<u>การทดลองที่ 5</u> การใช้งาน ADC

วัตถุประสงค์

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

1. Analog to Digital Converter (ADC)

ไมโครคอนโทรลเลอร์ STM32F411 มี ADC จำนวน 1 โมดูล ได้แก่ โมดูล ADC1 เชื่อมต่อกับบัส APB2 แปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลด้วยวิธี successive approximation โดยมีความละเอียดในการแปลงสูงสุด 12 บิต โมดูล ADC มีช่องสัญญาณแบบมัลติเพลกซ์ 19 ช่อง แบ่งเป็นช่องสัญญาณภายนอกจำนวน 16 ช่อง ช่องสัญญาณภายในจำนวน 2 ช่อง และช่องสัญญาณ V_{BAT} แรงดันสัญญาณอ้างอิงสูงสุดคือ 3.6 V

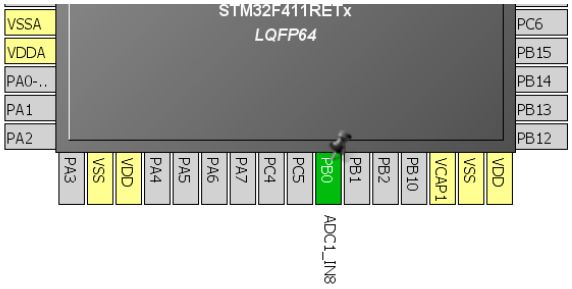
หากต้องการใช้งานโมดูล ADC เพื่อวัดแรงดันไฟฟ้าจากภายนอก สามารถศึกษาจาก Datasheet ได้ว่าช่องสัญญาณภายนอกจำนวน 16 ช่องนั้นเชื่อมต่อกับขาใดบ้างดังตัวอย่างรูปที่ 1.1 ซึ่งหากต้องการใช้ช่องสัญญาณ 8 ก็ให้เปลี่ยนขา PB0 ให้ทำหน้าที่ ADC1_8

Pin number					Pin name (function after reset)(1)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UQFN48	LQFP64	WLCSP49	LQFP100	UFBGA100L						
-	24	-	33	K5	PC4	I/O	FT	-	EVENTOUT	ADC1_14
-	25	-	34	L5	PC5	I/O	FT	-	EVENTOUT	ADC1_15
18	26	G5	35	M5	PB0	I/O	FT	-	TIM1_CH2N, TIM3_CH3, SPI5_SCK/I2S5_CK, EVENTOUT	ADC1_8
19	27	G4	36	M6	PB1	I/O	FT	-	TIM1_CH3N, TIM3_CH4, SPI5_NSS/I2S5_WS, EVENTOUT	ADC1_9

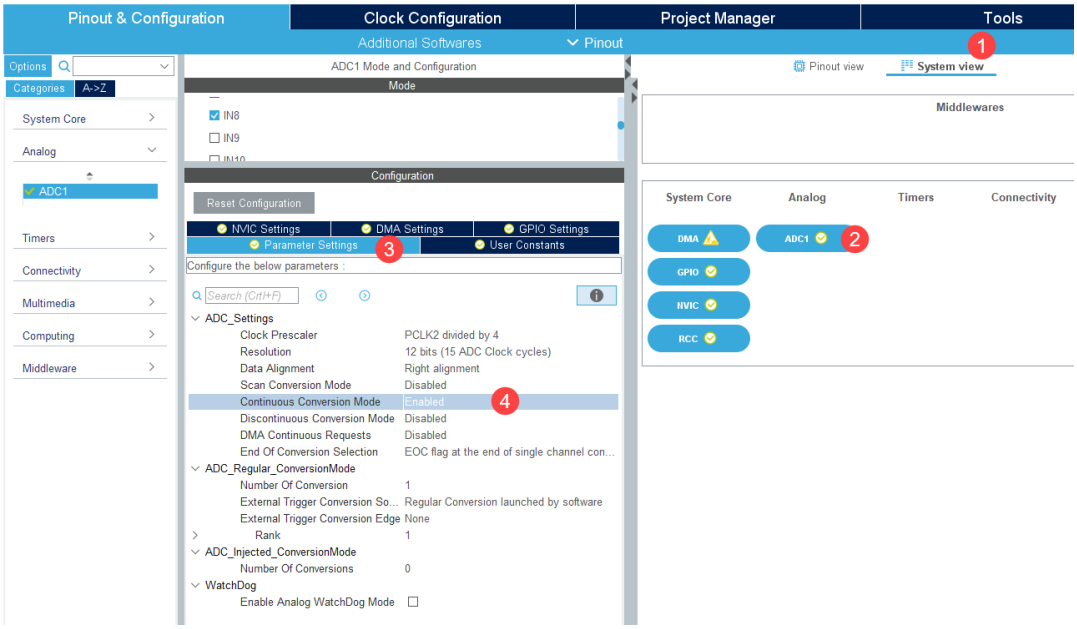
รูปที่ 1.1 แสดงการเชื่อมต่อช่องสัญญาณ ADC กับขาของไอซี

2. การตั้งค่าในโปรแกรม STM32CubeMX

การตั้งค่าสำหรับการทดลองครั้งนี้ต้องกำหนดให้ขา PB0 ทำหน้าที่เป็น ADC ดังรูปที่ 2.1 ซึ่งจะเชื่อมต่อวงจรปรับแรงดันไฟฟ้าเข้ามาที่ขานี้ จากนั้นตั้งค่า ADC ตามรูปที่ 2.2



รูปที่ 2.1 แสดงการตั้งค่าให้ PB0 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 B ในฟังก์ชัน `MX_GPIO_Init` ในไฟล์ `gpio.c` ดังรูปที่ 3.2 ส่วนการกำหนดให้ขา PB0 ทำหน้าที่เป็น Analog Input อยู่ในฟังก์ชัน `HAL_ADC_MspInit` ในไฟล์ `adc.c` ดังรูปที่ 3.3

```c
/* 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 = 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_SINGLE_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_8;
  sConfig.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()`

```c
void MX_GPIO_Init(void)
{

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

}
```

รูปที่ 3.2 แสดงการจ่ายสัญญาณนาฬิกาให้ GPIOB

```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_GPIOB_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PB0     ------> ADC1_IN8
    */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* USER CODE BEGIN ADC1_MspInit 1 */

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

รูปที่ 3.3 แสดงตั้งค่าให้ PB0 ทำหน้าที่รับสัญญาณอินพุตแบบแอนะล็อก

4. การอ่านค่าที่ได้จากการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอล

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

```
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. การทดลอง

1. แสดงตัวเลขในรูปเลขฐาน 16 ทาง UART2

จงสร้างฟังก์ชัน `displayHEX` ขึ้นมา โดยมี Function Prototype ดังนี้

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

เพื่อแปลงเลขจำนวนเต็มขนาด 32 บิตที่รับเข้ามาแล้วแสดงผลออกทาง UART2 ในรูปเลขฐาน 16 จำนวน 8 หลัก ดังตัวอย่างต่อไปนี้

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

โปแกรมจะแสดง **0x000001F5** ออกมาทาง UART2

2. วงจรปรับแรงดันไฟฟ้าโดยใช้ตัวต้านทานปรับค่าได้

จงต่อวงจรปรับแรงดันไฟฟ้าโดยเชื่อมต่อตัวต้านทานปรับค่าได้เข้ากับขาสัญญาณบนบอร์ด Nucleo411RET ตามรูปที่ 5.1 เพื่อใช้สำหรับการทดสอบการแปลงสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอลของโมดูล ADC

ขา 1 เชื่อมต่อ Ground

ขา 2 เชื่อมต่อ PB0

ขา 3 เชื่อมต่อ V_{DD}

รูปที่ 5.1 แสดงการใช้ตัวต้านทานปรับค่าได้เพื่อสร้างสัญญาณอินพุตให้โมดูล ADC ที่ขา PB0

3. การอ่านค่าที่แปลงได้จากโมดูล ADC

ภายหลังจากการต่อวงจรในการทดลองที่ 2 แล้วให้ใช้โปรแกรม STM32CubeMX สร้างโปรเจ็คขึ้นมา แล้วตั้งค่าขา PB0 ให้ทำหน้าที่รับสัญญาณอินพุตแบบแอนะล็อกดังรูปที่ 2.1 และ รูปที่ 2.2 จากนั้นให้เขียนโปรแกรมตามรูปที่ 4.2 เพิ่มเติมลงในฟังก์ชัน `main` เพื่ออ่านค่าผลลัพธ์จากการแปลง**สัญญาณของ**โมดูล ADC ที่ขา PB0 แล้วแสดงค่าที่แปลงได้ในโปรแกรม TeraTerm ผ่าน UART2 ด้วยฟังก์ชัน `displayHEX` ที่สร้างจากการทดลองที่ 1 กำหนดให้แสดงผลทุกๆ 300 ms ให้ทดลองหมุนปรับตัวต้านทานปรับค่าได้ในวงจรปรับแรงดันไฟฟ้าแล้วสังเกตและบันทึก

ค่าที่น้อยที่สุดที่แปลงได้ คือ .....

ค่าที่มากที่สุดที่แปลงได้ คือ .....

ทำไมค่าที่แปลงได้สูงสุดจึงไม่ใช่ 0xFFFFFFFF

.....

.....

.....

.....

4. การแสดงผลที่ได้จากโมดูล ADC เป็นช่วงๆ ด้วย LED

จงเขียนโปรแกรมเพื่อแสดงระดับของสัญญาณที่ได้จาก ADC ออกทาง LED จำนวน 4 ดวงที่ต่อเพิ่มบนบอร์ดทดลอง กำหนดให้ต่อ LED ที่ขา PC0 – PC3 โดยให้แบ่งระดับสัญญาณที่เป็นไปได้ออกเป็น 5 ระดับ เมื่อสัญญาณอยู่ระดับใดก็ให้ LED ติดดังตารางที่ 5.1 และให้ส่งค่าที่แปลงได้ออกทางพอร์ต UART ดังเช่นในการทดลองที่ 3

ตารางที่ 5.1 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

ใบตรวจการทดลองที่ 5

KU CSC Embedded System

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนิสิต _____ ชื่อ-นามสกุล _____

2. รหัสนิสิต _____ ชื่อ-นามสกุล _____

3. รหัสนิสิต _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 4 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. หากต้องแปลงสัญญาณ Analog ที่ channel 15 (ADC12_IN15) ต้องเชื่อมสัญญาณเข้ามาที่ขาใด

2. หากเปลี่ยน Data Alignment ในรูปที่ 2.2 เป็น Left Alignment ค่าสูงสุดและค่าต่ำสุดที่แปลงได้จะเปลี่ยนแปลงหรือไม่ ถ้าเปลี่ยนแปลงให้แสดงค่าที่เปลี่ยนแปลงนั้นด้วย