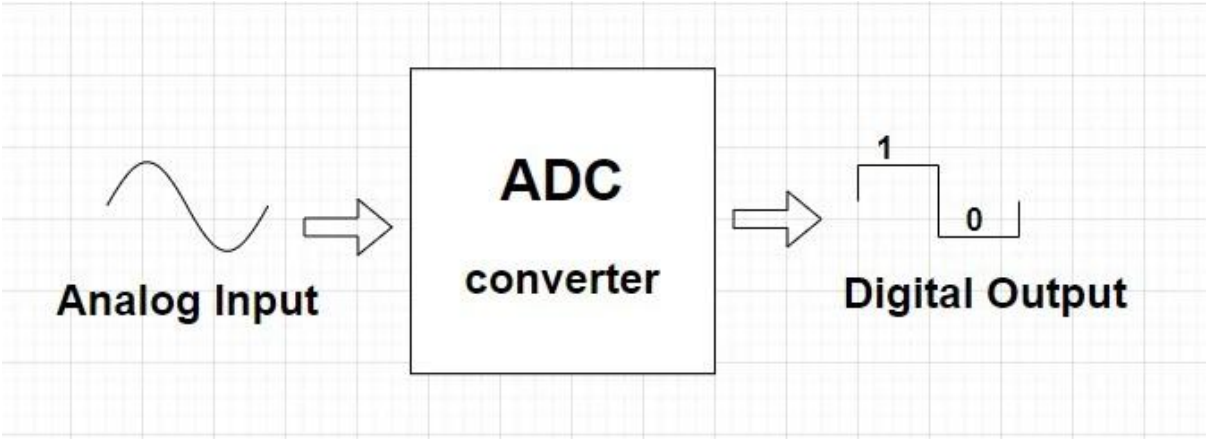
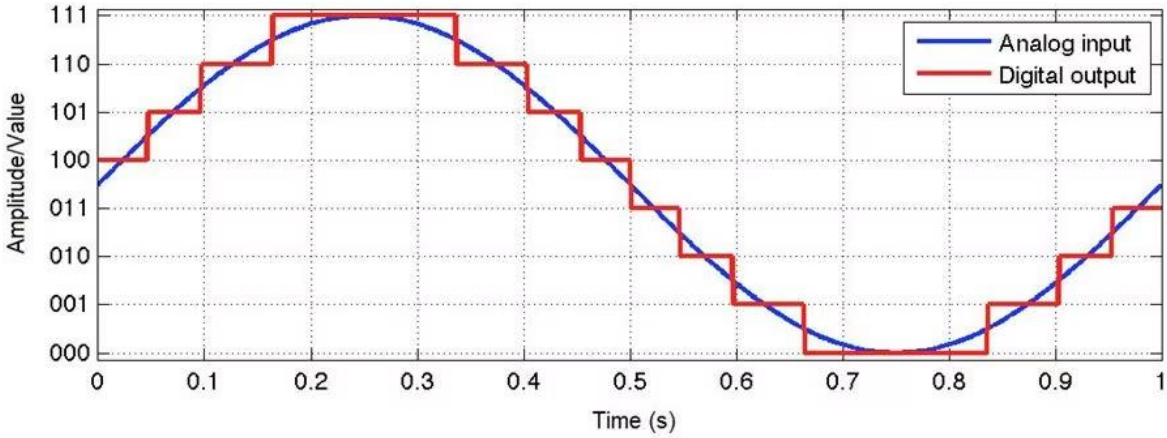
Bài 9: Analog Digital Converter (ADC)

Sơ lược về ADC (Bộ chuyển đổi tín hiệu tương tự sang số)

ADC (Analog-to-Digital Converter) là 1 mạch điện tử lấy điện áp tương tự là m đầu vào và chuyển đổi nó thành dữ liệu số (1 giá trị đại diện cho mức điện áp trong mã nhị phân).



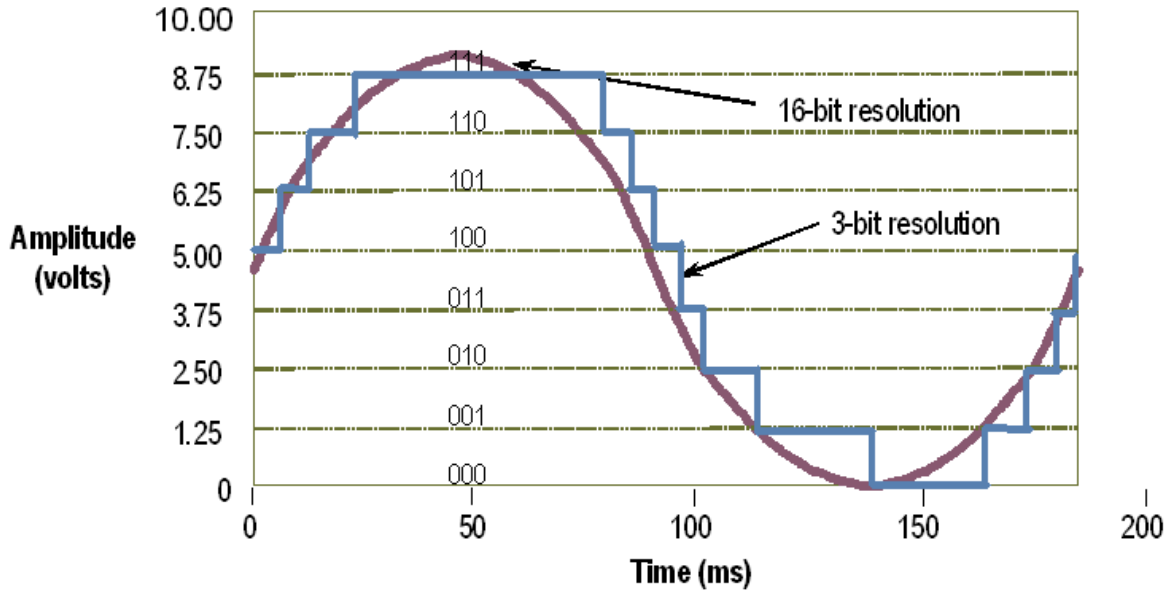
Về cơ bản, ADC hoạt động theo cách chia mức tín hiệu tương tự thành nhiều mức khác nhau. Các mức được biểu diễn bằng các bit nhị phân.



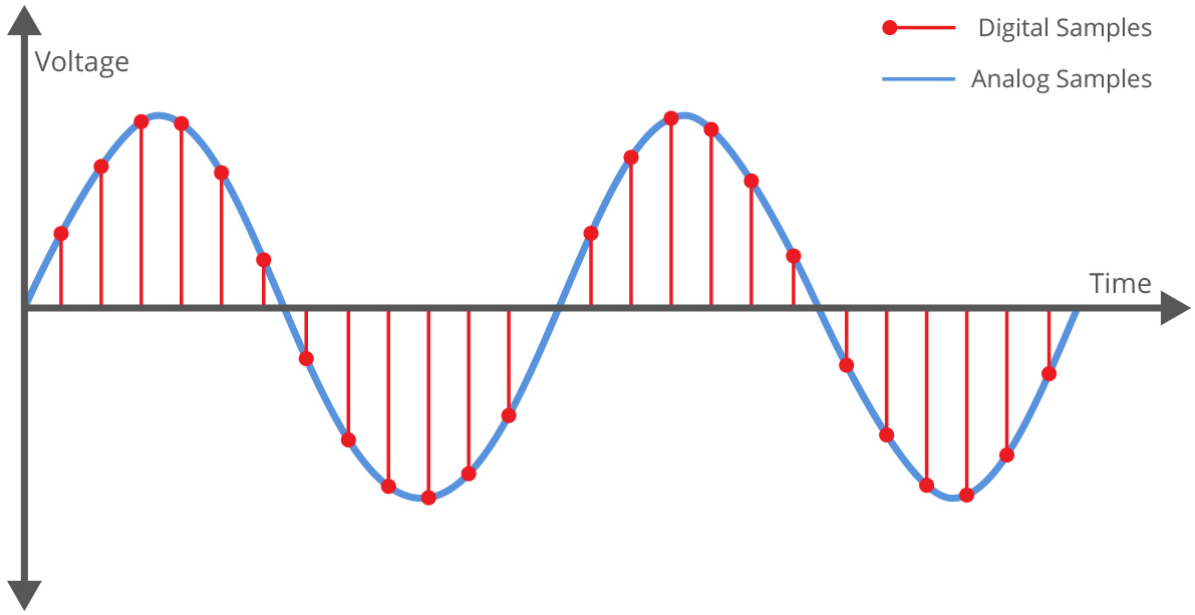
Bằng việc so sánh giá trị điện áp mỗi lần lấy mẫu với 1 mức nhất định, ADC chuyển đổi tín hiệu tương tự và mã hóa các giá trị về giá trị nhị phân tương ứng.

Các khái niệm cần biết:

Độ phân giải (resolution): dùng để chỉ số bit cần thiết để chứa hết các mức giá trị số (digital) sau quá trình chuyển đổi ở ngõ ra. Bộ chuyển đổi ADC của STM32F103C8T6 có độ phân giải mặc định là 12 bit, tức là có thể chuyển đổi ra 2^12= 4096 giá trị ở ngõ ra số.



Tần số lấy mẫu: là khái niệm được dùng để chỉ tốc độ lấy mẫu và số hóa của bộ chuyển đổi, thời gian giữa 2 lần số hóa càng ngắn độ chính xác càng cao. Khả năng tái tạo lại tín hiệu càng chính xác. Đó gọi là chu kỳ lấy mẫu. Được tính bằng : thời gian lấy mẫu tín hiệu+ thời gian chuyển đổi.

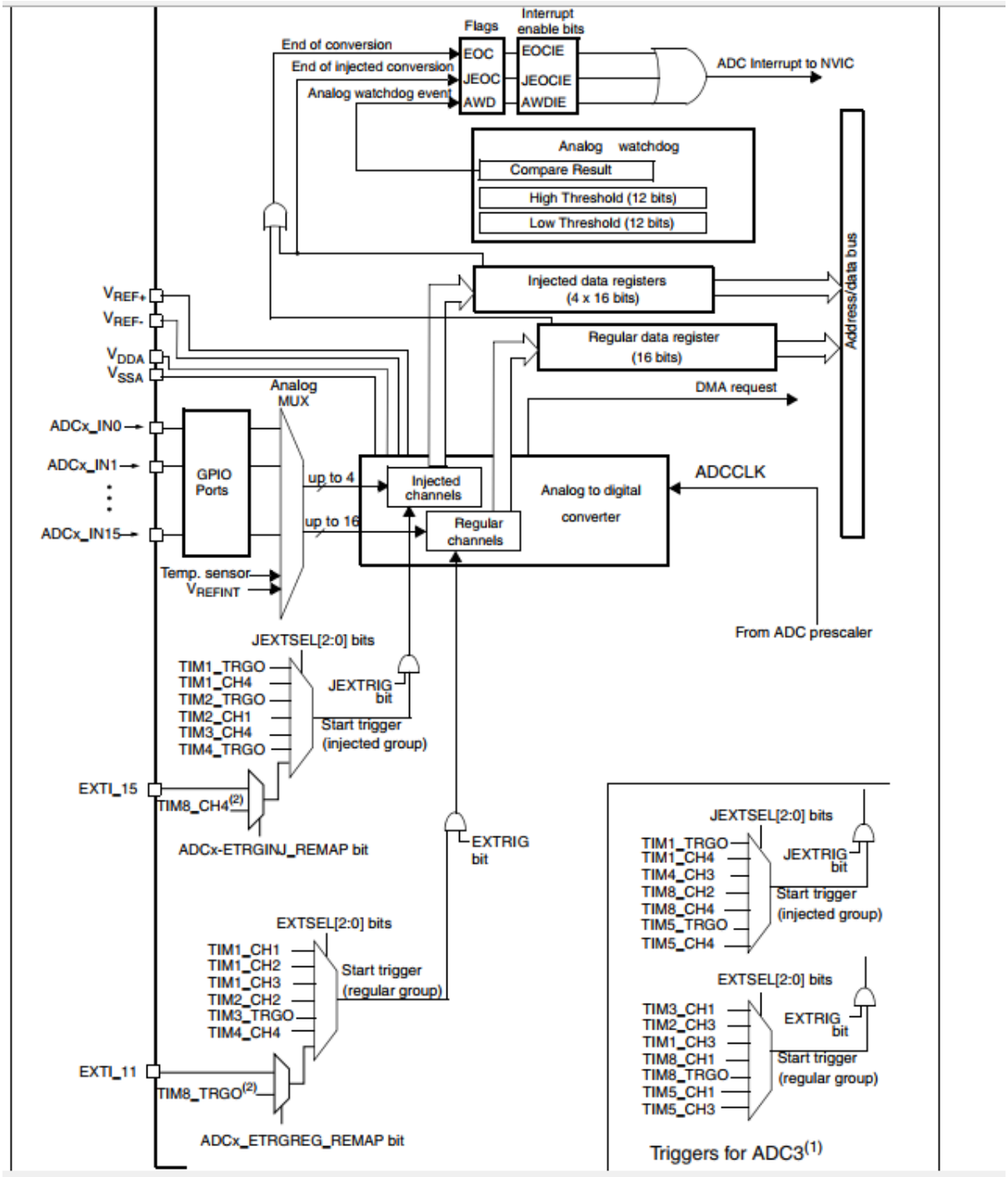


ADC trong STM32.

Dưới đây là khối ADC của Stm32f103. STM32F1 hỗ trợ 3 bộ ADC với nhiều kênh với các chế độ.

- Nguồn tín hiệu tương tự lấy thông qua cổng gpio port
- Bộ chuyển đổi tín hiệu tương tự sang số gồm 2 chế độ chuyển đổi chính Regular Channel (16 kênh) và Injected Channel (4 kênh)
- Mỗi Mode thì sử dụng một thanh ghi riêng 16 bit (Injected data register và Regular data register)

- Dữ liệu sau khi được chuyển đổi sẽ tạm ghi vào các thanh ghi này và sau đó đọc từ thanh ghi này ra.
- Xét về độ ưu tiên thì chế độ Injected có mức ưu tiên cao hơn Regular channel. Nghĩa là khi chế độ injected được kích hoạt thì sẽ tạm ừng chế độ regular để thực hiện injected trước.



Giá trị điện áp đầu vào bộ ADC được cung cấp trên chân VDDA và thường lấy bằng giá trị cấp nguồn cho vi điều khiển VDD(+3V3).

STM32F103C8 có 2 kênh ADC đó là ADC1 và ADC2, mỗi kênh có tối đa là 9 channel với nhiều mode hoạt động như: single, continuous,scan hoặc discontinuous. Kết quả chuyển đổi được lưu trữ trong thanh ghi 16 bit.

- Độ phân giải 12 bit tương ứng với giá trị maximum là 4095.
- Có các ngắt hỗ trợ.
- Single mode hay Continuous mode.
- Tự động calib và có thể điều khiển hoạt động ADC bằng xung Trigger.
- Thời gian chuyển đổi nhanh : 1us tại tần số 65Mhz.
- Điện áp cung cấp cho bộ ADC là 2.4V -> 3.6V. Nên điện áp Input của thiết bị có ADC 2.4V ≤ VIN ≤ 3.6V.
- Có bộ DMA giúp tăng tốc độ xử lí.

Giả sử ta cần đo điện áp tối thiểu là 0V và tối đa là 3.3V, trong STM32 sẽ chia 0 → 3.3V thành 4096 khoảng giá trị (từ 0 → 4095, do 2^12 = 4096), giá trị đo được từ chân IO tương ứng với 0V sẽ là 0, tương ứng với 1.65V là 2047 và tương ứng 3.3V sẽ là 4095.

Các Thanh ghi ADC
ADC_SQR2 - Thanh ghi lưu thứ tự kênh ADC (Regular Sequence Register 2)
Dùng để chỉ định thứ tự ưu tiên các kênh ADC (từ 0 đến 16, tùy vi điều khiển) sẽ được chuyển đổi AD (Analog to Digital) trong chế độ quét (Scan Mode)

ADC_JSQR - Thanh ghi thứ tự kênh ADC injected- các kênh này được kích hoạt hoạt động từ ngoài hoặc bằng software (Injected Sequence Register)
Cấu hình tối đa 4 kênh Injected để chuyển đổi AD, bao gồm thứ tự và số lượng kênh.
ADC_JDR1 - Thanh ghi dữ liệu các kênh Injected(Injected Data Register 1)
Sau khi chuyển đổi AD hoàn tất, bạn đọc ADC_JDR1 để lấy giá trị số hóa của tín hiệu Analog
ADC_DR - Thanh ghi dữ liệu thường (Regular Data Register)
Lưu kết quả sau mỗi lần chuyển đổi AD thường. Trong chế độ quét nhiều kênh, dữ liệu được ghi tuần tự vào ADC_DR và có thể dùng DMA để chuyển vào bộ nhớ.

Table 72. ADC register map and reset values (continued)

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
0x30	ADC_SQR2	Reserved		SQ12[4:0] 12th conversion in regular sequence bits					SQ11[4:0] 11th conversion in regular sequence bits					SQ10[4:0] 10th conversion in regular sequence bits					SQ9[4:0] 9th conversion in regular sequence bits					SQ8[4:0] 8th conversion in regular sequence bits					SQ7[4:0] 7th conversion in regular sequence 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		SQ6[4:0] 6th conversion in regular sequence bits					SQ5[4:0] 5th conversion in regular sequence bits					SQ4[4:0] 4th conversion in regular sequence bits					SQ3[4:0] 3rd conversion in regular sequence bits					SQ2[4:0] 2nd conversion in regular sequence bits					SQ1[4:0] 1st conversion in regular sequence 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]		JSQ4[4:0] 4th conversion in injected sequence bits					JSQ3[4:0] 3rd conversion in injected sequence bits					JSQ2[4:0] 2nd conversion in injected sequence bits					JSQ1[4:0] 1st conversion in injected sequence 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
0x40	ADC_JDR2	Reserved																	JDATA[15:0]														
	Reset value																		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
0x48	ADC_JDR4	Reserved																	JDATA[15:0]														
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x4C	ADC_DR	ADC2DATA[15:0]																	Regular DATA[15:0]														
	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

Xác định địa chỉ và con trỏ

- **Địa chỉ cơ sở của ADC**: Ví dụ, với ADC1 trên STM32, địa chỉ cơ sở thường là 0x40012000. Địa chỉ này có thể thay đổi tùy vi điều khiển, hãy kiểm tra datasheet hoặc reference manual.
- **Offset của từng thanh ghi**: Mỗi thanh ghi có một offset so với địa chỉ cơ sở. Ví dụ (giá trị giả định, cần xác minh từ tài liệu):
    - ADC_SQR2: offset 0x34
    - ADC_JSQR: offset 0x38
    - ADC_JDR1: offset 0x3C
    - ADC_DR: offset 0x4C

Sau đó, định nghĩa các con trỏ kiểu volatile uint32_t* để truy cập các thanh ghi này (vì chúng là các thanh ghi 32-bit và cần volatile để đảm bảo trình biên dịch không tối ưu hóa sai).

Cấu hình ADC cho MCU STM32F103.

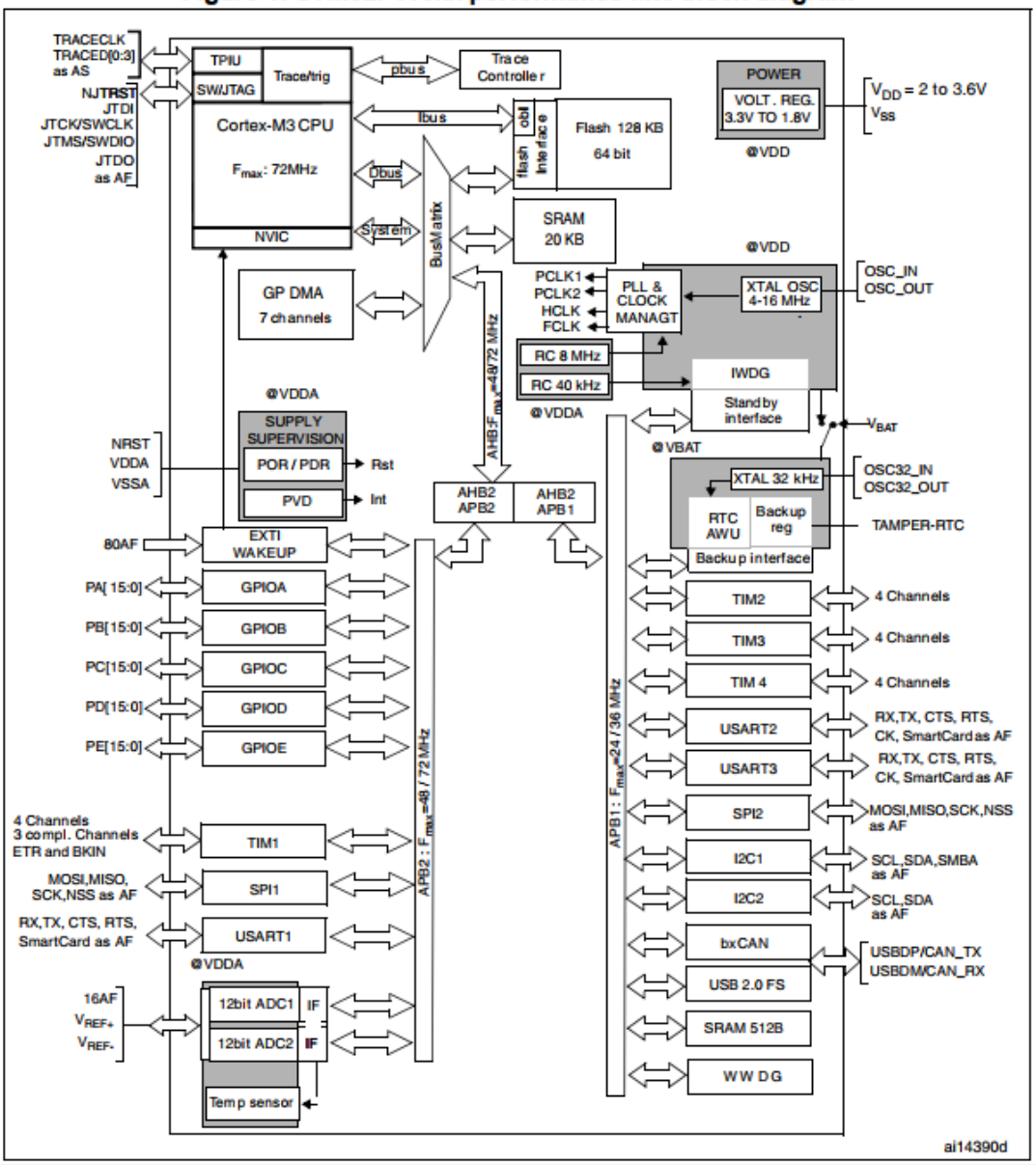
Cấu hình GPIO.

I/O	-	PA0	WKUP/ USART2_CTS(9)/ ADC12_IN0/ TIM2_CH1_ ETR(9)	-
I/O	-	PA1	USART2_RTS(9)/ ADC12_IN1/ TIM2_CH2(9)	-
I/O	-	PA2	USART2_TX(9)/ ADC12_IN2/ TIM2_CH3(9)	-
I/O	-	PA3	USART2_RX(9)/ ADC12_IN3/ TIM2_CH4(9)	-
S	-	V_{SS_4}	-	-
S	-	V_{DD_4}	-	-
I/O	-	PA4	SPI1_NSS(9)/ USART2_CK(9)/ ADC12_IN4	-
I/O	-	PA5	SPI1_SCK(9)/ ADC12_IN5	-
I/O	-	PA6	SPI1_MISO(9)/ ADC12_IN6/ TIM3_CH1(9)	TIM1_BKIN
I/O	-	PA7	SPI1_MOSI(9)/ ADC12_IN7/ TIM3_CH2(9)	TIM1_CH1N
I/O	-	PC4	ADC12_IN14	-
I/O	-	PC5	ADC12_IN15	-

ADC hỗ trợ rất nhiều kênh(ngõ vào), được cấu hình sẵn ở các chân GPIO của các Port và từ các chân khác.
Nếu sử dụng các kênh ngõ vào từ GPIO. Các chân dùng làm ngõ vào cho ADC sẽ được cấu hình Mode AIN.(Analoge Input).

Đầu tiên, các bộ ADC được cấp xung từ RCC APB2, để bộ ADC hoạt động cần cấp xung cho cả ADC để tạo tần số lấy mẫu tín hiệu và cấp xung cho GPIO của Port ngõ vào.

Figure 1. STM32F103xx performance line block diagram



Cấu hình ADC.

Các tham số cấu hình cho bộ ADC được tổ chức trong Struct ADC_InitTypeDef, Gồm:

- ADC_Mode: Cấu hình chế độ hoạt động cho ADC là đơn kênh(Independent) hay đa kênh, ngoài ra còn có các chế độ ADC chuyển đổi tuần tự các kênh (regularly) hay chuyển đổi khi có kích hoạt từ phần mềm hay các tín hiệu khác (injected).

- ADC_NbrOfChannel: Chọn kênh ADC để cấu hình, có 16 kênh tương ứng với 16 chân IO cấu hình sẵn (xem datasheet) và 2 kênh Vref và TempSensor để quy chiếu điện áp, đo điện áp Pin vv,.
- ADC_ContinuousConvMode: Cấu hình bộ ADC có chuyển đổi liên tục hay không, Enable để cấu hình ADC chuyển đổi lien tục, nếu cấu hình Disable, ta phải gọi lại lệnh đọc ADC để bắt đầu quá trình chuyển đổi.
- ADC_ExternalTrigConv: Enable để sử dụng tín hiệu ngoài để kích hoạt ADC, Disable nếu không sử dụng.
- ADC_ScanConvMode: Cấu hình chế độ quét ADC lần lượt từng kênh. Enable nếu sử dụng chế độ quét này.
- ADC_DataAlign: Cấu hình căn lề cho data. Vì bộ ADC xuất ra giá trị 12bit, được lưu vào biến 16 hoặc 32 bit nên phải căn lề các bit về trái hoặc phải.

Ngoài các tham số trên, cần cấu hình thêm thời gian lấy mẫu, thứ tự kênh ADC khi quét cho kênh ADC,
ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime):

- Rank: Thứ tự của kênh ADC.
- SampleTime: Thời gian lấy mẫu tín hiệu.

Ngoài ra, ADC trên STM32 còn hỗ trợ bộ Hiệu chuẩn giá trị (Calibrate) để giảm sai số do điện dung bên trong gây ra. Theo nhà sản xuất, nên hiệu chỉnh ADC ngay khi bật nguồn cho bộ ADC.

- Việc Hiệu chuẩn bắt đầu bằng việc đặt lại giá trị thanh ghi Calibrate. (Hàm ADC_ResetCalibration(ADC_TypeDef* ADCx))

- Sau khi thanh ghi reset xong, Bật chế độ Hiệu chauanr cho ADC và đợi cho việc hiệu chuẩn hoàn tất.

- Bắt đầu quá trình chuyển dổi ADC.

Chi tiết thành viên trong struct.

```
typedef struct
{
  uint32_t ADC_Mode;                    /*!< Configures the ADC to operate in independent or
                                             dual mode.
                                             This parameter can be a value of @ref ADC_mode */

  FunctionalState ADC_ScanConvMode;     /*!< Specifies whether the conversion is performed in
                                             Scan (multichannels) or Single (one channel) mode.
                                             This parameter can be set to ENABLE or DISABLE */

  FunctionalState ADC_ContinuousConvMode; /*!< Specifies whether the conversion is performed in
                                             Continuous or Single mode.
                                             This parameter can be set to ENABLE or DISABLE. */

  uint32_t ADC_ExternalTrigConv;        /*!< Defines the external trigger used to start the analog
                                             to digital conversion of regular channels. This parameter
                                             can be a value of @ref ADC_external_trigger_sources_for_regular_channels_conversion */

  uint32_t ADC_DataAlign;               /*!< Specifies whether the ADC data alignment is left or right.
                                             This parameter can be a value of @ref ADC_data_align */

  uint8_t ADC_NbrOfChannel;             /*!< Specifies the number of ADC channels that will be converted
                                             using the sequencer for regular channel group.
                                             This parameter must range from 1 to 16. */
}ADC_InitTypeDef;
/**
```

ADC_mode

Regular Conversion:

- **Single**: ADC chỉ đọc 1 kênh duy nhất, và chỉ đọc khi nào được yêu cầu.
  Bộ ADC có 16 kênh, chế độ single đọc 1 kênh duy nhất, đọc một lần và nó dừng không đọc nữa. giống như analogread() trong andruino nó chỉ đọc khi gọi hàm.
- **Single Continuous**: ADC sẽ đọc một kênh duy nhất, nhưng đọc dữ liệu nhiều lần liên tiếp (Có thể được biết đến như sử dụng DMA để đọc dữ liệu và ghi vào bộ nhớ).
  Khi gọi hàm ra, ADC được kích hoạt chuyển đổi nó sẽ đọc liên tục đến khi ngừng cấp nguồn cho vi điều khiển thì thôi.
- **Scan: Multi-Channels**: Quét qua và đọc dữ liệu nhiều kênh, nhưng chỉ đọc khi nào được yêu cầu.
  Đây là chế độ scan 1 lần , Nó sẽ thực hiện quét lần lượt 16 kênh ADC, ghi dữ liệu vào cùng 1 thanh ghi DR, bạn phải đọc dữ liệu ra ngay trước khi data cua kênh ADC2 được ghi đè vào. Lần lượt thực hiện cho 16 kênh.
- **Scan: Continuous Multi-Channels Repeat**: Quét qua và đọc dữ liệu nhiều kênh, nhưng đọc liên tiếp nhiều lần giống như Single Continous.
  Đây là chế độ quét nhiều lần, 16 kênh ADC sẽ được quét lần lượt liên tục. dữ liệu sẽ được ghi vào cùng 1 thanh ghi nên phải đọc dữ liệu ra ngay sau khi có cờ báo dữ liệu

Injected Conversion: (Thực hiện chuyển đổi tín hiệu analog khi có kích hoạt từ software hoặc phần cứng khác như timer, flag…)

Trong trường hợp nhiều kênh hoạt động. Khi kênh có mức độ ưu tiên cao hơn có thể tạo ra một ***Injected Trigger***. Khi gặp Injected Trigger thì ngay lập tức kênh đang hoạt động bị ngưng lại để kênh được ưu tiên kia có thể hoạt động.

```
/** @defgroup ADC_mode
```

```
  * @{
  Các chế độ ADC:
- Independent: Kênh đơn hoạt động độc lập
- regularly : chế độ chuyển đổi tín hiệu analog thông thường
- Injected: chuyển đổi khi kích hoạt từ phần mềm hoặc từ tín hiệu khác.
STM32 có 16 kênh ADC. Có thể thiết lập các chế độ
-  chuyển đổi AD trên 1 kênh (single mode) hoặc
-  Chuyển đổi AD quét trên toàn bộ 16 kênh (scan mode).
Mỗi chế độ này lại tách nhỏ thành 2 kiểu :
- Thực hiện chuyển AD 1 lần khi gọi hàm hoặc
- Chuyển đổi AD nhiều lần, thực hiện liên tục liên tục
Cụ thể:
- Regular Simultaneous with Alternate Trigger Mode:
Thực hiện chuyển đổi AD đồng thời cho 16 kênh ADC với tín hiệu kích hoạt xen kẽ.
Chế độ này phù hợp khi cần đồng bộ hóa dữ liệu nhưng với các sự kiện kích hoạt khác nhau.
- Injected Simultaneous with Slow Interleaved Mode:
Chuyển đổi AD đồng thời trên 16 kênh ADC khi có tín hiệu kích hoạt từ injected . nhưng với tốc độ xen kẽ chậm hơn, phù hợp với các ứng dụng không yêu cầu tốc độ cao
- Injected Simultaneous with Fast Interleaved Mode:
 Chuyển đổi AD đồng thời trên toàn bộ 16 kênh ADC khi có tín hiệu kích hoạt từ injected . nhưng với tốc độ xen kẽ nhanh hơn . Điều này giúp tăng tốc độ xử lý trong các ứng dụng cần phản hồi nhanh.
- Regular Simultaneous Mode:
Chuyển đổi AD đồng thời trên toàn bộ 16 kênh ADC kênh thường.
- Fast Interleaved Mode: Chuyển đổi AD chế độ xen kẽ nhanh
- Slow Interleaved Mode: Chuyển đổi AD chế độ xen kẽ chậm
- Alternate Trigger Mode: Trong chế độ này, việc kích hoạt chuyển đổi được xen kẽ giữa các ADC theo một trình tự cụ thể. Điều này hữu ích khi cần phối hợp các ADC theo các sự kiện kích hoạt khác nhau

}
  */

  #define ADC_Mode_Independent                     ((uint32_t)0x00000000)
//Kênh đơn, ADC hoạt động độc lập
  #define ADC_Mode_RegInjecSimult                  ((uint32_t)0x00010000)
// Mode: Regularly-Injected Simultanous
  #define ADC_Mode_RegSimult_AlterTrig             ((uint32_t)0x00020000)
// Regular Simultaneous with Alternate Trigger Mode
  #define ADC_Mode_InjecSimult_FastInterl          ((uint32_t)0x00030000)
  #define ADC_Mode_InjecSimult_SlowInterl          ((uint32_t)0x00040000)
  #define ADC_Mode_InjecSimult                     ((uint32_t)0x00050000)
  #define ADC_Mode_RegSimult                       ((uint32_t)0x00060000)
  #define ADC_Mode_FastInterl                      ((uint32_t)0x00070000)
  #define ADC_Mode_SlowInterl                      ((uint32_t)0x00080000)
  #define ADC_Mode_AlterTrig                       ((uint32_t)0x00090000)

  #define IS_ADC_MODE(MODE) (((MODE) == ADC_Mode_Independent) || \
```

```
                ((MODE) == ADC_Mode_RegInjecSimult) || \
                ((MODE) == ADC_Mode_RegSimult_AlterTrig) || \
                ((MODE) == ADC_Mode_InjecSimult_FastInterl) || \
                ((MODE) == ADC_Mode_InjecSimult_SlowInterl) || \
                ((MODE) == ADC_Mode_InjecSimult) || \
                ((MODE) == ADC_Mode_RegSimult) || \
                ((MODE) == ADC_Mode_FastInterl) || \
                ((MODE) == ADC_Mode_SlowInterl) || \
                ((MODE) == ADC_Mode_AlterTrig))
```

```
529 typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;
530 #define IS_FUNCTIONAL_STATE(STATE) (((STATE) == DISABLE) || ((STATE) == ENABLE))
```

```
FunctionalState ADC_ScanConvMode;
/*ADC_ScanConvMode:
ENABLE: Thực hiện Chuyển đổi AD trên toàn bộ các kênh ADC (Scan Mode).
DISABLE: Chuyển đổi AD trên một kênh duy nhất (Single Mode). */

FunctionalState ADC_ContinuousConvMode;
/*ADC_ContinuousConvMode:
ENABLE: Chuyển đổi AD liên tục, tự động lặp lại (Continuous Mode).
DISABLE: Chuyển đổi một lần duy nhất (Single Mode). */
```

ADC_ExternalTrigConv : Chọn nguồn kích hoạt ADC

adc_c.c | stm32f10x_adc.h | stm32f10x_adc.c | stm32f10x.h

```
118 /** @defgroup ADC_external_trigger_sources_for_regular_channels_conversion
119   * @{
120   */
121
122 #define ADC_ExternalTrigConv_T1_CC1              ((uint32_t)0x00000000) /*!< For ADC1 and ADC2 */
123 #define ADC_ExternalTrigConv_T1_CC2              ((uint32_t)0x00020000) /*!< For ADC1 and ADC2 */
124 #define ADC_ExternalTrigConv_T2_CC2              ((uint32_t)0x00060000) /*!< For ADC1 and ADC2 */
125 #define ADC_ExternalTrigConv_T3_TRGO             ((uint32_t)0x00080000) /*!< For ADC1 and ADC2 */
126 #define ADC_ExternalTrigConv_T4_CC4              ((uint32_t)0x000A0000) /*!< For ADC1 and ADC2 */
127 #define ADC_ExternalTrigConv_Ext_IT11_TIM8_TRGO  ((uint32_t)0x000C0000) /*!< For ADC1 and ADC2 */
128
129 #define ADC_ExternalTrigConv_T1_CC3              ((uint32_t)0x00040000) /*!< For ADC1, ADC2 and ADC3 */
130 #define ADC_ExternalTrigConv_None                ((uint32_t)0x000E0000) /*!< For ADC1, ADC2 and ADC3 */
131
132 #define ADC_ExternalTrigConv_T3_CC1              ((uint32_t)0x00000000) /*!< For ADC3 only */
133 #define ADC_ExternalTrigConv_T2_CC3              ((uint32_t)0x00020000) /*!< For ADC3 only */
134 #define ADC_ExternalTrigConv_T8_CC1              ((uint32_t)0x00060000) /*!< For ADC3 only */
135 #define ADC_ExternalTrigConv_T8_TRGO             ((uint32_t)0x00080000) /*!< For ADC3 only */
136 #define ADC_ExternalTrigConv_T5_CC1              ((uint32_t)0x000A0000) /*!< For ADC3 only */
137 #define ADC_ExternalTrigConv_T5_CC3              ((uint32_t)0x000C0000) /*!< For ADC3 only */
138
139 #define IS_ADC_EXT_TRIG(REGTRIG) (((REGTRIG) == ADC_ExternalTrigConv_T1_CC1) || \
140                                   ((REGTRIG) == ADC_ExternalTrigConv_T1_CC2) || \
141                                   ((REGTRIG) == ADC_ExternalTrigConv_T1_CC3) || \
142                                   ((REGTRIG) == ADC_ExternalTrigConv_T2_CC2) || \
143                                   ((REGTRIG) == ADC_ExternalTrigConv_T3_TRGO) || \
144                                   ((REGTRIG) == ADC_ExternalTrigConv_T4_CC4) || \
145                                   ((REGTRIG) == ADC_ExternalTrigConv_Ext_IT11_TIM8_TRGO) || \
146                                   ((REGTRIG) == ADC_ExternalTrigConv_None) || \
147                                   ((REGTRIG) == ADC_ExternalTrigConv_T3_CC1) || \
148                                   ((REGTRIG) == ADC_ExternalTrigConv_T2_CC3) || \
149                                   ((REGTRIG) == ADC_ExternalTrigConv_T8_CC1) || \
150                                   ((REGTRIG) == ADC_ExternalTrigConv_T8_TRGO) || \
151                                   ((REGTRIG) == ADC_ExternalTrigConv_T5_CC1) || \
152                                   ((REGTRIG) == ADC_ExternalTrigConv_T5_CC3))
153 /**
```

ADC_ExternalTrigConv_T1_CC1 (0x00000000**)**: Kích hoạt từ Timer 1, Capture/Compare 1, dành cho ADC1 và ADC2.

ADC_ExternalTrigConv_Ext_IT11_TIM8_TRGO (0x000C0000**)**: Kích hoạt từ ngắt ngoài 11 hoặc từ Timer 8, Trigger Output, dành cho ADC1 và ADC2.
ADC_ExternalTrigConv_None (0x000E0000**)**: Không dùng nguồn bên ngoài kích chuyển đổi AD, dành cho ADC1, ADC2 và ADC3.

Chế độ Capture

1. **Capture Input**: Khi bộ đếm của Timer đạt đến một giá trị nhất định, giá trị hiện tại của bộ đếm sẽ được "bắt" và lưu vào thanh ghi Capture/Compare.
2. **Ứng dụng**: Chế độ này thường được sử dụng để đo khoảng thời gian giữa các sự kiện, ví dụ như đo tần số hoặc chu kỳ của tín hiệu ngoại vi.

Chế độ Compare

1. **Compare Output**: Giá trị hiện tại của bộ đếm Timer được so sánh với giá trị trong thanh ghi Capture/Compare. Khi có sự khớp lệnh (so sánh bằng nhau), một sự kiện có thể được kích hoạt, như tạo ra một ngắt hoặc thay đổi trạng thái của một chân xuất ra.
2. **Ứng dụng**: Chế độ này thường được sử dụng để tạo tín hiệu PWM (Pulse Width Modulation), điều khiển động cơ, hoặc tạo sóng xung vuông.

Ví dụ về Sử dụng:

- **Đo tần số**: Sử dụng chế độ Capture để "bắt" giá trị của bộ đếm tại mỗi lần tín hiệu đầu vào thay đổi (rising/falling edge). Bằng cách đếm số lượng bộ đếm trong một khoảng thời gian nhất định, bạn có thể tính toán tần số của tín hiệu.
- **Tạo tín hiệu PWM**: Sử dụng chế độ Compare để thay đổi trạng thái của chân xuất ra khi bộ đếm đạt đến một giá trị cụ thể, tạo ra sóng xung với độ rộng xung điều chỉnh được.

```
157 /** @defgroup ADC_data_align
158   * @{
159   */
160
161 #define ADC_DataAlign_Right                     ((uint32_t)0x00000000)
162 #define ADC_DataAlign_Left                      ((uint32_t)0x00000800)
163 #define IS_ADC_DATA_ALIGN(ALIGN) (((ALIGN) == ADC_DataAlign_Right) || \
164                                   ((ALIGN) == ADC_DataAlign_Left))
```

ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)

```
/**
  * @brief  Configures for the selected ADC regular channel its corresponding
  *         rank in the sequencer and its sample time.
  * @param  ADCx: where x can be 1, 2 or 3 to select the ADC peripheral.
  * @param  ADC_Channel: the ADC channel to configure.
  *   This parameter can be one of the following values:
  *     @arg ADC_Channel_0: ADC Channel0 selected
  *     @arg ADC_Channel_1: ADC Channel1 selected
```

```
  *     @arg ADC_Channel_2: ADC Channel2 selected
  *     @arg ADC_Channel_3: ADC Channel3 selected
  *     @arg ADC_Channel_4: ADC Channel4 selected
  *     @arg ADC_Channel_5: ADC Channel5 selected
  *     @arg ADC_Channel_6: ADC Channel6 selected
  *     @arg ADC_Channel_7: ADC Channel7 selected
  *     @arg ADC_Channel_8: ADC Channel8 selected
  *     @arg ADC_Channel_9: ADC Channel9 selected
  *     @arg ADC_Channel_10: ADC Channel10 selected
  *     @arg ADC_Channel_11: ADC Channel11 selected
  *     @arg ADC_Channel_12: ADC Channel12 selected
  *     @arg ADC_Channel_13: ADC Channel13 selected
  *     @arg ADC_Channel_14: ADC Channel14 selected
  *     @arg ADC_Channel_15: ADC Channel15 selected
  *     @arg ADC_Channel_16: ADC Channel16 selected
  *     @arg ADC_Channel_17: ADC Channel17 selected
  * @param  Rank: The rank in the regular group sequencer. This parameter must
be between 1 to 16.
  * @param  ADC_SampleTime: The sample time value to be set for the selected
channel.
  *   This parameter can be one of the following values:
  *     @arg ADC_SampleTime_1Cycles5: Sample time equal to 1.5 cycles
  *     @arg ADC_SampleTime_7Cycles5: Sample time equal to 7.5 cycles
  *     @arg ADC_SampleTime_13Cycles5: Sample time equal to 13.5 cycles
  *     @arg ADC_SampleTime_28Cycles5: Sample time equal to 28.5 cycles
  *     @arg ADC_SampleTime_41Cycles5: Sample time equal to 41.5 cycles
  *     @arg ADC_SampleTime_55Cycles5: Sample time equal to 55.5 cycles
  *     @arg ADC_SampleTime_71Cycles5: Sample time equal to 71.5 cycles
  *     @arg ADC_SampleTime_239Cycles5: Sample time equal to 239.5 cycles
  * @retval None
  */
  void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel,
uint8_t Rank, uint8_t ADC_SampleTime)
  {
    uint32_t tmpreg1 = 0, tmpreg2 = 0;
    /* Check the parameters */
    assert_param(IS_ADC_ALL_PERIPH(ADCx));
    assert_param(IS_ADC_CHANNEL(ADC_Channel));
    assert_param(IS_ADC_REGULAR_RANK(Rank));
    assert_param(IS_ADC_SAMPLE_TIME(ADC_SampleTime));
    /* if ADC_Channel_10 ... ADC_Channel_17 is selected */
    if (ADC_Channel > ADC_Channel_9)
    {
      /* Get the old register value */
      tmpreg1 = ADCx->SMPR1;
      /* Calculate the mask to clear */
      tmpreg2 = SMPR1_SMP_Set << (3 * (ADC_Channel - 10));
      /* Clear the old channel sample time */
```

```
    tmpreg1 &= ~tmpreg2;
    /* Calculate the mask to set */
    tmpreg2 = (uint32_t)ADC_SampleTime << (3 * (ADC_Channel - 10));
    /* Set the new channel sample time */
    tmpreg1 |= tmpreg2;
    /* Store the new register value */
    ADCx->SMPR1 = tmpreg1;
  }
  else /* ADC_Channel include in ADC_Channel_[0..9] */
  {
    /* Get the old register value */
    tmpreg1 = ADCx->SMPR2;
    /* Calculate the mask to clear */
    tmpreg2 = SMPR2_SMP_Set << (3 * ADC_Channel);
    /* Clear the old channel sample time */
    tmpreg1 &= ~tmpreg2;
    /* Calculate the mask to set */
    tmpreg2 = (uint32_t)ADC_SampleTime << (3 * ADC_Channel);
    /* Set the new channel sample time */
    tmpreg1 |= tmpreg2;
    /* Store the new register value */
    ADCx->SMPR2 = tmpreg1;
  }
  /* For Rank 1 to 6 */
  if (Rank < 7)
  {
    /* Get the old register value */
    tmpreg1 = ADCx->SQR3;
    /* Calculate the mask to clear */
    tmpreg2 = SQR3_SQ_Set << (5 * (Rank - 1));
    /* Clear the old SQx bits for the selected rank */
    tmpreg1 &= ~tmpreg2;
    /* Calculate the mask to set */
    tmpreg2 = (uint32_t)ADC_Channel << (5 * (Rank - 1));
    /* Set the SQx bits for the selected rank */
    tmpreg1 |= tmpreg2;
    /* Store the new register value */
    ADCx->SQR3 = tmpreg1;
  }
  /* For Rank 7 to 12 */
  else if (Rank < 13)
  {
    /* Get the old register value */
    tmpreg1 = ADCx->SQR2;
    /* Calculate the mask to clear */
    tmpreg2 = SQR2_SQ_Set << (5 * (Rank - 7));
    /* Clear the old SQx bits for the selected rank */
    tmpreg1 &= ~tmpreg2;
```

```
    /* Calculate the mask to set */
    tmpreg2 = (uint32_t)ADC_Channel << (5 * (Rank - 7));
    /* Set the SQx bits for the selected rank */
    tmpreg1 |= tmpreg2;
    /* Store the new register value */
    ADCx->SQR2 = tmpreg1;
  }
  /* For Rank 13 to 16 */
  else
  {
    /* Get the old register value */
    tmpreg1 = ADCx->SQR1;
    /* Calculate the mask to clear */
    tmpreg2 = SQR1_SQ_Set << (5 * (Rank - 13));
    /* Clear the old SQx bits for the selected rank */
    tmpreg1 &= ~tmpreg2;
    /* Calculate the mask to set */
    tmpreg2 = (uint32_t)ADC_Channel << (5 * (Rank - 13));
    /* Set the SQx bits for the selected rank */
    tmpreg1 |= tmpreg2;
    /* Store the new register value */
    ADCx->SQR1 = tmpreg1;
  }
}
```

1. Nguyên lý hoạt động của bộ lọc Kalman

Bộ lọc Kalman hoạt động dựa trên hai nguồn thông tin:

1. **Mô hình hệ thống**: Mô tả cách hệ thống thay đổi trạng thái theo thời gian.
2. **Quan sát (measurement)**: Dữ liệu đo lường từ các cảm biến, thường chứa nhiễu.

Bộ lọc Kalman kết hợp hai nguồn thông tin này để ước lượng trạng thái của hệ thống một cách tối ưu, giảm thiểu sai số ước lượng.

---

2. Các bước hoạt động của bộ lọc Kalman

Bộ lọc Kalman hoạt động theo hai giai đoạn chính:

1. **Dự đoán (Prediction)**:
   - Dự đoán trạng thái tiếp theo của hệ thống dựa trên mô hình hệ thống.
   - Ước lượng độ không chắc chắn (uncertainty) của dự đoán.
2. **Cập nhật (Update)**:
   - Kết hợp dự đoán với quan sát từ cảm biến để cập nhật trạng thái ước lượng.
   - Điều chỉnh độ không chắc chắn dựa trên độ chính xác của quan sát.

---

3. Mô hình toán học của bộ lọc Kalman

a. Mô hình hệ thống

- Trạng thái hệ thống tại thời điểm kk được biểu diễn bằng vector xk$\mathbf{x}_k$.
- Mô hình hệ thống được mô tả bằng phương trình trạng thái:
  xk=Fkxk−1+Bkuk+wk$\mathbf{x}_k=\mathbf{F}_k\mathbf{x}_{k-1}+\mathbf{B}_k\mathbf{u}_k+\mathbf{w}_k$

  - Fk$\mathbf{F}_k$: Ma trận chuyển đổi trạng thái.
  - Bk$\mathbf{B}_k$: Ma trận điều khiển.
  - uk$\mathbf{u}_k$: Vector điều khiển.
  - wk$\mathbf{w}_k$: Nhiễu quá trình (process noise), thường được giả định là nhiễu Gaussian với trung bình 0 và ma trận hiệp phương sai Qk$\mathbf{Q}_k$.

b. Mô hình quan sát

- Quan sát tại thời điểm kk được biểu diễn bằng vector zk$\mathbf{z}_k$.
- Mô hình quan sát được mô tả bằng phương trình:
  zk=Hkxk+vk$\mathbf{z}_k=\mathbf{H}_k\mathbf{x}_k+\mathbf{v}_k$

  - Hk$\mathbf{H}_k$: Ma trận quan sát.
  - vk$\mathbf{v}_k$: Nhiễu quan sát (measurement noise), thường được giả định là nhiễu Gaussian với trung bình 0 và ma trận hiệp phương sai Rk$\mathbf{R}_k$.

---

4. Các bước chi tiết của bộ lọc Kalman

a. Khởi tạo

- Khởi tạo trạng thái ban đầu x0**x**0 và ma trận hiệp phương sai ban đầu P0**P**0.

b. Dự đoán (Prediction)

1. Dự đoán trạng thái tiếp theo:

   x^k−=Fkx^k−1+Bkuk**x**^*k*−=**F***k***x**^*k*−1+**B***k***u***k*

2. Dự đoán ma trận hiệp phương sai:
   Pk−=FkPk−1FkT+Qk**P***k*−=**F***k***P***k*−1**F***kT*+**Q***k*

c. Cập nhật (Update)

1. Tính độ lệch (innovation) giữa quan sát và dự đoán:

   yk=zk−Hkx^k−**y***k*=**z***k*−**H***k***x**^*k*−

2. Tính ma trận hiệp phương sai của độ lệch:
   Sk=HkPk−HkT+Rk**S***k*=**H***k***P***k*−**H***kT*+**R***k*

3. Tính ma trận Kalman Gain:
   Kk=Pk−HkTSk−1**K***k*=**P***k*−**H***kT***S***k*−1

4. Cập nhật trạng thái ước lượng:
   x^k=x^k−+Kkyk**x**^*k*=**x**^*k*−+**K***k***y***k*

5. Cập nhật ma trận hiệp phương sai:
   Pk=(I−KkHk)Pk−**P***k*=(**I**−**K***k***H***k*)**P***k*−

---

5. Ví dụ ứng dụng của bộ lọc Kalman

a. Ứng dụng trong định vị và dẫn đường

- Trong hệ thống GPS, bộ lọc Kalman được sử dụng để kết hợp dữ liệu từ các cảm biến (ví dụ: vận tốc, gia tốc) và dữ liệu GPS để ước lượng vị trí chính xác của vật thể.

b. Ứng dụng trong điều khiển tự động

- Trong robot di động, bộ lọc Kalman được sử dụng để ước lượng vị trí và hướng di chuyển của robot dựa trên dữ liệu từ cảm biến và mô hình động lực học.

c. Ứng dụng trong xử lý tín hiệu

- Trong xử lý tín hiệu, bộ lọc Kalman được sử dụng để loại bỏ nhiễu và ước lượng tín hiệu gốc từ tín hiệu quan sát nhiễu.

---

6. Ưu điểm của bộ lọc Kalman

- **Tối ưu hóa**: Bộ lọc Kalman cung cấp ước lượng tối ưu trong điều kiện nhiễu Gaussian.
- **Hiệu quả tính toán**: Thuật toán có độ phức tạp tính toán thấp, phù hợp với các hệ thống thời gian thực.
- **Linh hoạt**: Có thể áp dụng cho nhiều loại hệ thống động lực học khác nhau.

---

7. Nhược điểm của bộ lọc Kalman

- **Giả định nhiễu Gaussian**: Bộ lọc Kalman hoạt động tốt nhất khi nhiễu quá trình và nhiễu quan sát là Gaussian. Nếu nhiễu không phải Gaussian, hiệu quả của bộ lọc có thể giảm.
- **Yêu cầu mô hình chính xác**: Bộ lọc Kalman đòi hỏi mô hình hệ thống và mô hình quan sát phải chính xác. Nếu mô hình không chính xác, kết quả ước lượng có thể sai lệch.

---

8. Kết luận

Bộ lọc Kalman là một công cụ mạnh mẽ để ước lượng trạng thái của hệ thống động lực học trong điều kiện nhiễu. Nó kết hợp thông tin từ mô hình hệ thống và quan sát để cung cấp ước lượng tối ưu, giảm thiểu sai số. Tuy nhiên, việc áp dụng bộ lọc Kalman đòi hỏi hiểu biết về mô hình hệ thống và đặc tính nhiễu.

Hiện nay, có nhiều bộ lọc và phương pháp xử lý tín hiệu tiên tiến hơn bộ lọc Kalman truyền thống, đặc biệt trong các ứng dụng liên quan đến chuyển đổi tín hiệu tương tự sang số (ADC - Analog-to-Digital Conversion). Tùy thuộc vào yêu cầu cụ thể của ứng dụng (như độ chính xác, tốc độ xử lý, độ phức tạp tính toán, và loại nhiễu), các bộ lọc sau có thể được coi là tốt hơn hoặc phù hợp hơn bộ lọc Kalman:

1. Bộ lọc Kalman mở rộng (Extended Kalman Filter - EKF)

- **Ứng dụng**: Dùng cho hệ thống phi tuyến.
- **Ưu điểm**:
    - Mở rộng từ bộ lọc Kalman truyền thống để xử lý các hệ thống phi tuyến bằng cách tuyến tính hóa mô hình hệ thống tại từng bước.
    - Phù hợp với các hệ thống động lực học phức tạp hơn.
- **Nhược điểm**:
    - Độ phức tạp tính toán cao hơn so với bộ lọc Kalman truyền thống.
    - Hiệu suất phụ thuộc vào độ chính xác của quá trình tuyến tính hóa.

---

2. Bộ lọc Unscented Kalman Filter (UKF)

- **Ứng dụng**: Dùng cho hệ thống phi tuyến.
- **Ưu điểm**:
    - Không cần tuyến tính hóa mô hình hệ thống, thay vào đó sử dụng phép biến đổi Unscented để xấp xỉ phân phối xác suất.
    - Chính xác hơn EKF trong nhiều trường hợp phi tuyến.
- **Nhược điểm**:
    - Độ phức tạp tính toán cao hơn so với EKF.