การทดลองที่ 5 การใช้งาน ADC

วัตถุประสงค์

1) เข้าใจการทำงานของ Analog to Digital Converter
2) สามารถเขียนโปรแกรมควบคุมการทำงานของ Analog to Digital Converter

1. Analog to Digital Converter (ADC)

ไมโครคอนโทรลเลอร์ STM32F767 มี ADC จำนวน 3 โมดูล ได้แก่ โมดูล ADC1 ADC2 และ ADC3 เชื่อมต่อกับบัส APB2 แปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลด้วยวิธี successive approximation โดยมีความละเอียดในการแปลงสูงสุด 12 บิต โมดูล ADC มีช่องสัญญาณแบบมัลติเพลกซ์ 19 ช่อง แบ่งเป็นช่องสัญญาณภายนอกจำนวน 16 ช่อง ได้แก่ ช่องสัญญาณ 0 – ช่องสัญญาณ 15, ช่องสัญญาณ 16 ไม่ถูกใช้งาน, ช่องสัญญาณ 17 คือ V_{REFINT} (internal reference voltage), ส่วนช่องสัญญาณ 18 ใช้งานร่วมกันระหว่างเซนเซอร์วัดอุณหภูมิและ V_{BAT} แรงดันสัญญาณอ้างอิงสูงสุดคือ 3.6 V

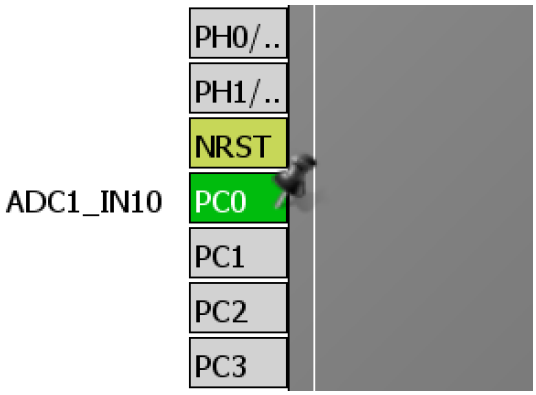
หากต้องการใช้งานโมดูล ADC เพื่อวัดแรงดันไฟฟ้าจากภายนอก สามารถศึกษาจาก Datasheet ได้ว่าช่องสัญญาณภายนอกจำนวน 16 ช่องนั้นเชื่อมต่อกับขาใดบ้างดังตัวอย่างรูปที่ 1.1 ซึ่งหากต้องการใช้ช่องสัญญาณ 10 ของทั้ง ADC1 ก็ให้เปลี่ยนขา PC0 ให้ทำหน้าที่ ADC1_10

Pin name (function after reset	Pin type	I/O structure	Notes	Alternate functions	Additional functions
PH1-OSC_OUT	I/O	FT	(3)	EVENTOUT	OSC_OUT
NRST	I/O	RST	-	-	-
PC0	I/O	FT	-	DFSDM1_CKIN0, DFSDM1_DATIN4, SAI2_FS_B, OTG_HS_ULPI_STP, FMC_SDNWE, LCD_R5, EVENTOUT	ADC1_IN10, ADC2_IN10, ADC3_IN10
PC1	I/O	FT	-	TRACED0, DFSDM1_DATIN0, SPI2_MOSI/I2S2_SD, SAI1_SD_A, DFSDM1_CKIN4, ETH_MDC, MDIOS_MDC, EVENTOUT	ADC1_IN11, ADC2_IN11, ADC3_IN11, RTC_TAMP3/ WKUP3
PC2	I/O	FT	-	DFSDM1_CKIN1, SPI2_MISO, DFSDM1_CKOUT, OTG_HS_ULPI_DIR, ETH_MII_TXD2, FMC_SDNE0, EVENTOUT	ADC1_IN12, ADC2_IN12, ADC3_IN12

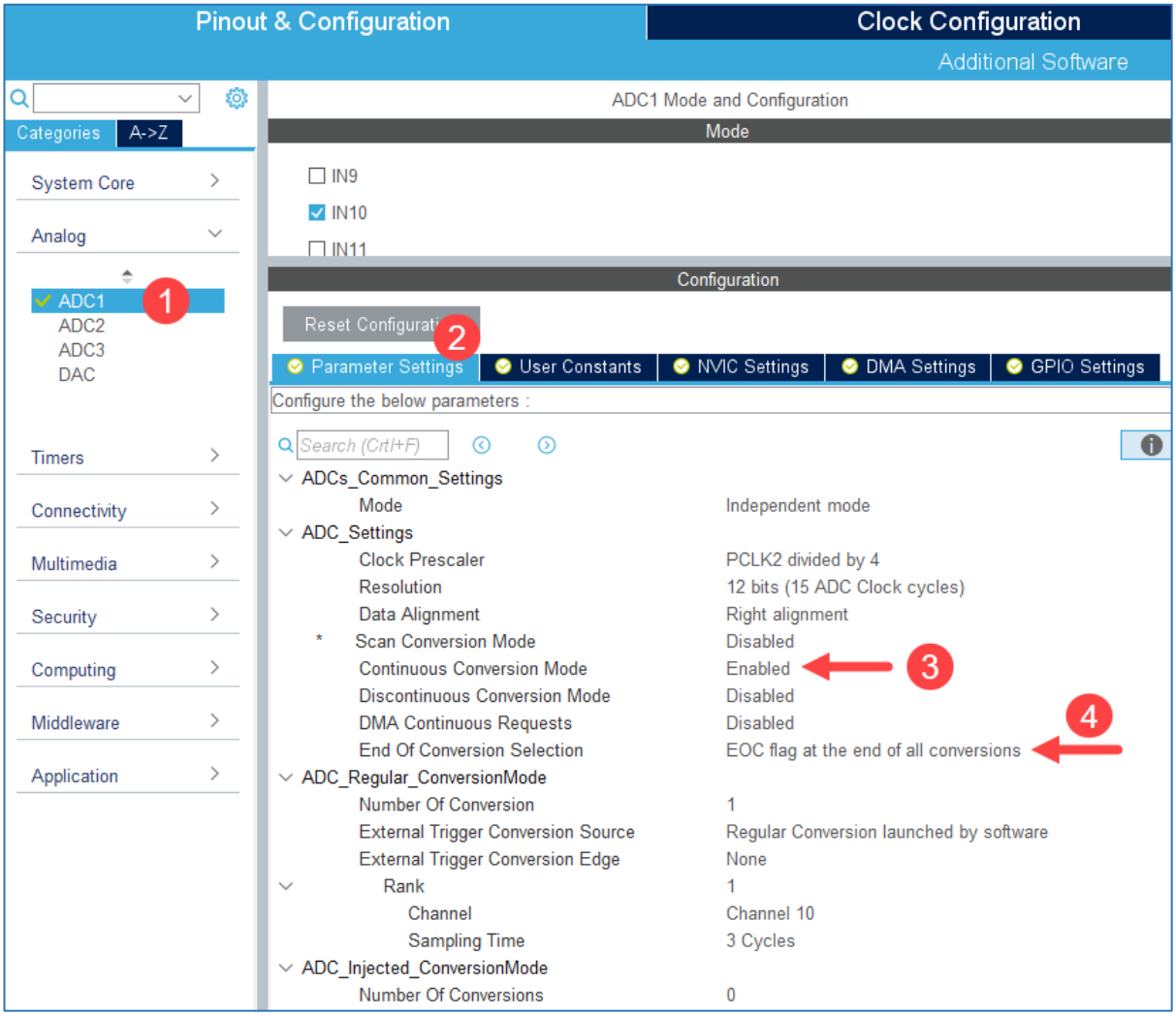
รูปที่ 1.1 แสดงการเชื่อมต่อช่องสัญญาณ ADC กับขาของไอซี

2. การตั้งค่าในโปรแกรม STM32CubeMX

การตั้งค่าสำหรับการทดลองครั้งนี้ต้องกำหนดให้ขา PC0 ทำหน้าที่เป็น Analog Input ดังรูปที่ 2.1 ซึ่งจะเชื่อมต่อวงจรปรับแรงดันไฟฟ้าเข้ามาที่ขานี้ จากนั้นตั้งค่า ADC ตามรูปที่ 2.2



รูปที่ 2.1 แสดงการตั้งค่าให้ PC0 ให้ทำหน้าที่รับสัญญาณแอนะล็อกจากภายนอก



รูปที่ 2.2 แสดงการตั้งค่าโมดูล ADC

3. อธิบายการทำงานของ ADC

โปรแกรม STM32CubeMX ตั้งค่า ADC ด้วยฟังก์ชัน `MX_ADC1_Init` ในไฟล์ `adc.c` ดังรูปที่ 3.1 ซึ่งกำหนดการทำงานของ ADC ให้ทำงานแบบ Regular จัดเรียงข้อมูลชิดขวา (`ADC_DATAALIGH_RIGHT`) แล้วจ่ายสัญญาณนาฬิกาให้กับ GPIO พอร์ต C ในฟังก์ชัน `MX_GPIO_Init` จากไฟล์ `gpio.c` ดังรูปที่ 3.2

ส่วนการกำหนดให้ขา PC0 ทำหน้าที่เป็น Analog Input อยู่ในฟังก์ชัน `HAL_ADC_MspInit` ในไฟล์ `adc.c` ดังรูปที่ 3.3

```
ADC_HandleTypeDef hadc1;

/* ADC1 init function */
void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  /** Configure the global features of the ADC
  (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel
  its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_10;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

รูปที่ 3.1 แสดงการตั้งค่า ADC ในฟังก์ชัน `MX_ADC1_Init()`

```
void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();

}
```

รูปที่ 3.2 แสดงการจ่ายสัญญาณนาฬิกาให้ GPIOC

```c
void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* ADC1 clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOC_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PC0     ------> ADC1_IN10
    */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }
}
```

รูปที่ 3.3 แสดงตั้งค่าให้ PC0 ทำหน้าที่รับสัญญาณอินพุตแบบแอนะล็อก

4. การอ่านค่าที่ได้จากการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลแบบ Polling

ฟังก์ชัน `HAL_ADC_GetValue` ใช้สำหรับการอ่านค่าดิจิตอลที่โมดูล ADC แปลงได้ โดยจะส่งค่าที่แปลงได้เป็นตัวเลขจำนวนเต็มไม่มีเครื่องหมายขนาด 32 บิต หรือ `uint32_t` ดังรูปที่ 4.1

HAL_ADC_GetValue

Function Name	**uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)**
Function Description	Get ADC regular group conversion result.
Parameters	• **hadc:** ADC handle
Return values	• Converted value
Notes	• Reading DR register automatically clears EOC (end of conversion of regular group) flag.

รูปที่ 4.1 แสดงรายละเอียดของฟังก์ชัน HAL_ADC_GetValue

ก่อนการเรียกใช้ฟังก์ชัน `HAL_ADC_GetValue` ต้องตรวจสอบว่าโมดูล ADC ได้แปลงสัญญาณเสร็จสิ้นแล้ว ด้วยการเรียกฟังก์ชัน HAL_ADC_PollForConversion ซึ่งจะส่งค่าสถานะ HAL_OK กลับมา แสดงตัวอย่างการอ่านค่าจากโมดูล ADC ได้ดังรูปที่ 4.2 โดยค่า 100 เป็นค่า Timeout ของฟังก์ชัน

```c
volatile uint32_t adc_val = 0;

HAL_ADC_Start(&hadc1);
while (1){
  while ( HAL_ADC_PollForConversion(&hadc1, 100) != HAL_OK ){}
  adc_val = HAL_ADC_GetValue(&hadc1);
}
```

รูปที่ 4.2 แสดงวิธีการอ่านค่าจากโมดูล ADC

5. การอ่านค่าที่ได้จากการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลแบบหลายช่องสัญญาณร่วมกับ DMA

การแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลแบบหลายช่องสัญญาณ (Multichannel ADC) ต้องคำนึงถึงลำดับในการอ่านข้อมูลหลังจากที่แปลงสัญญาณเสร็จซึ่งบางครั้งอาจจะทำได้ลำบากหากใช้การอ่านค่าแบบ polling เพราะขณะที่โมดูล ADC กำลังแปลงสัญญาณ ไมโครคอนโทรลเลอร์ก็อาจจะมีงานอื่นที่ต้องทำอยู่ด้วย ทำไมการอ่านค่าอาจมีการคลาดเคลื่อน เช่น โมดูล ADC แปลงสัญญาณครั้งต่อไปเสร็จแล้ว แต่ยังไม่ได้อ่านค่าเก่าไปเก็บไว้ เป็นต้น

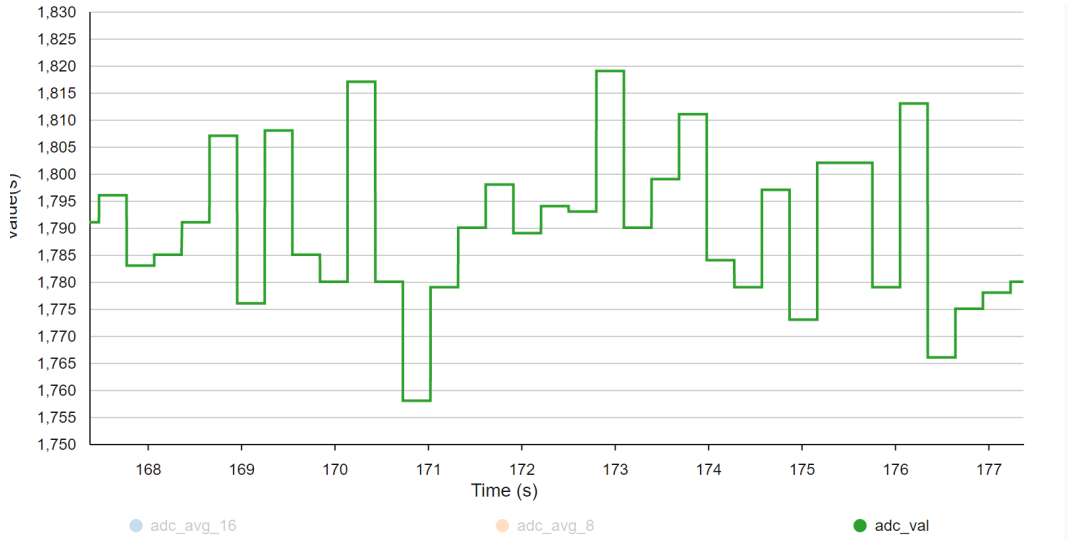
Direct Memory Access Controller (DMA) เป็นโมดูลที่ใช้สำหรับถ่ายโอนข้อมูล (data transfer) โดยไม่ผ่าน CPU เช่น สามารถใช้โมดูล DMA ในการถ่ายโอนข้อมูลปริมาณมากจาก peripheral ไปยังหน่วยความจำ โดยไม่ใช้ CPU ทำให้ CPU สามารถประมวลผลคำสั่งอื่นๆ ได้

ศึกษาการใช้งานโมดูล ADC แบบหลายช่องสัญญาณร่วกับ DMA ได้จากเอกสารประกอบการสอน

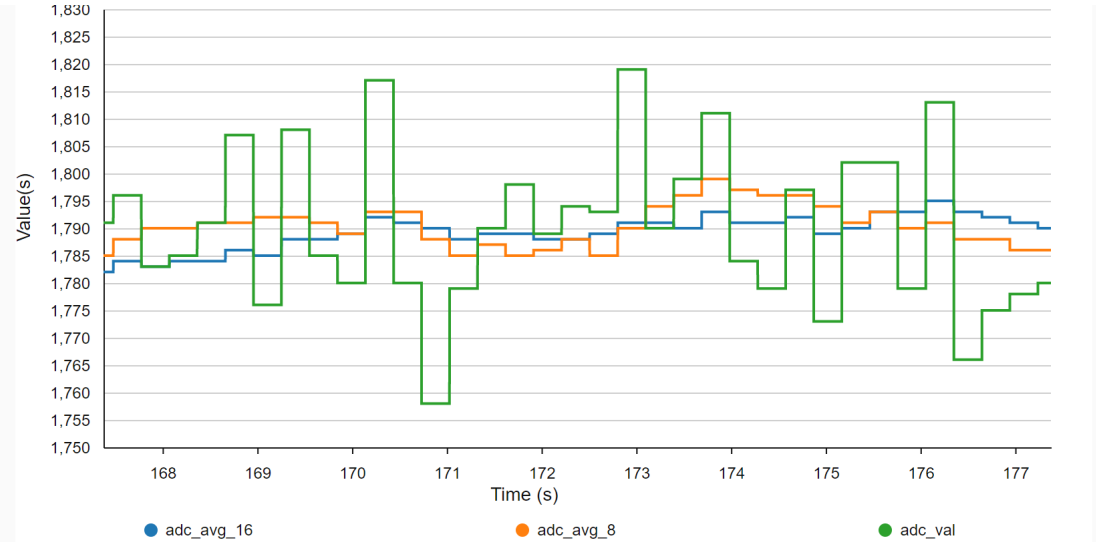
6. การลดการแกว่งของค่าที่ได้จากการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอล

ค่าที่ได้จากการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลจะมีการแกว่งดังรูปที่ 6.1 การลดการแกว่งอย่างง่ายสามารถทำได้โดยใช้วิธีหาค่าเฉลี่ย (Average) โดยนำค่าที่แปลงได้ค่าปัจจุบันไปรวมค่าก่อนหน้าแล้วใช้ค่าเฉลี่ยแทนค่าที่อ่านได้จากโมดูล ADC จะช่วยลดการแกว่งลงได้ทำให้ลดผลกระทบลงได้ รูปที่ 6.2 แสดงค่าล่าสุดที่อ่านได้จากโมดูล ADC (กราฟสีเขียว) เปรียบกับค่าเฉลี่ยของค่าล่าสุดกับ 7 ค่าก่อนหน้า (กราฟสีส้ม) และ 15 ค่าก่อนหน้า (กราฟสีน้ำเงิน) พบว่าการหาเฉลี่ยค่าล่าสุดกับ 16 ค่าก่อนหน้ามีการแกว่งน้อยที่สุด

รูปที่ 6.3 แสดงตัวอย่างโค้ดของฟังก์ชัน `average_8` และ `average_16` เพื่อใช้หาค่าเฉลี่ยเลขจำนวนเต็มจำนวน 8 และ 16 ค่า ตามลำดับ ในฟังก์ชันทั้งสองจะใช้ตัวแปรโลคอลแบบ static เพื่อเก็บค่าก่อนหน้า เพราะตัวแปรโลคอลแบบ static จะไม่ถูกทำลายเวลาฟังก์ชันนั้นๆ จบการทำงานลง ส่วนค่าที่อ่านได้ครั้งล่าสุดจะถูกเขียนทับลงใน array samples ไปเรื่อยๆ



รูปที่ 6.1 แสดงการแกว่งของค่าที่อ่านได้จากโมดูล ADC



รูปที่ 6.2 แสดงค่าที่อ่านได้จากโมดูล ADC เปรียบเทียบกับค่าเฉลี่ยจากการอ่าน 8 และ 16 ครั้ง

```
int average_8(int x) {
   static int samples[8];
   static int i = 0;
   static int total = 0;

   /* Update the moving average */
   total += x - samples[i];
   samples[i] = x;

   /* Update the index */
   i = (i==7 ? 0 : i+1);

   return total>>3;
}
```

```
int average_16(int x) {
   static int samples[16];
   static int i = 0;
   static int total = 0;

   /* Update the moving average */
   total += x - samples[i];
   samples[i] = x;

   /* Update the index */
   i = (i==15 ? 0 : i+1);

   return total>>4;
}
```

รูปที่ 6.3 แสดงตัวอย่างโค้ดในการหาค่าเฉลี่ย 8 และ 16 ค่าล่าสุด

7. การทดลอง

1. แสดงตัวเลขในรูปเลขฐาน 16 ทาง UART3

จงสร้างฟังก์ชัน `displayHEX` ขึ้นมา โดยมี Function Prototype ดังนี้

```
void displayHEX(uint32_t);
```

เพื่อแปลงเลขจำนวนเต็มขนาด 32 บิตที่รับเข้ามาแล้วแสดงผลออกทาง UART3 ในรูปเลขฐาน 16 จำนวน 8 หลัก ดังตัวอย่างต่อไปนี้

```
uint32_t hex1 = 501;
displayHEX(hex1);
```

โปแกรมจะแสดง **0x000001F5** ออกมาทาง UART3

สามารถใช้ฟังก์ชัน `sprintf` จากไลบรารี `stdio.h` เพื่อแปลงเลขฐานได้

2. วงจรปรับแรงดันไฟฟ้าโดยใช้ตัวต้านทานปรับค่าได้

จงต่อวงจรปรับแรงดันไฟฟ้าโดยเชื่อมต่อตัวต้านทานปรับค่าได้เข้ากับขาสัญญาณบนบอร์ด Nucleo767 ตามรูปที่ 7.1 เพื่อใช้สำหรับการทดสอบการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลของโมดูล ADC

ขา 1 เชื่อมต่อ Ground

ขา 2 เชื่อมต่อ PC0

ขา 3 เชื่อมต่อ V_{DD}

รูปที่ 7.1 แสดงการใช้ตัวต้านทานปรับค่าได้เพื่อสร้างสัญญาณอินพุตให้โมดูล ADC ที่ขา PC0

3. การอ่านค่าที่แปลงได้จากโมดูล ADC

ภายหลังจากการต่อวงจรในการทดลองที่ 2 แล้วให้ใช้โปรแกรม STM32CubeMX สร้างโปรเจ็คขึ้นมา แล้วตั้งค่าขา PC0 ให้ทำหน้าที่รับสัญญาณอินพุตแบบแอนะล็อกดังรูปที่ 2.1 และ รูปที่ 2.2 จากนั้นให้เขียนโปรแกรมตามรูปที่ 4.2 เพิ่มเติมลงในฟังก์ชัน `main` เพื่ออ่านค่าผลลัพธ์จากการแปลงสัญญาณของโมดูล ADC ที่ขา PC0 แล้วแสดงค่าที่แปลงได้ในโปรแกรม Tera Term ผ่าน UART3 ด้วยฟังก์ชัน `displayHEX` ที่สร้างจากการทดลองข้อ 1 พร้อมคำนวณและแสดงแรงดันไฟฟ้าที่ที่อ่านได้ (ทศนิยม 2 ตำแหน่ง) กำหนดให้แสดงผลทุกๆ 400 ms ให้ทดลองหมุนปรับตัวต้านทานปรับค่าได้ในวงจรปรับแรงดันไฟฟ้าแล้วสังเกตและบันทึกผล

ตัวอย่างการแสดงผล **ADC1_CH10 0x000001F5 Vin = 0.40 V**

ค่าที่น้อยที่สุดที่แปลงได้ คือ .....

ค่าที่มากที่สุดที่แปลงได้ คือ .....

ทำไมค่าที่แปลงได้สูงสุดจึงไม่ใช่ 0xFFFFFFFF

.....

.....

.....

4. การลดการแกว่งของค่าที่แปลงจากโมดูล ADC ด้วยวิธีหาค่าเฉลี่ย

จงเขียนโปรแกรมลดการแกว่งของค่าที่แปลงจากโมดูล ADC ด้วยวิธีหาค่าเฉลี่ย โดยใช้ฟังก์ชัน `average_8` และ `average_16` ดังรูปที่ 6.3 โดยให้สร้างตัวแปรโกลบอล adc_avg_8 และ adc_avg_16 ขึ้นมาเพื่อเก็บผลลัพธ์ของฟังก์ชันทั้งสอง โดยตอนส่งการทดลองให้แสดงกราฟเปรียบเทียบค่าที่อ่านได้จากโมดูล ADC และค่าเฉลี่ยทั้งสองในโปรแกรม STM32CubeMonitor ดังรูปที่ 6.2

5. การแสดงผลที่ได้จากโมดูล ADC เป็นช่วงๆ ด้วย LED

จงเขียนโปรแกรมเพื่อแสดงระดับของสัญญาณที่ได้จาก ADC ออกทาง LED จำนวน 4 ดวงที่ต่อเพิ่มจากบอร์ดทดลอง กำหนดให้ต่อ LED ที่ขา GPIO ใดก็ได้ โดยให้แบ่งระดับสัญญาณที่เป็นไปได้ออกเป็น 5 ระดับ เมื่อสัญญาณอยู่ระดับใดก็ให้ LED ติดดังตารางที่ 7.1 และให้ส่งค่าที่แปลงได้ออกทางพอร์ต UART ดังเช่นในการทดลองข้อ 3

ตารางที่ 7.1 แสดงระดับสัญญาณและและการติดสว่างของ LED

ระดับ	ผล
1	ไม่มี LED ติด
2	LED0 ติด
3	LED0 LED1 ติด
4	LED0 LED1 LED2 ติด
5	LED0 LED1 LED2 LED3 ติด

บันทึกผล

ระดับ	ช่วงของผลการแปลงจาก ADC
1	
2	
3	
4	
5	

6. การใช้งาน ADC ร่วมกับ DMA เพื่อแปลงหลายช่องสัญญาณ (Multichannel ADC with DMA)

จงใช้ตัวต้านทานปรับค่าได้จำนวน 2 ตัว **ให้ตัวต้านทานแต่ละตัวต่อเข้ากับ ADC1 จำนวน 4 ช่องสัญญาณ** พร้อมแสดงผลการแปลงออกทาง UART3 โดยกำหนดให้ใช้ DMA ในการถ่ายโอนข้อมูลจาก ADC เมื่อ DMA ทำการถ่ายโอนครึ่งหนึ่งแล้วให้ LD2 ติดค้าง และเมื่อถ่ายโอนข้อมูลอีกครึ่งที่เหลือเสร็จให้ LD2 ดับ

สามารถเลือกใช้ขาไหนของ ADC1 ก็ได้ โดยให้หลีกเลี่ยงขาที่มีการต่อใช้งานและมีสถานะทางไฟฟ้าแล้ว (ขาสีน้ำเงิน) เช่น PA7 เป็นต้น สามารถปรับความถี่ในการแสดงผล ความถี่การทำงานของ ADC และระยะเวลาในการ sampling ข้อมูลได้ตามสะดวก

ใบตรวจการทดลองที่ 5

Microcontroller Application and Development 2566

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

3. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 3&4 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

การทดลองข้อ 5 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

การทดลองข้อ 6 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. หากต้องแปลงสัญญาณ Analog ที่ channel 1 ของ ADC2 ต้องเชื่อมสัญญาณเข้ามาที่ขาใด (ระบุบอร์ดที่ใช้)

.....

.....

.....

2. จากการทดลองข้อ 5 หากเปลี่ยน Data alignment เป็น Left alignment จงหาช่วงของทั้ง 5 ระดับที่ทำให้ผลการทำงานเหมือนเดิม

ระดับ	ช่วงของผลการแปลงจาก ADC (จากการทดลองข้อ 5)	ช่วงของผลการแปลงจาก ADC (กรณีตั้งค่าเป็น Left Alignment)
1		
2		
3		
4		
5		