Use a Potentiometer to Provide Variable Voltage for ADC Input on the UCT STM32F051C6 Dev Board

1. Reference documents

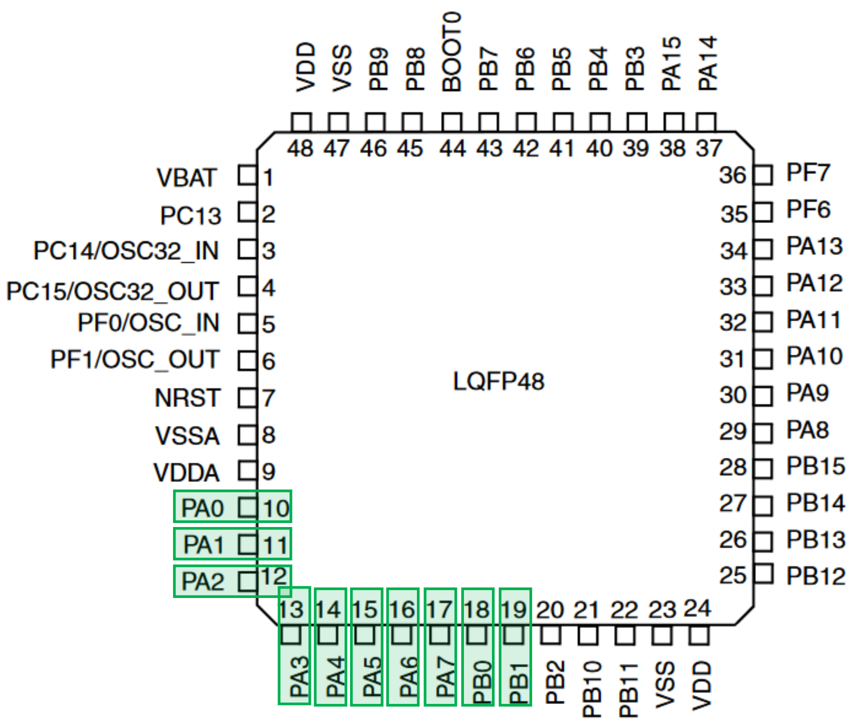
STM32F051xx Datasheet

STM32F0xx Reference manual

UCT STM32F051C6 Dev Board

2. STM32F051C6 ADC input pins

The ADC input pins can be identified from *Pin Definitions* section in the STM32F051xx Datasheet.



PIN	ADC input
10	ADC_IN0
11	ADC_IN1
12	ADC_IN2
13	ADC_IN3
14	ADC_IN4
15	ADC_IN5
16	ADC_IN6
17	ADC_IN7
18	ADC_IN8
19	ADC_IN9

3. Pin Selection

On the UCT STM32F051C6 Dev Board, the two potentiometers, POT0 and POT1, are connected to pins 15(PA5) and 16(PA6), respectively. In this example, I'll be using POT0 to provide variable voltage to pin PA5.

4. Libraries

After having decided on which ADC input pin(s) to use, open your preferred IDE and setup a new project. Once that's done, delete contents of the main.c file and let's then include all the libraries we'll need for our project.

```
#include “stm32f0xx.h
#include “lcd_stm32f0xx.h”
```

5. Configuring the ADC

Write to ADCAL and ADEN in the ADC_CR register if ADC is disabled i.e ADEN = 0
Write to ADSTART in the ADC_CR register if ADC is enabled and no pending request to disable the ADC
After enabling the ADC, wait until the ADRDY bit in the ADC_ISR is 1 allowing the ADC stabilization time

e.g enable ADC: `ADC1->CR  = ADC_CR_ADEN;`

6. Calibration of the ADC

Calibrating the ADC removes the offset error inherent in the ADC and calculates the calibration factor which will be applied until the next power-off. This process is done before the conversion process and can only be initiated if the ADC is disabled on the ADC channel of interest. The IDE must not use the ADC during this process.

e.g initiate ADC calibration:

```
ADC1->CR  &=  ~ADC_RC_ADEN;   // ensure the ADC is disabled first
ADC1->CR |=  ADC_CR_ADCAL;  // initiate ADC calibration
```

7. ADC Clock

The ADC has two independent sources of clock signal, the APB Clock(can bypass clock domain resynchronizations) and the ADC asynchronous clock(can reach max ADC clock freq).
To enable the APB clock write 1 to the ADCRST bit of the RCC_APB2ENR

e.g Enable ADC APB clock:

```
RCC->APB2ENR  |= RCC_APB2ENR_ADC1EN;
```

8. Channel selection

The STM32F052C6 microcontroller does analog-to-digital conversions on different channels for different pins, this means only one conversion can occur at a time because each pin has a dedicated channel. Analog Input 0 would use channel 0 and Analog Input 1 would use channel 1 and so on. Channels are selected by writing 1 to the corresponding CHSEL bit in the ADC_ CHSELR(Channel Selector Register)

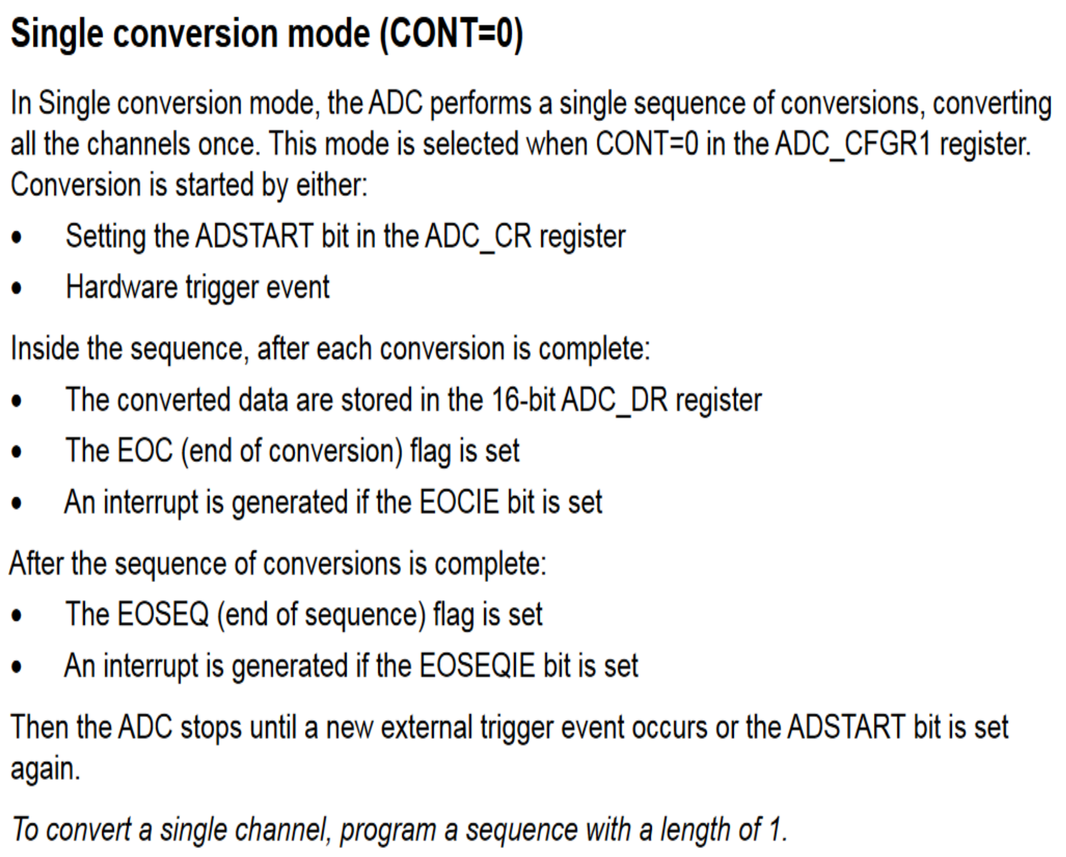
e.g selecting channel 1 if using PA1(ADC_IN1):

```
ADC->CHSELR  |=  ADC_CHSELR_CHSEL1;
```

The order in which the multiple channels will be scanned can be configured by programming the SCANDIR bit in the ADC_CFGR1 register:
•SCANDIR=0: forward scan Channel 0 to Channel 18
•SCANDIR=1: backward scan Channel 18 to Channel 0
e.g configure the ADC so that it scans the multiple ADC channels forward:

```
ADC->CFGR1 &=  ~ADC_CFGR1_SCANDIR;
```

9. Single and Continuous conversion modes



Single conversion mode (CONT=0)

In Single conversion mode, the ADC performs a single sequence of conversions, converting all the channels once. This mode is selected when CONT=0 in the ADC_CFGR1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

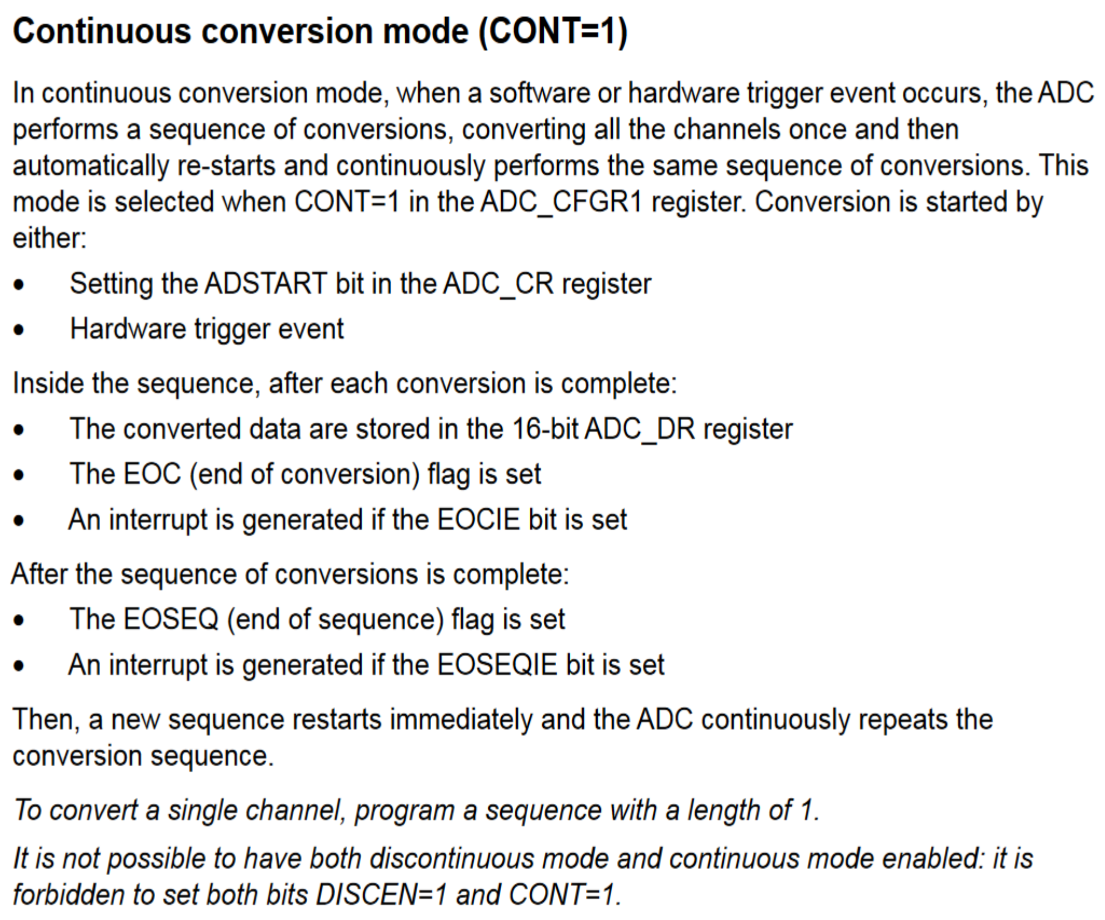
- The converted data are stored in the 16-bit ADC_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOSEQ (end of sequence) flag is set
- An interrupt is generated if the EOSEQIE bit is set

Then the ADC stops until a new external trigger event occurs or the ADSTART bit is set again.

To convert a single channel, program a sequence with a length of 1.



Continuous conversion mode (CONT=1)

In continuous conversion mode, when a software or hardware trigger event occurs, the ADC performs a sequence of conversions, converting all the channels once and then automatically re-starts and continuously performs the same sequence of conversions. This mode is selected when CONT=1 in the ADC_CFGR1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

- The converted data are stored in the 16-bit ADC_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOSEQ (end of sequence) flag is set
- An interrupt is generated if the EOSEQIE bit is set

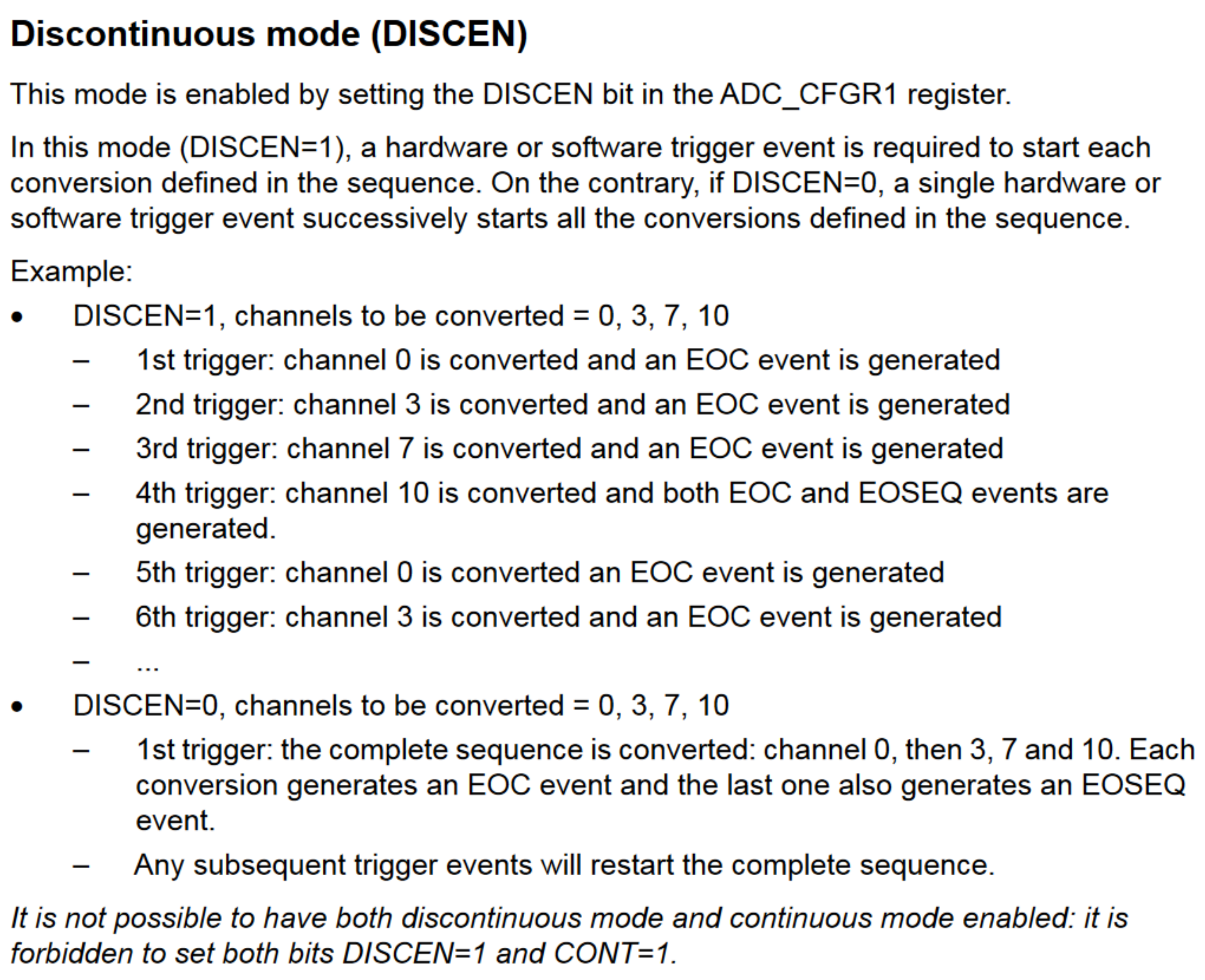
Then, a new sequence restarts immediately and the ADC continuously repeats the conversion sequence.

To convert a single channel, program a sequence with a length of 1.

It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN=1 and CONT=1.

e.g configure the ADC to continuous mode

```
ADC1->CFGR1  =  ADC_CFGR1_CONT
```



Discontinuous mode (DISCEN)

This mode is enabled by setting the DISCEN bit in the ADC_CFGR1 register.

In this mode (DISCEN=1), a hardware or software trigger event is required to start each conversion defined in the sequence. On the contrary, if DISCEN=0, a single hardware or software trigger event successively starts all the conversions defined in the sequence.

Example:

- DISCEN=1, channels to be converted = 0, 3, 7, 10
  - 1st trigger: channel 0 is converted and an EOC event is generated
  - 2nd trigger: channel 3 is converted and an EOC event is generated
  - 3rd trigger: channel 7 is converted and an EOC event is generated
  - 4th trigger: channel 10 is converted and both EOC and EOSEQ events are generated.
  - 5th trigger: channel 0 is converted an EOC event is generated
  - 6th trigger: channel 3 is converted and an EOC event is generated
  - ...
- DISCEN=0, channels to be converted = 0, 3, 7, 10
  - 1st trigger: the complete sequence is converted: channel 0, then 3, 7 and 10. Each conversion generates an EOC event and the last one also generates an EOSEQ event.
  - Any subsequent trigger events will restart the complete sequence.

It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN=1 and CONT=1.

10. Selecting ADC Resolution

STM32F051C6 has programmable ADC conversion resolution. It can be configured to 12, 10, 8 or 6 bits by writing to the RES[1:0] bit(3rd and 4th bits) in the ADC_CFGR1 register. The result of a conversion is always 12 bits wide with all unused LSBs set to zero.

RES[1:0] = 00 for 12 bit *res*

RES[1:0] = 01 for 10 bit *res*

RES[1:0] = 10 for 8 bit *res*

RES[1:0] = 11 for 6 bit *res*

e.g configure the ADC to a res of 6 bits

`ADC->CFGR1  |= 0b0000 0000 0000 0000 0000 0000 0000 1100;` or `ADC->CFGR1  |= 0b1100;`

Code

```
*/-----------------------------------------------------------------------------------------------------------------------------------
Libraries
-----------------------------------------------------------------------------------------------------------------------------------*/
#define STM32F051
#include “stm32f0xx.h”
#include “lcd_stm32f0.h”
*/-----------------------------------------------------------------------------------------------------------------------------------
DEFINES
-----------------------------------------------------------------------------------------------------------------------------------*/

*/-----------------------------------------------------------------------------------------------------------------------------------
GLOBAL VARIABLES
-----------------------------------------------------------------------------------------------------------------------------------*/

*/-----------------------------------------------------------------------------------------------------------------------------------
FUNCTIONS DECLARATIONS
-----------------------------------------------------------------------------------------------------------------------------------*/
void main(void);
void init_GPIOA(void);
void init_ADC(void);
void lcd_ADCdisplay(uint8_t number);

*/-----------------------------------------------------------------------------------------------------------------------------------
MAIN FUNCTION
-----------------------------------------------------------------------------------------------------------------------------------*/
void main(void)
{
            init_GPIO();
            init_ADC();
            lcd_init();

           while(1)
           {
                      ADC1->CR  |=  ADC_CR_ADSTART;  // start conversion
                      while( (ADC->ISR & ADC_ISR_EOC) == 0 );  // wait until conversion is complete
                      LCD_ADCdisplay(ADC->DR);  // display conversion result on the LCD
           }
}
```

```
*/-------------------------------------------------------------------------------------------------------------------------------
FUNCTIONS DEFINITIONS
-------------------------------------------------------------------------------------------------------------------------------*/
void init_GPIOA(void)
{
        GPIOA->MODER  |= GPIO_MODER_MODER5;  // configure pin PA5 to analog input mode
}

void init_ADC(void)
{
        ADC1->CR &= ~ADC_CR_ADEN;  // ensure ADC is disabled before calibration
        ADC1->CR |= ADC_CR_ADCAL;  // initiate ADC calibration

        RCC->APB2ENR  |=  RCC_APB2ENR_ADCEN;  // connect ADC to APB clock
        ADC1->CR  |=  ADC_CR_ADEN;  // enable ADC
        ADC1->CHSELR  |=  ADC_CHSELR_CHSEL5;  // select ADC channel 5 (for ADC_IN5 = PA5)
        ADC1->CFGR1  &= ~ADC_CFGR1_CONT;  // set to single conversion mode
        ADC1->CFGR1  |= 0b0000;  // configure ADC to res of 12 bits
        while ( (ADC1->ISR & ADC_ISR_ADRDY) == 0 ){ }  // exit when ADC has stabilized
}

void lcd_ADCdisplay(uint8_t number)
{
        /* for 12 bit res, we expect a decimal number of up to four digits */
        uint8_t first_digit = number/1000;
        uint8_t second_digit = (number – first_digit*1000 )/100;
        uint8_t third_digit = (number – second_digit*100 – first_digit*1000)/10;
        uint8_t fourth_digit = number – third_digit*10 - second_digit*100 – first_digit*1000;

        lcd_command(LCD_CURSOR_HOME);  // reset cursor to first position
        lcd_num( 48 + first_digit );
        lcd_num( 48 + second_digit );
        lcd_num( 48 + third_digit );
        lcd_num( 48 + fourth_digit );
}
```