Décimo proyecto
ADC

Alejandro Ramírez Jaramillo

Universidad Nacional de Colombia sede Manizales Email: alramirezja@unal.edu.co

Introducción

Los ADC (Analog to digital converter) son periféricos del microcontrolador que permiten convertir una señal de voltaje en un pin en un valor numérico, en otras palabras permiten llevar una señal física a un formato digital que puede ser usado/interpretado por el microcontrolador. Por ejemplo en la imagen de abajo se ilustra una señal de voltaje (color negro) que es muestreada por el ADC (puntos rojos) de manera periódica, estos valores muestreados son los que se convierten a un formato digital. Este ejemplo fue diseñado para una tarjeta STM32L476 y se hará en el lenguaje de programación assembler.

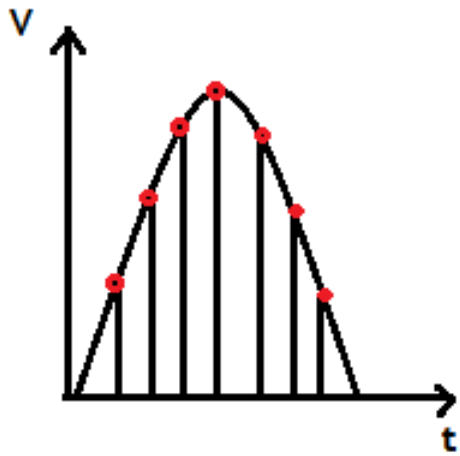

Figure 1. Muestreo de una señal de voltaje con el ADC

Contenido

1 Registros

Address:

+ RCC: 0x40021000
+ GPIOA: 0x48000000
+ ADC1: 0x0x50040000

1.1 Inicialización de los relojes de los periféricos

1.1.1 RCC_AHB2ENR

+ Offset:0x4C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	HASHE N	AESEN (1)
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DCMIE N	ADCEN	OTGFS EN	Res.	Res.	Res.	GPIOIE N	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
	rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 2. Distribución bits AHB2ENR, RM0351 Reference Manual, Pag. 251

Para el ejercicio el usuario usará el pin 1 del puerto A.

1.1.2 RCC_CCIPR

+ Offset: 0x88

Peripherals independent clock configuration register, se usa para escoger la fuente del reloj del ADC, en otras palabra que reloj va a usarse en el ADC, con los bits 29 y 28.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DFSDM 1 SEL	SWP MI1 SEL	ADCSEL[1:0]		CLK48SEL[1:0]		SAI2SEL[1:0]		SAI1SEL[1:0]		LPTIM2SEL[1:0]		LPTIM1SEL[1:0		I2C3SEL[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I2C2SEL[1:0]		I2C1SEL[1:0]		LPUART1SEL [1:0]		UART5SEL [1:0]		UART4SEL [1:0]		USART3SEL [1:0]		USART2SEL [1:0]		USART1SEL [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 3. Distribución bits CCIPR, RM0351 Reference Manual, Pag. 269

1.2 Configuración de entradas y salidas

1.2.1 GPIOx_MODER

+ Offset:0x00

Con este registro establezca el pin 1 del puerto A como un pin analógico para que funcione como la entrada del ADC.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 4. Distribución bits MODER, RM0351 Reference Manual, Pag. 305

1.2.2 GPIOx_ASCR

+ Offset: 0x2C

GPIO port analog switch control register, conecta el switch analógico a la entrada del ADC.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASC15	ASC14	ASC13	ASC12	ASC11	ASC10	ASC9	ASC8	ASC7	ASC6	ASC5	ASC4	ASC3	ASC2	ASC1	ASC0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5. Distribución bits ASCR, RM0351 Reference Manual, Pag. 312

1.3 Configuración del ADC

1.3.1 ADC1_CR

+ Offset: 0x08

ADC control register, registro que maneja el ADC, el inicio y el final de las conversiones. Este registro se usará para activar el regulador de voltaje del ADC y habilitar el ADC.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADCAL	ADCALDIF	DEEPPWD	ADVREGEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
rs	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	JADSTP	ADSTP	JADSTART	ADSTART	ADDIS	ADEN
										rs	rs	rs	rs	rs	rs

Figure 6. Distribución bits ADC_CR, RM0351 Reference Manual, Pag. 589

1.3.2 ADC1_SQR1

+ Offset: 0x30

ADC regular sequence register 1, lo primeros 4 bits determinan el número de conversiones que se harán en una secuencia, los bits 6:10 escogen que canal se va a leer en la primera conversión (en este ejercicio se usa el canal 6).

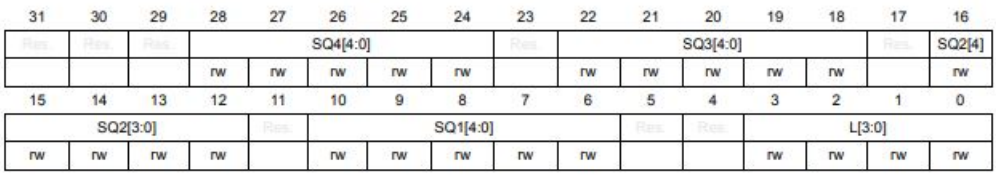

Figure 7. Distribución bits ADC_SQR1, RM0351 Reference Manual, Pag. 602

1.3.3 ADC_SMPR1

+ Offset: 0x14

ADC sample time register 1, este registro determina el tiempo de muestreo del ADC para un canal específico, si nos remitimos a la imagen en la introducción seria la distancia en tiempo entre cada punto rojo basándose en el reloj del ADC.

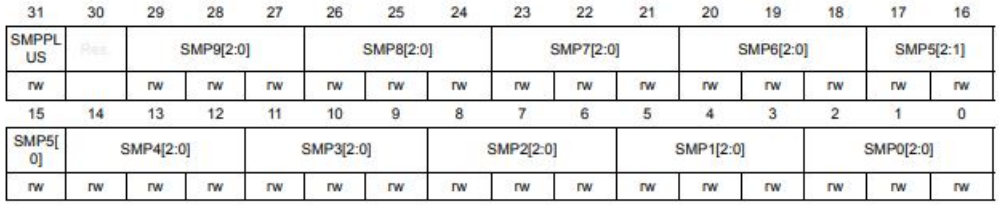

Figure 8. Distribución bits ADC_SMPR1, RM0351 Reference Manual, Pag. 598

1.3.4 ADC1_ISR

+ Offset: 0x00

ADC interrupt and status register, este registro contiene múltiples banderas del ADC que indican, por ejemplo, el final de una secuencia de conversión. Va a ser usado para identificar cuando se puede leer el valor del ADC en el registro ADC_DR.

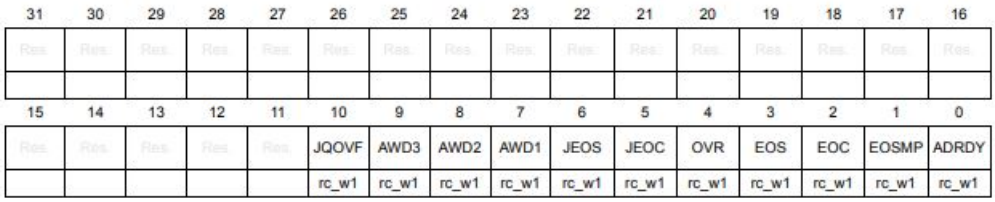

Figure 9. Distribución bits ADC1_ISR, RM0351 Reference Manual, Pag. 585

1.3.5 ADC1_DR

+ Offset: 0x40

ADC regular data register, este registro contiene el valor medido por el ADC, solo puede ser leído.

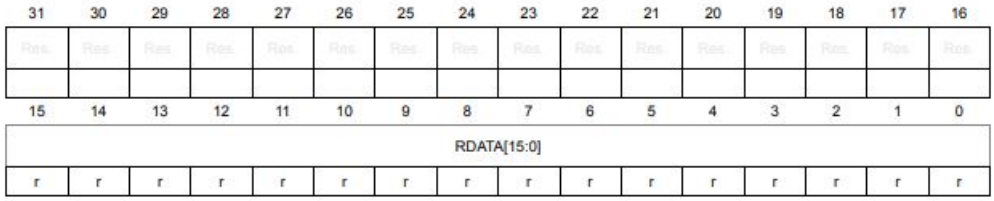

Figure 10. Distribución bits ADC1_DR, RM0351 Reference Manual, Pag. 606

2 Código ADC

El código que se utilizará el ADC para medir un valor de voltaje en el pin 1 del puerto A.

Operaciones:

- orr → OR lógico
- and & = → AND lógico

2.1 Dirección de registros

Se inicializan los registros usando los valores de offset y direcciones que se encuentran en la sección Registros.

```
//       RCC   base address is 0x40021000
//   AHB2ENR register offset is 0x4C
.equ     RCC_AHB2ENR,   0x4002104C // RCC AHB2 peripheral clock reg

//       RCC   base address is 0x40021000
//   CCIPR register offset is 0x88
.equ     RCC_CCIPR,   0x40021088 // RCC CCIPR peripheral independent clock reg

//       GPIOA base address is 0x48000000
//   MODER register offset is 0x00
//   ASCR  register offset is 0x2C
.equ     GPIOA_MODER,   0x48000000 // GPIOA port mode register
.equ     GPIOA_ASCR,    0x4800002C // GPIOA analog switch data register

//       ADC1  base address is 0x50040000
//   CR    register offset is 0x08
//   SQR1  register offset is 0x30
//   SMPR1 register offset is 0x14
//   ISR   register offset is 0x00
//   DR    register offset is 0x40
.equ     ADC1_CR,     0x50040008 // ADC1 control register
.equ     ADC1_SQR1,   0x50040030 // ADC1 regular sequence register
.equ     ADC1_SMPR1,  0x50040014 // ADC1 sample time register
.equ     ADC1_ISR,    0x50040000 // ADC1 interrupt and status register
.equ     ADC1_DR,     0x50040040 // ADC1 regular data register
```

2.2 Relojes

Como se va a usar el pin 1 del puerto A como la entrada del ADC se activa el reloj para el puerto A:

```
// Enable GPIOA & ADC Peripheral Clock (bit 0 & 13 in AHB2ENR register)
ldr r6, = RCC_AHB2ENR       // Load peripheral clock reg address to r6
ldr r5, = 0x2001            // Set bit 0 to enable GPIOA clock and bit 13 ADC clock enable
str r5, [r6]                // Store result in peripheral clock register
```

Después escogemos el reloj del ADC y lo habilitamos:

```
// Enable ADC Clock (bit 29 & 28 in CCIPR register)
ldr r6, = RCC_CCIPR         // Load peripheral clock reg address to r6
ldr r5, [r6]                // Read its content to r5
orr r5, 0x30000000          // Set bit 29 & 28 to enable system clock as ADC clock
str r5, [r6]                // Store result in peripheral clock register
```

2.3 Configuración GPIOs

Se configura el pin 1 del puerto A en modo análogo.

```
// Make GPIOA Pin1 as analog pin (bits 3:2 in MODER register)
ldr r6, = GPIOA_MODER       // Load GPIOA MODER register address to r6
ldr r5, [r6]                // Read its content to r5
```

```
and r5, 0xFFFFFFFF          // Write 11 to bits 3, 2 for P1
str r5, [r6]                // Store result in GPIOA MODER register
```

Y se conecta el pin 1 al switch análogo:

```
// Make GPIOA Pin1 as analog switch pin (bits 3:2 in ASCR register)
ldr r6, = GPIOA_ASCR        // Load GPIOA ASCR register address to r6
ldr r5, [r6]                // Read its content to r5
orr r5, 0x00000002          // Write 10 to bit 1 for P1
str r5, [r6]                // Store result in GPIOA ASCR register
```

2.4 Configuración ADC

Comenzamos habilitando el regulador de voltaje del ADC1 usando el registro de control.

```
// Enable ADC1 voltage regulator (bit 28 in CR register)
ldr r6, = ADC1_CR           // Load peripheral clock reg address to r6
ldr r5, = 0x10000000         // Set bit 1 to enable ADVREGEN
str r5, [r6]                // Store result in peripheral clock register
```

Usando el registro SQR1 escogemos el canal usado para la primera secuencia de conversión (Channel 6) y con el registro SMPR1 configuramos la frecuencia de las conversiones para este canal.

```
// Set regular secuence chanel ADC1 (bit 10:6 in SQ1 register)
ldr r6, = ADC1_SQR1         // Load peripheral clock reg address to r6
ldr r5, [r6]                // Read its content to r5
orr r5, 0x180               // Set bit 8:7 to enable CH6
str r5, [r6]                // Store result in peripheral clock register

// Set sample time ADC1 CH6 (bit 20:18 in SMP6 register)
ldr r6, = ADC1_SMPR1        // Load peripheral clock reg address to r6
ldr r5, [r6]                // Read its content to r5
orr r5, 0x001C0000          // Set 640.5 ADC clock cycles
str r5, [r6]                // Store result in peripheral clock register
```

El ADC requiere algo de tiempo para inicializarse, por lo cual sera necesario usar un delay, este cargara una variable DELAY en un registro, a este registro se le restara 1 (comando sub) y cuando llegue a cero (comando cmp para comparar el valor del registro con el número cero) seguira con la medición del ADC, en caso contrario seguirá restando 1 al registro.

```
        // wait time while ADC initialize
        ldr r4, = DELAY                     // Load 800000 ticks to r4
delay1:
        sub r4, r4, #1                          // Subtract data content in r4 less 1
        cmp r4, #0                                  // Compare data content in r4 is 0
        bne delay1                                  // branch instruction: jump to lazo 1 if r4 is differe
```

2.5 ADC

Para dar inicio a la conversión de ADC se usa el bit 2 del registro de control (CR).

```
loop:

        // ADC START
        ldr r6, = ADC1_CR           // Load reg address to r6
        ldr r5, [r6]                // Read its content to r5
        orr r5, 0x4                 // Set bit 1 to enable ADSTART
        str r5, [r6]                // Store result in peripheral clock register

        // ADC wait end of conversion
```

Después de iniciar la conversión es necesario esperar hasta que esta termine para leer el valor medido por el ADC, para identificar cuando termina se acude al registro ISR en donde se encuentra la bandera de final de conversión (EOC). Entonces se hará una comparación entre el valor de este registro y una constante para conocer el estado de la bandera (1 o 0).

```
eoc:
        ldr r6, = ADC1_ISR          // Load reg address to r6
        ldr r5, [r6]                // Read its content to r5
        and r5, 0x4
        cmp r5, #4                  // Compare data content in r4 is 4
        bne eoc                     // brach while conversion finish
```

Si embargo el registro ISR tiene otras banderas distintas cuyos estados no conocemos y esto afecta la comparación, pues si una sola bandera está en un estado diferente a su bit correspondiente en la constante con la que se esta comparando el registro, la comparación ya no dependerá solamente de la bandera EOC y no necesariamente indicará si ya termino o no la conversión.
Entonces para hacer una comparación adecuada es necesario convertir los valores desconocidos de los otros bits del registro en valores que si conozcamos (obviamente esta operación se aplicará al registro en el que se almacenará el valor de ADC_ISR, pues no debo modificar el registro original para evitar cambiar el comportamiento del ADC).

```
        and r5, 0x4
        cmp r5, #4                  // Compare data content in r4 is 4
```

Para hacer esto se hace una operación "and" que convertirá todos los demás bits en cero sin importar su valor original mientras el valor del bit EOC permanece igual, entonces se podrá hacer la comparación con la constante. Hasta que la conversión no termine se seguirá haciendo la comparación y cuando se termine se almacenará el valor del registro DR en otro registro.

```
        ldr r6, = ADC1_DR           // Load reg address to r6
        ldr r5, [r6]                // Read its content to r5
```

Aunque la conversión haya terminado, no necesariamente ha terminado la secuencia, por esto se replica lo que se hizo con la bandera EOC pero esta vez usando la bandera EOS (End of regular Sequence flag):

```
eos:
        ldr r6, = ADC1_ISR          // Load reg address to r6
        ldr r5, [r6]                // Read its content to r5
        and r5, 0x8
        cmp r5, #8                  // Compare data content in r4 is 8
        bne eos                     // brach while conversion finish
```

Cuando la secuencia termine es necesario limpiar la bandera EOS poniendo un uno en ella, lo mismo aplicaría para la bandera EOC pero esta se limpia automáticamente cuando se lee el registro CR.

```
        ldr r6, = ADC1_ISR          // Load reg address to r6
        ldr r5, [r6]                // Read its content to r5
        orr r5, 0x8                 // Set bit 3 to enable EOS
        str r5, [r6]                // Store result in register

        b loop                      // Jump to loop
```

3 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ MB1136 Schematic board.