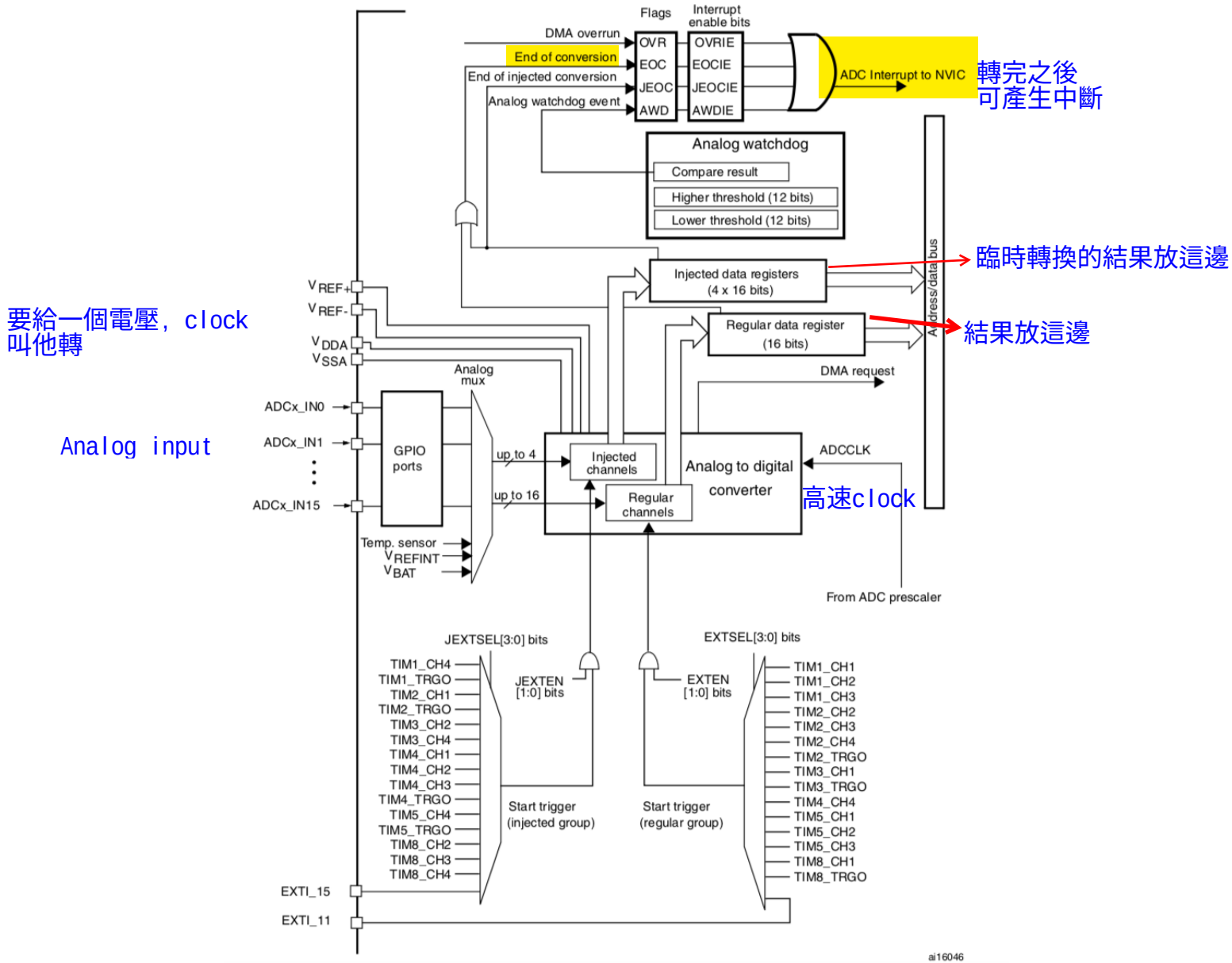
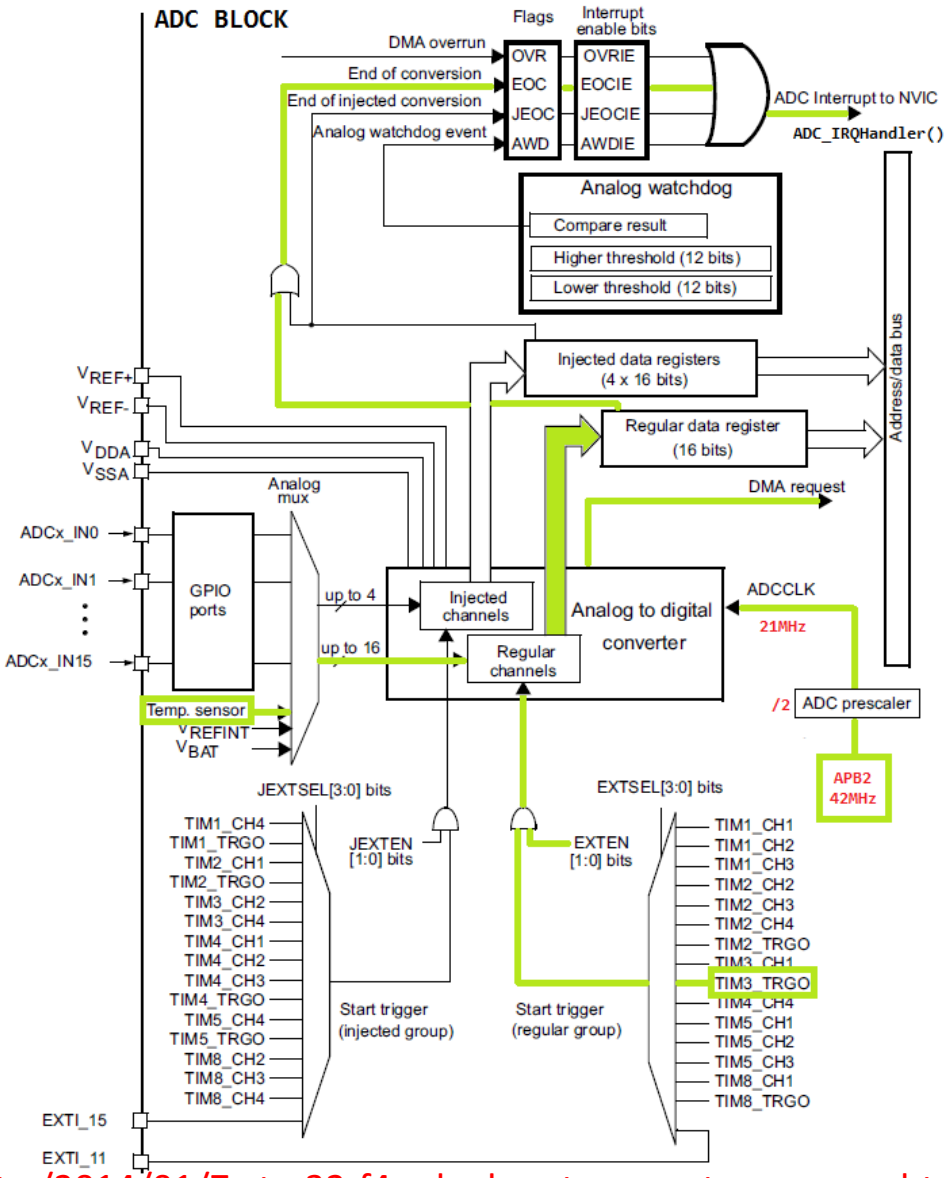
Analog to Digital Converter (ADC)


Flags
Interrupt enable bits
DMA overrun
End of conversion
End of injected conversion
Analog watchdog event
OVR
EOC
JEOC
AWD
OVRIE
EOCIE
JEOCIE
AWDIE
ADC Interrupt to NVIC
轉完之後
可產生中斷
Analog watchdog
Compare result
Higher threshold (12 bits)
Lower threshold (12 bits)
Address/data bus
臨時轉換的結果放這邊
Injected data registers
(4 x 16 bits)
Regular data register
(16 bits)
結果放這邊
DMA request
VREF+
VREF-
VDDA
VSSA
Analog mux
要給一個電壓, clock
叫他轉
ADCx_IN0
ADCx_IN1
ADCx_IN15
Analog input
GPIO ports
up to 4
up to 16
Injected channels
Regular channels
Analog to digital converter
ADCCLK
高速clock
Temp. sensor
VREFINT
VBAT
From ADC prescaler
JEXTSEL[3:0] bits
EXTSEL[3:0] bits
TIM1_CH4
TIM1_TRGO
TIM2_CH1
TIM2_TRGO
TIM3_CH2
TIM3_CH4
TIM4_CH1
TIM4_CH2
TIM4_CH3
TIM4_TRGO
TIM5_CH4
TIM5_TRGO
TIM8_CH2
TIM8_CH3
TIM8_CH4
JEXTEN [1:0] bits
EXTEN [1:0] bits
Start trigger (injected group)
Start trigger (regular group)
TIM1_CH1
TIM1_CH2
TIM1_CH3
TIM2_CH2
TIM2_CH3
TIM2_CH4
TIM2_TRGO
TIM3_CH1
TIM3_TRGO
TIM4_CH4
TIM5_CH1
TIM5_CH2
TIM5_CH3
TIM8_CH1
TIM8_TRGO
EXTI_15
EXTI_11
ai16046



This is what hardware is doing in the diagram:

- Temp. sensor is ON
- ADC is ON
- TIM3 triggers at 2kHz
- ADC converts the voltage and stores it in the regular data register
- ADC makes a DMA request
- DMA moves data to memory
- ADC raises EOC flag which generates an ADC interrupt
- ADC IRQ handler is called at 2kHz toggling a pin at 1kHz (indirectly testing my calculations)

Software will do the following:

- Calibrating the sensor 校正
- Averaging N ADC samples (VSENSE is the average value)
- Calculating the absolute temperature
- Calculating temperature variations

$$\text{Temp (in °C)} = \frac{(\text{VSENSE} - \text{V25})}{\text{Avg\_Slope}} + 25$$

(The rest is just for fun)

- Setting some threshold temperatures
- If the measured temp. is below/above the threshold values, some LEDs light up :)

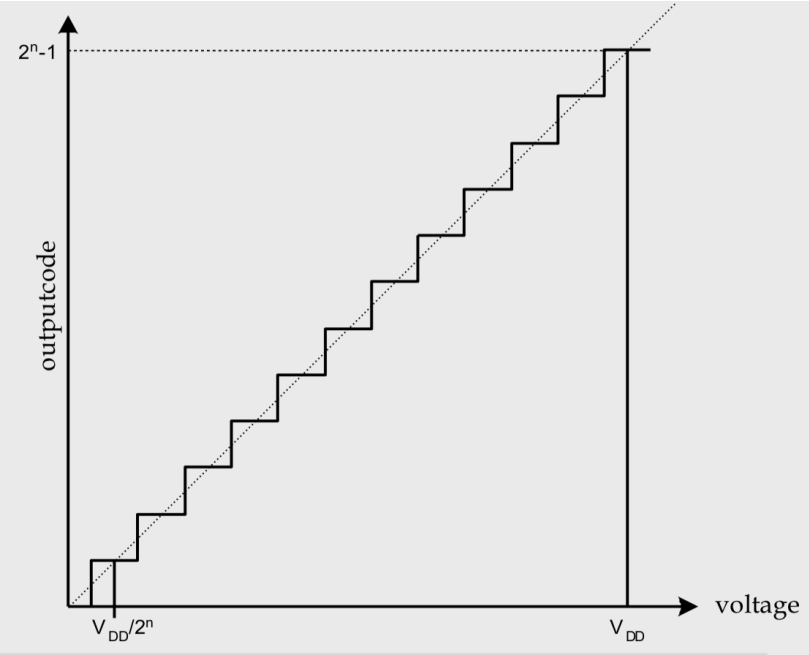
http://00xnor.blogspot.tw/2014/01/7-stm32-f4-adc-dma-temperature-sensor.html

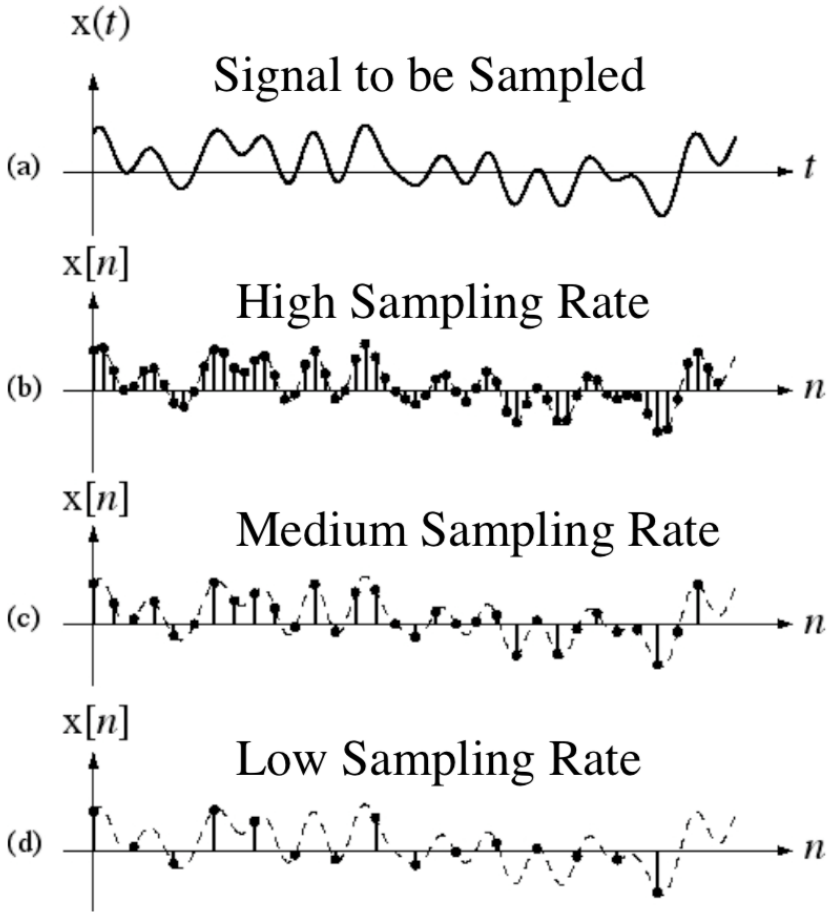
STM32L476

- Up to 3x ADCs, out of which two of them can operate in dual mode
- 12, 10, 8 or 6-bit configurable resolution 精度可調
- Channel-wise programmable sampling time 取樣頻率
- Start-of-conversion can be initiated:
  - by software
  - by hardware triggers with configurable polarity
- Conversion modes 單通道，多通道
  - Each ADC can convert a single channel or can scan a sequence of channels
  - Single mode converts selected inputs once per trigger
  - Continuous mode converts selected inputs continuously

Different Sample Rates vs. Resolutions

resolutions，用幾個bit紀錄



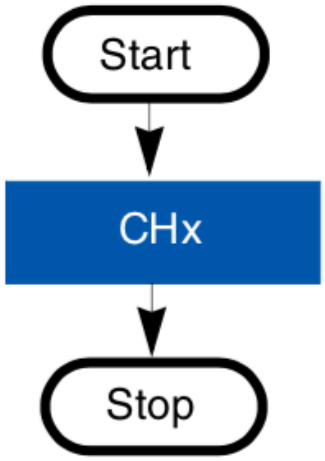


Source: http://control.ucsd.edu/mauricio/courses/mae143a/lectures/8sampling.pdf

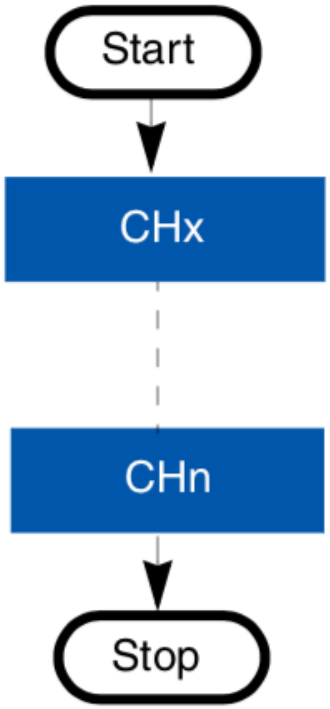
Conversion modes

可有不同的取樣頻率

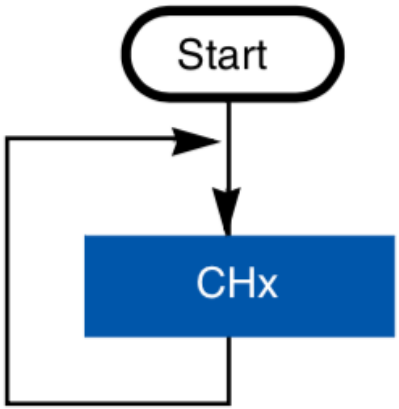
單次性，單通道



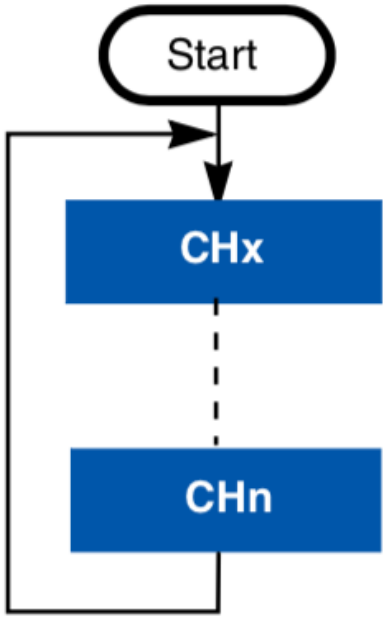
單次性，多通道



多次性，單通道



多次性，多通道



ADCx_ISR	ADC Interrupt and Status Register
ADCx_IER	ADC Interrupt Enable Register
ADCx_CR	ADC Control Register
ADCx_CFGR	ADC ConFiGuration Register
ADCx_SMPR1..2	ADC SaMPle time Register 1..2
ADCx_SQR1..4	ADC Regular SeQuence Register 1..4
ADCx_DR	ADC Regular Data Register
ADCx_CALFACT	ADC CALibration FACTors
ADCx_DIFSEL	ADC DIFferential Mode SELection Register 2

Sampling time setting

設定哪些頻道ADC要看

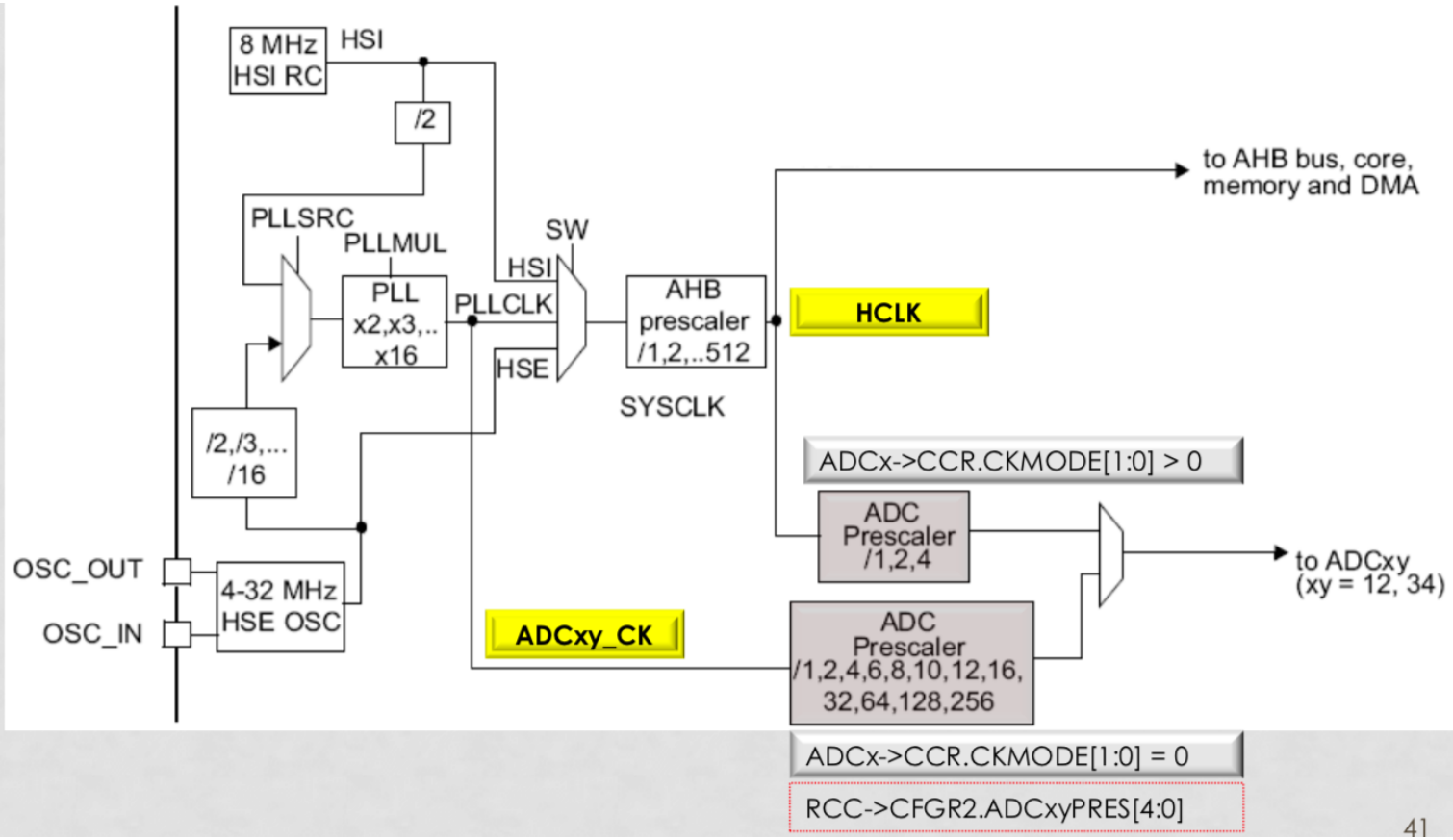
結果

校正參數

ADCx_CR reset value: 0x2000 0000
ADVREGEN[1:0] ='10': ADC Voltage regulator disabled

Register	Bits	Name	Function
ADCx_ISR	3	EOS	End of regular sequence flag
	2	EOC	End of conversion flag
	0	ADRDY	ADC ready
ADCx_CR	31	ADCAL	Start ADC calibration
	30	ADCALDIF	Differential mode for calibration
	29:28	ADVREGEN[1:0]	ADC voltage regulator enable (Reset value: '01' : disabled)
	2	ADSTART	ADC start of regular conversion
	1	ADDIS	ADC disable command
	0	ADEN	ADC enable control
ADCx_CFGR	13	CONT	Single / continuous conversion mode for regular conversions
	5	ALIGN	Right/Left data alignment
	4:3	RES[1:0]	Data resolution (0:12, 1:10, 2:8, 3:6)
ADCx_SMPR1:2	29:0	SMPx[2:0]	Channel x sampling time selection (ADC clock cycles: 0:1.5; 1:2.5, 2:4.5, 3:7.5, 4:19.5; 5:61.5; 6:181.5; 7:601.5)
ADCx_SQR1:4		SQx[4:0]	x conversion in regular sequence (channel to be converted)
		L[3:0]	Regular channel sequence length (0: 1 conversions, 1: 2 conversions,…)
ADCx_DR	15:0	RDATA[15:0	Regular Data converted
ADCx_CALFACT	6:0	CALFACT_S[6:0]	Calibration factor

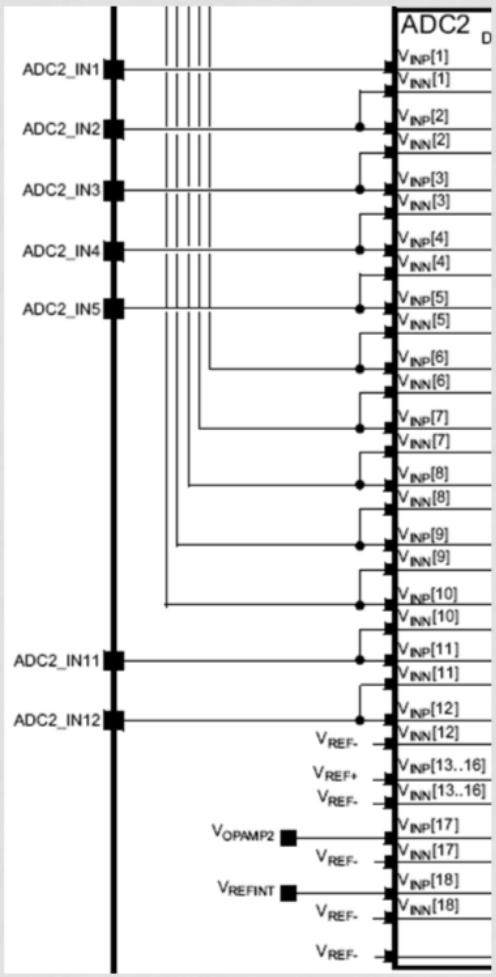
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**ADC_SR**	Reserved																										OVR	STRT	JSTRT	JEOC	EOC	AWD
	Reset value																											0	0	0	0	0	0
0x04	**ADC_CR1**	Reserved					OVRIE	RES[1:0]		AWDEN	JAWDEN	Reserved						DISC NUM [2:0]			JDISCEN	DISCEN	JAUTO	AWD SGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					
	Reset value						0	0	0	0	0							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**ADC_CR2**	Reserved	SWSTART	EXTEN[1:0]		EXTSEL [3:0]				Reserved	JSWSTART	JEXTEN[1:0]		JEXTSEL [3:0]				Reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
	Reset value		0	0	0	0	0	0	0		0	0	0	0	0	0	0					0	0		0							0	0
0x0C	**ADC_SMPR1**	Sample time bits SMPx_x																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**ADC_SMPR2**	Sample time bits SMPx_x																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**ADC_JOFR1**	Reserved																				JOFFSET1[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	**ADC_JOFR2**	Reserved																				JOFFSET2[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**ADC_JOFR3**	Reserved																				JOFFSET3[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x20	**ADC_JOFR4**	Reserved																				JOFFSET4[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x24	**ADC_HTR**	Reserved																				HT[11:0]											
	Reset value																					1	1	1	1	1	1	1	1	1	1	1	1
0x28	**ADC_LTR**	Reserved																				LT[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**ADC_SQR1**	Reserved								L[3:0]				Regular channel sequence SQx_x bits																			
	Reset value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x30	**ADC_SQR2**	Reserved	Regular channel sequence SQx_x bits																														
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x34	**ADC_SQR3**	Reserved	Regular channel sequence SQx_x bits																														
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x38	**ADC_JSQR**	Reserved										JL[1:0]		Injected channel sequence JSQx_x bits																			
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	**ADC_JDR1**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x40	**ADC_JDR2**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44	**ADC_JDR3**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x48	**ADC_JDR4**	Reserved																JDATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x4C	**ADC_DR**	Reserved																Regular DATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

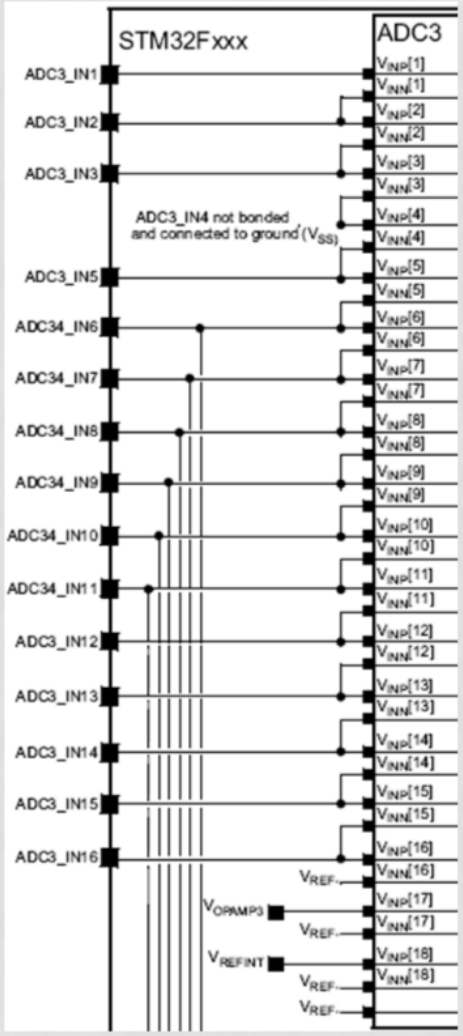

8 MHz
HSI RC
HSI
/2
PLLSRC
PLLMUL
SW
HSI
PLL
x2,x3,..
x16
PLLCLK
HSE
AHB
prescaler
/1,2,..512
SYSCLK
to AHB bus, core,
memory and DMA
HCLK
/2,/3,...
/16
ADCx->CCR.CKMODE[1:0] > 0
ADC
Prescaler
/1,2,4
to ADCxy
(xy = 12, 34)
OSC_OUT
OSC_IN
4-32 MHz
HSE OSC
ADCxy_CK
ADC
Prescaler
/1,2,4,6,8,10,12,16,
32,64,128,256
ADCx->CCR.CKMODE[1:0] = 0
RCC->CFGR2.ADCxyPRES[4:0]

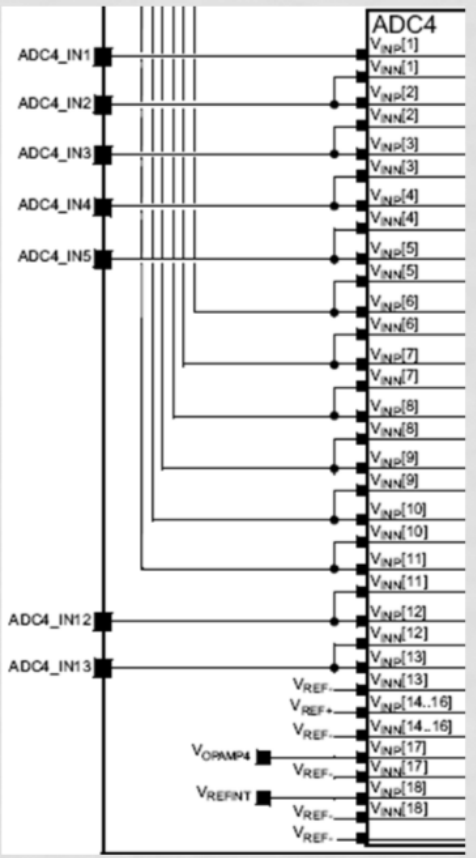
Channel Selection (SQRx)

16個來自GPIO,要把GPIO設成analog

- There are up to 19 multiplexed channels per ADC:
- Before any conversion of an input channel coming from GPIO pads, it is necessary to configure the corresponding GPIOx_ASCR register in the GPIO, in addition to the I/O configuration in analog mode.
- The regular channels and their order in the conversion sequence must be selected in the ADCx_SQR registers. The total number of conversions in the regular group must be written in the L[3:0] bits in the ADCx_SQR1 register.


ADC2
ADC2_IN1
ADC2_IN2
ADC2_IN3
ADC2_IN4
ADC2_IN5
ADC2_IN11
ADC2_IN12
VINP[1]
VINN[1]
VINP[2]
VINN[2]
VINP[3]
VINN[3]
VINP[4]
VINN[4]
VINP[5]
VINN[5]
VINP[6]
VINN[6]
VINP[7]
VINN[7]
VINP[8]
VINN[8]
VINP[9]
VINN[9]
VINP[10]
VINN[10]
VINP[11]
VINN[11]
VINP[12]
VINN[12]
VREF-
VINP[13..16]
VREF+
VINN[13..16]
VREF-
VOPAMP2
VINP[17]
VREF-
VINN[17]
VREFINT
VINP[18]
VREF-
VINN[18]
VREF-


STM32Fxxx
ADC3
ADC3_IN1
ADC3_IN2
ADC3_IN3
ADC3_IN4 not bonded and connected to ground (VSS)
ADC3_IN5
ADC34_IN6
ADC34_IN7
ADC34_IN8
ADC34_IN9
ADC34_IN10
ADC34_IN11
ADC3_IN12
ADC3_IN13
ADC3_IN14
ADC3_IN15
ADC3_IN16
VINP[1]
VINN[1]
VINP[2]
VINN[2]
VINP[3]
VINN[3]
VINP[4]
VINN[4]
VINP[5]
VINN[5]
VINP[6]
VINN[6]
VINP[7]
VINN[7]
VINP[8]
VINN[8]
VINP[9]
VINN[9]
VINP[10]
VINN[10]
VINP[11]
VINN[11]
VINP[12]
VINN[12]
VINP[13]
VINN[13]
VINP[14]
VINN[14]
VINP[15]
VINN[15]
VINP[16]
VINN[16]
VREF-
VINP[17]
VOPAMP3
VREF-
VINN[17]
VINP[18]
VREFINT
VREF-
VINN[18]
VREF-


ADC4
ADC4_IN1
ADC4_IN2
ADC4_IN3
ADC4_IN4
ADC4_IN5
ADC4_IN12
ADC4_IN13
VINP[1]
VINN[1]
VINP[2]
VINN[2]
VINP[3]
VINN[3]
VINP[4]
VINN[4]
VINP[5]
VINN[5]
VINP[6]
VINN[6]
VINP[7]
VINN[7]
VINP[8]
VINN[8]
VINP[9]
VINN[9]
VINP[10]
VINN[10]
VINP[11]
VINN[11]
VINP[12]
VINN[12]
VINP[13]
VINN[13]
VREF-
VREF+
VINP[14..16]
VREF-
VINN[14..16]
VOPAMP4
VINP[17]
VREF-
VINN[17]
VREFINT
VINP[18]
VREF-
VINN[18]
VREF-

Channel-wise Programmable Sampling Time (SMPR1, SMPR2)

- Before starting a conversion, the ADC must establish a direct connection between the voltage source under measurement and the embedded sampling capacitor of the ADC.
- Each channel can be sampled with a different sampling time which is programmable using the SMP[2:0] bits in the ADCx_SMPR1 and ADCx_SMPR2 registers.
  - SMP = 000: 2.5 ADC clock cycles
  - SMP = 001: 6.5 ADC clock cycles
  - SMP = 010: 12.5 ADC clock cycles
  - SMP = 011: 24.5 ADC clock cycles
  - SMP = 100: 47.5 ADC clock cycles
  - SMP = 101: 92.5 ADC clock cycles
  - SMP = 110: 247.5 ADC clock cycles
  - SMP = 111: 640.5 ADC clock cycles

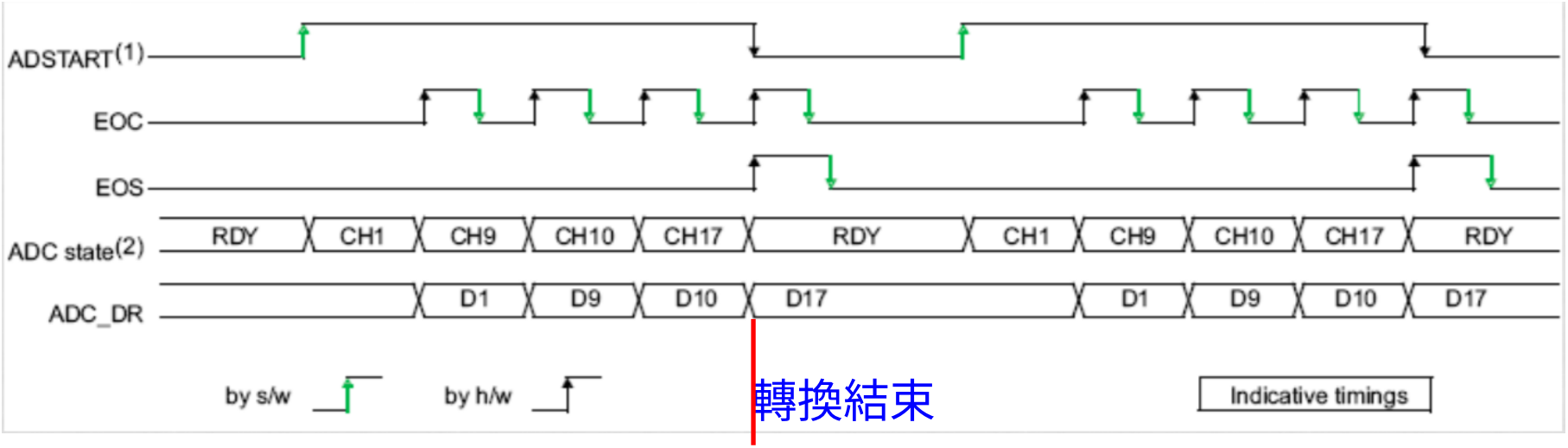
SMP	ADC clock cycles
000	1.5
001	2.5
010	4.5
011	7.5
100	19.5
101	61.5
110	181.5
111	601.5

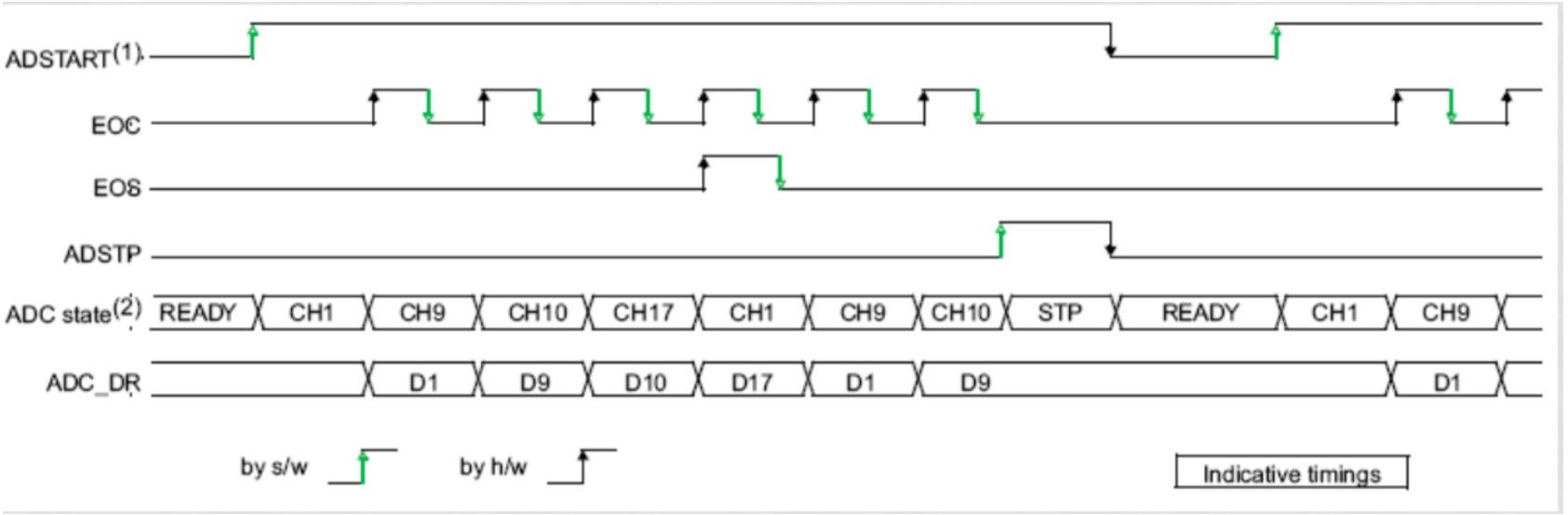
- The total conversion time is calculated as follows (resolution = 12 bits):
  - Tconv = Sampling time + 12.5 ADC clock cycles　　轉換耗時　＋　轉換頻率
- Example:
  - With F_{ADC_CLK} = 72 MHz and a sampling time of 1.5 ADC clock cycles:
    - Tconv = (1.5 + 12.5) ADC clock cycles = 14 ADC clock cycles = 0.194 µs (for fast channels)

Conversion Mode

轉完就停了

- In single conversion mode, the ADC performs once all the conversions of the channels.
- In continuous conversion mode, when a software or hardware regular trigger event occurs, the ADC performs once all the regular conversions of the channels and then automatically re-starts and continuously converts each conversions of the sequence.


ADSTART(1)
EOC
EOS
ADC state(2)
RDY
CH1
CH9
CH10
CH17
RDY
CH1
CH9
CH10
CH17
RDY
ADC_DR
D1
D9
D10
D17
D1
D9
D10
D17
by s/w
by h/w
轉換結束
Indicative timings


ADSTART(1)
EOC
EOS
ADSTP
ADC state(2)
READY
CH1
CH9
CH10
CH17
CH1
CH9
CH10
STP
READY
CH1
CH9
ADC_DR
D1
D9
D10
D17
D1
D9
D1
by s/w
by h/w
Indicative timings

Programmable Resolution

- It is possible to perform faster conversion by reducing the ADC resolution.
- The resolution can be configured to be either 12, 10, 8, or 6 bits by programming the control bits RES[1:0].

End of Conversion

- The ADC sets the EOC flag as soon as a new regular conversion data is available in the ADCx_DR register. An interrupt can be generated if bit EOCIE is set. EOC flag is cleared by the software either by writing 1 to it or by reading ADCx_DR.

可以用pulling或是interrupt的方式檢查

Procedure

- To start ADC operations, it is first needed to exit deep-power-down mode by setting bit DEEPPWD=0.
- It is mandatory to enable the ADC internal voltage regulator by setting the bit ADVREGEN=1 into ADCx_CR register.
- Once DEEPPWD=0 and ADVREGEN=1, the ADC can be enabled
  - Clear the ADRDY bit in the ADC_ISR register by writing ‘1’.
  - Set ADEN=1.
  - Regular conversion can then start by setting ADSTART=1

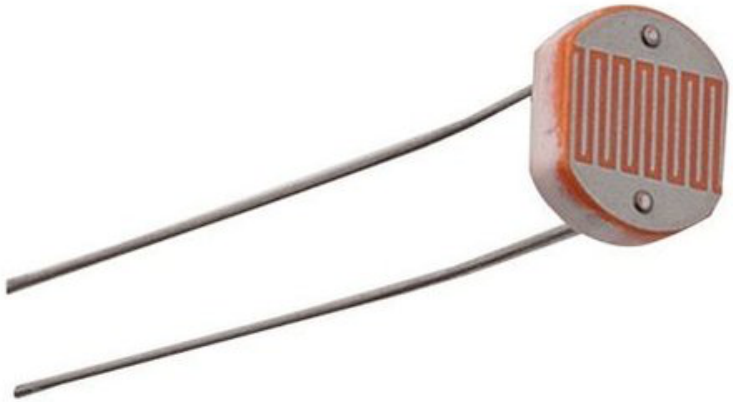
```c
#include "stm32f30x.h"

void Delay (uint32_t nTime);

uint16_t  ADC1ConvertedValue = 0;
uint16_t  ADC1ConvertedVoltage = 0;
uint16_t  calibration_value = 0;
Volatile uint32_t TimingDelay = 0;

int main(void)
{
  // At this stage the microcontroller clock tree is already configured
  RCC->CFGR2 |= RCC_CFGR2_ADCPRE12_DIV2; // Configure the ADC clock
  RCC->AHBENR |= RCC_AHBENR_ADC12EN; // Enable ADC1 clock
  // Setup SysTick Timer for 1 µsec interrupts
  if (SysTick_Config(SystemCoreClock / 1000000))
  {
    // Capture error
    while (1)
    {}
  }
```

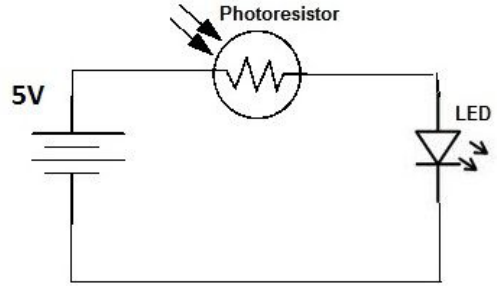
設定clock source

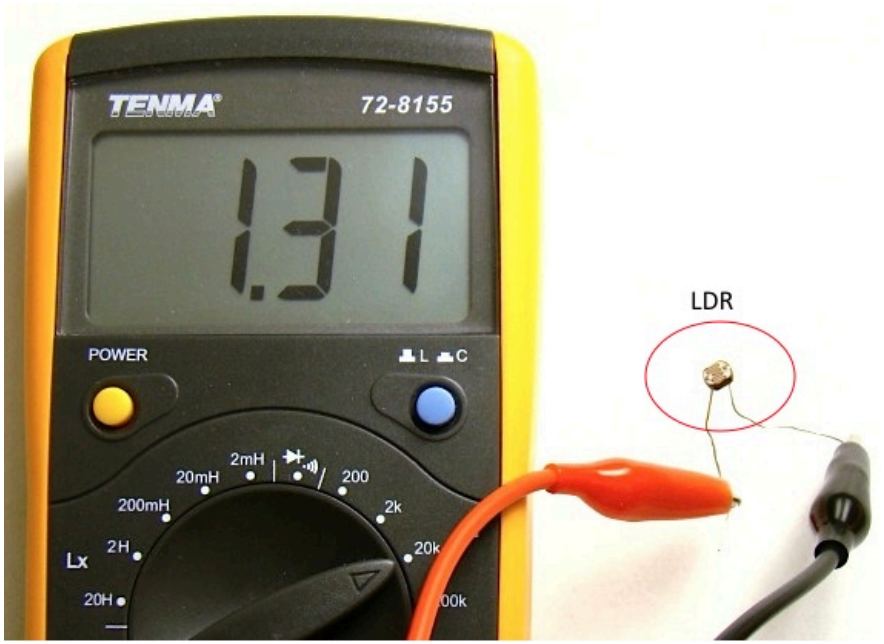
```
// ADC Channel configuration  PC1 in analog mode
RCC->AHBENR |= RCC_AHBENR_GPIOCEN; // GPIOC Periph clock enable
GPIOC->MODER |= 3 << (1*2); // Configure ADC Channel7 as analog input

/* Calibration procedure */
ADC1->CR &= ~ADC_CR_ADVREGEN;
ADC1->CR |= ADC_CR_ADVREGEN_0; // 01: ADC Voltage regulator enabled
Delay(10); // Insert delay equal to 10 µs
ADC1->CR &= ~ADC_CR_ADCALDIF; // calibration in Single-ended inputs Mode.
ADC1->CR |= ADC_CR_ADCAL; // Start ADC calibration
// Read at 1 means that a calibration in progress.
while (ADC1->CR & ADC_CR_ADCAL); // wait until calibration done
calibration_value = ADC1->CALFACT; // Get Calibration Value ADC1
```

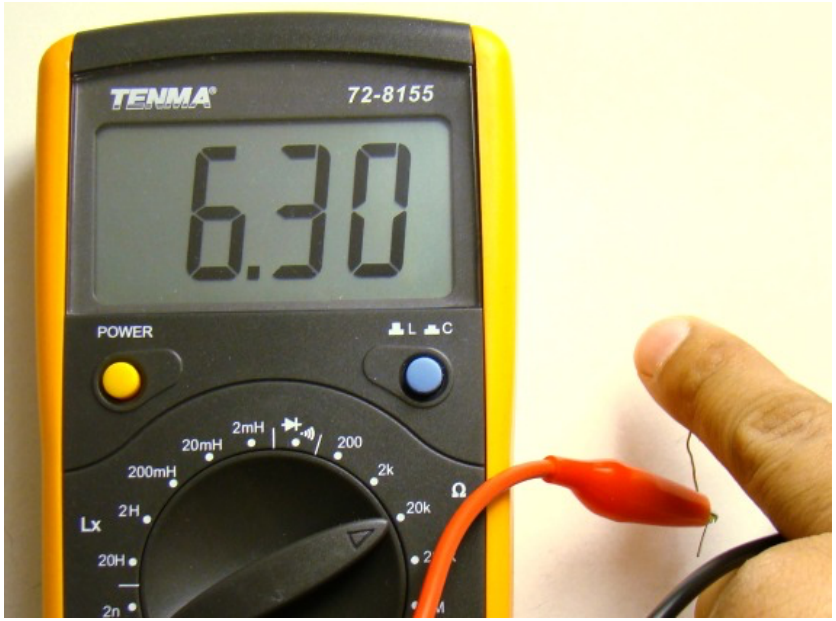
```
  // ADC configuration
  ADC1->CFGR |= ADC_CFGR_CONT;  // ADC_ContinuousConvMode_Enable
  ADC1->CFGR &= ~ADC_CFGR_RES;  // 12-bit data resolution
  ADC1->CFGR &= ~ADC_CFGR_ALIGN;  // Right data alignment

  /* ADC1 regular channel7 configuration */
  ADC1->SQR1 |= ADC_SQR1_SQ1_2 | ADC_SQR1_SQ1_1 | ADC_SQR1_SQ1_0;  // SQ1 = 0x07, start converting ch7
  ADC1->SQR1 &= ~ADC_SQR1_L;  // ADC regular channel sequence length = 0 => 1 conversion/sequence
  ADC1->SMPR1 |= ADC_SMPR1_SMP7_1 | ADC_SMPR1_SMP7_0;  // = 0x03 => sampling time 7.5 ADC clock cycles
  ADC1->CR |= ADC_CR_ADEN;  // Enable ADC1
  while(!ADC1->ISR & ADC_ISR_ADRD);  // wait for ADRDY

  ADC1->CR |= ADC_CR_ADSTART;  // Start ADC1 Software Conversion

  while (1)
  {
    while(!(ADC1->ISR & ADC_ISR_EOC));  // Test EOC flag
    ADC1ConvertedValue = ADC1->DR;  // Get ADC1 converted data
    ADC1ConvertedVoltage = (ADC1ConvertedValue *3300)/4096;  // Compute the voltage
  }
}
```

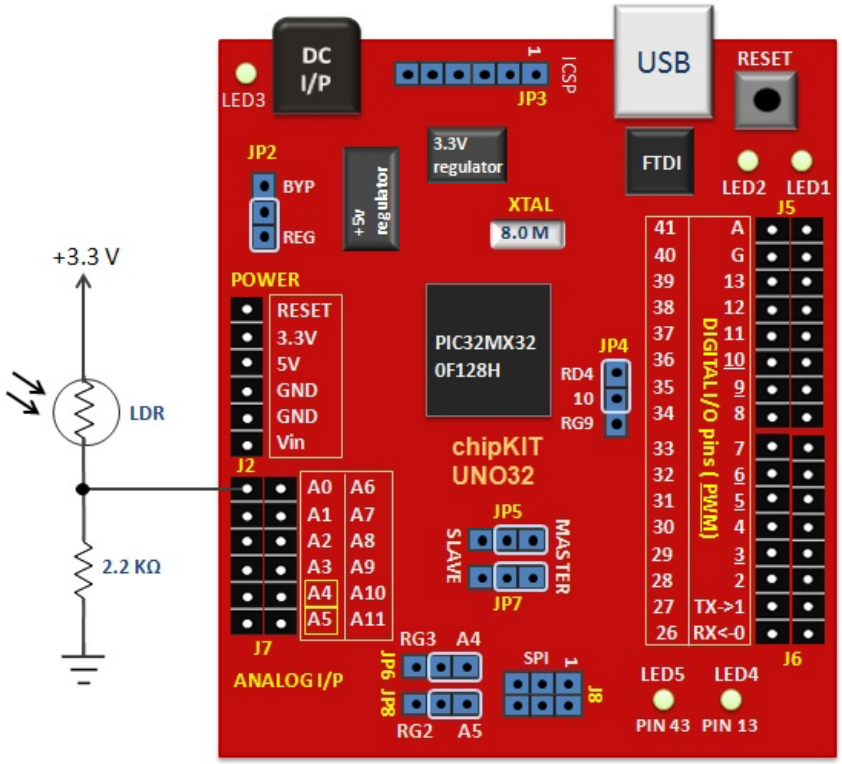
polling




Photoresistor
5V
LED


TENMA
72-8155
1.31
POWER
L C
LDR
2mH
20mH
200
200mH
2k
2H
20k
Lx
20H


TENMA
72-8155
6.30
POWER
L C
2mH
20mH
200
200mH
2k
Ω
2H
20k
Lx
20H


+3.3 V
LDR
2.2 KΩ
DC I/P
LED3
JP3
ICSP
USB
RESET
JP2
BYP
REG
+5v regulator
3.3V regulator
FTDI
LED2
LED1
XTAL
8.0 M
J5
POWER
RESET
3.3V
5V
GND
GND
Vin
PIC32MX32 0F128H
JP4
RD4
10
RG9
chipKIT UNO32
DIGITAL I/O pins (PWM)
41 A
40 G
39 13
38 12
37 11
36 10
35 9
34 8
33 7
32 6
31 5
30 4
29 3
28 2
27 TX->1
26 RX<-0
J6
J2
A0 A6
A1 A7
A2 A8
A3 A9
A4 A10
A5 A11
J7
ANALOG I/P
JP5
SLAVE
MASTER
JP7
RG3 A4
JP6
JP8
RG2 A5
SPI
J8
LED5
LED4
PIN 43
PIN 13

```
int main()
{

  TIM3_Config();
  ADC_Config();

  while (1)
  {
    R = Get_R();
    ...
  }

}
```

```
void TIM3_Config(void)
{
  TIM_TimeBaseInitTypeDef TIM3_TimeBase;


  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

  TIM_TimeBaseStructInit(&TIM3_TimeBase);
  TIM3_TimeBase.TIM_Period        = (uint16_t)49; // Trigger = CK_CNT/(49+1) = 2kHz
  TIM3_TimeBase.TIM_Prescaler     = 420;          // CK_CNT = 42MHz/420 = 100kHz
  TIM3_TimeBase.TIM_ClockDivision = 0;
  TIM3_TimeBase.TIM_CounterMode   = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM3, &TIM3_TimeBase);
  TIM_SelectOutputTrigger(TIM3, TIM_TRGOSource_Update);


  TIM_Cmd(TIM3, ENABLE);
}
```

```
void ADC_Config(void)
{
  ADC_InitTypeDef       ADC_INIT;
  ADC_CommonInitTypeDef ADC_COMMON;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  ADC_COMMON.ADC_Mode             = ADC_Mode_Independent;
  ADC_COMMON.ADC_Prescaler        = ADC_Prescaler_Div2;
  ADC_COMMON.ADC_DMAAccessMode    = ADC_DMAAccessMode_Disabled;
  ADC_COMMON.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_COMMON);

  ADC_INIT.ADC_Resolution           = ADC_Resolution_12b;
  ADC_INIT.ADC_ScanConvMode         = DISABLE;
  ADC_INIT.ADC_ContinuousConvMode   = DISABLE; // ENABLE for max ADC sampling frequency
  ADC_INIT.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising;
  ADC_INIT.ADC_ExternalTrigConv     = ADC_ExternalTrigConv_T3_TRGO;
  ADC_INIT.ADC_DataAlign            = ADC_DataAlign_Right;
  ADC_INIT.ADC_NbrOfConversion      = 1;
  ADC_Init(ADC1, &ADC_INIT);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_3Cycles);
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
  ADC_Cmd(ADC1, ENABLE);
}
```

```
float Get_R(void)
{
  uint8_t i;

  for(i = 0; i < NS; i++)
  {
    Sample_ADC_Raw[i] = ADC_Raw[i];
  }

  return R;
}
```

```
   /**Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION12b;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = EOC_SINGLE_CONV;
  HAL_ADC_Init(&hadc1);
```

```
    /**Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
  //sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);
```

```
void ConfigureADC() {
__HAL_RCC_ADC_CLK_ENABLE();
// Clear Deep Sleep
CLEAR_BIT(ADC1->CR, ADC_CR_DEEPPWD);
// Turn on Voltage Regulator
SET_BIT(ADC1->CR, ADC_CR_ADVREGEN);
delay_us(200);
// Prescaler
MODIFY_REG(ADC123_COMMON->CCR, ADC_CCR_PRESC|ADC_CCR_CKMODE, ADC_CCR_CKMODE_0);
MODIFY_REG(ADC1->CFGR, ADC_CFGR_FIELDS_1, ADC_CFGR_CONT);
CLEAR_BIT(ADC1->CFGR2, ADC_CFGR2_ROVSE);
CLEAR_BIT(ADC1->SQR1, ADC_SQR1_L);
MODIFY_REG(ADC1->SQR1, (0xF)|(0x1F<<24)|(0x1F<<18)|(0x1F<<12)|(0x1F<<6), (0x1<<6)); // Channel 1, Rank 1
MODIFY_REG(ADC1->SMPR1, (0x3FFFFFFFFF), (0x6<<3)); // Channel 1, Sampling Time: 247.5 ADC cycles
}

void startADC() {
while (!(ADC1->ISR & ADC_ISR_ADRDY)) ADC1->CR |= ADC_CR_ADEN;  // TURN ON
delay_us(5000);
ADC1->ISR = ADC_ISR_EOC | ADC_ISR_EOS | ADC_ISR_OVR; // Clear flags
SET_BIT(ADC1->CR, ADC_CR_ADSTART); // START CONV
}

void light() {
startADC();
while (!(ADC1->ISR & ADC_ISR_EOC));
adcVal = ADC1->DR;}
```

References

- RM0351 Reference manual, STM32L4x6 advanced ARM®-based 32-bit MCUs
- STM32L476xx, Datasheet