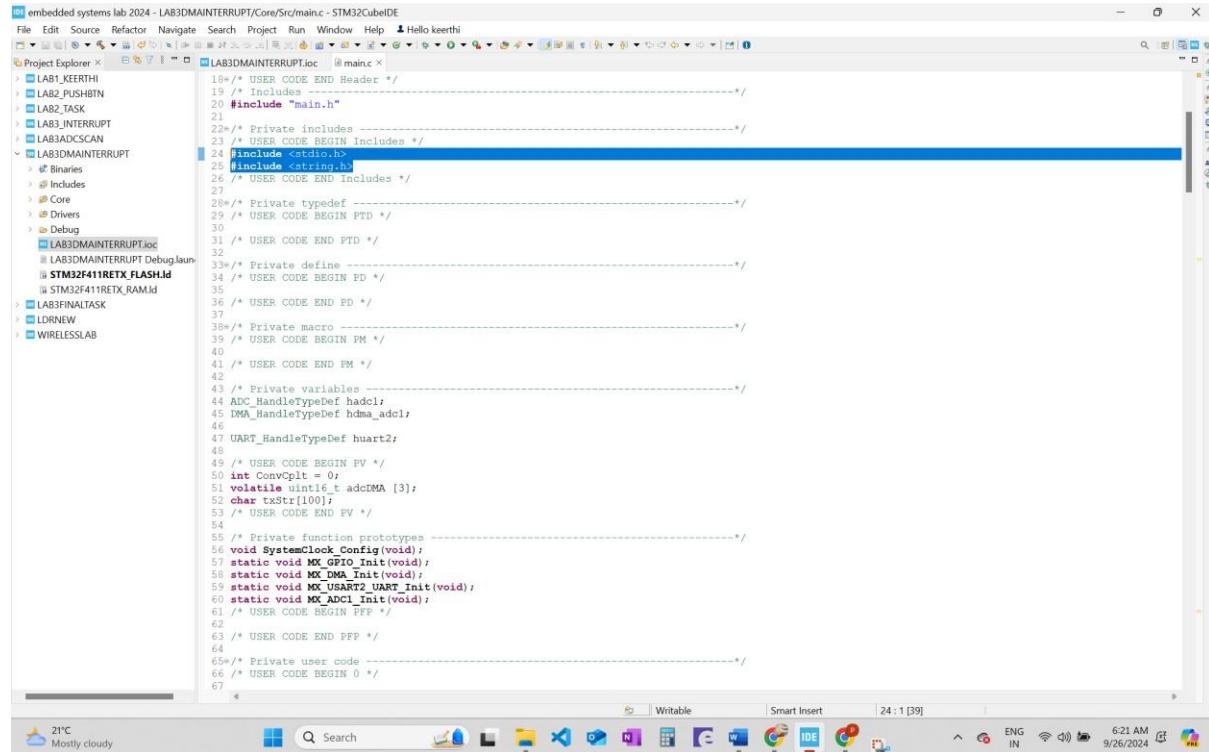
This function starts the ADC to read three channels and store the results in the adc DMA array using DMA (Direct Memory Access).

Sends the formatted string containing ADC values over UART.

Formats the ADC conversion results into a string for transmission.

### Part 3. ADC Scan Mode with DMA (& Interrupt)

- The code initializes the ADC in DMA mode, which allows the ADC to continuously convert analog inputs and store the results in memory without requiring constant CPU intervention.
- The callback function is triggered whenever an ADC conversion is complete. It processes the conversion results, sends them to the UART for display, and restarts the ADC DMA to continue the conversion process.
- This approach is efficient for applications that require continuous ADC measurements without significant CPU overhead.



The screenshot shows the STM32CubeIDE interface with the project 'LAB3DMAINTERRUPT' open. The main.c file is displayed in the center, showing the code for ADC scan mode with DMA. The code includes headers for stdio.h and string.h, and defines for GPIO, DMA, USART, and ADC. A large blue bracket highlights the section of code from line 24 to line 65, which includes the inclusion of stdio.h and string.h, and the definition of various handles and function prototypes.

```
/* USER CODE BEGIN Header */
#include "main.h"
#include <stdio.h>
#include <string.h>
/* USER CODE BEGIN Includes */

/* Private includes */
/* USER CODE BEGIN Includes */

/* Private typedef */
/* USER CODE BEGIN PTD */

/* Private define */
/* USER CODE BEGIN PD */

/* Private macro */
/* USER CODE BEGIN PM */

/* Private variables */
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adcl;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
int ConvCplt = 0;
volatile uint16_t adcDMA [3];
char txstr[100];
/* USER CODE END PV */

/* Private function prototypes */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code */
/* USER CODE BEGIN 0 */

```

#### Code:

```
/* USER CODE BEGIN Includes */
```

```
#include <stdio.h>
```

```
#include <string.h>
```



These headers provide functions for standard input/output operations and string manipulation.

```
/* USER CODE END Includes */
```

```

42     /* Private variables */
43     ADC_HandleTypeDef hadc1;
44     DMA_HandleTypeDef hdma_adcl;
45
46
47     UART_HandleTypeDef huart2;
48
49     /* USER CODE BEGIN PV */
50     int ConvCplt = 0;
51     volatile uint16_t adcDMA[3];
52     char txStr[100];
53     /* USER CODE END PV */
54
55     /* Private function prototypes */
56     void SystemClock_Config(void);
57     void MX_GPIO_Init(void);
58     static void MX_DMA_Init(void);
59     static void MX_USART2_UART_Init(void);
60     static void MX_ADC1_Init(void);
61     /* USER CODE BEGIN FPF */
62
63     /* USER CODE END FPF */
64
65     /* Private user code */
66
67     /* USER CODE END */

```

- ConvCplt: A flag variable used to indicate whether an ADC conversion is complete.
- adcDMA[3]: An array of 3 uint16\_t elements to store the ADC conversion results.
- txStr[100]: A character array to store the formatted string for UART transmission.

```

87     /* USER CODE END Init */
88
89     /* Configure the system clock */
90     SystemClock_Config();
91
92     /* USER CODE BEGIN SysInit */
93
94     /* USER CODE END SysInit */
95
96     /* Initialize all configured peripherals */
97     MX_GPIO_Init();
98     MX_DMA_Init();
99     MX_USART2_UART_Init();
100    MX_ADC1_Init();
101
102    /* USER CODE BEGIN 2 */
103
104    /* USER CODE END 2 */
105
106    /* Infinite loop */
107    /* USER CODE BEGIN WHILE */
108    HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3);
109    while (1)
110    {
111        /* USER CODE END WHILE */
112
113        /* USER CODE BEGIN 3 */
114    }
115    /* USER CODE END 3 */
116 }
117
118 /**
119 * @brief System Clock Configuration
120 * @retval None
121 */
122 void SystemClock_Config(void)

```

```

/* USER CODE BEGIN WHILE */

HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3);

while (1) → indicates an infinite loop where the main
{                                                 program will run continuously.

/* USER CODE END WHILE */

```

This function starts the ADC to read three channels and store the results in the `adcDMA` array using DMA (Direct Memory Access).

```

/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    ConvCplt = 1; // Sets the flag indicating that the ADC conversion has completed.

    HAL_UART_Transmit(&huart2, (uint8_t *) "Interrupt Fired: DMA Buffer filled.\r\n",
                      strlen("Interrupt Fired: DMA Buffer filled.\r\n"), HAL_MAX_DELAY);
    sprintf(txStr, "CH1 = %lu\tCH2 = %lu\tCH3 = %lu\r\n", adcDMA[0], adcDMA[1],
            adcDMA[2]);

    HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adcDMA, 3);
    // Restart the ADC DMA to keep reading values
}

/* USER CODE END 4 */

```

This is the callback function that is automatically called by the HAL when an ADC conversion is complete.

ConvCplt = 1; → Sets the flag indicating that the ADC conversion has completed.

Sends a message to the UART indicating that the ADC conversion is complete and the DMA buffer is filled.

Formats the ADC conversion results into a string for transmission.

Sends the formatted string containing ADC values over UART.

Restarts the ADC DMA to continue reading values.

## Part 4. ADC Continuous Mode with DMA (& Interrupt)

- The code initializes the ADC in DMA mode, which allows the ADC to continuously convert analog inputs and store the results in memory without requiring constant CPU intervention.
  - The main loop periodically (every 500 milliseconds) checks the ADC conversion results and sends them to the UART for display.

## Code:

/\* USER CODE BEGIN Includes \*/

```
#include <stdio.h>
```

```
#include <string.h>
```

These headers provide functions for standard input/output operations and string manipulation.

/\* USER CODE END Includes \*/

- ConvCplt: A flag variable used to indicate whether an ADC conversion is complete.
  - adcDMA[3]: An array of 3 uint16\_t elements to store the ADC conversion results.
  - txStr[100]: A character array to store the formatted string for UART transmission.

```

embeded systems lab 2024 - LAB3FINALTASK/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
Project Explorer X LAB3FINALTASK @ main.c
> LAB1_KEERTHI
> LAB2_PUSHBTN
> LAB2_TASK
> LAB3_INTERRUPT
> LAB3DCCSCAN
> LAB3DMAINTERRUPT
> LAB3FINALTASK
  > Binaries
  > Includes
  > Core
    > Inc
      > main.c
      > stm32f4xx_hal_msp.c
      > stm32f4xx_it.c
      > syscalls.c
      > system.c
      > system_stm32f4xx.c
    > Startup
  > Drivers
  > Debug
  > LAB3FINALTASK Debug launch
  > STM32F411RETX_FLASH.Jd
  > LDNEW
  > WIRELESSLAB

```

```

96  /* USER CODE END SysInit */
97
98  /* Initialize all configured peripherals */
99  MX_GPIO_Init();
100 MX_DMA_Init();
101 MX_USART2_UART_Init();
102 MX_ADC1_Init();
103 /* USER CODE BEGIN 2 */
104
105 /* USER CODE END 2 */
106
107 /* Infinite loop */
108 /* USER CODE BEGIN WHILE */
109 HAL_ADC_Start_DMA(&hadc1, (uint16_t *) adcDMA, 3);
110 while (1)
111 {
112     /* USER CODE END WHILE */
113     HAL_Delay(500);
114     HAL_UART_Transmit(&huart2, (uint8_t *) "Interrupt Fired: DMA Buffer filled.\r\n", strlen("Interrupt Fired: DMA Buffer filled.\r\n"), HAL_MAX_DELAY);
115     sprintf(txStr, "CH1 = %lu\tCH2 = %lu\tCH3 = %lu\r\n", adcDMA[0],
116             adcDMA[1], adcDMA[2]);
117     HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr),
118                     HAL_MAX_DELAY);
119     /* USER CODE BEGIN 3 */
120 }
121 /* USER CODE END 3 */
122
123 /**
124 * @brief System Clock Configuration
125 * @retval None
126 */
127 */
128 void SystemClock_Config(void)
129 {
130     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
131     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
132
133     /* Configure the main internal regulator output voltage */

```

/\* USER CODE BEGIN WHILE \*/

HAL\_ADC\_Start\_DMA(&hadc1, (uint16\_t \*) adcDMA, 3);

while (1) → **An infinite loop that keeps the main program running indefinitely.**

**Starts the ADC in DMA mode, using hadc1 as the ADC handle, adcDMA as the destination buffer for conversion results, and 3 as the number of channels to be converted.**

/\* USER CODE END WHILE \*/

HAL\_Delay(500);

**Introduces a 500 millisecond delay, allowing the system to pause briefly between UART transmissions.**

HAL\_UART\_Transmit(&huart2, (uint8\_t \*) "Interrupt Fired: DMA Buffer filled.\r\n", strlen("Interrupt Fired: DMA Buffer filled.\r\n"),

HAL\_MAX\_DELAY);

sprintf(txStr, "CH1 = %lu\tCH2 = %lu\tCH3 = %lu\r\n", adcDMA[0],  
adcDMA[1], adcDMA[2]);

HAL\_UART\_Transmit(&huart2, (uint8\_t \*) txStr, strlen(txStr),

HAL\_MAX\_DELAY);

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

**Sends a message to the UART indicating that the ADC conversion is complete and the DMA buffer is filled.**

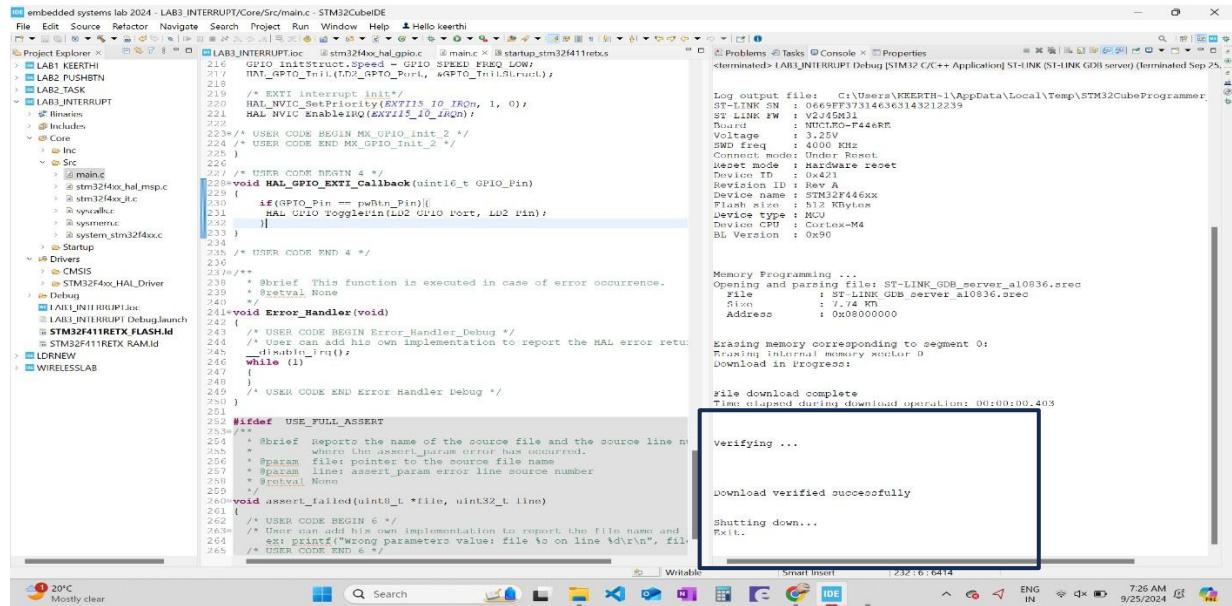
**Formats the ADC conversion results into a string for transmission.**

**Sends the formatted string to the UART.**

## Results:

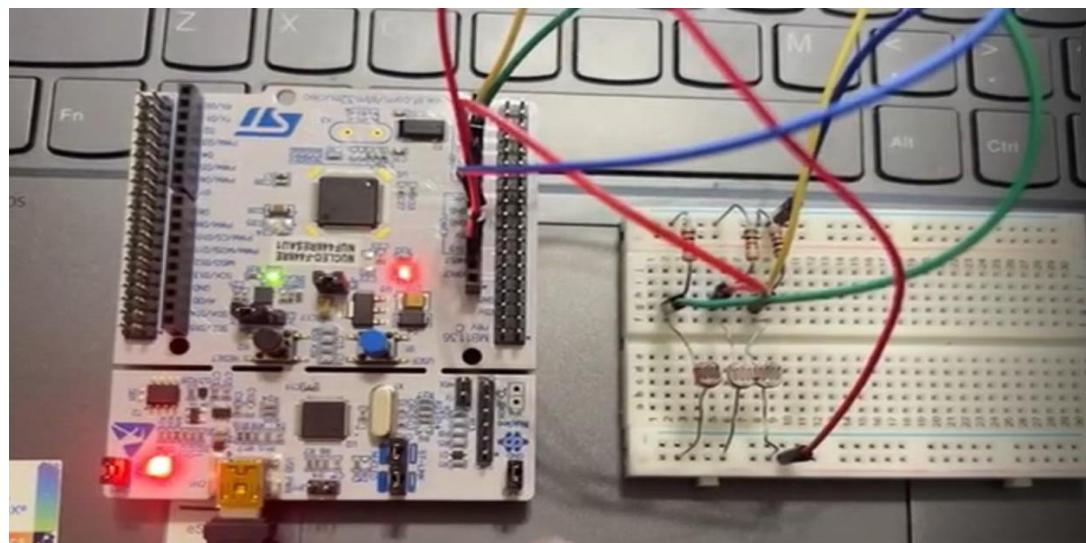
### Part 1. GPIO External Interrupt (EXTI) Configuration

- After saving the file and writing the code in main.c and connecting the hardware setup, I have successfully debugged the file and the output is shown in console window.



The screenshot shows the STM32CubeIDE interface. On the left, the Project Explorer displays the source files for the project, including main.c, startup\_stm32441rtxs.s, and various HAL and CMSIS files. On the right, a terminal window shows the results of a memory dump operation. The log output includes device information (ST-LINK SN: 0664F373146363143212239, Device ID: 0x421, Device name: STM32F46xx, Flash size: 512 Kbytes, Device type: Cortex-M4, BU Version: 0x90), memory programming details (Opening and parsing file: ST-LINK\_GDB\_server\_a10836.srec, File size: 7.14 KB, Address: 0x00000000), and download status (File download complete, Time elapsed during download operation: 00:00:00.403). A large red box highlights the terminal window.

- After running the code, I have observed that when I push the button externally the LED glows and when I again push the button the LED turns off indicating the external interrupt.  
➤ When external interrupt is given through push button LED glows on and off.



- A video demonstrating the configuration of the GPIO External Interrupt (EXTI) is provided below.

[https://drive.google.com/file/d/1FIYNUDNJH3u3hqsnTS2RaTpOfxiPfY98/](https://drive.google.com/file/d/1FIYNUDNJH3u3hqsnTS2RaTpOfxiPfY98/view?usp=drive_link)  
[view?usp=drive\\_link](https://drive.google.com/file/d/1FIYNUDNJH3u3hqsnTS2RaTpOfxiPfY98/view?usp=drive_link)

## Part 2. ADC Scan Mode Configuration

- After successfully compiling, in the console window to display the values on the serial monitor. I have selected the command shell console and then selected the serial port and added new connection name (This connection name is named as **nucleo**).

```

lab_3_2.ioc main.c
86     SystemClock_Config();
87
88 /* USER CODE BEGIN SysInit */
89
90 /* USER CODE END SysInit */
91
92 /* Initialize all configured peripherals */
93 MX_GPIO_Init();
94 MX_USART2_UART_Init();
95 MX_ADC1_Init();
96 /* USER CODE BEGIN 2 */
97 volatile uint16_t adcDMA [3];
98 char txStr[100];
99 /* USER CODE END 2 */
100
101 /* Infinite loop */
102 /* USER CODE BEGIN WHILE */
103 while (1)
104 {
105     /* USER CODE END WHILE */
106     HAL_ADC_Start_DMA(hadc1, (uint16_t *) adcDMA, 3);
107     sprintf(txStr, "CH1 = %lu\CH2 = %lu\CH3 = %lu\r\n", adcDMA[0],
108     HAL_UART_Transmit(&huart2, (uint8_t *) txStr, strlen(txStr), HAL_
109     HAL_Delay(50);
110     /* USER CODE BEGIN 3 */
111 }
112 /* USER CODE END 3 */
113 }
114
115 */
116 /* @brief System Clock Configuration

```

- The values appear on the serial monitor as soon as the remote connection is established successfully.

CH1	CH2	CH3
531	458	553
540	472	566
558	483	562
582	506	602
581	512	606
602	520	614
616	542	636

- The values that are displayed on serial monitor are the LDR values. I have made channel 0, channel 1 and channel 4. These channel values resemble the LDR values. As the LDR is connected to ADC pin on board, ADC converts the analog value to digital values and display those digital values on the serial monitor. The values are constantly shown, and they are adjusted and mirrored on the serial monitor whenever the LDR is covered. Data from the ADC channels is displayed in the serial terminal, showing values in real-time.

## Part 3. ADC Scan Mode with DMA (& Interrupt)

- After successfully compiling, in the console window to display the values on the serial monitor. I have selected the command shell console and then selected the serial port and added new connection name (This connection name is named as NUCLEO41).

The screenshot shows the STM32CubeIDE interface. The Project Explorer on the left lists various source files and projects. The main workspace shows the code for `main.c`. On the right, a terminal window displays the output of a download operation. A callout box highlights the message "Download verified successfully". The status bar at the bottom shows the date and time as 9/26/2024 6:01 AM.

```

embedded systems lab 2024 - LAB3DMAINTERRUPT/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Search Project Run Window Help Hello keerthi

Project Explorer
LAB1_KEERTHI
LAB2_PUSHBTN
LAB2_TASK
LAB3_INTERRUPT
LAB3ADCSCAN
LAB3DMAINTERRUPT
  Binaries
  Includes
  Core
  Drivers
  Debug
  LAB3DMAINTERRUPT.oic
  LAB3DMAINTERRUPT_Debug.lau
  STM32F41RETX_FLASHId
  STM32F41RETX_RAMD
  LAB3FINALTASK
  LDRNEW
  WIRELESSLAB

308 /*Configure GPIO pin : LD2_Pin */
309 GPIO_InitStruct.Pin = LD2_Pin;
310 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
311 GPIO_InitStruct.Pull = GPIO_NOPULL;
312 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
313 HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
314
315 /* USER CODE BEGIN MX_GPIO_Init_2 */
316 /* USER CODE END MX_GPIO_Init_2 */
317 }
318 }
319
320 /* USER CODE BEGIN 4 */
321 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
322 {
323 ConvCplt = 1;
324 HAL_UART_Transmit(&huart2, (uint8_t *)"Interrupt Fired: DMA Buffer filled.\r\n", HAL_MAX_DELAY);
325 strlen("Interrupt Fired: DMA Buffer filled.\r\n");
326 sprintf(txStr, "CH1 = %lu\r\nCH2 = %lu\r\nCH3 = %lu\r\n", adcDMA[0], adcDMA[1]
327 adcDMA[2]);
328 HAL_UART_Transmit(&huart2, (uint8_t *)txStr, strlen(txStr), HAL_MAX_DELAY);
329 HAL_ADC_Start_DMA(hadc1, (uint32_t*)adcDMA, 3);
330 // Restart the ADC DMA to keep reading values
331 }
332 /* USER CODE END 4 */
333
334 /**
335 * @brief This function is executed in case of error occurrence.
336 * @retval None
337 */
338 void Error_Handler(void)
339 {
340 /* USER CODE BEGIN Error_Handler_Debug */
341 /* User can add his own implementation to report the HAL error return state */
342 __disable_irq();
343 while (1)
344 {
345 }
346 /* USER CODE END Error_Handler_Debug */
347 }
348
349 #ifdef USE_FULL_ASSERT
350 /**
351 * @brief Reports the name of the source file and the source line number
352 * where the assert_param error has occurred.
353 * @param file: pointer to the source file name
354 * @param line: assert_param error line source number
355 * @retval None
356 */
357 void assert_failed(uint8_t *file, uint32_t line)

```

This screenshot is nearly identical to the one above, showing the STM32CubeIDE interface with the same code in `main.c`. The terminal window on the right shows the download verification message "Download verified successfully". The status bar at the bottom shows the date and time as 9/26/2024 6:02 AM.

```

embedded systems lab 2024 - LAB3DMAINTERRUPT/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Search Project Run Window Help Hello keerthi

Project Explorer
LAB1_KEERTHI
LAB2_PUSHBTN
LAB2_TASK
LAB3_INTERRUPT
LAB3ADCSCAN
LAB3DMAINTERRUPT
  Binaries
  Includes
  Core
  Drivers
  Debug
  LAB3DMAINTERRUPT.oic
  LAB3DMAINTERRUPT_Debug.lau
  STM32F41RETX_FLASHId
  STM32F41RETX_RAMD
  LAB3FINALTASK
  LDRNEW
  WIRELESSLAB

308 /*Configure GPIO pin : LD2_Pin */
309 GPIO_InitStruct.Pin = LD2_Pin;
310 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
311 GPIO_InitStruct.Pull = GPIO_NOPULL;
312 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
313 HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
314
315 /* USER CODE BEGIN MX_GPIO_Init_2 */
316 /* USER CODE END MX_GPIO_Init_2 */
317 }
318 }
319
320 /* USER CODE BEGIN 4 */
321 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
322 {
323 ConvCplt = 1;
324 HAL_UART_Transmit(&huart2, (uint8_t *)"Interrupt Fired: DMA Buffer filled.\r\n", HAL_MAX_DELAY);
325 strlen("Interrupt Fired: DMA Buffer filled.\r\n");
326 sprintf(txStr, "CH1 = %lu\r\nCH2 = %lu\r\nCH3 = %lu\r\n", adcDMA[0], adcDMA[1]
327 adcDMA[2]);
328 HAL_UART_Transmit(&huart2, (uint8_t *)txStr, strlen(txStr), HAL_MAX_DELAY);
329 HAL_ADC_Start_DMA(hadc1, (uint32_t*)adcDMA, 3);
330 // Restart the ADC DMA to keep reading values
331 }
332 /* USER CODE END 4 */
333
334 /**
335 * @brief This function is executed in case of error occurrence.
336 * @retval None
337 */
338 void Error_Handler(void)
339 {
340 /* USER CODE BEGIN Error_Handler_Debug */
341 /* User can add his own implementation to report the HAL error return state */
342 __disable_irq();
343 while (1)
344 {
345 }
346 /* USER CODE END Error_Handler_Debug */
347 }
348
349 #ifdef USE_FULL_ASSERT
350 /**
351 * @brief Reports the name of the source file and the source line number
352 * where the assert_param error has occurred.
353 * @param file: pointer to the source file name
354 * @param line: assert_param error line source number
355 * @retval None
356 */
357 void assert_failed(uint8_t *file, uint32_t line)

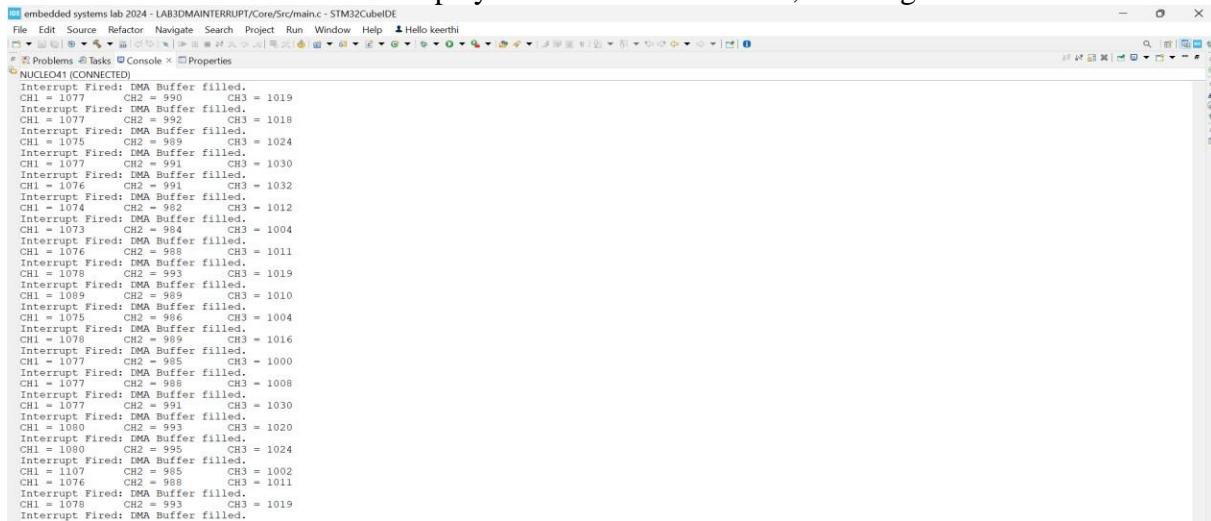
```

The screenshot shows the STM32CubeIDE interface. On the left, the Project Explorer displays various source files and projects. In the center, the code editor shows a C file named main.c with several lines of code related to GPIO and DMA initialization. A modal dialog box is open over the code editor, prompting for a 'Select Remote Connection' from a list of available options. On the right, the terminal window shows the output of a 'HAL\_DMA1\_IRQHandler' function, which includes details about the DMA buffer being filled and interrupt firing. The terminal also shows the progress of a memory download and verification process.

- The values appear on the serial monitor as soon as the remote connection is established successfully.

This screenshot is similar to the previous one but shows the serial monitor window instead of the terminal. The serial monitor displays a continuous stream of data, specifically showing interrupt logs for DMA buffer fills. The data includes interrupt numbers (CH1, CH2, CH3) and their corresponding values (e.g., CH1 = 1016, CH2 = 987, CH3 = 1007). The serial monitor window has a header indicating the baud rate (115200) and other communication parameters.

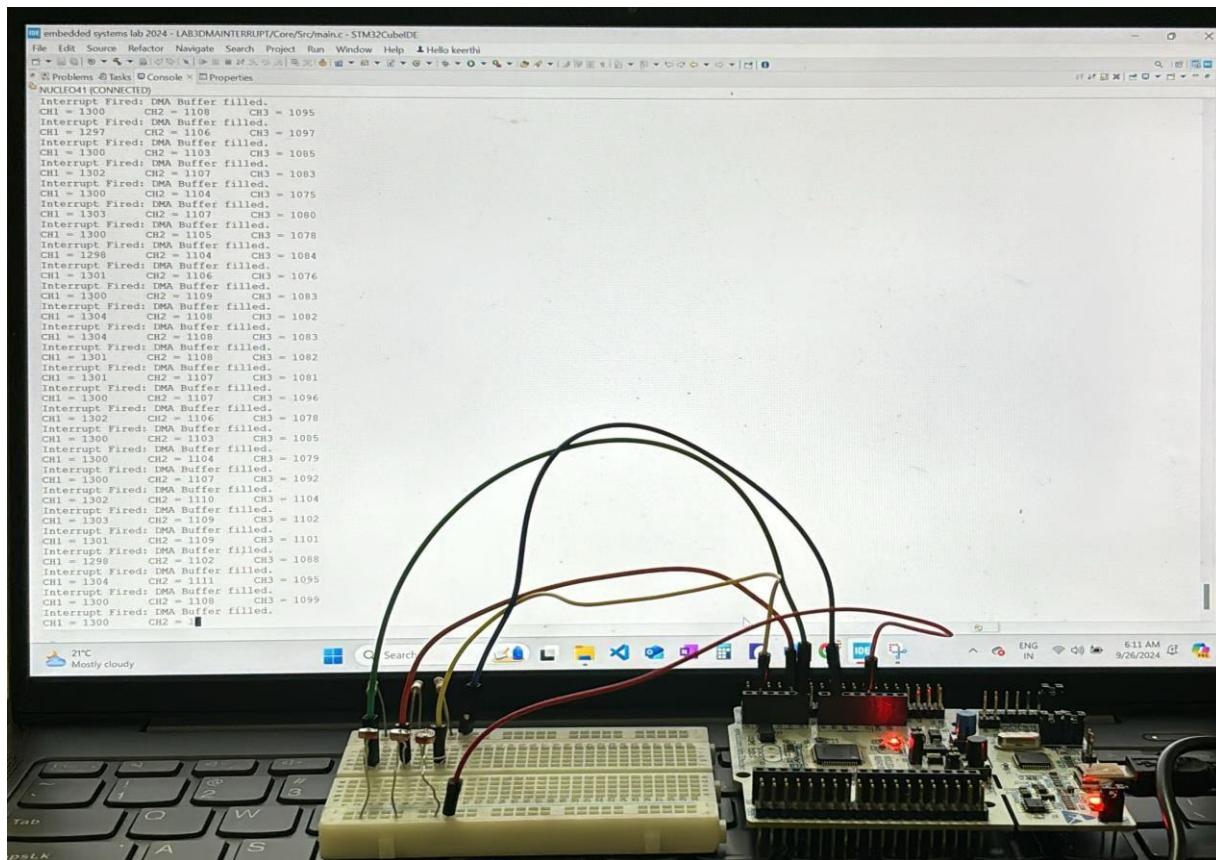
- The values that are displayed on serial monitor are the LDR values. I have **Enabled** the **scan conversion mode** and increased the **Number of conversions** to **3** and I have made channel 0, channel 1 and channel 4. These channel values resemble the LDR values. As the LDR is connected to ADC pin on board, ADC converts the analog value to digital values and display those digital values on the serial monitor. The values are constantly shown, and they are adjusted and mirrored on the serial monitor whenever the LDR is covered. Data from the ADC channels is displayed in the serial terminal, showing values in real-time.



```

embedded systems lab 2024 - LAB3DMAINTERRUPT/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
problems Tasks Console Properties
NUCLEO41 (CONNECTED)
Interrupt Fired: DMA Buffer filled.
CH1 = 1077 CH2 = 990 CH3 = 1019
Interrupt Fired: DMA Buffer filled.
CH1 = 1078 CH2 = 991 CH3 = 1018
Interrupt Fired: DMA Buffer filled.
CH1 = 1075 CH2 = 984 CH3 = 1024
Interrupt Fired: DMA Buffer filled.
CH1 = 1076 CH2 = 985 CH3 = 1030
Interrupt Fired: DMA Buffer filled.
CH1 = 1076 CH2 = 991 CH3 = 1032
Interrupt Fired: DMA Buffer filled.
CH1 = 1074 CH2 = 986 CH3 = 1012
Interrupt Fired: DMA Buffer filled.
CH1 = 1073 CH2 = 984 CH3 = 1004
Interrupt Fired: DMA Buffer filled.
CH1 = 1076 CH2 = 984 CH3 = 1011
Interrupt Fired: DMA Buffer filled.
CH1 = 1076 CH2 = 991 CH3 = 1019
Interrupt Fired: DMA Buffer filled.
CH1 = 1089 CH2 = 989 CH3 = 1010
Interrupt Fired: DMA Buffer filled.
CH1 = 1080 CH2 = 990 CH3 = 1004
Interrupt Fired: DMA Buffer filled.
CH1 = 1078 CH2 = 999 CH3 = 1016
Interrupt Fired: DMA Buffer filled.
CH1 = 1077 CH2 = 988 CH3 = 1000
Interrupt Fired: DMA Buffer filled.
CH1 = 1077 CH2 = 988 CH3 = 1008
Interrupt Fired: DMA Buffer filled.
CH1 = 1077 CH2 = 991 CH3 = 1030
Interrupt Fired: DMA Buffer filled.
CH1 = 1080 CH2 = 993 CH3 = 1020
Interrupt Fired: DMA Buffer filled.
CH1 = 1080 CH2 = 995 CH3 = 1024
Interrupt Fired: DMA Buffer filled.
CH1 = 1109 CH2 = 988 CH3 = 1002
CH1 = 1076 CH2 = 988 CH3 = 1011
Interrupt Fired: DMA Buffer filled.
CH1 = 1078 CH2 = 993 CH3 = 1019
Interrupt Fired: DMA Buffer filled.

```



- A video demonstrating the configuration of the GPIO External Interrupt (EXTI) is provided below.

[https://drive.google.com/file/d/1x3q4iTDF7nkQmfFzUDWT\\_1YAHEf7H9zl/view?usp=drive\\_link](https://drive.google.com/file/d/1x3q4iTDF7nkQmfFzUDWT_1YAHEf7H9zl/view?usp=drive_link)

## Part 4. ADC Continuous Mode with DMA (& Interrupt)

- After successfully compiling, in the console window to display the values on the serial monitor. I have selected the command shell console and then selected the serial port and added new connection name (This connection name is named as NUCLEO41).

The screenshot shows the STM32CubeIDE interface. The left pane displays the Project Explorer with the file main.c selected. The right pane shows the terminal window with the following content:

```

Log output file : C:\Users\KEERTH-1\AppData\Local\Temp\STM32CubeProgrammer\terminated\LAB3FINALTASK Debug [STM32 C++ Application] ST-LINK (ST-LINK GDB server) [terminated]
ST-LINK SN : 066CF505271754867073353
ST-LINK FW : V245M31
Board : NUCLEO-F411RE
Voltage : 3.35V
SWD freq : 4000 KHz
Connect mode : Under Reset
Reset mode : Hardware reset
Device ID : 0x431
Revision ID : Rev A
Device name : STM32F411xC/E
Flash size : 512 KBbytes
Device type : MCU
Device CPU : Cortex-M4
BL Version : 0xD0

Memory Programming ...
Opening and parsing file: ST-LINK_GDB_server_a22040.srec
File : ST-LINK_GDB_server_a22040.srec
Size : 15.98 KB
Address : 0x08000000

Erasing memory corresponding to segment 0:
Erasing internal memory sector 0
Download in Progress:

File download complete
Time elapsed during download operation: 00:00:00.488

Verifying ...
Download verified successfully
Shutting down...
Exit.

```

The status bar at the bottom indicates the date and time as 9/26/2024 6:37 AM.

The screenshot shows the STM32CubeIDE interface. The left pane displays the Project Explorer with the file main.c selected. The right pane shows the terminal window with the following content:

```

Log output file : C:\Users\KEERTH-1\AppData\Local\Temp\STM32CubeProgrammer\terminated\LAB3FINALTASK Debug [STM32 C++ Application] ST-LINK (ST-LINK GDB server) [terminated]
ST-LINK SN : 066CF505271754867073353
ST-LINK FW : V245M31
Board : NUCLEO-F411RE
Voltage : 3.35V
SWD freq : 4000 KHz
Connect mode : Under Reset
Reset mode : Hardware reset
Device ID : 0x431
Revision ID : Rev A
Device name : STM32F411xC/E
Flash size : 512 KBbytes
Device type : MCU
Device CPU : Cortex-M4
BL Version : 0xD0

Memory Programming ...
Opening and parsing file: ST-LINK_GDB_server_a22040.srec
File : ST-LINK_GDB_server_a22040.srec
Size : 15.98 KB
Address : 0x08000000

Erasing memory corresponding to segment 0:
Erasing internal memory sector 0
Download in Progress:

File download complete
Time elapsed during download operation: 00:00:00.488

Verifying ...
Download verified successfully
Shutting down...
Exit.

```

The status bar at the bottom indicates the date and time as 9/26/2024 6:38 AM.

The screenshot shows the STM32CubeIDE interface with the project **LAB3FINALTASK** selected. The **main.c** file is open in the editor. A **Select Remote Connection** dialog box is displayed, listing a connection named **NUCLEO41**. The right panel displays connection information and memory programming progress, indicating a download in progress.

```

21°C Mostly cloudy
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
Project Explorer Problems Tasks Console Properties
LAB3FINALTASK.cproj main.c
100 /* Initialize all configured peripherals */
101 MX_GPIO_Init();
102 MX_DMA_Init();
103 MX_USART2_UART_Init();
104 MX_ADC1_Init();
105 /* USER CODE BEGIN 2 */
106
107 /* Infinite loop */
108 /* USER CODE BEGIN WHILE */
109 HAL_ADC_Start_DMA(shadcl, (uint16_t *) adcDMA, 3);
110 while (1)
111 {
112     /* USER CODE END WHILE */
113     HAL_Delay(500);
114     HAL_UART_Transmit(&uart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);
115     HAL_UART_Transmit(&uart2, (uint8_t *) txStr, strlen(txStr), HAL_MAX_DELAY);
116     /* USER CODE BEGIN 3 */
117 }
118 /* USER CODE END 3 */
119
120 /* USER CODE BEGIN 4 */
121
122 }
123
124 /**
125 * @brief System Clock Configuration
126 * @retval None
127 */
128 void SystemClock_Config(void)
129 {
130     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
131     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
132
133     /** Configure the main internal regulator output voltage
134     */
135     __HAL_RCC_PWR_CLK_ENABLE();
136     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
137
138     /** Initializes the RCC Oscillators according to the specified parameters
139     * in the RCC_OscInitTypeDef structure.
140     */
141     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
142     RCC_OscInitStruct.HSISState = RCC_HSI_ON;
143     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
144     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
145     /* Prevent ODRottage */
146 }

```

- The values appear on the serial monitor as soon as the remote connection is established successfully.

The screenshot shows the STM32CubeIDE interface with the project **LAB3FINALTASK** selected. The **main.c** file is open in the editor. The right panel displays the serial monitor output, which is filled with interrupt messages from the NUCLEO41 board, indicating DMA buffer fills for channels CH1, CH2, and CH3.

```

21°C Mostly cloudy
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
Project Explorer Problems Tasks Console Properties
LAB3FINALTASK.cproj main.c
NUCLEO41 (CONNECTED)
Interrupt Fired: DMA Buffer filled.
CH1 = 1130 CH2 = 1132 CH3 = 1144
Interrupt Fired: DMA Buffer filled.
CH1 = 1130 CH2 = 1130 CH3 = 1152
Interrupt Fired: DMA Buffer filled.
CH1 = 1130 CH2 = 1131 CH3 = 1120
Interrupt Fired: DMA Buffer filled.
CH1 = 1125 CH2 = 1131 CH3 = 1139
Interrupt Fired: DMA Buffer filled.
CH1 = 1123 CH2 = 1133 CH3 = 1153
Interrupt Fired: DMA Buffer filled.
CH1 = 1126 CH2 = 1131 CH3 = 1140
Interrupt Fired: DMA Buffer filled.
CH1 = 1125 CH2 = 1127 CH3 = 1126
Interrupt Fired: DMA Buffer filled.
CH1 = 1131 CH2 = 1138 CH3 = 1137
Interrupt Fired: DMA Buffer filled.
CH1 = 1128 CH2 = 1128 CH3 = 1124
Interrupt Fired: DMA Buffer filled.
CH1 = 1132 CH2 = 1125 CH3 = 1149
Interrupt Fired: DMA Buffer filled.
CH1 = 1140 CH2 = 1140 CH3 = 1139
Interrupt Fired: DMA Buffer filled.
CH1 = 1137 CH2 = 1138 CH3 = 1146
Interrupt Fired: DMA Buffer filled.
CH1 = 1139 CH2 = 1139 CH3 = 1153
Interrupt Fired: DMA Buffer filled.
CH1 = 1138 CH2 = 1135 CH3 = 1163
Interrupt Fired: DMA Buffer filled.
CH1 = 1139 CH2 = 1137 CH3 = 1137
Interrupt Fired: DMA Buffer filled.
CH1 = 1150 CH2 = 1152 CH3 = 1029
Interrupt Fired: DMA Buffer filled.
CH1 = 753 CH2 = 922 CH3 = 943
Interrupt Fired: DMA Buffer filled.
CH1 = 942 CH2 = 1007 CH3 = 1052
Interrupt Fired: DMA Buffer filled.
CH1 = 894 CH2 = 993 CH3 = 1061
Interrupt Fired: DMA Buffer filled.
CH1 = 690 CH2 = 682 CH3 = 808
Interrupt Fired: DMA Buffer filled.
CH1 = 640 CH2 = 572 CH3 = 797
Interrupt Fired: DMA Buffer filled.
CH1 = 645 CH2 = 553 CH3 = 828
Interrupt Fired: DMA Buffer filled.
CH1 = 640 CH2 = 583 CH3 = 791
Interrupt Fired: DMA Buffer filled.
CH1 = 626 CH2 = 572 CH3 = 767
Interrupt Fired: DMA Buffer filled.
CH1 = 616 CH2 = 569 CH3 = 768

```

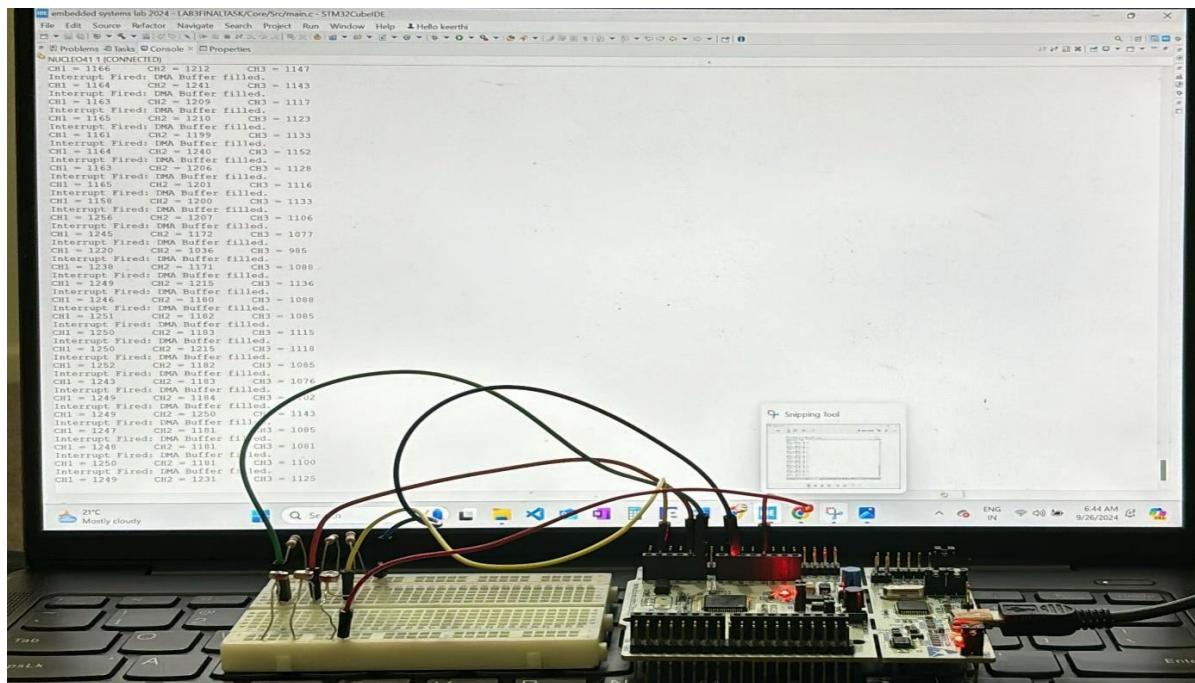
- The values that are displayed on serial monitor are the LDR values. I have **Enabled** the **continuous mode and DMA Continuous request Mode** and increased the **Number of conversions** to 3 and I have made channel 0, channel 1 and channel 4. These channel values resemble the LDR values. As the LDR is connected to ADC pin on board, ADC converts the analog value to digital values and display those digital values on the serial monitor. The values are constantly shown, and they are adjusted and mirrored on the serial monitor whenever the LDR is covered. Data from the ADC channels is displayed in the serial terminal, showing values in real-time.

```

embedded systems lab 2024 - LAB3FINALTASK/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help Hello keerthi
Problems Tasks Console Properties
NUCLEO411 (CONNECTED)
CH1 = 1147 CH2 = 1049 CH3 = 1002
Interrupt Fired: DMA Buffer filled.
CH1 = 1144 CH2 = 1086 CH3 = 1085
Interrupt Fired: DMA Buffer filled.
CH1 = 1142 CH2 = 1084 CH3 = 1051
Interrupt Fired: DMA Buffer filled.
CH1 = 1147 CH2 = 1086 CH3 = 1088
Interrupt Fired: DMA Buffer filled.
CH1 = 1141 CH2 = 1075 CH3 = 1038
Interrupt Fired: DMA Buffer filled.
CH1 = 1140 CH2 = 1074 CH3 = 1042
Interrupt Fired: DMA Buffer filled.
CH1 = 1152 CH2 = 1067 CH3 = 1018
Interrupt Fired: DMA Buffer filled.
CH1 = 1149 CH2 = 1090 CH3 = 1090
Interrupt Fired: DMA Buffer filled.
CH1 = 1144 CH2 = 1074 CH3 = 1040
Interrupt Fired: DMA Buffer filled.
CH1 = 1189 CH2 = 1096 CH3 = 1106
Interrupt Fired: DMA Buffer filled.
CH1 = 1187 CH2 = 1094 CH3 = 1062
Interrupt Fired: DMA Buffer filled.
CH1 = 1188 CH2 = 1096 CH3 = 1077
Interrupt Fired: DMA Buffer filled.
CH1 = 1190 CH2 = 1097 CH3 = 1062
Interrupt Fired: DMA Buffer filled.
CH1 = 1189 CH2 = 1096 CH3 = 1094
Interrupt Fired: DMA Buffer filled.
CH1 = 1188 CH2 = 1098 CH3 = 1073
Interrupt Fired: DMA Buffer filled.
CH1 = 1194 CH2 = 1090 CH3 = 1094
Interrupt Fired: DMA Buffer filled.
CH1 = 1189 CH2 = 1092 CH3 = 1078
Interrupt Fired: DMA Buffer filled.
CH1 = 1194 CH2 = 1099 CH3 = 1082
Interrupt Fired: DMA Buffer filled.
CH1 = 1107 CH2 = 1091 CH3 = 1056
Interrupt Fired: DMA Buffer filled.
CH1 = 1178 CH2 = 1082 CH3 = 1066
Interrupt Fired: DMA Buffer filled.
CH1 = 1180 CH2 = 1084 CH3 = 1043
Interrupt Fired: DMA Buffer filled.
CH1 = 1174 CH2 = 1089 CH3 = 1080
Interrupt Fired: DMA Buffer filled.
CH1 = 1178 CH2 = 1088 CH3 = 1060
Interrupt Fired: DMA Buffer filled.
CH1 = 1180 CH2 = 1091 CH3 = 1093
Interrupt Fired: DMA Buffer filled.
CH1 = 1175 CH2 = 1086 CH3 = 1059
Interrupt Fired: DMA Buffer filled.
CH1 = 1182 CH2 = 1089 CH3 = 1088

```

21°C Mostly cloudy      Search      ENG IN      6:39 AM 9/26/2024



- A video demonstrating ADC Scan Mode with DMA (& Interrupt) can be found below.  
[https://drive.google.com/file/d/1sOf1NgYR3-ucHL-2H4emA4luLDapTFdR/view?usp=drive\\_link](https://drive.google.com/file/d/1sOf1NgYR3-ucHL-2H4emA4luLDapTFdR/view?usp=drive_link)

## Challenges and Solutions:

**Challenge 1:** The error "Failed to execute MI command: target remote localhost:61234" in STM32CubeIDE usually indicates a failure in connecting the debugger/ ST-LINK GDB.

**Resolution 1:** After changing the properties and fixing the STM- LINK GDB connection we got the proper result in the console window.

**Challenge 2:** After building the project 2 we have found that CH3 is always displaying 0.

**Resolution 2:** After doing the proper connections on board with LDR and rebuilding the project the result got displayed.

```

286     huart2.Init.WordLength = UART_WORDLENGTH_8B;
287     huart2.Init.StopBits = UART_STOPBITS_1;
288     huart2.Init.Parity = UART_PARITY_NONE;
289     huart2.Init.Mode = UART_MODE_TX_RX;
290     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
291     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
292     if (HAL_UART_Init(&huart2) != HAL_OK)
293     {
294         Error_Handler();
295     }
296     /* USER CODE BEGIN USART2_Init_2 */
297
298     /* USER CODE END USART2_Init_2 */
299
300 }
```

```

302/* */
303     /* Enable DMA controller clock */
304     /* */
305     static void MX_DMA_Init(void)
306     {
307
308         /* DMA controller clock enable */
309         __HAL_RCC_DMA2_CLK_ENABLE();
310
311         /* DMA2 Stream0 IRQn interrupt configuration */
312         HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
313         HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
314
315     }
316
317
318/* */
319     /* Brief GPIO Initialization Function
320     * @param None
321     * @retval None
322     */
323     static void MX_GPIO_Init(void)
324     {
325         GPIO_InitTypeDef GPIO_InitStruct = {0};
326     /* USER CODE BEGIN MX_GPIO_Init_1 */
327     /* USER CODE END MX_GPIO_Init_1 */
328
329     /* GPIO Ports Clock Enable */
330     __HAL_RCC_GPIOC_CLK_ENABLE();
331     __HAL_RCC_GPIOH_CLK_ENABLE();
332     __HAL_RCC_GPIOA_CLK_ENABLE();
333     __HAL_RCC_GPIOB_CLK_ENABLE();
334
335     /*Configure GPIO pin Output Level */

```

**Challenge 3:** We got this COMB error so many times because of the port connection with the windows.

**Resolution 3:** After changing the port connection settings in device manager of my laptop, the port connection error is solved.

## **Conclusion:**

The experiments successfully illustrated the practical applications of interrupts and ADC configurations in the STM32 microcontroller environment. Working together as a group, we effectively set up an external interrupt on pin PC13, demonstrating how user inputs could initiate actions like toggling an LED without requiring constant monitoring in the main loop. This approach enhanced responsiveness and highlighted the benefits of using interrupt-driven programming in embedded systems.

Moreover, our exploration of ADC functionalities showcased the STM32's capability to handle multiple input channels efficiently. The incorporation of DMA in the later experiments underscored its significance in streamlining data transfer and processing, particularly for real-time applications. Throughout the process, we encountered challenges, such as configuring DMA settings and debugging UART communication, which enriched our hands-on experience. Overall, the insights gained reinforced the vital role of interrupts and DMA in improving system performance and set the stage for future projects that involve more complex sensor integrations and data management tasks.

## **Critical thinking**

### **1. What is the difference between Lab Experiment 2 last week and Lab Experiment 3 (this one), how are they different?**

- In Lab 2, you set up a push button and one LDR (Light Dependent Resistor) as an interrupt to control a light. The interrupt method switched off the light exposing LDR senses an alteration of light intensity. It used interrupt-driven control, where the system responded to light changes sensed by the LDR, immediately. The push button was probably making manual control or reset this lab would be a primer to function bases on how interrupts are integrated in sensors with the STM32 Nucleo board.

The complexity of the setup started with Lab 3, where a parallel circuit was used, and three channels in total using LDRs. The LDRs were likely made for interrupting but here they controlled the blinking of lights, possibly dependent on ambient light. Each was likely a single channel LDR, doing its thing on separate channels — potentially the same way using timers or PWM to blink away. By learning multi-channel control and parallel processing.

### **2. How is the LED being controlled without any input monitoring condition within the main function loop?**

- The LED in the STM32F411RE can be controlled without any input monitoring in the main function loop by utilizing interrupts triggered by external devices like a push button and LDRs (Light Dependent Resistors). When the push button or LDR detects a change, an interrupt service routine (ISR) is triggered, which controls the LED based on the input event, without needing to constantly monitor the inputs in the main loop. Additionally, hardware timers can be used to manage LED behaviour, such as blinking, at regular intervals. This allows the push button and LDRs to function as independent triggers for the LED, bypassing the need for direct input monitoring in the main function loop.

### **3. What is the difference between External Interrupt Mode and External Event Mode?**

- The External Interrupt Mode creates an interrupt request (IRQ) when it detects a specific change (like a rise or fall) in a signal on a chosen pin. On the other hand, the External Event Mode triggers an event without causing an interrupt, which means it can help synchronize tasks or events without stopping the current process to deal with a high-priority event.

### **4. What'd happen if you had chosen Pull-up in the GPIO settings?**

- When the pull-up resistor is enabled, the GPIO pin will stay high (logic level 1) when the button is not pressed. Pressing the button connects the pin to ground (logic level 0). This configuration helps prevent floating pin states and ensures a clear logic level when the button is not active.

## **5. What are the use cases of ADC Scan Mode?**

- ADC Scan Mode is useful for applications that need to monitor several analog signals at the same time. It's great for systems with multiple sensors, like environmental monitoring, where different factors such as temperature, humidity, and light must be tracked together. In industrial control, it allows efficient sampling of various signals like pressure and flow rates, improving process management. It's also important in medical devices that track multiple physiological signals for a complete health overview. Other applications include data acquisition systems that require quick sampling from several inputs and consumer electronics that need smooth interaction with various analog controls. Overall, ADC Scan Mode makes it easier to collect and process data from multiple sources efficiently.

## **6. Unlike previous experiments, why was HAL\_ADC\_Start\_DMA function never called after starting ADC in the final experiment?**

- HAL\_ADC\_Start\_DMA is called just once because the ADC is in continuous mode, which lets it keep reading signals without needing a restart. This first call also sets up the DMA (Direct Memory Access) to automatically transfer data from the ADC to memory. Once set up, the DMA handles the data transfer on its own. This makes the system more efficient, allowing the CPU to work on other tasks or enter low-power mode without needing to keep starting the ADC. The system also uses interrupts to process the data only when it's ready, ensuring everything runs smoothly without extra calls to start the ADC again.