Corso di Microcontrollori e DSP

Laura Magistrale in Ingegneria Meccatronica

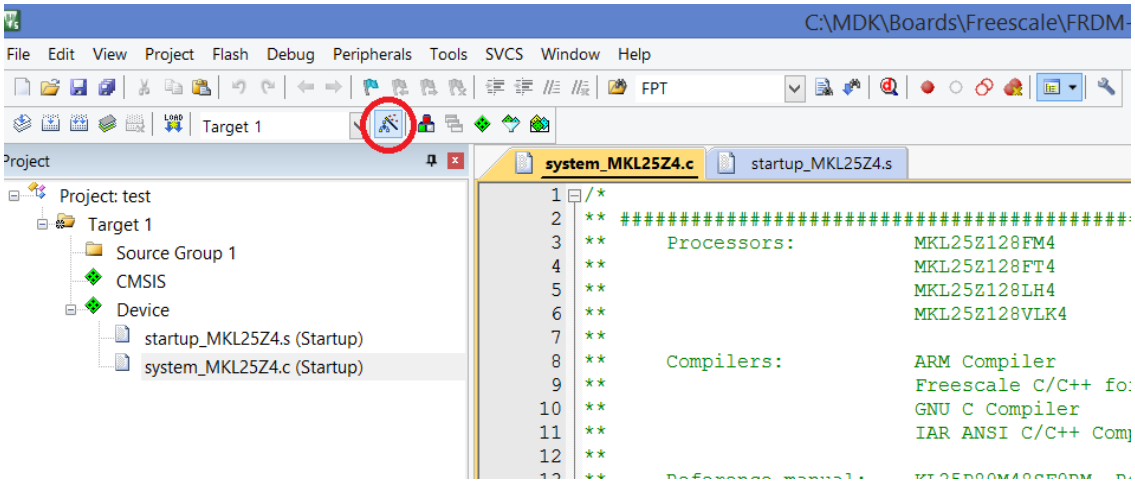
A.A. 2018/2019

IMPLEMENTAZIONE DI UN FILTRO IIR DEL PRIMO ORDINE SU SCHEDA KL25Z

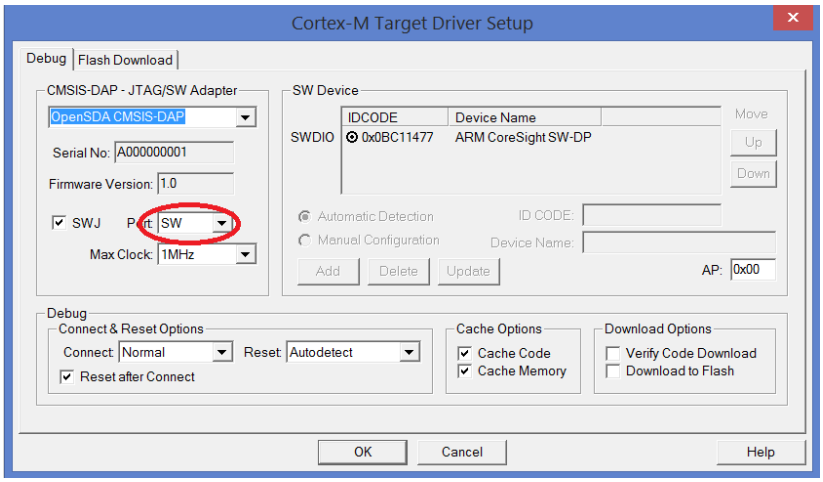
Passo 1: Inizializzazione del progetto

Si inizializzi un nuovo progetto, scegliendo come dispositivo *target* MKL25Z128xxx4 e selezionando come componenti software da utilizzare CMSIS-CORE e Device-Startup.

Si acceda al menù *Options for Target* utilizzando il pulsante sotto riportato e si effettuino le seguenti modifiche.



- Nella sezione *Debug* si selezioni **CMSIS-DAP Adapter**. Dopo aver collegato la scheda **KL25Z** al pc mediante la porta USB **SDA,** si prema sul tasto *Settings* (riportato di fianco a CMSIS-DAP Adapter) e nella nuova finestra si imposti **SW** come porta. Se la rilevazione del dispositivo è avvenuta correttamente, verrà visualizzato il valore *IDCODE* del dispositivo. Si chiudano tutte le finestre premendo su OK.



A questo punto è necessario editare i file di inizializzazione: **startup_MKL25Z**, **system_MKL25Z**.

Per quanto riguarda il file assembler **startup_MKL25Z.s**, a valle dei *Core Vectors*, e quindi appena prima della *label: Default_Handler*, si inserisca il seguente frammento di codice che consente di catturare l'interrupt sollevato dall'ADC e che richiama una routine **ADC0routine** (da definire successivamente):

```
ADC0_IRQHandler     PROC
                    EXPORT  ADC0_IRQHandler        [WEAK]
                    IMPORT  ADC0routine
                    LDR     R0, =ADC0routine
                    BX      R0
                    ENDP
```

Il file **system_MKL25Z.c** (scritto in linguaggio c) è da analizzare in maniera congiunta al file **system_MKL25Z.h** (header file) in cui sono presenti solo delle definizioni (utilizzando delle apposite direttive). In quest'ultimo file si associ il valore appropriato alla costante **CLOCK_SETUP** come segue:

```
#define CLOCK_SETUP     1
```

Questo consentirà di impostare il sistema alla massima velocità (Core clock = 48MHz e Bus clock = 24MHz).

Passo 2: Definizione del file principale di progetto e gestione Reset

Nel riquadro Project (a destra del terminale) si selezioni *Source Group 1* con il testo destro e si scelga "Add new item", impostando un nuovo file C come sorgente. Il file generato dovrà contenere:

- Definizione delle costanti o delle variabili utili al progetto oltre all'inclusione del file di definizione del microcontrollore (#include "MKL25Z4.h");
- Una routine per la gestione dell'interrupt di reset (in questo caso si verifichi la corrispondenza con il nome associato a questa routine nel file *startup_MKL25Z.s*);
- Una routine per la gestione dell'interrupt dell'ADC;
- 3 routine che consentano di inizializzare il convertitore A/D, il convertitore D/A e una delle porte GPIO (da utilizzare per l'accensione di un LED).

Gestione interrupt di reset. Ad ogni reset è necessario inizializzare nuovamente tutte le periferiche. È perciò necessario:

a) richiamare *SystemCoreClockUpdate*(), contenuta nel file *system_MKL25Z.c* e che inizializza l'unità MCG (Multipurpose Clock Generator);
b) richiamare le 3 routine di inizializzazione del convertitore A/D, convertitore D/A e GPIO (descritte in seguito);

c) implementare un loop infinito che resti in attesa di interrupt.

A titolo di esempio si riporta la struttura per la routine del main:

```
int main (void) {
    SystemCoreClockUpdate();

    Istruzioni da eseguire…..
}
```

Da riportare nella relazione:

- Codice implementato per la routine di gestione del reset con descrizione sintetica. La descrizione può essere anche contenuta nel codice stesso, purché sia facilmente leggibile nella relazione (ad esempio utilizzando colori differenti).

Passo 3. Inizializzazione ADC

La fase di inizializzazione del convertitore A/D prevede la configurazione dei moduli SIM (System Integration Module), ADC (Analog to Digital Converter), NVIC (Nestled Vector Interrupt Controller).

Per quanto riguarda la periferica SIM, il settaggio del registro SIM_SCGC6 consente di abilitare il clock dell'ADC. A tale scopo si ricorda che la sintassi per modificare 1-bit di un registro è la seguente:

```
SIM->SCGC6 = (1UL << n);
```

Dove ***n*** è l'opportuno campo, indicato in tabella, che si vuole porre ad 1.

SIM_SCGC6 field descriptions

Field	Description
31 DAC0	DAC0 Clock Gate Control This bit controls the clock gate to the DAC0 module. 0 Clock disabled 1 Clock enabled
30 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
29 RTC	RTC Access Control This bit controls software access and interrupts to the RTC module. 0 Access and interrupts disabled 1 Access and interrupts enabled
28 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
27 ADC0	ADC0 Clock Gate Control This bit controls the clock gate to the ADC0 module. 0 Clock disabled 1 Clock enabled
26 TPM2	TPM2 Clock Gate Control This bit controls the clock gate to the TPM2 module. 0 Clock disabled 1 Clock enabled
25 TPM1	TPM1 Clock Gate Control This bit controls the clock gate to the TPM1 module. 0 Clock disabled 1 Clock enabled
24 TPM0	TPM0 Clock Gate Control This bit controls the clock gate to the TPM0 module. 0 Clock disabled 1 Clock enabled
23 PIT	PIT Clock Gate Control This bit controls the clock gate to the PIT module. 0 Clock disabled 1 Clock enabled
22–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
15 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

Field	Description
14–2 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
1 DMAMUX	DMA Mux Clock Gate Control This bit controls the clock gate to the DMA Mux module. 0 Clock disabled 1 Clock enabled
0 FTF	Flash Memory Clock Gate Control This bit controls the clock gate to the flash memory. Flash reads are still supported while the flash memory is clock gated, but entry into low power modes is blocked. 0 Clock disabled 1 Clock enabled

Si configuri il registro **ADC0_CFG1**, per una conversione a **10 bit**, in **Long Time mode**, operante ad una frequenza **BusClock/2**.

È utile ricordare che al fine di settare ad 1 più bit contemporaneamente si può utilizzare l'operatore **bitwise OR**:

ADC0->CFG1 = (1UL << ***n1***) | (1UL << ***n2***)|(1UL << ***n3***);

ADCx_CFG1 field descriptions

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 ADLPC	Low-Power Configuration Controls the power configuration of the successive approximation converter. This optimizes power consumption when higher sample rates are not required. 0 Normal power configuration. 1 Low-power configuration. The power is reduced at the expense of maximum clock speed.
6–5 ADIV	Clock Divide Select ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK. 00 The divide ratio is 1 and the clock rate is input clock. 01 The divide ratio is 2 and the clock rate is (input clock)/2. 10 The divide ratio is 4 and the clock rate is (input clock)/4. 11 The divide ratio is 8 and the clock rate is (input clock)/8.
4 ADLSMP	Sample time configuration ADLSMP selects between different sample times based on the conversion mode selected. This bit adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption if continuous conversions are enabled and high conversion rates are not required. When ADLSMP=1, the long sample time select bits, (ADLSTS[1:0]), can select the extent of the long sample time. 0 Short sample time. 1 Long sample time.
3–2 MODE	Conversion mode selection Selects the ADC resolution mode. 00 When DIFF=0:It is single-ended 8-bit conversion; when DIFF=1, it is differential 9-bit conversion with 2's complement output. 01 When DIFF=0:It is single-ended 12-bit conversion ; when DIFF=1, it is differential 13-bit conversion with 2's complement output. 10 When DIFF=0:It is single-ended 10-bit conversion ; when DIFF=1, it is differential 11-bit conversion with 2's complement output. 11 When DIFF=0:It is single-ended 16-bit conversion; when DIFF=1, it is differential 16-bit conversion with 2's complement output.
1–0 ADICLK	Input Clock Select Selects the input clock source to generate the internal clock, ADCK. Note that when the ADACK clock source is selected, it is not required to be active prior to conversion start. When it is selected and it is not active prior to a conversion start, when CFG2[ADACKEN]=0, the asynchronous clock is activated at the start of a conversion and deactivated when conversions are terminated. In this case, there is an associated clock startup delay each time the clock source is re-activated. 00 Bus clock 01 (Bus clock)/2 10 Alternate clock (ALTCLK) 11 Asynchronous clock (ADACK)

Inoltre, si imposti il registro ADC0_SC3 in modo che l'ADC esegua una conversione in **Continuous mode**.

ADCx_SC3 field descriptions (continued)

Field	Description
7 CAL	Calibration Begins the calibration sequence when set. This field stays set while the calibration is in progress and is cleared when the calibration sequence is completed. CALF must be checked to determine the result of the calibration sequence. Once started, the calibration routine cannot be interrupted by writes to the ADC registers or the results will be invalid and CALF will set. Setting CAL will abort any current conversion.
6 CALF	Calibration Failed Flag Displays the result of the calibration sequence. The calibration sequence will fail if SC2[ADTRG] = 1, any ADC register is written, or any stop mode is entered before the calibration sequence completes. Writing 1 to CALF clears it. 0 Calibration completed normally. 1 Calibration failed. ADC accuracy specifications are not guaranteed.
5–4 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
3 ADCO	Continuous Conversion Enable Enables continuous conversions. 0 One conversion or one set of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion. 1 Continuous conversions or sets of conversions if the hardware average function is enabled, that is, AVGE=1, after initiating a conversion.
2 AVGE	Hardware Average Enable Enables the hardware average function of the ADC. 0 Hardware average function disabled. 1 Hardware average function enabled.
1–0 AVGS	Hardware Average Select Determines how many ADC conversions will be averaged to create the ADC average result. 00 4 samples averaged. 01 8 samples averaged. 10 16 samples averaged. 11 32 samples averaged.

Il registro ADC0_SC10 dovrà essere settato per l'acquisizione in **Single-end mode** sul canale **DADP0**, con **Interrupt abilitato**.

ADCx_SC1*n* field descriptions (continued)

Field	Description
	This is a read-only field that is set each time a conversion is completed when the compare function is disabled, or SC2[ACFE]=0 and the hardware average function is disabled, or SC3[AVGE]=0. When the compare function is enabled, or SC2[ACFE]=1, COCO is set upon completion of a conversion only if the compare result is true. When the hardware average function is enabled, or SC3[AVGE]=1, COCO is set upon completion of the selected number of conversions (determined by AVGS). COCO in SC1A is also set at the completion of a calibration sequence. COCO is cleared when the respective SC1n register is written or when the respective Rn register is read. 0 Conversion is not completed. 1 Conversion is completed.
6 AIEN	Interrupt Enable Enables conversion complete interrupts. When COCO becomes set while the respective AIEN is high, an interrupt is asserted. 0 Conversion complete interrupt is disabled. 1 Conversion complete interrupt is enabled.
5 DIFF	Differential Mode Enable Configures the ADC to operate in differential mode. When enabled, this mode automatically selects from the differential channels, and changes the conversion algorithm and the number of cycles to complete a conversion. 0 Single-ended conversions and input channels are selected. 1 Differential conversions and input channels are selected.
4–0 ADCH	Input channel select Selects one of the input channels. The input channel decode depends on the value of DIFF. DAD0-DAD3 are associated with the input pin pairs DADPx and DADMx. **NOTE:** Some of the input channel options in the bitfield-setting descriptions might not be available for your device. For the actual ADC channel assignments for your device, see the Chip Configuration details. The successive approximation converter subsystem is turned off when the channel select bits are all set, that is, ADCH = 11111. This feature allows explicit disabling of the ADC and isolation of the input channel from all sources. Terminating continuous conversions this way prevents an additional single conversion from being performed. It is not necessary to set ADCH to all 1s to place the ADC in a low-power state when continuous conversions are not enabled because the module automatically enters a low-power state when a conversion completes. 00000 When DIFF=0, DADP0 is selected as input; when DIFF=1, DAD0 is selected as input. 00001 When DIFF=0, DADP1 is selected as input; when DIFF=1, DAD1 is selected as input. 00010 When DIFF=0, DADP2 is selected as input; when DIFF=1, DAD2 is selected as input. 00011 When DIFF=0, DADP3 is selected as input; when DIFF=1, DAD3 is selected as input. 00100 When DIFF=0, AD4 is selected as input; when DIFF=1, it is reserved. 00101 When DIFF=0, AD5 is selected as input; when DIFF=1, it is reserved. 00110 When DIFF=0, AD6 is selected as input; when DIFF=1, it is reserved. 00111 When DIFF=0, AD7 is selected as input; when DIFF=1, it is reserved. 01000 When DIFF=0, AD8 is selected as input; when DIFF=1, it is reserved. 01001 When DIFF=0, AD9 is selected as input; when DIFF=1, it is reserved. 01010 When DIFF=0, AD10 is selected as input; when DIFF=1, it is reserved. 01011 When DIFF=0, AD11 is selected as input; when DIFF=1, it is reserved. 01100 When DIFF=0, AD12 is selected as input; when DIFF=1, it is reserved. 01101 When DIFF=0, AD13 is selected as input; when DIFF=1, it is reserved. 01110 When DIFF=0, AD14 is selected as input; when DIFF=1, it is reserved. 01111 When DIFF=0, AD15 is selected as input; when DIFF=1, it is reserved. 10000 When DIFF=0, AD16 is selected as input; when DIFF=1, it is reserved. 10001 When DIFF=0, AD17 is selected as input; when DIFF=1, it is reserved. 10010 When DIFF=0, AD18 is selected as input; when DIFF=1, it is reserved. 10011 When DIFF=0, AD19 is selected as input; when DIFF=1, it is reserved. 10100 When DIFF=0, AD20 is selected as input; when DIFF=1, it is reserved. 10101 When DIFF=0, AD21 is selected as input; when DIFF=1, it is reserved. 10110 When DIFF=0, AD22 is selected as input; when DIFF=1, it is reserved. 10111 When DIFF=0, AD23 is selected as input; when DIFF=1, it is reserved. 11000 Reserved. 11001 Reserved. 11010 When DIFF=0, Temp Sensor (single-ended) is selected as input; when DIFF=1, Temp Sensor (differential) is selected as input. 11011 When DIFF=0,Bandgap (single-ended) is selected as input; when DIFF=1, Bandgap (differential) is selected as input. 11100 Reserved. 11101 When DIFF=0,V_{REFSH} is selected as input; when DIFF=1, $-V_{REFSH}$ (differential) is selected as input. Voltage reference selected is determined by SC2[REFSEL]. 11110 When DIFF=0,V_{REFSL} is selected as input; when DIFF=1, it is reserved. Voltage reference selected is determined by SC2[REFSEL]. 11111 Module is disabled.

È importante ricordare che, di default, il convertitore A/D utilizza come riferimento di tensione la tensione unipolare fornita dalla scheda KL25Z, $\mathbf{V_{REFH} = 3.3V}$.

Infine, nel vettore interruzioni, è necessario agire sul registro NVIC_ISER, asserendo a 1 il bit ADC0_IRQn. Questo può essere fatto richiamando la libreria *NVIC_EnableIRQ e passando come argomento ADC0_IRQn*:

NVIC_EnableIRQ(ADC0_IRQn);

Da riportare nella relazione:

- Codice implementato con descrizione sintetica.

Passo 4. Inizializzazione LED e DAC

Per inizializzare il LED, si eseguano le configurazioni dei seguenti registri:

- Nel registro SIM->SCGC5, impostare ad 1 il campo PORTD (bit numero 12) al fine di abilitare il clock sulla porta D;
- Nel registro PORTD->PCR[1] impostare ad 001 il campo MUX (10-8) al fine di selezionare PTD1 come porta GPIO;
- Nel registro FPTD->PDDR impostare ad 1 il bit 1, in modo che tale porta sia configurata come output;
- Nel registro FPTD->PCOR impostare ad 1 il bit 1. Questo consente di azzerare PTD1 e di accendere il LED (a tal proposito si ricorda che i LED presenti sulla scheda di KL25Z sono già configurati con una rete di pull-up).

Per inizializzare il convertitore D/A, si eseguano le seguenti configurazioni dei registri:

- Nel registro SIM->SCGC6, impostare ad 1 il campo DAC0 (31) per abilitare il clock sul DAC (si rammenta che questo registro viene modificato anche durante la configurazione dell'ADC e per tal motivo è necessario utilizzare l'operatore di assegnazione **|=**);
- Nel registro DAC0->C0, impostare ad 1 i campi DACEN (7) e DACTRGSEL (5) per abilitare il DAC e utilizzare un trigger di tipo software.
- Nel registro DAC0->C1, impostare ad 1 i campi DACBFEN (0) al fine di abilitare il funzionamento "Buffer Mode".
- Nel registro DAC0->C2, impostare ad 1 i campi DACBFUP (0) per impostare la posizione "top" del buffer ad 1. Si noti che il buffer del DAC è composto da due soli registri (in posizione 0 e posizione 1).

Da riportare nella relazione:

- Codice implementato con descrizione sintetica.

Passo 4. Implementazione routine di gestione interrupt ADC

Il nome di questa routine deve coincidere con quello riportato nel passo 1 (ADC0routine). In questa fase è sufficiente leggere il valore acquisito dall'ADC e utilizzarlo come ingresso del DAC, utilizzando il microcontrollore in modalità "buffer".

Il dato può essere letto dall'ADC, accedendo al seguente registro:

ADC0->R[0]

Al fine di rendere più rapida l'elaborazione dei dati, possiamo avvalerci di uno o più registri del processore (r0-r12). Per fare ciò è necessario definire i registri, all'inizio del file, come:

register unsigned int r0 __asm("r0");

Si ricorda che il suffisso **__asm** può essere utilizzato ogni qualvolta sia necessario inserire del codice in linguaggio assembly.

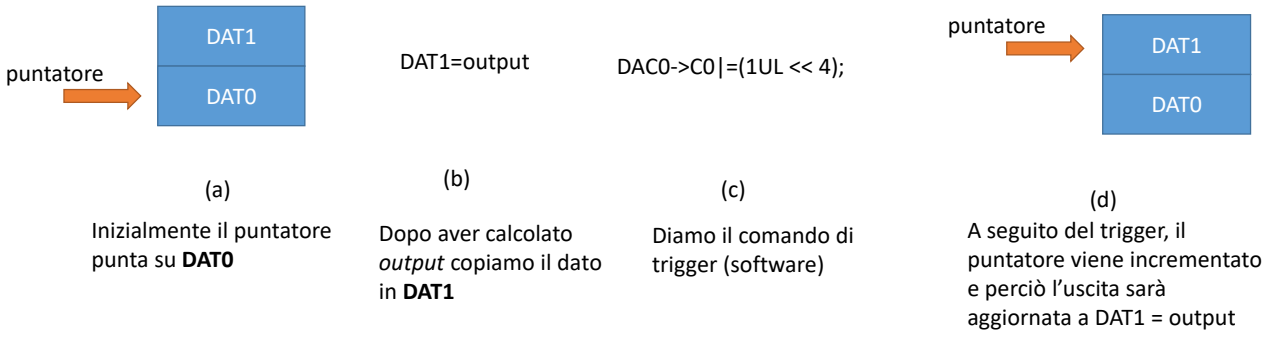
A questo punto possiamo trasferire il valore letto su *ADC0->R[0]* al registro *r0*, semplicemente come:

r0=ADC0->R[0];

Il contenuto di *r0* deve essere a sua volta copiato nei registri *DAC0->DAT->DATL* e *DAC0->DAT->DATH* del DAC. È opportuno osservare che avendo attivato la modalità buffer, saranno presenti due coppie di registri su cui scrivere:

- *DAC0->DAT[0].DATL* e *DAC0->DAT[0].DATH*
- *DAC0->DAT[1].DATL* e *DAC0->DAT[1].DATH*

Che permetteranno di scrivere rispettivamente nella locazione 0 ed 1 del buffer. La figura successiva descrive schematicamente i passi da seguire per il riempimento del buffer



(a)
Inizialmente il puntatore punta su **DAT0**

(b)
Dopo aver calcolato *output* copiamo il dato in **DAT1**

(c)
Diamo il comando di trigger (software)

(d)
A seguito del trigger, il puntatore viene incrementato e perciò l'uscita sarà aggiornata a DAT1 = output

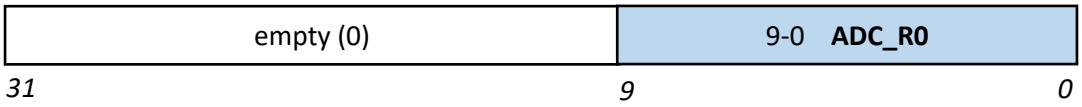
Si ricorda che per verificare la posizione del puntatore è sufficiente implementare un costrutto **if**:

```
if ((DAC0->C2 & DAC_C2_DACBFRP_MASK)==0)
```

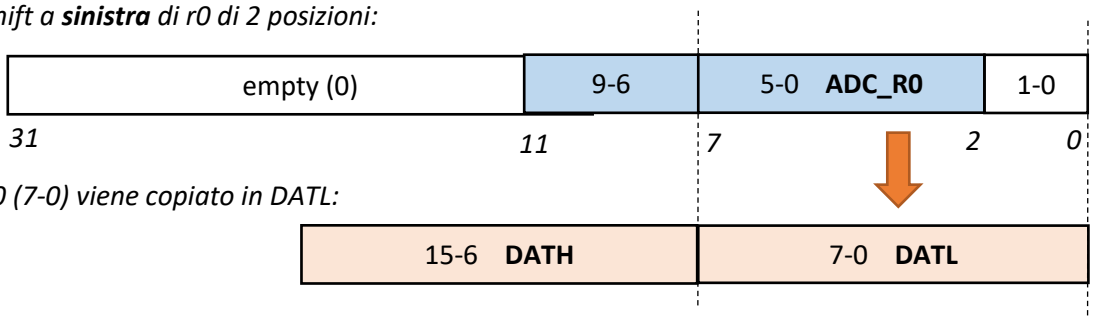
che consentirà di testare il bit DACBFRP presente all'interno del registro DAC0_C2 (ovvero il bit in cui è contenuto il valore del puntatore).

Si noti che la conversione A/D è eseguita con una risoluzione di 10-bit, mentre la conversione D/A è fissa a 12-bit. Le operazioni da implementare per copiare il contenuto di r0 su DATL e DATH sono schematicamente riportate:

contenuto iniziale di r0:

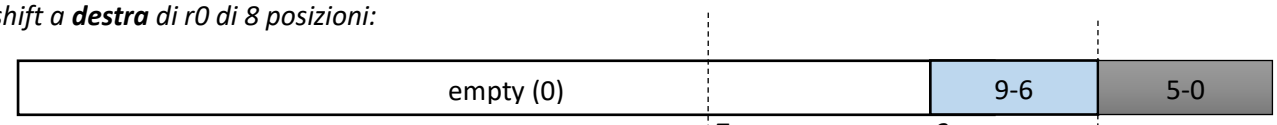


*shift a **sinistra** di r0 di 2 posizioni:*

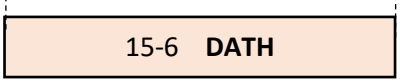


r0 (7-0) viene copiato in DATL:

*shift a **destra** di r0 di 8 posizioni:*



r0 (7-0)viene copiato in DATH:



Il dato deve essere prima "*left adjusted*", in modo da utilizzare i bit più significativi del convertitore:

```
r0=r0 << 2
```

A questo punto è possibile copiare gli 8-bit meno significati di *r0* in *DATL*

```
DAC0->DAT[0].DATL =r0;        //in questo caso scriviamo nella posizione 0 del buffer. Per scrivere
                              nella posizione 1 si utilizza invece: DAC0->DAT[1].DATL
```

Successivamente, per accedere ai 4 bit più significativi di *r0* (da copiare in *DATH*) è necessario eseguire uno shift a destra di 8 posizioni e poi copiare:

```
r0=r0 >> 8;
DAC0->DAT[0].DATH=r0;         //oppure DAC0->DAT[1].DATH
```

Per fornire il comando di trigger (e quindi incrementare del puntatore) si usi il seguente codice:

```
DAC0->C0 |= (1UL << 4);
```

Si verifichi la funzionalità del progetto, compilandolo ed eseguendo il *debug* dello stesso. A tale scopo, si inserisca un *breakpoint* nella routine *ADC0routine* e si verifichino le configurazioni dei registri inerenti alle periferiche ADC, DAC, NVIC, MCG.

Da riportare nella relazione:

- Codice implementato con descrizione sintetica;

Passo 5. Test del microcontrollore in modalità buffer.

Innanzitutto, si localizzino il terminale di ingresso dell'ADC (**PTE20**), il terminale di uscita del DAC (**PTE30**) ed un terminale di **GND** (tali informazioni sono evidenziate sul retro della scheda).

Utilizzando degli appositi cavi, si applichi all'ingresso dell'ADC una sinusoide (generata mediante generatore di funzioni) con frequenza di **10kHZ**, ampiezza picco-picco **2V** e offset di **1.5V**.

Note:

- Prima di collegare l'uscita del generatore di funzioni al microcontrollore verificare mediante oscilloscopio che il segnale abbia le caratteristiche desiderate, ed in particolare che esso non fuoriesca dal range 0-3.3V.

- È necessario configurare il generatore di funzioni in modalità "alta impedenza".

Sia il segnale di ingresso che l'uscita dell'ADC dovranno essere visualizzati e confrontati mediante oscilloscopio. È inoltre necessario eseguire una stima del ritardo introdotto dal microcontrollore. A tal fine, utilizzando il datasheet del microcontrollore si calcoli il tempo di campionamento del DAC e si esegua una sommaria stima del tempo richiesto per l'esecuzione dell'interrupt dell'ADC.

Da riportare nella relazione:

- Immagini dell'oscilloscopio da cui è possibile evincere le informazioni di ampiezza e frequenza delle due sinusoidi sovrapposte;
- Una misura sperimentale del ritardo tra le due sinusoidi ed un confronto (approssimativo) con il valore atteso.

Passo 6. Implementazione del filtro IIR

Caratteristiche richieste:

Fitro passa basso;

Frequenza di taglio: f_t= 10kHz;

Attenuazione a 100kHz: 20dB.

Guadagno unitario.

Noto il tempo di campionamento T_c e dato un filtro IIR nella forma:

$y(k) = a \cdot y(k-1) + b \cdot x(k)$,

con $b = \frac{T_c}{T_c+\tau} =$ $a = 1 - b$

si calcolino i coefficienti *a* e *b* per ottenere la frequenza di taglio desiderata.

Essendo i due coefficienti frazionari (<1), e scegliendo di considerare dati a 16 bit, possiamo lavorare normalizzando agli interi. Per fare ciò è sufficiente moltiplicare i coefficienti calcolati per 2^{16}.

Note: si ricorda che la normalizzazione all'intero è equivalente all'adozione della notazione in virgola fissa in cui 2^{16} rappresenta proprio il fattore di scala.

Si proceda all'implementazione del codice eseguendo in sequenza il prodotto $b \cdot x(k)$, il prodotto $a \cdot y(k-1)$ e poi la somma dei due. È necessario definire una variabile *output* in cui accumulare il valore di uscita (che sarà salvata nella SRAM):

static unsigned int output=0;

I due prodotti dovranno essere memorizzati in due registri (r0 e r1) precedentemente definiti. Si ricordi inoltre, che a valle di ogni moltiplicazione è necessario shiftare il risultato a destra di 16 posizioni (equivalente a dividere per 2^{16} e perciò ad annullare la precedente normalizzazione).

Una volta completate le due moltiplicazioni sarà sufficiente accumulare la somma nel registro r0, copiare il risultato in *output* (affinché sia disponibile per il successivo interrupt) e trasferire il contenuto di r0 sul DAC (implementando le opportune precauzioni descritte al passo 4).

In alternativa all'utilizzo di una variabile *output* salvata nella memoria RAM, è possibile prevedere l'utilizzo di un registro (r3-r11) per memorizzare y(k).

Si compili il programma e si verifichi il corretto funzionamento.

Da riportare nella relazione:

- I calcoli eseguiti per la determinazione dei coefficienti del filtro e la gestione della normalizzazione all'intero;
- Il codice implementato con una descrizione sintetica.

Passo 7. Test del filtro IIR

Analogamente al passo 5, si effettuino gli opportuni collegamenti per visualizzare sull'oscilloscopio le forme d'onda in ingresso e in uscita.

Si costruisca un diagramma di bode "sperimentale", variando la frequenza del segnale in ingresso e misurando l'attenuazione e lo sfasamento dei segnali.

Da riportare nella relazione:

- Immagini dell'oscilloscopio da cui è possibile evincere le informazioni di ampiezza e frequenza delle sinusoidi alle frequenze di 1Hz, 10kHz e 100kHz;
- Diagramma di bode "sperimentale" (modulo e fase), aggiungendo ulteriori punti in frequenza a quelli sopra elencati;
- Misura dello sfasamento a 10kHz e confronto con il dato atteso (dato dalla somma tra il tempo di campionamento, ritardo introdotto dal codice e sfasamento intrinseco al filtro passa basso).

Attività facoltative:

1) Si implementi un filtro **FIR** a **media mobile** con caratteristiche simili (f_t=10zHz, attenuazione a 100kHz = 20dB). Suggerimenti:

   a) Per determinare il numero di coefficienti richiesto, si faccia uso della funzione matlab **freqz** che consente di tracciare il diagramma di bode nel caso di sistemi discreti.

   [h,f]=freqz(b,a,k,fs)
   b: vettore coefficienti b;
   a: vettore coefficienti a (il primo coefficiente a è inteso come coefficiente di y(k), per tale motivo nel caso di filtro FIR sarà necessario impostare a=1);
   k: numero di punti in cui discretizzare l'intervallo di frequenza 0-fs;
   fs: frequenza di campionamento

   h: risposta in frequenza del filtro
   f: vettore delle frequenze

   b) Si preveda un opportuno array per memorizzare gli N ingressi precedenti.
   c) Piuttosto che generare diverse versioni del codice, è possibile utilizzare delle direttive che consentono la compilazione condizionata dal codice.

```
#if (condizione)
        Istruzione;
        ….
#elif (condizione)
        Istruzione;
        ….
#else
        Istruzione;
        ….
#endif
```

2) Si provi ad utilizzare il **DAC senza buffer** e si verifichi il funzionamento complessivo del filtro. Questo può essere fatto usando solo la coppia di registri *DAC0->DAT[0].DATL* e *DAC0->DAT[0].DATH* e lasciando fissa la posizione del puntatore su di essi.