

# REPORT

---

Title: ADC-IR Reflective Sensor



Submission Date	2021.11.19	Major	Mechanical and control engineering
Subject	Embedded Controller	Professor	Young-Keun Kim
Name	Yeong-Won Song	Student Number	21700375
Partner Name	Sang-Hyeon-Kim	Partner Number	21700102

# Contents

<b>I. Introduction .....</b>	<b>3</b>
1.1 Purpose .....	3
1.2 Parts List.....	3
 <b>II. Procedure .....</b>	 <b>4</b>
2.1 Create EC_HAL functions .....	4
2.2 IR Reflective Sensor (TCRT 5000).....	5
2.3 Configuration .....	7
2.4 Line Tracing .....	9
 <b>III. Conclusion .....</b>	 <b>13</b>

## Appendix

# I. Introduction




## 1.1 Purpose

In this lab, we are required to create a simple application that uses ADCs to implement the line tracing mission for an RC car. The analog measurement of reflection values from two IR reflective sensors are used. The ADCs are triggered by a timer of given sampling rate.

## 1.2 Parts List

Parts	Specification	Quantity
IR-Reflective sensor	TCRT 5000	2
breadboard	-	1
NUCLEO-F411RE	Arm®(a) Cortex®-M4 with FPU	1

**Table 1. Part List**

		
NUCLEO-F411RE	IR-Reflective sensor	Breadboard

**Table 2. experiment equipment**

## II. Procedure

### 2.1 Create EC\_HAL functions

Specific for given Output Pins

Include File	Function	Description
ecTIM.h,c	<pre>uint32_t is_UIF(TIM_TypeDef *timx); void clear_UIF(TIM_TypeDef *timx); void TIM_INT_enable(TIM_TypeDef* timx); void TIM_INT_disable(TIM_TypeDef* timx);</pre>	Initialize timer counter period of usec. For Timerx= TIM1, TIM2
ecADC.h,c	<pre>void ADC_init(GPIO_TypeDef *port, int pin, int type); void ADC_continue(int contmode); void ADC_TRGO (TIM_TypeDef* TIMx, int msec, int edge). void ADC_sequence(int length, int *seq); void ADC_start(void); uint32_t ADC_read(); uint32_t ADC_pinmap(GPIO_TypeDef *port, int pin); uint32_t is_ADC_EOC(ADC_TypeDef *ADCx); uint32_t is_ADC_OVR(ADC_TypeDef *ADCx); void clear_ADC_OVR(ADC_TypeDef *ADCx); uint32_t ADC_read();</pre>	// ADC_pinmap() will be provided

### 2.2 IR Reflective Sensor (TCRT 5000)

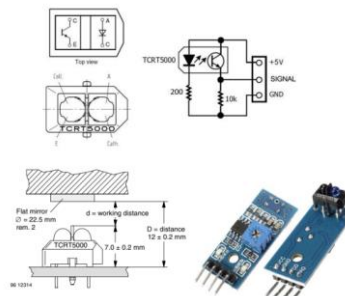


Figure 1. IR Reflective Sensor

*The HC-SR04 Ultrasonic Range Sensor Features:*

- Input Voltage: 5V
- Detector type: phototransistor
- Operating range within > 20 % relative collector current: 0.2 mm to 15 mm
- Emitter wavelength: 950 nm

#### APPLICATIONS

- Position sensor for shaft encoder
- Detection of reflective material such as paper, IBM cards, magnetic tapes etc.
- Limit switch for mechanical motions in VCR
- General purpose - wherever the space is limited

## 2.3 Configuration

Create a new project named as “LAB\_ADC\_IR”.

Name the source file as “LAB\_ADC\_IR.c”

*Configuration Input and Output pins*

TIMER	
Timer 3 Up-Counter, Counter CLK 1kHz OC1M: Output Compare 1 mode (PWM mode 1) Master Mode selection: (TRGO) OC1REF	
ADC	GPIO
ADC_IN8 (1st channel) ADC_IN9 (2nd channel) ADC Clock Prescaler /8 12-bit resolution, right alignment, Single conversation mode Scan mode: Two channels in regular group External trigger (Timer3 TRGO) @ 1kHz Trigger Detection on Rising Edge	PB_0, PB_1: Analog Mode ADC_IN8 (PB_0) ADC_IN9 (PB_1) No Pull-up Pull-down

## 2.4 Line Tracing

- Create a logic to trace a dark line on white background surface for your RC car.
- Use 2 IR reflective sensors to detect if the black line is in between the sensors. It should display whether the system needs to move Left or Right to keep the line between sensors.
- Set the ADC sampling rate trigger to be 1KHz, to decrease burden to your CPU.
- Determine the threshold value to differentiate dark and white surface of the object.
- Display (1) reflection value of IR1 and IR2 (2) print 'GO LEFT' or 'GO RIGHT' on serial monitor of Tera-Term. Print the values every second

### - Flow Chart

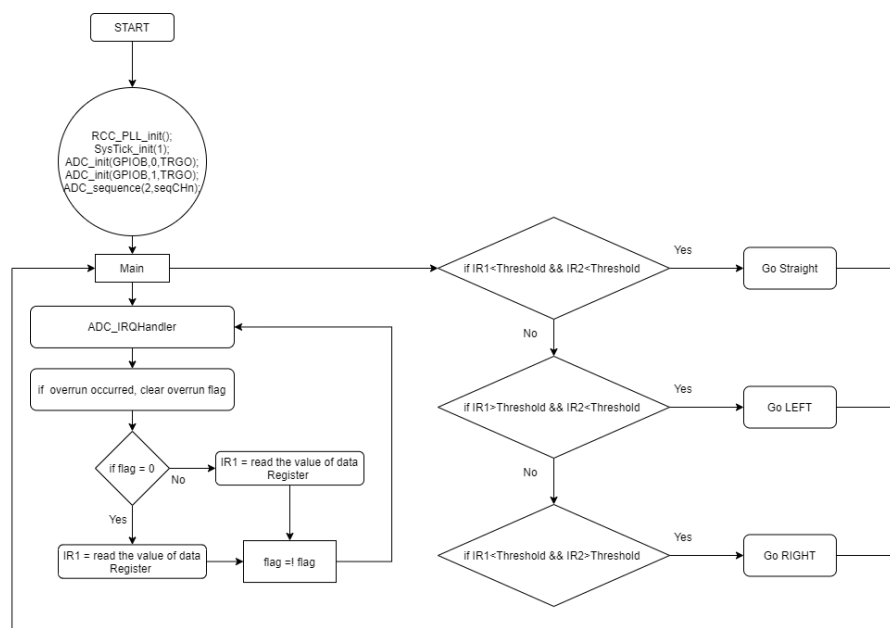


Figure 2. Flow chart

### - External Circuit

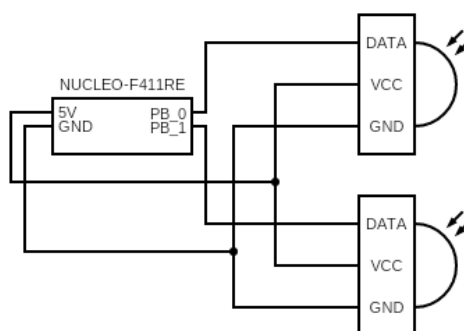


Figure 3. External circuit

- Main source code

```

/**
 * @author SONG YEONG WON
 * @Mod 2021-11-19 by YWSONG
 * @brief Embedded Controller: LAB : ADC-IR Reflective Sensor
 */
*/
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecSysTick.h"
#include "ecUART.h"
#include "ecADC.h"
#include "math.h"
#include "ecPWM.h"

//IR parameter//
static int flag =0 ;
static uint32_t IR1=0;
static uint32_t IR2=0;

static int seqCHn[2] = {8,9};

void setup(void);
void ADC_IRQHandler(void);

int main(void) {
    // Initialization -----
    setup();
    ADC_start();

    // Infinite Loop -----
    while(1){

        printf("IR1 = %d\r\n",IR1);
        printf("IR2 = %d\r\n",IR2);
        if(IR1<1500 && IR2<1500){
            printf("Go Straight\r\n");
        }
        if(IR1>1500 && IR2<1500){
            printf("Go LEFT\r\n");
        }
        if(IR1<1500 && IR2>1500){
            printf("Go RIGHT\r\n");
        }
        delay_ms(1000);
    }

    // InitializationW
    void setup(void)
    {
        RCC_PLL_init(); // System Clock = 84MHz
        UART2_init();
        SysTick_init(1);

        ADC_init(GPIOB,0,TRGO);
        ADC_init(GPIOB,1,TRGO);

        ADC_sequence(2,seqCHn);
    }

    void ADC_IRQHandler(void){
        if(is_ADC_OVR()){
            clear_ADC_OVR();
        }
        if(is_ADC_EOC()){ //after finishing sequence
            if(flag==0){
                IR1 = ADC1->DR;
            }
            else if(flag==1){
                IR2 = ADC1->DR;
            }
            flag =!flag;
        }
    }
}

```

- Discussion

1) How would you change the code if you need to use 3 Analog sensors?

To use 3 analog sensor, three digital pin should be set. The pin of three ADC pinout map channel in the set does not overlap. In the case of regular mode, the sequence of the channel must be determined. For each of the pin number sequentially channel by channel number, 1-D array sequence for the input it. To use three analog sensors, one more flag must be added in ADC\_IRQHandler.

2) Which registers should be modified if you need to use Injection Groups instead of regular groups for 2 analog sensors?

The Regular Group has 16 channels and shares one regular data register. On the other hand, the Injected Group has four channels, and each channel has an independent Injected data register. Using Injected Groups does not require flag on the code because it has an independent data register. To use Injected Groups, a total of five registers needs to be modified.

Determine the Injected Sequence length with ADC1\_JSQR: JL [1:0] register.

Bits 21:20 **JL[1:0]**: Injected sequence length  
 These bits are written by software to define the total number of conversions in the injected channel conversion sequence.  
 00: 1 conversion  
 01: 2 conversions  
 10: 3 conversions  
 11: 4 conversions

**Figure 4. Injected Sequence Length**

And then determine the Channel sequence with ADC1\_JSQR: JSQn [4:0] register.

Bits 19:15 **JSQ4[4:0]**: 4th conversion in injected sequence (when JL[1:0]=3, see note below)  
 These bits are written by software with the channel number (0..18) assigned as the 4th in the sequence to be converted.  
 Bits 14:10 **JSQ3[4:0]**: 3rd conversion in injected sequence (when JL[1:0]=3, see note below)  
 Bits 9:5 **JSQ2[4:0]**: 2nd conversion in injected sequence (when JL[1:0]=3, see note below)  
 Bits 4:0 **JSQ1[4:0]**: 1st conversion in injected sequence (when JL[1:0]=3, see note below)

**Figure 5. channel sequence**

Determine Conversion start with ADC1\_CR2: JSWSTART register.

In addition, we determine the Trigger source in ADC1\_CR2: JEXTSEL register.



Bits 21:20 **JEXTEN**: External trigger enable for injected channels

These bits are set and cleared by software to select the external trigger polarity and enable the trigger of an injected group.

00: Trigger detection disabled

01: Trigger detection on the rising edge

10: Trigger detection on the falling edge

11: Trigger detection on both the rising and falling edges

**Figure 6. Trigger polarity**

The trigger polarity is determined in the ADC1\_JEXTEN register.

Bits 19:16 **JEXTSEL[3:0]**: External event select for injected group

These bits select the external event used to trigger the start of conversion of an injected group.

0000: Timer 1 CC4 event

0001: Timer 1 TRGO event

0010: Timer 2 CC1 event

0011: Timer 2 TRGO event

0100: Timer 3 CC2 event

0101: Timer 3 CC4 event

0110: Timer 4 CC1 event

0111: Timer 4 CC2 event

1000: Timer 4 CC3 event

1001: Timer 4 TRGO event

1010: Timer 5 CC4 event

1011: Timer 5 TRGO event

1100: Reserved

1101: Reserved

1110: Reserved

1111: EXTI line15

**Figure 7. Tirgger source**

The trigger source is determined in the ADC1\_JEXTSEL register. Finally, read the value of the Injected data register for 1-4 channels from ADC1\_JDRn register.

### III. Conclusion

Line tracking through RC car's IR Sensor was attempted using the ADC concept. TRGO external trigger using Timer was used, and through this, the sampling time of ADC could be determined. In addition, it was possible to analyze and compare the differences between the Regular Group and the Injected Group. It was confirmed in code that each mode was performed through control over different registers.

#### - Troubleshooting

**Q.** How did you set the threshold value and how did you print the RC car's line tracking direction?

**A.** Data values were measured with direct IR sensors. The value was measured using light reflectance according to black and white, and an appropriate threshold value was determined. Three cases were dealt with: GO straight, go right, and go left according to the threshold value.

## Appendix

- Video demo Link

Click [here](#) watch video.

- Source code

---

```

#include "stm32f411xe.h"
#include "ecSysTick.h"
#include "ecADC.h"
#include "ecGPIO.h"
#include "ecTIM.h"
#include <stdint.h>
uint32_t xresult;

void ADC_init(GPIO_TypeDef *port, int pin, int trigmode){ //mode 0 : SW, 1 : TRGO
// 0. Match Port and Pin for ADC channel
int CHn = ADC_Pinner(port, pin); // ADC Channel <->Port/Pin mapping

// GPIO configuration -----
// 1. Initialize GPIO port and pin as ANALOG, no pull up / pull down
GPIO_init(port, pin, ANALOG); // ANALOG = 3
GPIO_pupdr(port, pin, NOPUPD); // EC_NONE = 0

// ADC configuration -----
// 1. Total time of conversion setting
// Enable ADC peripheral clock
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable the clock of RCC_APB2ENR_ADC1EN

// Configure ADC clock pre-scaler
ADC->CCR &= ~(3UL<<16);
ADC->CCR |= 1<<16; // 0001: PCLK2 divided by 4 (21MHz)

// Configure ADC resolution
ADC1->CR1 &= ~(3UL<<24); // 00: 12-bit resolution (15cycle+)

// Configure channel sampling time of conversion.
// Software is allowed to write these bits only when ADSTART=0 and JADSTART=0 !!
// ADC clock cycles @42MHz = 2us
if(CHn < 10) ADC1->SMPR2 |= 4U << 3*CHn; // sampling time conversion : 84
else ADC1->SMPR1 |= 4U << 3*(CHn%10);

// 2. Regular / Injection Group
//Regular: SQRx, Injection: JSQx

// 3. Repetition: Single or Continuous conversion
//ADC1->CR2 |= ADC_CR2_CONT; // Enable Continuous conversion mode
ADC1->CR2 &= ~(1UL<<1); // default : Single conversion mode

// 4. Single Channel or Scan mode
// - Single Channel: scan mode, right alignment
ADC1->CR1 |= 1<<8; // 1: Scan mode enable

// Configure the sequence length
ADC1->SQR1 &= ~(15UL<<20); // 0000: 1 conversion in the regular channel conversion sequence

// Configure the channel sequence
ADC1->SQR3 &= ~(0x1F<<0); // SQR1 clear bits
ADC1->SQR3 |= (CHn & ADC_SQR3_SQ1); // Choose the channel to convert firstly
//ADC1->SQR3 |= (CHn <<0);

// 5. Interrupt Enable
// Enable EOC(conversion) interrupt.
ADC1->CR1 &= ~(1UL<<5); // Interrupt reset
ADC1->CR1 |= 1<<5; // Interrupt enable

// Enable ADC_IRQn
NVIC_SetPriority(ADC_IRQn,1); // Set Priority to 2
NVIC_EnableIRQ(ADC_IRQn); // Enable interrupt form ADC1 peripheral

/* -----*/
// HW TRIGGER MODE
/* -----*/

// TRGO Initialize : TIM3, 1msec, RISE edge
if(trigmode==TRGO) ADC_TRGO(TIM3, 1, RISE);
}

```

---

```

void ADC_TRGO(TIM_TypeDef* TIMx, int msec, int edge){
    // set timer
    int timer = 0;
    if(TIMx==TIM2) timer=2;
    else if(TIMx==TIM3) timer=3;

    // Single conversion mode (disable continuous conversion)
    ADC1->CR2 &= ~(1UL<<1); // Discontinuous conversion mode
    ADC1->CR2 |= 1<<10; // Enable EOCS

    // HW Trigger configuration -----

    // 1. TIMx Trigger Output Config
    // Enable TIMx Clock
    TIM_init(TIMx, msec);
    TIMx->CR1 &= ~(1UL<<0); //counter disable

    // Set PSC, ARR
    TIM_period_ms(TIMx, msec);

    // Master Mode Selection MMS[2:0]: Trigger output (TRGO)
    TIMx->CR2 &= ~(7UL<<4); // reset MMS
    TIMx->CR2 |= 4<<4; //100: Compare - OC1REF signal is used as trigger output (TRGO)

    // Output Compare Mode
    TIMx->CCMR1 &= ~(7UL<<4); // OC1M : output compare 1 Mode
    TIMx->CCMR1 |= 6<<4; // OC1M = 110 for compare 1 Mode chl

    // OC1 signal
    TIMx->CCER |= 1<<0; // CC1E Capture enabled
    TIMx->CCER = (TIMx->ARR)/2; // duty ratio 50%

    // Enable TIMx
    TIMx->CR1 |= 1<<0; //counter enable

    // 2. ADC HW Trigger Config.
    // Select Trigger Source
    ADC1->CR2 &= ~ADC_CR2_EXTSEL; // reset EXTSEL
    ADC1->CR2 |= (timer*2+2)<<24; // TIMx TRGO event (ADC : TIM2, TIM3 TRGO)

    //Select Trigger Polarity
    ADC1->CR2 &= ~ADC_CR2_EXTEN; // reset EXTEN, default
    if(edge==RISE) ADC1->CR2 |= ADC_CR2_EXTEN_0; // trigger detection rising edge
    else if(edge==FALL) ADC1->CR2 |= ADC_CR2_EXTEN_1; // trigger detection falling edge
    else if(edge==BOTH) ADC1->CR2 |= ADC_CR2_EXTEN_Msk; // trigger detection both edge
}

void ADC_continue(int contmode){
    if(contmode==CONT){
        // Repetition: Continuous conversion
        ADC1->CR2 |= 1<<1; // Enable Continuous conversion mode
        ADC1->CR1 &= ~ADC_CR1_SCAN; // 0: Scan mode disable
    }
    else //if(contmode==SINGLE)
    {
        // Repetition: Single conversion
        ADC1->CR2 &= ~ADC_CR2_CONT; // Disable Continuous conversion mode
        ADC1->CR1 |= ADC_CR1_SCAN; // 1: Scan mode enable
    }
}

void ADC_sequence(int length, int *seq){
    ADC1->SQR1 &= ~(0xF<<20); // reset length of conversions in the regular channel
    ADC1->SQR1 |= (length-1)<<20; // conversions in the regular channel conversion sequence

    for(int i = 0; i<length; i++){
        if (i<6){
            ADC1->SQR3 &= ~(0x1F<<i*5); // SQn clear bits
        }
    }
}

```

---

```

        ADC1->SQR3 |= seq[i]<<(i*5);        // Choose the channel to convert sequence
    }
    else if (i < 12){
        ADC1->SQR2 &= ~(0x1F<<(i-6)*5);    // SQn clear bits
        ADC1->SQR2 |= seq[i]<<(i-6)*5;      // Choose the channel to convert sequence
    }
    else{
        ADC1->SQR1 &= ~(0x1F<<(i-12)*5);    // SQn clear bits
        ADC1->SQR1 |= seq[i]<<(i-12)*5;     // Choose the channel to convert sequence
    }
}

void ADC_start(void){
    // Enable ADON, SW Trigger-----
    ADC1->CR2 |= 1<<0;
    ADC1->CR2 |= 1<<30;
}

uint32_t is_ADC_EOC(void){
    return ((ADC1->SR & 1<<1) != 0);
}

uint32_t ADC_read(void){
    return ADC1->DR &= 0xFFFF;
}

uint32_t is_ADC_OVR(void){
    return ((ADC1->SR & 1<<5) != 0);
}

void clear_ADC_OVR(void){
    ADC1->SR &= ~(1UL<<5);
}

uint32_t ADC_pinmap(GPIO_TypeDef *Port, int Pin){
    if(Port == GPIOA){
        if(Pin == 0)    return 0;
        else if(Pin == 1) return 1;
        else if(Pin == 4) return 4;
        else if(Pin == 5) return 5;
        else if(Pin == 6) return 6;
        else if(Pin == 7) return 7;
        else             while(1);
    }
    else if(Port == GPIOB){
        if(Pin == 0)    return 8;
        else if(Pin == 1) return 9;
        else             while(1);
    }
    else if(Port == GPIOC){
        if(Pin == 0)    return 10;
        else if(Pin == 1) return 11;
        else if(Pin == 2) return 12;
        else if(Pin == 3) return 13;
        else if(Pin == 4) return 14;
        else if(Pin == 5) return 15;
        else             while(1);
    }
}

```

---

Figure 8. ADC.c code

```
#ifndef __MY_ADC_H
#define __MY_ADC_H
#include "stm32f4llxe.h"

// ADC trigmode
#define SW 0
#define TRGO 1

// ADC contmode
#define CONT 0
#define SINGLE 1

// Edge Type
#define RISE 1
#define FALL 2
#define BOTH 3

#define _DEFAULT 0

// ADC setting
void ADC_start(void);
void ADC_init(GPIO_TypeDef *port, int pin, int mode); // trigmode : SW , TRGO
void ADC_continue(int contmode); // contmode : CONT, SINGLE / Operate both ADC,JADC
void ADC_TRGO(TIM_TypeDef* TIMx, int msec, int edge);
void ADC_sequence(int length, int *seq);
void TIM_TRGO_init(TIM_TypeDef* timx, uint32_t msec);

uint32_t is_ADC_EOC(void);
uint32_t is_ADC_OVR(void);
void clear_ADC_OVR(void);
uint32_t ADC_read(void);
uint32_t ADC_pinmap(GPIO_TypeDef *port, int pin);

#endif
```

Figure 9. ADC.h code

## - Documentation

# ADC

## ADC\_init()

Initializes GPIO port and pin number, Trigger mode.

```
void ADC_init(GPIO_TypeDef *port, int pin, int mode);
```

### Parameters

- **Port:** Port Number, GPIOA~GPIOH
- **pin:** pin number (int) 0~15
- **mode:** trigmode : SW , TRGO

### Example code

```
ADC_init(GPIOB,0,TRGO);
ADC_init(GPIOB,1,TRGO);
```

---

## ADC\_continue()

Set the Single mode or Continuous mode

```
void ADC_continue(int contmode);
```

### Parameters

- **contmode:** contmode : CONT, SINGLE

### Example code

```
ADC_continue(SINGLE)
```

## ADC\_TRGO()

Initializes TRGO with default setting.

```
void ADC_TRGO(TIM_TypeDef* TIMx, int msec, int edge);
```

### Parameters

- **TIM:** Select Timer
- **msec:** period of timer
- **edge:** Select Trigger Polarity

### Example code

```
ADC_TRGO(TIM3, 1, RISE);
```

---

---

## ADC\_sequence()

Set the Sequence length and channel sequence.

```
void ADC_sequence(int length, int *seq);
```

### Parameters

- **length**: length of conversions
- **seq**: the number of conversion sequence

### Example code

```
static int seqCHn[2] = {8,9};  
ADC_sequence(2,seqCHn);
```

## ADC\_start()

Software Trigger.

```
void ADC_start(void);
```

### Parameters

- **void**

### Example code

```
ADC_start();
```

## is\_ADC\_EOC()

check the conversion ends and read the data of register ADC.

```
uint32_t is_ADC_EOC(void);
```

### Parameters

- **void**

### Example code

```
if(is_ADC_EOC()){           //after finishing sequence  
    if(flag==0){  
        IR1 = ADC1->DR;  
    }else if(flag==1){  
        IR2 = ADC1->DR;  
    }  
    flag =!flag;  
}
```



---

### is\_ADC\_OVR()

Check the overrun is occurred.

```
uint32_t is_ADC_OVR(void);
```

#### Parameters

- void

#### Example code

```
if(is_ADC_OVR()){
    clear_ADC_OVR();
}
```

### clear\_ADC\_OVR()

Clear the overrun flag.

```
void clear_ADC_OVR(void);
```

#### Parameters

- void

#### Example code

```
if(is_ADC_OVR()){
    clear_ADC_OVR();
}
```

### ADC\_read()

Read the data register of ADC.

```
uint32_t ADC_read(void);
```

#### Parameters

- void

#### Example code

```
IR1 = ADC_read();
```

### ADC\_pinmap()

Match the channel number according to the port and pin number.

```
uint32_t ADC_pinmap(GPIO_TypeDef *Port, int Pin);
```

#### Parameters

- **port** : Port Number, GPIOA~C
- **pin** : pin number (int) 0~15

#### Example code

```
channel = ADC_pinmap(GPIOA, 4); // channel is 4 when GPIOA port and pin 4.
```

---

**Figure 10. documentation of ADC**