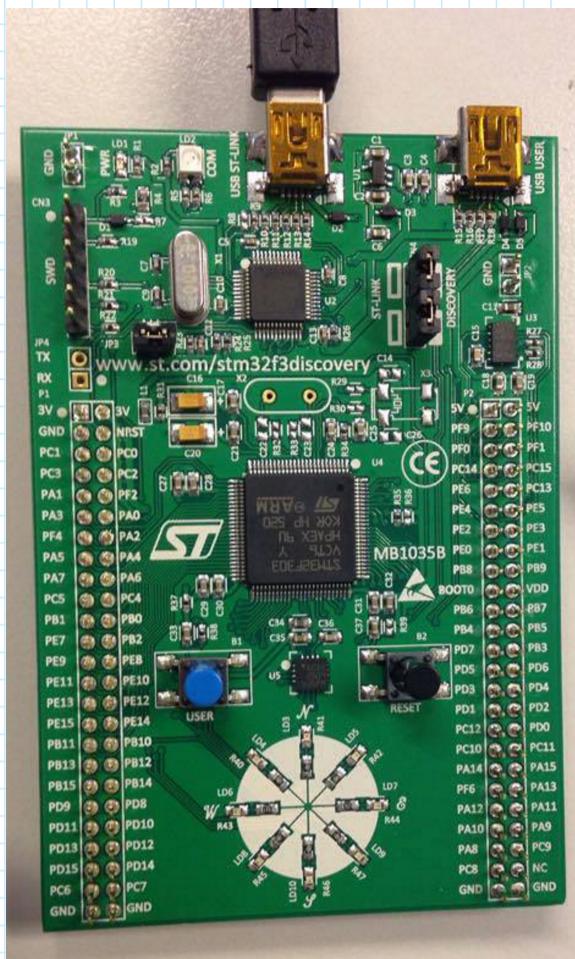
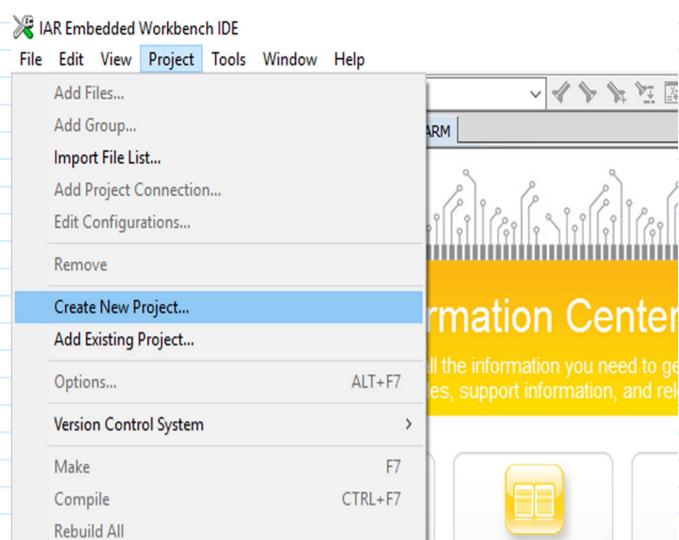


# LABORATORIO 1

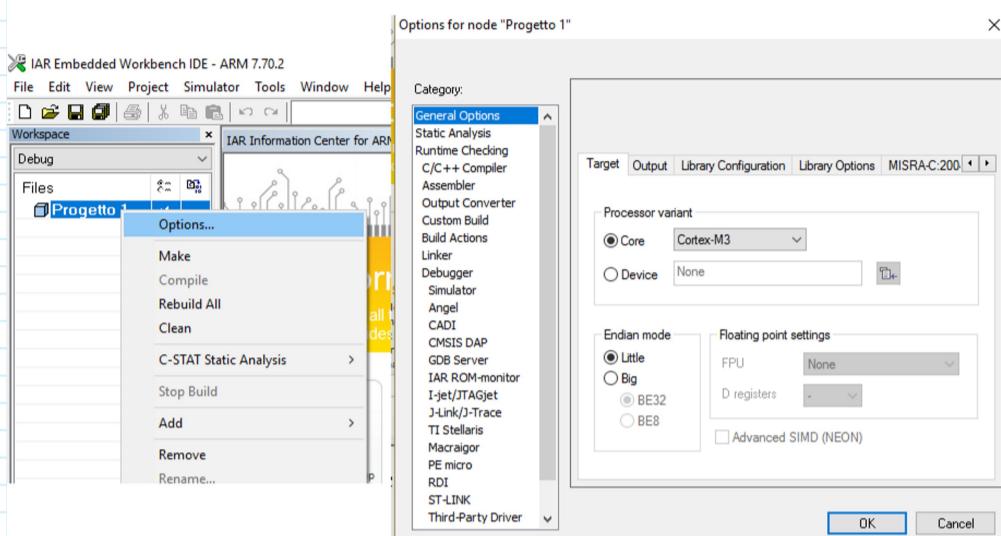
lunedì 26 settembre 2016





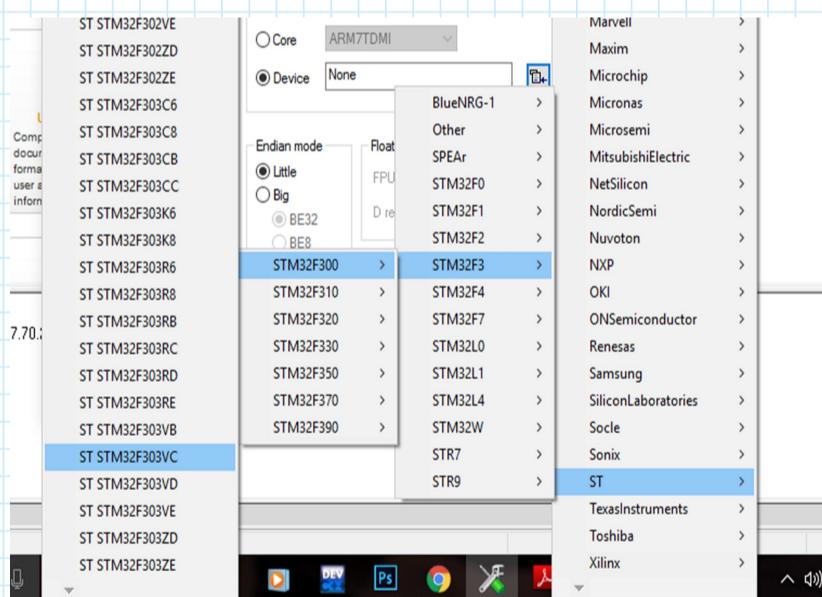
Configurazione del progetto (tutto scritto nel documento GettingStarted).

- 1) Cliccare col tasto destro sul progetto e scegliere "Options", si aprirà una finestra.



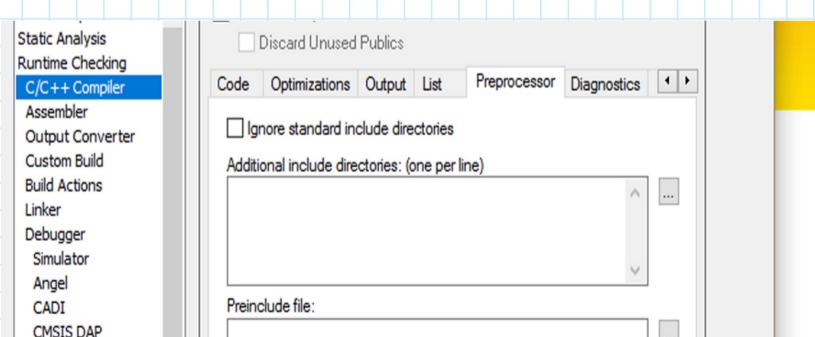
- 2) Iniziamo a dire chi è il microcontrollore.

Vado su General Options->Target->Device-> Apro il menù a tendina e cerco la ST (casa costruttrice), all'interno della ST ci sono tante famiglie di microcontrollori ARM la nostra è STM32F300 della famiglia degli F300 il nostro microcontrollore è F303XC, dove X sta per qualunque lettera (le versioni più recenti hanno la lettera V invece della X).



- 3) Dobbiamo ora configurare C/C++ Compiler.

Scorriamo le pagine fino a quella del preprocessor, noi non inseriremo degli header file (file.h) nel nostro progetto, qualora dovessimo inserirli il compilatore li va a cercare nella sua cartella include di default, qualora dobbiamo usare degli altri .h che si trovano in altre cartelle, le cartelle dobbiamo indicarle qui. Quindi in questo riquadro dovranno essere indicate le cartelle di default in cui si trovano gli header file che useremo, a parte quello di default dell'ambiente.

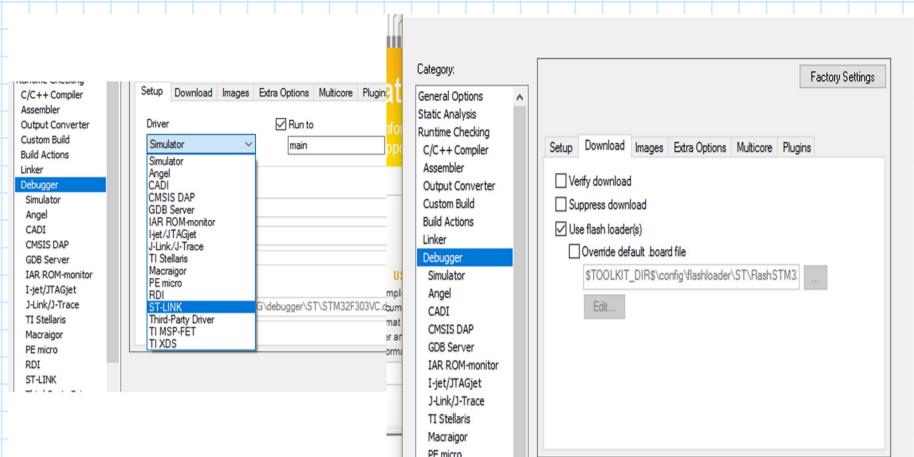


- 4) Configuriamo il Debugger.

Il debug è quel processo per cui andiamo a testare il nostro codice, cerchiamo eventuali errori e cerchiamo di rimuoverli. Si può fare in due modi.

- a) In simulazione (impostazione di default): questo ambiente simula il microcontrollore e sul computer si esegue il programma e si vede se si fanno degli errori (il microcontrollore non è collegato, è tutto simulato io faccio eseguire le istruzioni).
- b) ST-Link: Si adopera il collegamento usb per scaricare il programma sulla

flash del microcontrollore e il debug si fa man mano che il microcontrollore esegue le operazioni. Per fare questo non basta cliccare su ST-LINK, ma bisogna andare sulla pagina Download e spuntare su "Use flash loader" (Scarica il programma sulla flash del microcontrollore).



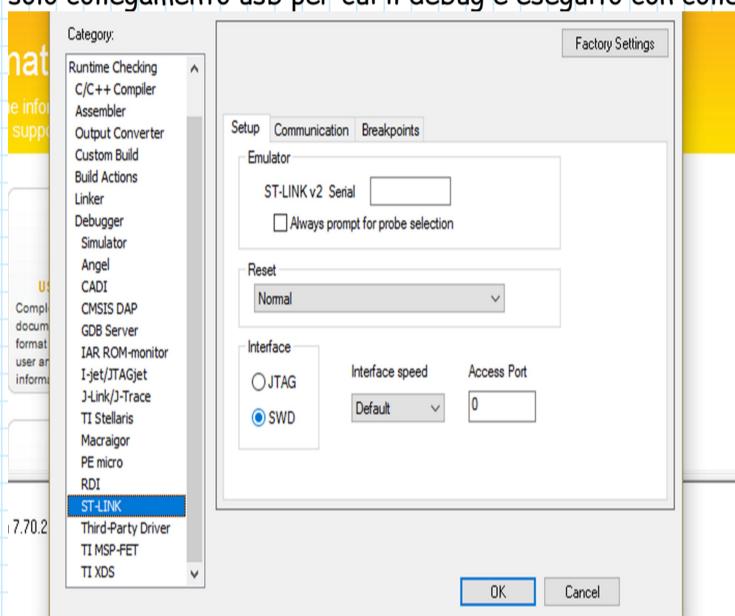
- 5) Configurare ST-LINK, collegamento usb tra il microcontrollore e il computer (nel caso in cui si sceglie per il debug st-link, se si sceglie simulator, questo passaggio non va fatto).

Ci sono due modi per fare il debug di circuiti integrati:

a) Standard Jtag

B) Tramite il "collegamento seriale"

Su questa scheda utilizzeremo il collegamento seriale, non essendoci un connettore Jtag, o quanto meno non viene adoperato, c'è un solo collegamento usb per cui il debug è eseguito con collegamento seriale. Quindi clicchiamo su SWD.



Fine configurazione.

Per avere un'idea su cosa c'è sul microcontrollore, apriamo il DataSheet.

A pag. 11 c'è uno schema a blocchi su cosa c'è nel chip.

C'è il microcontrollore vero e proprio, intorno ci sono una marea di moduli che eseguono operazioni diverse, che vengono chiamati periferiche. Ci sono due convertitori analogico-digitali, delle porte di general purpose i/o (porte di input e output), ci sono dei timer, una usb 2.0fullspeed questa è la porta di comunicazione del microcontrollore verso l'esterno secondo diversi protocolli. Il microcontrollore manda comandi a tutte queste periferiche a seconda di quello che noi scriviamo nel programma.

Ora tutte le periferiche sono collegate al microcontrollore mediante 3 bus principali: AHB bus, APB2 bus, APB1 bus.

Sapere i nomi dei bus è importante perchè se si vuole far comunicare il microcontrollore con una periferica, bisogna sapere quest'ultima a quale bus è collegata, per poter stabilire il collegamento con quella periferica.

Apriamo poi il reference manual.

Il reference manual dice come far eseguire le operazioni al microcontrollore.

Come si può notare, c'è un capitolo per ogni periferica, a seconda della periferica che si dovrà operare andremo a studiare quella particolare sezione del reference manual. Ogni sezione è costituita da una introduzione, principali caratteristiche di quella periferica, descrizione di come funziona e registri.

A che servono I registri: Il microcontrollore stabilisce quello che deve fare in base al valore che trova scritto in alcuni registri di memoria., quindi per far fare delle determinate operazioni al microcontrollore dobbiamo scrivere degli opportuni valori (in base a quello che vogliamo fargli fare) in questi registri.

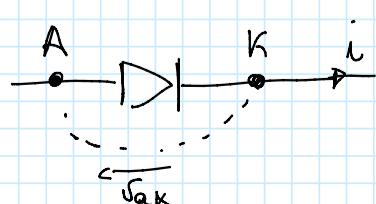
Allora per ogni periferica sono indicati I registri: cioè se vuoi adoperare una determinata periferica chi sono I registri da usare e come bisogna configuarli.

Progetto del giorno: cercare di accendere un led sulla scheda adoperando e usando la periferica general purpose i/o.

In particolare quello che faremo è una programmazione a basso livello, lavoriamo a livello hardware (siamo molto legati all'hardware), quindi nella domanda "accendere un led" bisogna chiedersi dal punto di vista circuitale come si accende un led e quindi il microcontrollore cosa deve fare.

Che cos'è un led: Un diodo che si illumina

Cos'è un diodo?



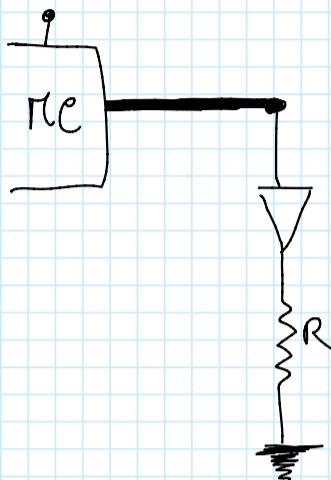
A : ANODO (+)  
K : CATODO (-)

Un diodo è un componente particolare, con un comportamento binario. I suoi morsetti si chiamano ànodo (il positivo) e càtodo (il negativo). Se la differenza di tensione (anodo-catodo) è maggiore di un certo valore di soglia ( $V_s$ ), allora il diodo è in conduzione, cioè conduce corrente, presentando una resistenza bassissima (quindi è assimilabile ad un corto circuito), dove questa resistenza, tendente a 0, è detta resistenza diretta.

Se la tensione  $V_{AK}$  è minore del valore di soglia, il diodo si dice interdetto (o in interdizione), non lascia passare corrente, perché è equivalente ad una resistenza tendente all'infinito (molto grande).

Ci sono alcuni diodi che quando sono in conduzione emettono luce -> led.

Sullo schedino ci sono otto led, collegati al microcontrollore in questo modo:



Per ogni led viene adoperata una linea (nel disegno l'ho fatta un po' più spessa delle altre per far capire quale fosse), del microcontrollore-> uno dei pin attorno al microcontrollore. Si avranno quindi otto linee del microcontrollore collegate così ad ogni led. Considerando che alla fine ho un collegamento a massa, quindi a tensione 0, allora se la linea del microcontrollore è a tensione 0, il diodo è interdetto, non conduce e il led è spento. Se questa linea va ad una tensione che è maggiore della tensione di soglia di questo led, allora circola corrente, quindi il led conduce e si accende.

Questa è una linea del microcontrollore, quindi siamo noi nel nostro programma che diciamo a che tensione deve portare questa linea. Non conviene scegliere un valore analogico, perché significherebbe che noi qui dobbiamo essere in grado di dare una tensione analogica, la cosa più semplice da fare è trattare questa linea come una linea digitale che può essere o alta (si porta ad una tensione con la quale alimento lo schedino, questo schedino è alimentato ad una tensione pari a 3.3 V) o bassa (e la tensione è 0 V).

Quindi lo scopo di oggi è quello di trattare questa linea come una linea digitale che può assumere solo due valori: logicamente può essere alta o bassa (led spento); ma fisicamente questi valori sono due valori di tensione: 3.3V e 0 (rispettivamente).

Linea alta (tensione 3.3 V) led acceso.

Linea bassa (tensione 0) led spento.

Allora per poter accendere questo led oggi dobbiamo dire al microcontrollore, che questa linea la useremo come linea digitale, che sarà un output (perchè la pilotiamo noi, la pilota il microcontrollore, è lui a decidere a che valore deve stare questa linea) e che deve avere valore alto, così accendiamo il led.

A tale scopo, oggi adopereremo la periferica general purpose I/O (GPIO), visto che voglio solo utilizzare delle linee di output (in particolare) senza fare conteggi di tempo o altro (vedi pag 11 del DataSheet). Il microcontrollore ha 100 linee, per questioni di semplicità nella programmazione e per risparmiare memoria, queste linee vengono raggruppate in quelle che nel reference manual vengono chiamate **GPIO PORT**:

- GPIO PORT A: PA [15; 0], significa che 16 linee vengono raggruppate tutte insieme in un unico blocco che chiamiamo GPIO PORT A. Queste linee vengono chiamate PA0, PA1, PA2..., PA15.
  - GPIO PORT B: PB [15; 0], periferica di 16 linee che indicherò con I nomi PB0, PB1,...,PB15.
  - .....
  - GPIO PORT F: Diversa dalle altre, poichè raggruppa solo otto linee chiamate PF0, PF1,..,PF7.

Obiettivo del giorno : Adoperare la periferica GPIOE e in particolare vogliamo indicare come digitale e alzare la linea PE8.

Visto che so quale periferica voglio adoperare, apro il reference manual e vado nella sezione General Purpose IO (pag. 134 e in particolare 135 per quello che sto per scrivere).

Perchè si chiama General Purpose ? Perchè a seconda di come configuriamo I registri, il pin può essere un output (scrivo se la voglio alta o bassa, nell'output data register, e automaticamente viene collegato il valore di tensione desiderato al pin); oppure questo pin può essere un input (il circuito esterno stabilisce se la linea è alta o bassa, noi possiamo solo leggerne il valore, non impostarlo), in questo caso il livello della linea, viene portato all'input data register (se ho selezionato la linea come digitale, perchè volendo potrei anche impostarla come digitale). Potrei anche usare la linea come una funzione alternativa.

Il microcontrollore può fare tante cose, così tante che le 100 linee che ha per interfacciarsi con l'esterno non gli bastano, quindi ogni linea può collegare il pin esterno al microcontrollore a diverse periferiche, perciò le linee possono essere output, input, digitale, etc....

Osservazione : cosa voglio far fare a questa linea del modulo GPIOE, devo scriverlo negli opportuni registri, per cui se voglio usare questo modulo, devo conoscere tutti I registri, capire quali devo configurare e che valori devo mettere.

Il primo registro che incontriamo (pag.143 reference manual) è il MODE REGISTER (`GPIOx_MODER`, dove x sta per la lettera della porta che vogliamo adoperare), per cui il primo registro in cui andiamo a scrivere è `GPIOE_MODER`, serve ad impostare il modo delle linee della periferica `GPIOE`, in particolare `GPIE` contiene 16 linee e considerando che i registri sono a 32 bit (perchè il microcontrollore è a 32 bit, quindi le locazioni di memoria sono a 32 bit), possiamo suddividere il registro in coppie di bit che si chiamano `Moder0`, `Moder1`..., `Moder15`.

I due bit di Moder0 definiscono cosa deve fare la linea PE0, quelli di Moder1 per la linea PE1, etc..

Per ogni linea si hanno quindi due bit, perchè si hanno quattro scelte.

- 00: la linea è un input digitale, stato di reset, a reset tutte le linee sono di input digitale.
- 01: la linea è un output digitale
- 10: funzione alternativa
- 11: modalità analogica

Il nostro scopo è quello di prendere PE8 come output digitale, devo scrivere 01 nella coppia di bit indicata con MODER8. In bit il valore di MODER8 è (Partendo da moder15 fino a moder0) 000000000000000010000000000000000000000 Come scrivo questo valore?

Vari modi:

- GPIOE\_MODER= 0X00010000

Notazione esadecimale, comoda perchè ogni cifra esadecimale è un gruppo di quattro bit, in questo caso ho un valore a 32 bit con tutti I bit a 0 tranne un uno sul 16 (0X è per far capire al compilatore che è la notazione esadecimale), quindi devo trovarmi 8 cifre esadecimali.

- GPIOE\_MODER= 1<<16 (Uno shiftato a sinistra di 16 posizioni)

Comoda quando tutti I bit sono zero tranne uno come in questo caso.

- Notazione decimale.

Quindi per ora abbiamo capito cosa dover scrivere per rendere la linea PE8 un output digitale, resta da capire dove scrivere queste cose: nell'indirizzo del registro GPIOE\_MODER, per poter scrivere al suo interno questo valore. Questo è possibile perchè il microcontrollore ha la memoria organizzata con I registri delle periferiche a delle locazioni di memoria fisse.

Nel reference manual sotto al titolo di ogni registro, si troverà sempre un Address offset (nel caso di moder è 0).

Address offset: I registri relativi alle periferiche sono organizzati in sezioni che occupano una determinata area di memoria. Gli indirizzi relativi ad ogni periferica partono da un indirizzo che si chiama INDIRIZZO BASE.

Andando a pagina 42 del reference manual, si troverà una tabella "mappa della memoria e indirizzi dei confini dei registri". Questa tabella dice che tutti I registri relativi ai convertitori analogico-digitali 3 e 4, si trovano tra l'indirizzo esadecimale 0x5000 0400 e l'indirizzo 0x5000 07FF.

Tutti i registri relativi a GPiOE partono dall'indirizzo base : 0x48001000

→ INDIRIZZO BASE DELLA PERIFERICA GPiOE: 0x48001000

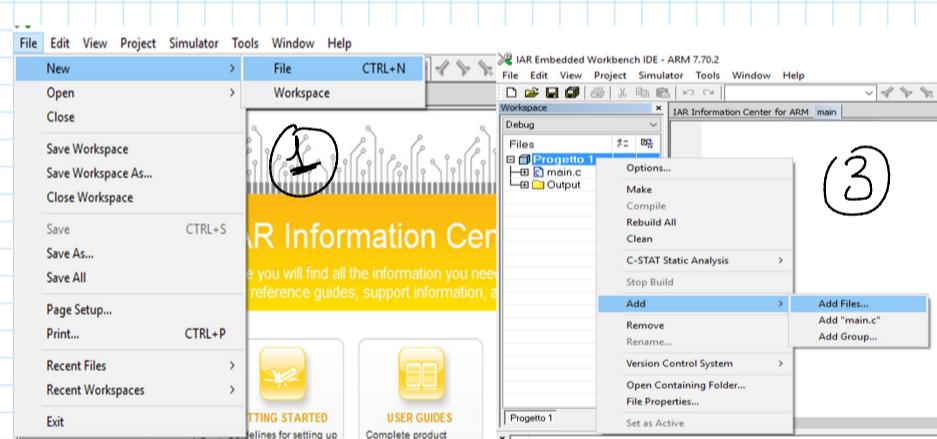
Si chiama indirizzo base perchè da questo partono gli indirizzi di tutti I registri di quella periferica (nel nostro caso GPiOE).

Tornando a pag 143, si può osservare che per MODER non è l'unico registro di GPIO, ce ne sono vari. L'informazione che abbiamo ottenuto è che quello di moder è l'indirizzo di partenza, per tutti gli altri registri è riportato l'offset rispetto all'indirizzo base.

Ora a partire dal registro base, ho l'offset per ogni indirizzo. Quindi l'indirizzo del mode register sarà data dalla somma dell'indirizzo base e l'address offset (per mode è 0x00, perciò l'indirizzo di mode\_register è il primo da cui parte tutto).

Abbiamo l'indirizzo e sappiamo cosa scrivere nel suo interno, per dire al microcontrollore che deve mettere la linea PE8 come output digitale → torno al progetto su iar.

Prima cosa da fare è quella di creare un file e di aggiungerlo al progetto, dopo averlo salvato (io ho chiamato questo file main.c).



Inizio programmazione:

```
unsigned int*puntatore;  
/* dichiaro il puntatore perchè conosco l'indirizzo di memoria del registro che mi serve e so inoltre cosa voglio scrivere all'interno del registro. I registri sono ovviamente senza segno*/
```

```
void main ()  
{  
    puntatore=0x48001000; //indirizzo al quale voglio scrivere
```

```
*puntatore=0x00010000;  
/* Scrivo poi I bit che mi servono per dire che la mia linea deve essere un  
output digitale, all'indirizzo puntato dal puntatore, che sarebbe l'indirizzo di  
di mode8.*/  
While (1);  
/* Il microcontrollore esegue continuamente quello che viene scritto nella sua  
memoria flash, se voglio che venga eseguita una parte di codice e poi voglio  
che il microcontrollore si fermi, devo mettere un'istruzione che lo faccia  
bloccare. L'istruzione perfetta, in questo caso e quasi sempre è while (1), cioè
```

finchè 1 è maggiore di 0. In pratica che cosa succede, il microcontrollore esegue le prime due istruzioni e poi si allunga nel while, in questo modo si ha la certezza che le prime due istruzioni, non verranno più eseguite, in questo senso il microcontrollore è stato fermato. \*/

/\*Osservo che se eseguo ora mi da un errore. Infatti il puntatore è un puntatore ad interi senza segno, ma le costanti vengono interpretate come interi con segno.

Errore: un valore di tipo intero non può essere assegnato ad un entità di tipo puntatori ad interi senza segno. Per questo bisogna convertire puntatore= 0x..... A

puntatore= (unsigned int\*)0x480010000; -> Questo è il valore corretto da mettere nel programma. \*/

}

Una volta scritto il programma, bisogna compilare. Cliccare su Download e Debug (freccia verde) per poter compilare e scaricare nella flash del microcontrollore.



Clicchiamo sul debugg.

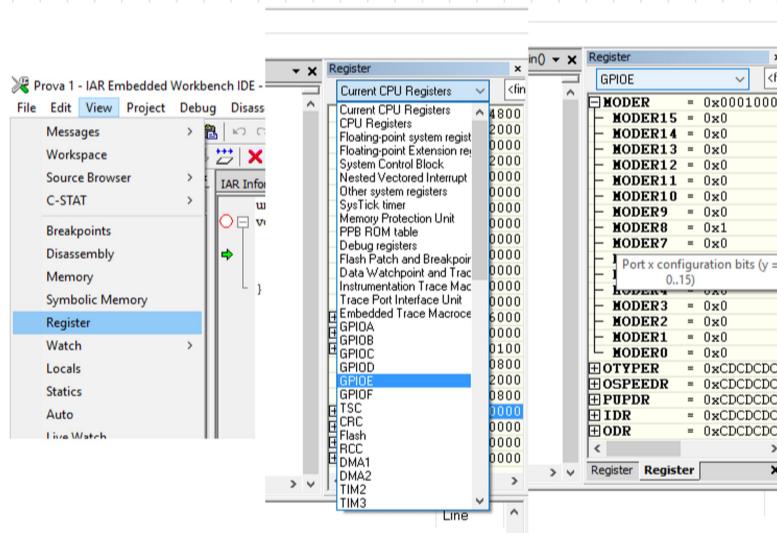
Vediamo gli strumenti di debugg:

1. Reset (resettere il programma)
2. Step over eseguire un'istruzione alla volta
3. Step in to entra in una funzione
4. Step out esci dalla funzione
5. Next Statement (prossima condizione)
6. Run to Cursor Esegui fino al cursore
7. Go Esegui
8. Infine la X per interrompere il debug



Per vedere se il programma fa quello che vogliamo, andiamo nella sezione view e possiamo vedere varie cose tra cui:

- I valori che assumono le variabili con il watch, meglio del watch è il live watch, che aggiorna il valore delle variabili real time (meglio del watch perchè per vedere il valore della variabile del watch bisogna mettere in pausa l'esecuzione del programma).
- I registri, cosa importante perchè da la possibilità di vedere I registri e verificare se li sto configurando come penso. Quindi apro il menù a tendina dove sono elencate tutte le periferiche, io sto lavorando su GPIOE e in particolare su Moder. Espandendo Moder, riusciamo a vedere le coppie di bit su ogni linea. Se le cose vanno come dovrebbero andare, allora moder8 dovrebbe passare da 0 a 1. Quindi esegui passo passo.



A lezione abbiamo collegato il pc al microcontrollore, e andando ad eseguire La coppia di bit di moder8 non è passata da 0 a 1.

Il problema era dovuto al fatto che per questioni di risparmio energetico, tutte le periferiche sono disabilitate, ma allora per fare in modo che il microcontrollore possa comunicare con una periferica BISOGNA preventivamente ABILITARLA (Ricorda, quando il registro non ti calcola dopo che gli hai scritto su, allora probabilmente non hai abilitato la periferica).

Come si abilita? Sezione del reference manual Reset and clock control (RCC) a pag. 94 e per I registri da pag. 105.

In RCC lavoreremo solo con tre registri: AHB enable register, APB2 enable register, APB1 enable register.

I nomi dei tre registri, ricordano quelli dei tre bus principali, non a caso, infatti ogni registro (di questi tre) contiene il bit per attivare la periferica collegata a quel bus. GPIOE è collegata al bus AHB, quindi andrà a cercare, per abilitare la periferica, il registro AHB enable register sul reference manual (pag. 116). In questo registro si possono abilitare I convertitori analogico-digitali e tutti I moduli GPIO, che chiama IOPF...IOPE....

Il bit sul quale voglio agire io è il bit 21, devo quindi abilitare il clock della porta IOE, in particolare se metto questo bit a 1-> Abilito il clock di GPIOE. Abilitare il clock significa abilitare la comunicazione tra il microcontrollore e la periferica.

Allora prima ancora di agire sul MODER di GPIOE bisogna abilitare GPIOE , per fare ciò dobbiamo andare sul registro AHB enable register e dobbiamo fare in modo che si alzi il bit 21.

Possiamo scrivere 1<<21 oppure la corrispondente cifra esadecimale: 0x00200000

Ma dove dobbiamo scriverlo?

L'indirizzo base del blocco di registri RCC = 0x4002 1000 (pag 43, AHB1, peripheral RCC). Tutti gli indirizzi relativi ad rcc stanno in questo intervallo 0x4002 1000 - 0x4002.

AHB enable register ha un offset di 0x14--> Indirizzo di AHB enable register sarà 0X1014

Quindi tornando al codice, il programma modificato sarà:

```
Unsigned int*puntatore;
void mani (){

//Abilito prima la periferica
Puntatore=(unsigned int*)0x4002 1014;
*puntatore=0x00200000;

//cambio valore al puntatore, per passarlo al moder
Puntatore=(unsigned int*)0x48001000;
*puntatore=0x00010000;
While (1);
}
```

## LABORATORIO 2

lunedì 3 ottobre 2016 14:48

L'Obiettivo della prima esercitazione era quello di adoperare una linea del microcontrollore come un output digitale per accendere un led. A tale scopo abbiamo usato la linea PE8 della periferica GPIOE. Abbiamo attivato il clock della periferica GPIOE, abbiamo poi aperto il reference manual nella sezione della periferica General Purpose IO e abbiamo visto I registri da configurare e il modo in cui configurarli.

Il primo registro che abbiamo incontrato è il MODE REGISTER (MODER)--> se voglio adoperare la linea otto come output digitale, devo scrivere 01 nella coppia di bit indicati come moder8 (cioè il mode della linea 8).

Oggi vedremo come accendere due o più led contemporaneamente e come spegnerne uno lasciando inalterati gli altri.

Il registro successivo al Mode ha offset 4 (poichè gli indirizzi si incrementano ad ogni byte, quindi dopo 32 bit l'offset è quattro, infatti 32bit <->4byte e si chiama OTYPE REGISTER pag.143. Qui si possono impostare 16 bit uno per ogni linea. Per default è un output push-pull (eroga ed assorbe corrente). Allo stato di reset la linea è push pull, per cui visto che il nostro obiettivo di oggi è quello di accendere e spegnere un led, non è importante settare questo registro (lo lasciamo allo stato di reset.)

Registro ancora successivo è OSPEED REGISTER con offset 8 -> consente di selezionare la velocità delle linee impostando dei determinati valori (pag 144): x0 la linea cambia stato a bassa velocità ( 2 MHz); 01 la linea cambia stato a velocità media (10 MHz); 11 a velocità elevata (50 MHz). Naturalmente più elevata è la velocità, più si consuma, per questo viene lasciata la possibilità di scegliere. Per lo scopo del giorno (accendere e spegnere un led richiede un secondo), va benissimo lasciare lo stato di reset sullo speed.

Registro PBDR (pag. 144) permette di inserire sulla linea un pull-up/pull-down. Per capire...Se io non fisso che la linea deve avere un valore di tensione o alto o basso, il livello logico che ha quella linea è indefinito. Consideriamo una linea di input, se non ho collegato a niente questa linea, il livello è indifinito. I pull-up/pull-down servono a fissare un livello per le linee di input. In particolare il pull up è una resistenza che collega la linea in modo tale che sia sempre alta (se non viene toccata). Il pull down è una resistenza che fissa il potenziale della linea a massa, per cui se nessuno dall'esterno la tocca, questa linea è vista come bassa (perchè sta a potenziale 0). Noi la linea la adopereremo com output, quindi pull-up/pull-down non ci servono, perchè saremo noi dal microcontrollore a pilotare lo stato logico della linea.

Infine registri IDR e ODR, in questi due registri c'è un bit per ogni linea. INPUT DATA REGISTER riporta il livello logico delle linee che sono state configurate come input (infatti questa r indica che questi bit sono solo in lettura, noi non possiamo fare niente che non sia leggerli per capire se il livello logico della linea è alto o basso, ma sarà il circuito hw ad abbassare o alzare la linea).

Fissando la linea come OUTPUT DATA REGISTER, il bit che si va a scrivere si ripercuote sul livello della linea. Si può scrivere e non solo leggere. Sono io a gestire la linea e dire quando deve alzarsi e quando non deve farlo.

Ci sono altri registri nel seguito: GPIO port bit set/reset register pag 146

Anche questi servono ad alzare o abbassare una particolare linea. Questo registro lo uso quando voglio alzare una singola linea, perchè opera un singolo bit. Mentre utilizzando il registro Odr possiamo alzare anche più linee contemporaneamente.

Oggi dobbiamo operare con l'odr.

Ultima osservazione prima di iniziare: sul pdf Discovery pag 21, c'è una tabella su cui viene indicato tutto quello che sta sulla scheda, e sono quindi elencate anche le linee del microcontrollore

Scorrendo la tabella fino a pag. 28 si scopre dove sono collegati I led (led blu su pe8, led rosso su pe9 etc...).

Tutte le otto linee più significative di gpioe sono collegate a dei led.

Detto ciò, passiamo alle esercitazioni.

ESERCIZIO 1) Accendere il led blu.

```
unsigned int * puntatore;  
void main() {  
    puntatore= (unsigned int*) 0x40021014; //Abilito il clock  
    *puntatore= 0x00200000;
```

```
puntatore= (unsigned int*) 0x48001000; //Imposto la linea pe8 come output digitale (01)  
*puntatore=0x00010000;
```

```
puntatore= (unsigned int*) 0x48001014; //Per accendere il led blu (uno sull'8 dell'ODR)  
*puntatore=1<<8; //Nota: per alzare il bit 16-> 1<<16; per alzare il bit 18-> 1<<18; per alzare il bit 8-> 1<<8
```

```
While (1);  
}
```

ESERCIZIO 2) Accendere contemporaneamente il led rosso e il led blu.

Abbiamo già acceso il led blu, ora vogliamo accendere il led rosso che è collegato alla linea pe9. Affinchè il livello logico di pe9 si alzi, bisogna impostare nel moder che la linea pe9 è un output digitale (come prima cosa), quindi bisogna scrivere 01 sul moder9 (quindi alzare il bit 18). Successivamente bisogna agire sul registro ODR e in particolare sul bit 9.

Osservazione: vogliamo accendere il led rosso, senza che il led blu si spenga, quindi nel moder vogliamo alzare il bit 18 lasciando inalterati tutti gli altri. Analogamente vogliamo alzare il bit 9 dell'odr lasciando inalterati tutti gli altri.

Vale questa regola: ogni volta che vogliamo alzare un bit lasciando inalterati tutti gli altri, facciamo una or con un numero che ha tutti 0 ed un 1 nella posizione del bit che si vuole alzare.

Esempio.

Ho  $\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \text{---} & 7 & 6 & 5 & 4 & 3 & 2 \end{array}$  voglio alzare il bit in posizione 2

Allora faccio una or con il numero

$\hookrightarrow 00000100$  e ottengo

$\left\{ \begin{array}{c} 1001001 \\ | \\ 0000010 \end{array} \right. \rightarrow \begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ \text{---} & 7 & 6 & 5 & 4 & 3 & 2 \end{array} \rightarrow \begin{array}{l} \text{Ho alzato il bit 2} \\ \text{lasciamolo tutti gli} \\ \text{altri bit inalterati} \end{array}$

Soluzione esercizio 2).

```
unsigned int *puntatore;
```

```
void main() {
```

```
    //Prima parte uguale ad esercizio 1)
```

```
    puntatore = (unsigned int*) 0x40021014;
```

```
    *puntatore = 0x00200000;
```

```
    puntatore = (unsigned int*) 0x48001000;
```

```
    *puntatore = 0x00010000;
```

```
    puntatore = (unsigned int*) 0x48001014;
```

```
    *puntatore = 1<<8;
```

```
//Inizio esercizio 2)
```

```
puntatore = (unsigned int*) 0x48001000;
```

```
*puntatore |= (1<<18); //Alzo il bit 18 lasciando tutto inalterato
```

```
puntatore = (unsigned int*) 0x48001014;
```

```
*puntatore |= (1<<9); //Per alzare il 9 lasciando tutto inalterato
```

```
while (1);
```

```
}
```

ESERCIZIO 3) Lasciare acceso il led rosso e spegnere il blu.

Spegnere il led blu  $\leftrightarrow$  abbassare pe8  $\leftrightarrow$  Azzerare un bit lasciando inalterati tutti gli altri.

Vale la seguente regola: per azzerare un bit lasciando inalterati tutti gli altri, bisogna fare una and con un numero che ha tutti 1 e solo uno 0 nella posizione del bit che si vuole azzerare.

Esempio

Ho  $\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ \text{---} & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{array}$  voglio azzerare il bit 1

Allora faccio una and del mio numero con

$\hookrightarrow 1111111110$  e ottengo

$\left\{ \begin{array}{c} 1001100101 \\ | \\ 1111111110 \end{array} \right. \rightarrow \begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ \text{---} & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{array} \quad \begin{array}{l} \text{Ho abbassato il bit} \\ 1 \text{ lasciandolo tutti} \\ \text{gli altri inalterati} \end{array}$

La soluzione al problema si otterrà scrivendo: \*puntatore&= varie opzioni a seconda della notazione

Soluzione finale:

```
unsigned int *puntatore;
```

```
void main() {
    //Prima parte uguale ad esercizio 1) e 2)
    punitatore= (unsigned int*) 0x40021014;
    *punitatore= 0x00200000;

    punitatore= (unsigned int*) 0x48001000;
    *punitatore= 0x00010000;

    punitatore= (unsigned int*) 0x48001014;
    *punitatore|= (1<<8);

    punitatore= (unsigned int*) 0x48001000;
    *punitatore|= (1<<18);

    punitatore= (unsigned int*) 0x48001014;
    *punitatore|= (1<<9);

    //Inizio parte 3)
    punitatore= (unsigned int*) 0x48001014;
    *punitatore&= ~(1<<8);
    // Faccio una and con un numero che è il negato di un valore con tutti 0 ed un 1 in posizione 8
    /*Soluzione alternativa in esadecimale:
     *punitatore&= 0xFFFFFEFF; */

    /*Se si vuole abbassare anche il moder 9:
     punitatore= (unsigned int*) 0x48001000;
     *punitatore&= ~(1<<18); */

    While(1);
}
```

#### OSSERVAZIONI FINALI.

- 1) Errore: mai fare una and con un numero  $0<<x$  (con  $x$  qualunque numero), perché azzerà tutto il registro.
- 2) Per capire se si sta proseguendo bene, guardare sempre I registri.

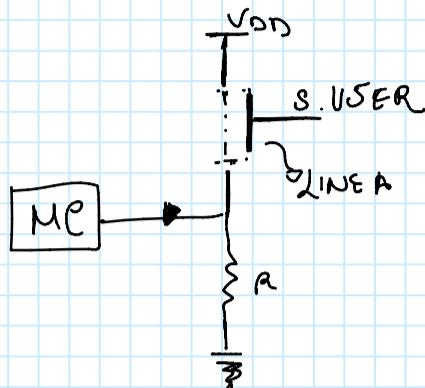
## LABORATORIO 3

Lunedì 17 ottobre 2016

Progetto del giorno: lavorare sull'input digitale.

Il pulsante azzurro etichettato come user sulla Demo Board ci permette di lavorare con gli input digitali, in particolare questo pulsante cambia esternamente il livello della linea che adopereremo come input digitale.

Dal punto di vista digitale



Ho una linea collegata a massa tramite una resistenza di pull down.  
Normalmente questa linea è a potenziale di massa (0 V) e quindi il livello logico è basso

Quando premo lo switch user (il pulsantino azzurro), si chiude quella parte di circuito e la linea va alla tensione alta (3,3 V).

Quando rilascio il pulsantino la linea torna di nuovo a massa.

Il microcontrollore si accorge che il pulsante è stato premuto dal livello della linea che è alto-. Quando è basso nessuno sta premendo il pulsante.

La prima cosa da capire è a quale linea è collegato lo switch user.

Su Discovery Board pag 21 (parte la tabella che mappa tutto ciò che si ha a disposizione sulla demo board con le relative linee del microcontrollore a cui sono collegate), si ricava che il pulsante USER è collegato a PA0 (linea 0 della periferica General Purpose I/O A).

Osservazione: la linea è di input digitale quindi si adopera sempre la periferica GPIO.

ESERCIZIO 1) Realizzare un programma che quando piglio il pulsante si accende un led e quando lo rilascio si spegne

Prima cosa da fare: Abilito I clock delle periferiche che servono.

Vado nel Reference Manual sezione RCC (pag.116), e cerco I bit che mi servono per abilitare GPIOA per il pulsantino e GPIOE per il led. Le periferiche GPIO sono collegate al bus AHB. Per abilitare la E devo alzare il bit 21, per abilitare la A devo alzare anche il bit 17.

Successivamente devo impostare nel Modo di GPIOA per dire che la linea 0 deve essere un input digitale (vado a scrivere 00-> che è anche la modalità di reset).

Dopo aver impostato la linea 0 come input digitale, per vedere se è alta o bassa, considero l'Input Data Register. In questo registro si vedrà (sulla linea 0) 1 se la linea è alta, 0 se la linea è bassa (ricordando che quando si tratta di input non siamo noi a dire se la linea è alta o bassa, ma possiamo solo leggere).

Quindi bisogna vedere se l'IDR0 passa a 0 o a 1 e si vuole inoltre che quando si preme il tasto (quindi la linea passa a 1) deve accendersi il led. Come scriverlo in codice, cioè nel programma come faccio a dire se qualcuno ha premuto il tasto? Bisogna fare un if.

Se si vanno a guardare I registri, si vedranno gli stati delle varie linee (idr15, idr14,...) a me interessa IDR0 (poiché il tastino azzurro è collegato alla linea 0), allora per capire se il tasto è stato premuto, devo vedere se l'ultimo bit di IDR è 1, ed è questa l'informazione che devo riportare nell'if. In effetti voglio isolare l'ultimo bit del registro idr e vedere se questo è 1 o 0. Per isolare un bit basta fare una and, con un numero che è tutti zero tranne un 1 in corrispondenza del bit di cui voglio sapere lo stato. Il risultato di questa and darà un 1 sul bit 0 se il pulsante è stato piggiato, altrimenti se è stato rilasciato da 0.

SOLUZIONE PER L'IF :      if ((IDR & 1) == 1) {  
              istruzione per accendere il led;  
    }

NOTA: Dopo aver sviluppato questo programma, per poterne migliorare la leggibilità si potrebbero definire l'indirizzo per ogni periferica, attraverso il "define". Modificando il programma si dovrà avere una situazione del genere:

```
#define RCC_AHBENR (unsigned int *) 0x40021014; //Ogni volta che il compilatore vede RCC_AHBENR sostituisce quel valore  
#define GPIOA_MODER (unsigned int *) 0x48000000;  
#define GPIOA_IDR (unsigned int *) 0x48000010;  
#define GPIOE_MODER (unsigned int *) 0x48001000;  
#define GPIOE_ODR (unsigned int *) 0x48001014;  
  
void main (){...}
```

In questo modo, quando nel main si vuole assegnare un indirizzo alla variabile "puntatore", si potrà scrivere direttamente il nome dell'indirizzo in questione.

Esempio :

Non scriverò più ->puntatore= 0x40021014, ma direttamente puntatore=RCC\_AHBENR.

## LABORATORIO 4

lunedì 24 ottobre 2016

Esercizio finale del Laboratorio 3 (17 ottobre): Scrivere di nuovo tutto il programma per accendere un led dopo aver premuto il tasto, per poi spegnerlo al rilascio, utilizzando il "define" per ogni puntatore al registro che si vuole utilizzare. Ovviamente con l'aumento delle periferiche (con cui si lavora), andare a definire di volta in volta tutti I registri può risultare "pesante", anche perchè ci sono molte periferiche che sono "replicate" nel microcontrollore, ad esempio la periferica GPIO ha sempre gli stessi registri, tuttavia con questo metodo si dovrebbe fare un define per il moder di GPIOE, uno per il moder di GPIOA, uno per l'odr di GPIOE, uno per l'odr di GPIOA... e così via in base a ciò che serve. Alla fine si avrebbero una marea di define per delle periferiche I cui registri sono fatti tutti allo stesso modo.

### 1) Modo ottimizzato per programmare I microcontrollori.

Sezione del reference manual della periferica GPIO (pag. 143).

Tutti gli indirizzi che fanno riferimento ad una periferica generica di IO sono organizzati in memoria nel seguente modo:

① GPIOx : BASE ADDRESS

→ Immutato di base di partenza

② MODER : ADDRESS OFFSET 0x00 (questi <=> immutato base)

↳ Si setta i 2 mooolo delle linee delle generica porta x.  
Registro lungo 32 bit

③ OTYPER : Dopo le byte (<=> 32 bit) dal moder, quindi offset 0x04  
Anche questo Registro è lungo 32 bit

④ OSPEEDR : Altri 32 bit dopo OTYPER => ADDRESS OFFSET 0x08.

↳ Permette di settare la velocità delle linee di output  
lungo 32 bit.

⑤ PUPDR : OFFSET 0x0C (12 byte dal Base Address)

↳ Registro dei RLC-UP e PULL-DOWN

⑥ IDR : OFFSET 0x20 , lungo 32 bit di cui solo 16 significativi

⑦ ODR : 4 byte dopo IDR , offset 0x14 . Anche questo lungo 32 bit,  
di cui solo 16 significativi

Questa organizzazione (per quanto riguarda gli offset) è la stessa per tutte le periferiche GPIO (A,B,C,D,E,F) cambia solo l'indirizzo base, però fissato questo indirizzo di partenza la posizione dei registri è sempre la stessa rispetto all'indirizzo di partenza (quindi Moder avrà sempre offset 0x00, ODR sempre 0x14, ... indipendentemente dalla periferica GPIOx).

Una soluzione al problema di ripetere tanti define, per periferiche organizzate allo stesso modo, sarebbe di definire un tipo GPIO\_type attraverso il "typedef" e dire che questo tipo è una struttura composta da diversi campi, che sono poi tutti I registri della periferica GPIO.

Typedef{

struct {

unsigned MODER: 32; //Il primo campo di questa struttura è un senza segno (che si chiama moder) a 32 bit <-> unsigned int MODER;

unsigned int OTYPER;

unsigned int OSPEEDR;

unsigned int PUPDR;

unsigned int IDR;

unsigned int ODR;

unsigned int BSRR; //Bit set/reset register

unsigned int LCKR; //Serve per bloccare le linee

unsigned int AFRL; //setta funzioni alternative

unsigned int AFRH; //setta funzioni alternative

}

}GPIO\_TYPE;

Osservazione: l'ordine con cui sono stati messi I campi è l'ordine dei registri in memoria, in questo modo, si definisce una struttura GPIO\_TYPE che ad ogni 32 bit c'è un nuovo campo. Questa sequenza di campi deve essere la stessa con cui I registri sono allocati in memoria e questo tipo GPIO\_TYPE è valido per tutte le periferiche di GPIOx (indifferentemente da A,B,...) tanto sono organizzate tutte nello stesso modo.

Una volta creata questa struttura, si semplifica tutto, perchè basta definire un unico define per la struttura e non più tanti define per ogni registro

Es. Per GPIOA:

```
#define GPIOA (GPIO_TYPE*) 0x48000000; //Definizione di un puntatore di tipo GPIO_Type che punta all'indirizzo base di GPIOA.
// GPIOA è un puntatore al tipo GPIO_TYPE
```

In modo analogo si può definire GPIOE come puntatore alla struttura di tipo GPIO\_TYPE all'indirizzo base di GPIOE.

```
#define GPIOE (GPIO_TYPE*) 0x48001000;
```

Come accedere ad un registro di una periferica (ad esempio GPIOE\_ODR) ?

Con queste definizioni per poter accedere al registro ODR della periferica GPIOE si può scrivere

`(*GPIOE).ODR=0;` o in modo equivalente `GPIOE->ODR=0;`

Il puntatore arriva all'indirizzo base, il campo ODR ha offset 0x14 (4 byte di moder+4byte di otype+...+4byte di idr). L'offset del registro rispetto all'indirizzo base lo si definisce SOLO con la posizione del registro all'interno della struttura.

In questo modo è comodissimo programmare, infatti non bisogna più scrivere gli indirizzi fisici di ogni registro di tutte le periferiche, ma basta considerare ad esempio scrivere `GPIOx->R` per poter accedere al particolare registro R che si vuole considerare .

Bisogna però definire queste strutture per tutte le periferiche del microcontrollore.

La ST fornisce una libreria, in cui sono definite appunto tutte queste strutture per ogni periferica del microcontrollore.

Nome file header:

`stm32f30x.h`

Su questo file non sono solo dichiarati tutti I tipi (per tutte le periferiche, ordinate in ordine alfabetico) , ma vengono dichiarati anche tutti gli indirizzi base (per ogni periferica) e tutti I puntatori ai tipi delle varie strutture.

GPIOA è un puntatore ad una struttura di tipo `GPIO_typedef` che punta all'indirizzo base di GPIOA.

Il tipo è sempre lo stesso, per ogni periferica `GPIOx`, cambiano solo gli indirizzi base che pure sono definiti in questo file.

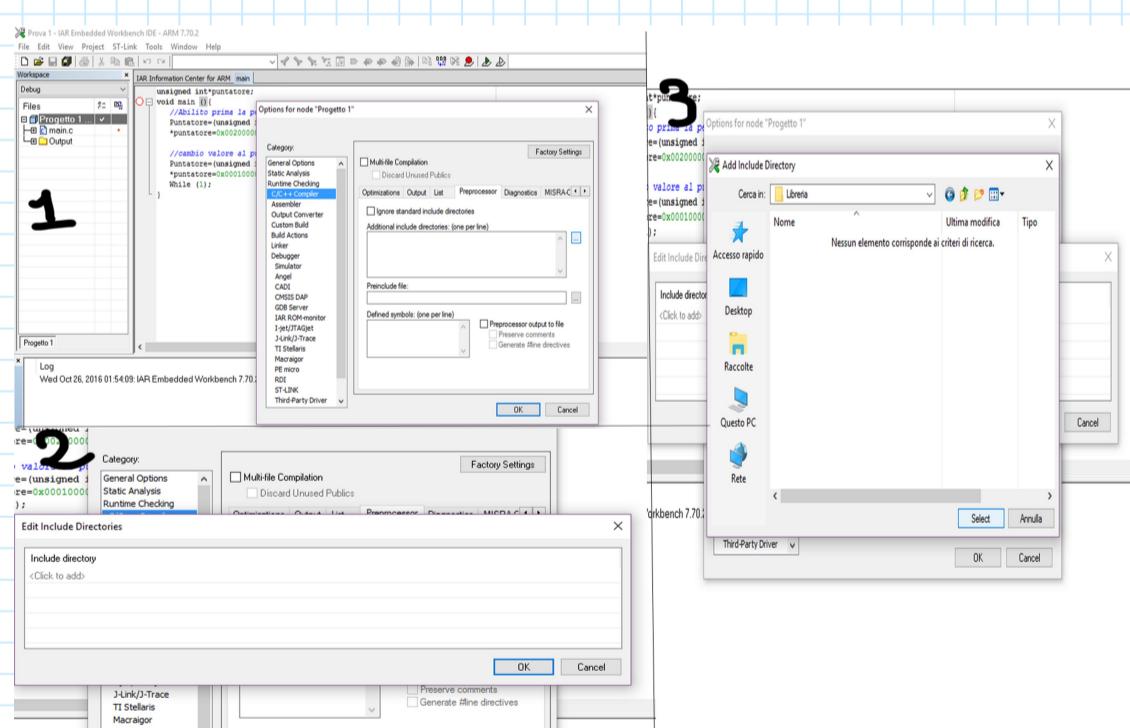
Da questo momento in poi in ogni progetto andrà inclusa questa libreria, in modo tale da poter trascurare gli indirizzi fisici. Si useranno solo dei nomi simbolici, dove ogni nome sarà del tipo : `PERIFERICA->REGISTRO` (andando a vedere sempre il reference manual).

Esempio di nome simbolico (per abilitare il clock di GPIOE): `RCC->AHBENR|=1<<21;`

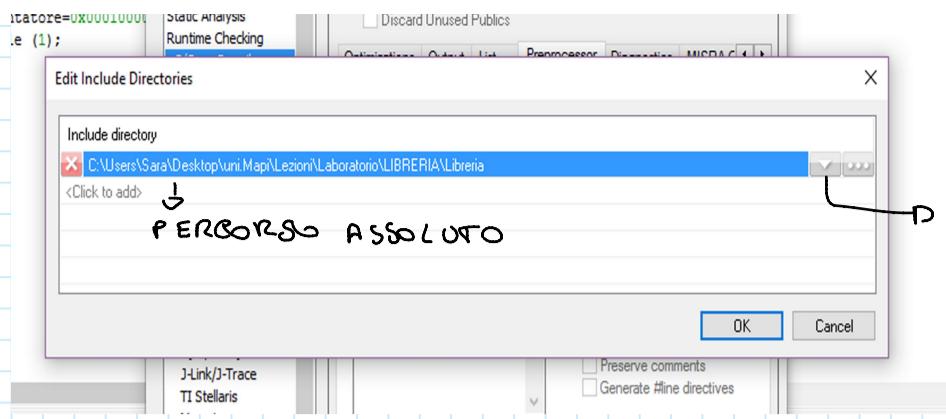
Per includere la libreria bisogna seguire due passi:

1) Poichè si sta includendo un file che non è contenuto in quelli standard di C, bisogna aggiungerlo.

Opzioni progetto (tasto destro progetto... vedi lab. 1)-> C/C++ COMPILER -> PREPROCESSOR: in cui vanno aggiunte le cartelle di include diverse da quelle standard del C (include addizionali). Si va ad inserire quindi il percorso della cartella in cui si trova il file.h

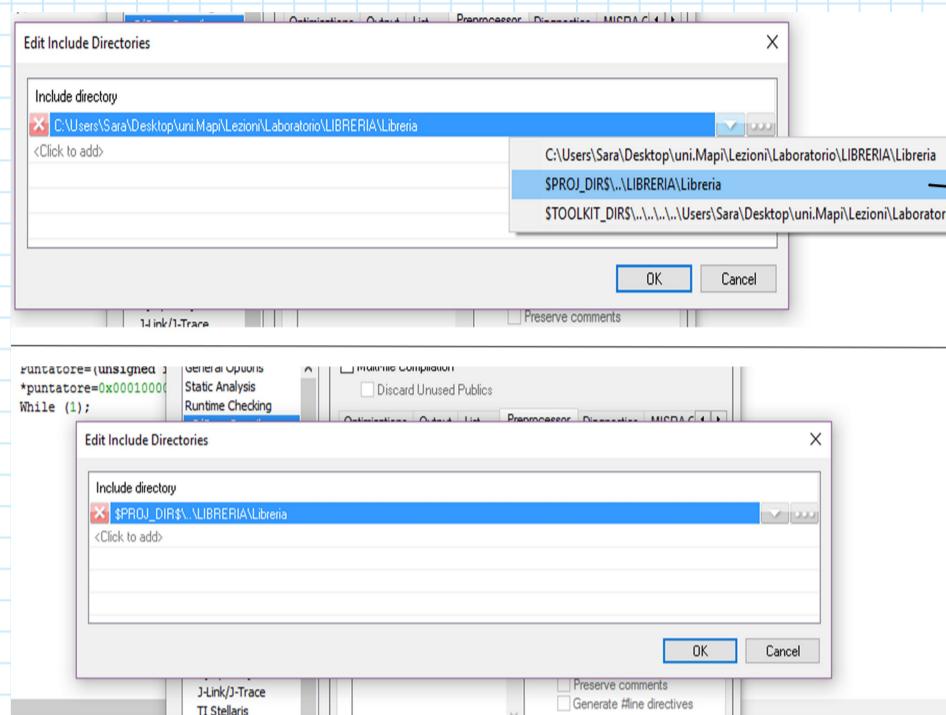


Il primo percorso che esce è il percorso assoluto, quindi se si passa il progetto su un altro computer, questo non funziona. Consiglio: considerare sempre I percorsi relativi, in modo che la libreria si trovi sempre nella cartella del progetto attivo.



CLICCO SU QUESTA FRECCIA PER IL PERCORSO RELATIVO

Percorso relativo :



RELATIVO: Dov'è QUESTO  
DOLUSO (che sia ad  
immediata da quella  
del progetto)

2) Scrivere nel file.c: #include <stm32f30x.h>

**ESERCIZIO 1:**

RISCRIVERE IL PROGRAMMA DEL LABORATORIO 3 CON QUESTA LIBRERIA: Premo -> Accende un led, Smetto-> Spegne il led

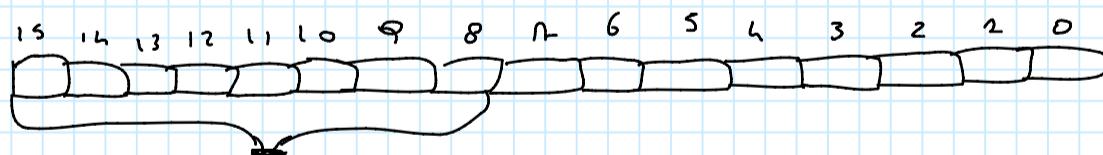
Soluzione: "Esercizio lab 3 con libreria"

**ESERCIZIO 2:**

CONTATORE BINARIO CHE CONTA IL NUMERO DI VOLTE CHE SI PREME IL PULSANTE.

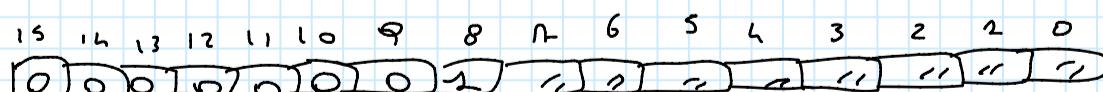
Si vuole contare quante volte è stato premuto il tasto. Servirà sicuramente una variabile contatrice che ogni volta che viene premuto il tasto deve incrementarsi. Si adoperano poi gli 8 led come se fossero gli 8 bit di un contatore ad 8 bit.

Quindi, se questo è il registro odr di GPIOE:



I LED STANNO SU QUESTI 8 bit  
Più significativi (Da PE8 a PE15)

-Cont (la variabile contatrice) parte da 0, quando si preme il tasto cont passa ad 1. Se si va a scrivere il valore di cont sugli otto bit più significativi, scrivo tutti 0 ed 1 (finale) e si accende il led blu.



-Dopo aver rilasciato il pulsante, quando si preme di nuovo, cont passa a 2, e andando a scrivere il valore di cont in esadecimale, negli ultimi otto bit di odr si avrà: 00000010 (2 in binario) e si accende il led rosso.



-E così via per tutti I conteggi successivi, ad ogni incremento di cont deve accendersi una luce.

In questo senso si sta parlando di un **CONTATORE BINARIO**: Gli otto led codificano in binario il valore del conteggio.

Errore: se si tiene premuto il pulsante user, il valore di cont non deve aumentare all'infinito, questo significa che solo dopo aver rilasciato il pulsante, cont si deve incrementare. Altrimenti c'è il rischio che il contatore vada in overflow più volte fino ad arrivare ad un numero che non si può controllare.

Osservazione: Una volta arrivato ad otto, bisogna tornare al punto di partenza, cioè far accendere di nuovo il led blu (cont = 9), poi il led rosso (cont =10), ..., fino a cont=16 (si accende l'ultimo led) e ripetere tutto. Quindi cont deve contare "otto pressioni alla volta", il motivo è ovvio, I led sono otto e si trovano solo su otto registri di odr. Oltre ad otto non ci sono led.

Soluzione: "Laboratorio 4 progetto\_Contatore Binario".

# LABORATORIO 5

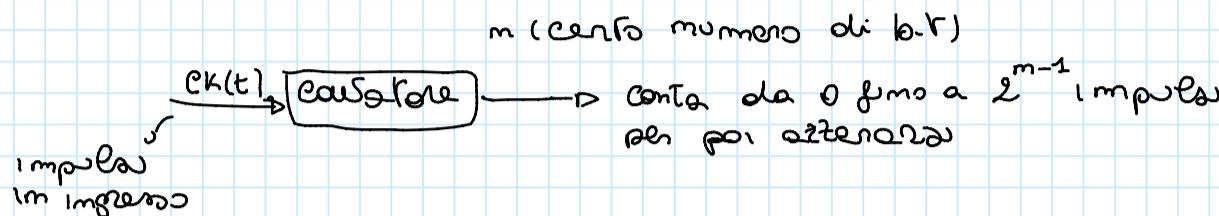
venerdì 28 ottobre 2016

## Modulo Timer.

Sul Data Sheet a pag. 11, si può osservare l'architettura generale di tutte le periferiche, in particolare ci sono una serie di timer sul lato destro (timer2, timer3, timer4) ed altri sul lato sinistro (timer 15, timer16, timer17, timer 1).

Ci sono quindi vari moduli timer, lo scopo di oggi è capire cosa sono questi moduli, a cosa servono e iniziare ad usarli.

Il cuore di un timer è un contatore ad un certo numero di bit ( $n$ ). Questo contatore conta degli impulsi che gli arrivano in ingresso (nel microcontrollore tali impulsi provengono da un segnale di clock e si vedrà che conoscendo la frequenza è possibile anche selezionare chi è questo segnale di clock), partendo da 0 fino a  $2^{n-1}$ . Arrivato al valore  $2^n$ , il contatore si azzera e comincia a contare di nuovo da 0, dato che  $n$  è il numero di bit e non può andare oltre.



Esistono due modi di adoperare il contatore.

- 1) Modalità Base dei Tempi.
- 2) Modalità Contatore (o Misuratore di Tempo).

### Modalità Base dei Tempi. $\rightarrow$ INCOGNITA $N$

Nella modalità Base dei Tempi, si aspettano degli intervalli di tempo uguali per poter effettuare una certa operazione (ad esempio il campionamento del segnale, o il blink del led che si accende ad intervalli regolari, ...), quindi si sfrutta il contatore per attendere un intervallo di tempo prefissato prima di una determinata operazione. Come si fa? Conoscendo la frequenza del clock e quindi il periodo del clock e l'intervalllo di tempo che si vuole aspettare prima di eseguire l'operazione.

Es.

$\Theta$   $T_{ck}$  = periodo del clock = 1 ms (tempo di clock)

$\Theta$   $\Delta t$  desiderato = 1 ms  $\rightarrow$  bisogna aspettare 1 ms prima di eseguire un'operazione.

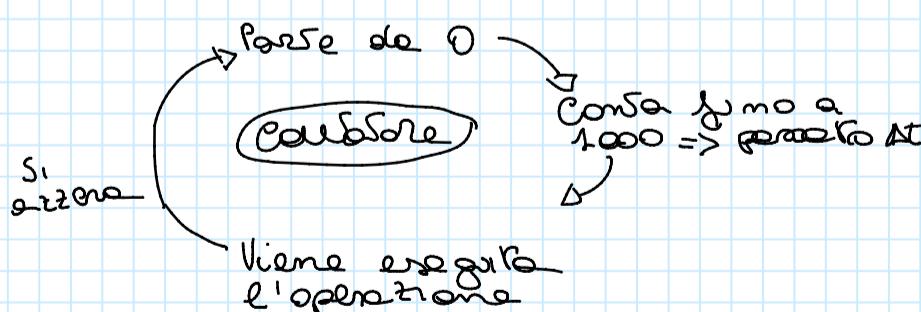
Sapendo che durante tale intervallo il clock invia al contatore una serie di impulsi a frequenza fissa  $f_{ck} = 1/T_{ck}$ , il numero  $N$  degli impulsi connessi risulta proporzionale allo stesso  $\Delta t$ .

$\Rightarrow$  Sapendo il periodo di  $ck$  e l'intervalllo di tempo che si vuole aspettare prima di eseguire un'operazione, si può determinare il numero di conteggio che deve effettuare il contatore

$$N = \frac{\Delta t}{T_{ck}} = \Delta t \cdot f_{ck}$$

Nell'esempio  $N = \frac{10^{-3}}{10^6 \text{ s}} = 10^{-3} \cdot 10^6 = 10^3 = 1000$

In pratica inizia il seguente ciclo.



Si ha quindi un modo per sapere quando è passato  $\Delta t$  per poter eseguire l'operazione.

### Modalità Contatore $\rightarrow$ INCOGNITA $\Delta t$

Si adopera il contatore per misure il tempo, quindi permette di misurare intervalli di tempo utilizzando il contatore numerico.

Es.

Supponiamo di premere il pulsante User e di mantenerlo premuto per un certo intervallo di tempo, per poi rilasciarlo. Si vuole sapere quanto tempo è passato dal momento in cui il pulsante è stato premuto, fino al suo rilascio, o meglio quanto è durato il tempo in cui abbiamo premuto il pulsante?

L'intervalllo di tempo trascorso  $\Delta t$  sarà dato da:

$$\Delta t = n \cdot T_{ck} \rightarrow \text{periodo del segnale di clock}$$

L'intervallo di tempo trascorso  $\Delta t$  sarà dato da:

$$\Delta t = N \cdot T_{\text{en}} \rightarrow \text{periodo del segnale di clock}$$

*↳ numero di conteggi che il contatore ha fatto durante tale intervallo*

Nel senso della quantizzazione perché l'intervallo di tempo  $\Delta t$  viene quantizzato in base al periodo del segnale di clock.

Osservazione:

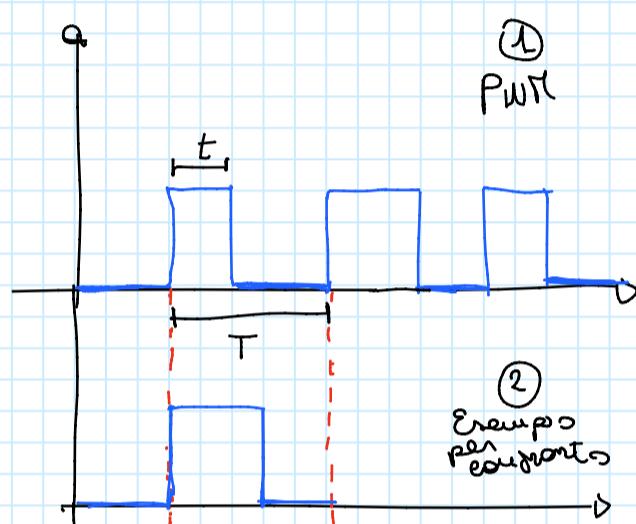
La risoluzione con cui si misura l'intervallo di tempo è proprio  $T_{\text{ck}}$ , per cui risulta evidente che il segnale che viene selezionato è anche quello che determina la risoluzione con cui si misura il tempo  $\Delta t$ .

Fatta questa introduzione, si può iniziare a vedere nei dettagli cosa sono I Timer e come sono fatti.

Sul Reference Manual ci sono tre capitoli dedicati ai timer:

- 1) Advanced-Control Timers (timers per il controllo avanzato) -> Periferiche indicate con Tim1 e Tim8 (capitolo 16)
- 2) General Purpose Timers -> Tim2, Tim3, Tim4, Tim15, Tim16, Tim17 (capitolo 17, 18)
- 3) Basic Timers -> Tim6, Tim7 (capitolo 20)

1) Gli Advanced-Control Timers sono timers che controllano un'altra periferica, ad esempio tim1 e tim8 vengono adoperati per generare segnali PWM.



Confronto (1) e (2)  
a parità di periodo  
(2) ha valore medio più alto.

Segnali PWM (Pulse Width Modulation ovvero Modulazione a variazione della larghezza di impulso)

Un segnale PWM è un'onda quadra di Duty Cycle variabile.

Il Duty Cycle (ciclo di lavoro o ciclo di lavoro utile) è uguale al rapporto tra il periodo di segnale "alto"  $t$  (ovvero la durata del tetto) e il periodo di segnale totale  $T$  e serve ad esprimere per quanta porzione di periodo il segnale è a livello alto (intendendo con alto il livello attivo)

$$d = \frac{t}{T}$$

Il valore di  $d$  è sempre un numero compreso tra 0 e 1. In particolare se  $d=0$  o  $d=1$  si è in presenza di segnali continui, infatti se  $d=0$  allora  $t=0$  e quindi si ha un livello basso per tutto il tempo (segnale continuo a livello basso); se invece  $t=1$  allora  $t=T$  (hanno lo stesso valore) e quindi per tutto il periodo il segnale è alto (segnale continuo a livello alto).

Osservazione: se  $d=0,5$  -> per metà del periodo il segnale è alto per l'altra metà il segnale è basso, si è quindi in presenza di un'onda quadra.

In base alla definizione di Duty Cycle, questo risulta essere variabile se a parità di periodo, varia la durata del tetto. Al variare della durata del tetto, varia il valor medio del segnale (perciò varia la durata del tetto).

Allora in altre parole I segnali PWM sono dei segnali il cui Duty Cycle viene variato in base al valor medio che si vuole avere.

Per questi segnali, si necessitano quindi due timers in modalità base dei tempi, per poter impostare da un lato la durata del tetto dall'altro il periodo. C'è quindi una modalità di funzionamento dei timers avanzati (Tim1 e Tim8), che permette di impostare questi due tempi in modo tale che Tim1 conta il tempo  $t$  dopo il quale il segnale PWM viene abbassato mentre Tim8 conta il periodo, dopo il quale il segnale PWM viene alzato.

Nel nostro corso non ci occuperemo di questi segnali (Non abbiamo il tempo di generare segnali PWM, nè abbiamo circuiti che possiamo controllare con questi segnali).

## 2) General Purpose Timers (TIM2, TIM3, TIM4 -> Reference Manual capitolo 17, inizio pag 452).

Sono dei timers utilizzati per vari scopi (timers multiuso), tra cui ad esempio per misurare la durata degli impulsi di un segnale di input o anche per la generazione di forme d'onda in uscita.

I timers di questa sezione sono Timer2 (a 32 bit, quindi può contare fino a  $2^{32-1}$ ), Timer3 e Timer4 (entrambi a 16 bit, quindi possono contare fino a  $2^{16-1}$ ), che sono completamente indipendenti tra loro e non condividono alcuna risorsa. Possono inoltre essere sincronizzati tra loro.

Nota: Il numero di bit del timer <-> al numero di bit del contatore.

Abbiamo visto che ci sono due modalità per adoperare I timers.

### 2.1) Modalità Base dei tempi (pag 454) -> in questa modalità I registri fondamentali sono tre:

- Counter Register (TIMx\_CNT): è il contatore, ha memoria del conteggio.
- Prescaler Register (TIMx\_PSC): divisore di frequenza.
- Auto-Reload Register (TIMx\_ARR): contiene il valore "limite" impostato a priori, a cui deve arrivare il conteggio prima che il contatore si azzeri e riprenda a contare (basta ricordare il discorso fatto per cui nella modalità base dei tempi, si vuole aspettare un intervallo di tempo stabilito, per poter eseguire un'operazione e quindi bisogna aspettare che il contatore arrivi ad un certo valore, inserito nel registro ARR).

$\boxed{\text{CNT}}$   $\longrightarrow \boxed{\text{ARR}}$

Nota: un flag è una variabile booleana che può



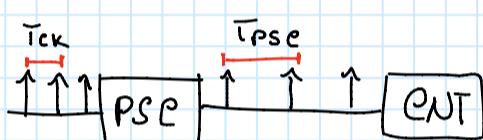
Nota: un flag è una variabile booleana che può assumere solo due stati. Segnala con il suo valore se un evento si è verificato oppure no, o se il sistema è in un certo stato oppure no. Viene utilizzato per eseguire determinate operazioni solo al raggiungimento di un determinato stato o al verificarsi di un evento.

Il PSC è stato definito come "divisore di frequenza", resta da capire in che senso e quindi a cosa serve -> Esempio teorico.

- ①  $T_{CK} = 1\mu s$
- ② Voglio contare 1s ( $\Rightarrow \Delta t = 1s$ ) utilizzando Timer3, che è a 16 bit.  
Sapendo che  $N = \frac{\Delta t}{T_{CK}} = \frac{1s}{10^{-6}s} = 10^6 = 1000000$   
Allora in ARR devo scrivere 1000000

N rappresenta il numero massimo di conteggi da fare e quindi il valore da inserire in ARR. Tuttavia, risulta ovvio che non posso impostare 1000000 in ARR, questo perché il Timer3 è a 16 bit, quindi il contatore arriva massimo a 65535 per poi azzerarsi al conteggio successivo, quindi non arriverà mai ad 1000000.

Il problema però mi chiede di contare fino ad 1000000 utilizzando TIM3. Cosa posso fare? La soluzione è quella di aumentare il periodo del segnale di clock (e quindi ridurre la frequenza), che fa incrementare il contatore. Bisogna dunque passare ad un divisore di frequenza che è proprio il PSC.



Nel PSC entra il segnale di clock ed esce un treno di impulsi, la cui distanza (Tpse) è più grande di quella del periodo di clock. Nello specifico se nel PSC inserisco un numero x Tpsc è il valore che ho scritto nel PSC +1 per Tck.

$$Tpse = (x + 1) T_{CK}$$

Allora se metto il PSC a 0 al contatore sto mandando proprio gli impulsi di clock

$$Tpse = (0 + 1) T_{CK} = T_{CK}$$

se invece scrivo 1 il periodo del segnale è il doppio del periodo del clock (ho dimezzato la frequenza -> raddoppiato il periodo).

$$Tpse = (1 + 1) T_{CK} = 2 T_{CK}$$

Qualunque numero metto nel psc sono limitata ai 16 bit anche qui, divido la frequenza del segnale che invio al contatore.

Detto ciò torno all'esempio precedente.

- ①  $T_{CK} = 1\mu s$
- ②  $\Delta t = 1s$  utilizzando TIM3
- ③ imposto IL PSC a 999  $\Rightarrow Tpse = (999 + 1) T_{CK} = 1000 \mu s$  (periodo degli impulsi che arrivano al contatore)  
 $= 1ms$

$$\text{In ARR devo scrivere } N = \frac{1s}{1ms} = 1000$$

Adesso 1000 impulsi de 1ms danno 1s. Posso scrivere 1000 in un registo a 16 bit? Sì!

Aumentando il periodo, sono riuscita a risolvere il mio problema.

Osservazione: Nel PSC non può essere inserito un valore a caso, ma va scelto il giusto compromesso tra la durata dell'intervalllo di tempo che si vuole aspettare e la miglior risoluzione possibile. Infatti nel passaggio da Tck a Tpsc cambia la risoluzione, si ha la risoluzione di Tpsc (Nota: la risoluzione relativa è 1/N, quindi 1 sul valore che scrivo in ARR)

Questi sono I tre registri da adoperare di cui vengono poi descritti tutti I bit (sempre a pag 454). In particolare il conteggio è abilitato da un particolare bit che si trova nel registro Control Register 1 (TIMx\_CR1 register) e questo bit si chiama CEN (counter enable).

Note aggiunte a pag 455 del Reference Manual->Diagramma: conteggio in funzione del PSC

## 2.2) Modalità Contatore (pag 455)-> Serve per misurare il tempo fondamentalmente

Il contatore può contare sia incrementandosi che decrementandosi: se lo si imposta in upcounting conterà da 0 al valore fissato in ARR; se lo si imposta in downcounting conterà dal valore in ARR a 0. C'è anche un'altra modalità per il conteggio: Allineato al centro, si incrementa e quando arriva all'overflow si decrementa.

Nella modalità contatore, non viene fissato un punto di arrivo del contatore, ma si va a vedere a che conteggio è arrivato per poter misurare un intervallo di tempo.

### 2.3) Selezione del clock (Pag 463)-> Per impostare Tck.

Le impostazioni di default sono quelle per cui Tck viene dal clock interno che, per come stiamo lavorando, è praticamente quello dell'oscillatore esterno allo schedino (che vale 8MHz).

A parte le impostazioni di default, si può selezionare come segnale di clock un segnale che va collegato ad un pin della scheda, un segnale di trigger interno o un segnale che viene da un'altra periferica che ha svolto un operazione.

Altri modi per selezionare un segnale di clock sono ad esempio:

- 1) Input Capture (pag. 468), grazie al quale si può dire al microcontrollore di fermare il timer quando si verifica un evento.  
Permette di ottenere il tempo trascorso fino al verificarsi di uno specifico evento sul segnale di input.  
Es. "ferma il timer quando il segnale che ti ho collegato a pa2 passa da basso ad alto", automaticamente questo evento ferma il timer e quindi si ha l'intervallo di tempo desiderato.
- 2) Using one timer to start another timer (pag. 486). Un timer può fare a sua volta da evento trigger, ovvero da evento di abilitazione per altre periferiche.  
Es. "io timer, quando arrivo ad un determinato conteggio, mando un segnale di abilitazione ad un altro timer, che inizia a contare". Caso che ci interesserà è quando il timer viene usato per abilitare un convertitore analogico/digitale: "timer, quando sei arrivato a questo conteggio, manda un segnale di abilitazione al convertitore analogico/digitale, che deve effettuare una conversione". Il timer diventa quindi il master di un'altra periferica .

### REGISTRI (pag492)

Come per tutte le altre periferiche, anche per I timers bisogna studiare I vari registri, da settare in determinati modi, per poterli usare (I timers). Si possono distinguere due prime categorie di registri:

- Periferica\_CR : Registri di controllo, vengono settati per configurare la periferica)
- Periferica\_SR : Status Register, si leggono per vedere se si sono verificati dei determinati eventi.

Scopo delle pagine successive: analizzare I registri dei General Purpose Timers

#### 1) Control Register 1 (TIMx\_CR1)-> pag. 492

È un registro a 16 bit molti dei quali non vengono adoperati.

Address offset: 0x00 Reset value: 0x0000															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIF RE-MAP	Res.	CKD[1:0]	ARPE	CMS	DIR	OPM	URS	UDIS	CEN		
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bit 11 (UIFREMAP: UIF status bit remapping): Primo bit più significativo, dice quando si è verificato l'evento di update del timer, cioè quando il contatore è arrivato al valore impostato in ARR (lo si può interpretare come il flag che manda il registro ARR).

Nel caso di upcounting: UIF è 0, per il conteggio che va da 0 al valore in ARR-1; si alza (quindi passa a 1) quando arriva al valore di ARR.

Nel caso di downcounting: UIF è 0, per il conteggio che va da ARR a 1; passa a 1 quando il conteggio arriva a 0.

Perchè update? Perchè in entrambi I casi (up e down) arrivato ad ARR o a 0 (a seconda dell'amodalità) poi si azzera o si rimette al valore di ARR e conta di nuovo.

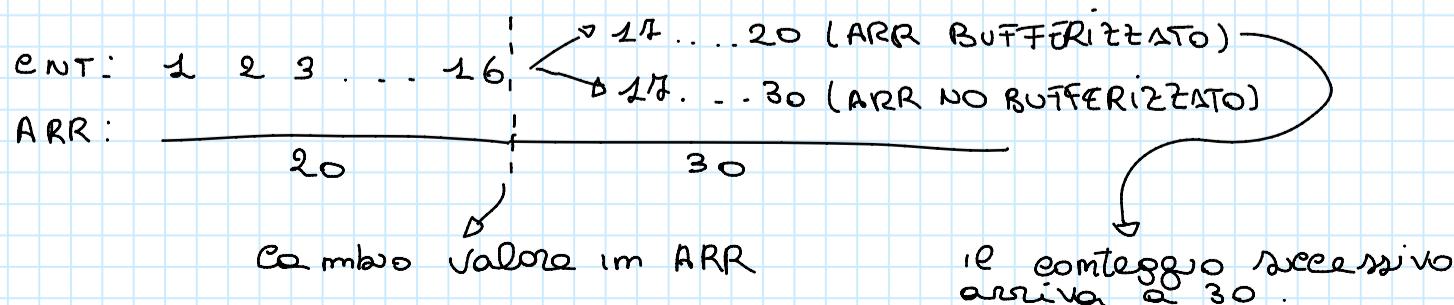
- Bit 9 e 8 (CKD: Clock division): permettono di dividere ulteriormente (oltre al PSC per intenderci) la frequenza del segnale di clock (dividere per 1, per 2 o per 4).
- Bit 7 (ARPE:Auto-reload preload enable): se questo bit sta a 0 il registro ARR non è bufferizzato se questo bit sta a 1 il registro è bufferizzato.

In che senso il registro ARR è bufferizzato o meno ?

Es. Supponiamo di aver scritto in ARR il valore 20. Supponiamo poi di scrivere un altro valore in ARR oppure che il programma faccia un'operazione per cui setta nuovamente ARR e che questo nuovo valore sia 30. Nel tempo che va dalla prima impostazione di ARR fino alla sua modifica, il contatore ha fatto dei conteggi, in particolare supponiamo che quando il valore di ARR è stato modificato il contatore era arrivato a 16.

Quindi il valore di ARR passa a 30 e nel frattempo il contatore ha fatto 16 conteggi. Problema: Il contatore ora, prima di azzerarsi e riprendere il conteggio da 0, arriva fino a 20 o fino a 30? In altri termini, il valore di UIF si alzerà a 20 o a 30?

Dipende se ARR è stato bufferizzato o meno: se è stato bufferizzato il contatore continua il conteggio da 16 fino a 20, arrivato a 20 si azzera e riprende il conteggio successivo questa volta da 0 a 30; se non è stato bufferizzato, allora il contatore considera direttamente l'ultimo ARR scritto e quindi conta da 16 fino a 30 già dalla prima volta.



- Bit 6 e 5 (CMS: Center-aligned mode selection): Nel caso in cui si vuole adoperare un conteggio allineato al centro. Ci sono quattro possibili valori codificati con 2 bit:
  - 00 -> impostazione di default, quando non si vuole utilizzare questa modalità.
  - 01 -> modo 1-> il contatore conta avanti e indietro in maniera alternativa.  
Le uscite delle interrupt flag dei canali configurati in uscita sono impostate solo quando il contatore conta a decrescere.
  - 10 -> modo 2-> il contatore conta avanti e indietro in maniera alternativa.  
Le uscite delle interrupt flag dei canali configurati in uscita sono impostate solo quando il contatore conta a crescere.
  - 11 -> modo 3 -> il contatore conta avanti e indietro in maniera alternativa.  
Le uscite delle interrupt flag dei canali configurati in uscita sono impostate solo quando il contatore conta a crescere o a decrescere.
- Bit 4 (DIR: Direction): Per impostare la modalità del conteggio -> 0 conta a crescere  
-> 1 conta a decrescere
- Bit 3 (OPM: One-pulse mode): Serve per dire al contatore se una volta arrivato al valore impostato in ARR, questo deve continuare il conteggio (e quindi ripartire da 0 ad ARR) oppure se lo deve fermare. In particolare:
  - 0-> Impostazione di default, continua il conteggio.
  - 1-> Ferma il conteggio, liberando il CEN.
- Bit 2 (URS: Update request source): Questo bit viene adoperato per cancellare un evento del timer quando questo si verifica (per questa fase lo tralasciamo, perchè per poterlo utilizzare bisogna prima vedere cosa sono gli eventi e come si configurano).
- Bit 1 (UDIS: Update disable): sempre in riferimento agli eventi e quindi lo tralasciamo (per ora).
- Bit 0 (CEN: Counter Enable): Bit meno significativo permette di abilitare (1) o disabilitare il conteggio (0)

## 2) Control Register 2 (TIMx\_CR2)-> pag 494

Control Register 2 (TIMx_CR2)															
Address offset: 0x04															
Reset value: 0x0000															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TI1S	MMS[2:0]		CCDS	Res.	Res.	Res.	
								rw	rw	rw	rw	rw			

- Bit 7 (TI1S: TI1 selection): Riguarda segnali che provengono dall'esterno.
- Bit 3 (CCDS: Capture/compare DMA selection): Riguarda richieste da inviare ai DMA.
- Bit 6, 5, 4 (MMS: Master mode selection): Vengono adoperati quando si vuole che il timer piloti altre periferiche.  
Il timer genera un segnale di trigger (TRGO: Trigger output) in corrispondenza di particolari eventi. Questi eventi che fanno generare il trigger al timer vengono stabiliti in questi tre bit, e sono:
  - 000 -> Il reset del timer.
  - 001 -> L'abilitazione del conteggio.
  - 010 -> L'update del timer.
  - 100 -> Il compare pulse, che riguarda il conteggio con segnali che provengono dall'esterno.

Quindi ipoteticamente se si vuole pilotare il convertitore analogico/digitale in modo che ogni secondo esegua una conversione, basterà dire che un timer è il master di quel convertitore e in particolare che questo timer deve inviare un TRGO ad ogni evento di update. Ovviamente bisogna anche impostare PSC e ARR in modo da avere un update ogni secondo.

## 3) TIMx slave mode control register (TIMx\_SMCR)-> Pag. 495.

Registro che va configurato se si vuole che un timer sia uno "slave", cioè se si vuole che il timer debba ricevere un trigger da un altro timer (Nota: ogni timer può fare da trigger ad altri timer).  
Lo ignoriamo.

## 4) TIMx DMA/Interrupt enable register (TIMx\_DIER) -> Pag. 498

In ogni periferica si ha la possibilità di configura una interrupt, per dire alla periferica che "quando si verifica un dato evento deve scatenare una interrupt". Nel caso del timer il registro che permette di fare ciò è il TIMx\_DIER (DIER perchè alcuni bit riguardano il DMA e altri l' Interrupt Enable).

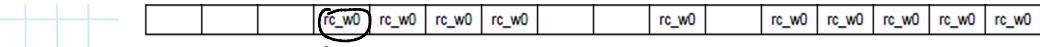
In particolare di questo registro useremo quasi sicuramente il bit meno significativo:

UIE (Update Interrupt Enable) -> "in occorrenza di un update scatena una interrupt perchè quando è trascorso un certo tempo vogliamo fare una certa sequenza di operazioni e la vogliamo fare in interrupt".

## 5) TIMx status register (TIMx\_SR)-> Pag. 499

Registri dei flag.

Status Register (TIMx_SR)															
Address offset: 0x10															
Reset value: 0x0000															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			(rc_w0)	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	



Si leggono le operazioni che avranno luogo sui bit (si tratta su tutti i registratori)

- r lettura
- w scrittura

Per quanto riguarda i flag, si capisce subito che sono dei bit che devono segnalare degli eventi, infatti si legge "rc" che sta per "read clear": si possono leggere o al massimo possono essere cancellati (quando si vuole dire alla periferica "ho capito che è successo questo evento". In particolare viene detto come si cancella il flag, ovvero scrivendo 0, cosa non troppo banale perché altri flag si cancellano con 1.

La parte più significativa riguarda tutte le funzionalità che non adoperiamo, di questo registro in fondo l'unica funzionalità che adoperiamo è quella data dal bit meno significativo:

Bit 0 (UIF: Update Interrupt flag): Serve a capire quando si è andati in update. Questo bit è settato dall'hardware su un evento di update ed è cancellato via software (quindi dobbiamo cancellarlo noi).

## 6) TIMx counter (TIMx\_CNT) -> Pag. 509

Address offset: 0x24															
Reset value: 0x0000															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31] or UIFCPY															
CNT[30:16] (depending on timers)															
rw or r	rw														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Del registro counter sono significativi tutti i 32 bit solo per il timer 2, per gli altri due solo i 16 bit più significativi.

Il bit 31 può essere impostato come il bit 31 del counter (se UIFREMAP in TIMx\_CR1 è 0) o come la copia di UIFREMAP (se questo è pari a 1 in TIMx\_CR1). Questo significa che per semplicità di programmazione si può sacrificare un bit del counter e avere qui il valore del flag dell'update di UIFREMAP, invece di andare a vedere lo status. Ovviamente questo va configurato nel timer, per cui il conteggio avverrà su 31 bit. Nella modalità di default il conteggio è su 32 bit.

Come impostare che il bit 31 del contatore è l'update flag. Questa impostazione va nel control register, poiché è una configurazione, in particolare si va ad impostare il bit 11 del CR1, UIFREMAP: normalmente è 0, se lo si mette a 1 si chiede di rimappare il bit UIF e quindi di metterlo nel bit 31 del contatore.

## 7) A pag. 510 abbiamo infine i due registri :

### 17.4.11 TIMx prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 PSC[15:0]: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

→ anche questi 16 bit per tutti, Tranne per Timer 2 ai 32 bit

### 17.4.12 TIMx auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
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
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 ARR[31:16]: High auto-reload value (on TIM2).

Bits 15:0 ARR[15:0]: Low Auto-reload Prescaler value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 17.3.1: Time-base unit on page 454](#) for more details about ARR update and behavior.

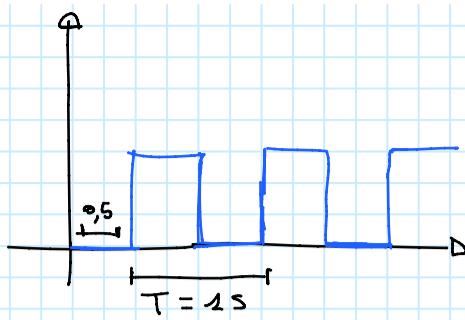
The counter is blocked while the auto-reload value is null.

## ESECIZIO DEL GIORNO

Adoperare il timer con modalità base dei tempi, in particolare :



Si vuole generare questa onda quadra con un pin di output digitale di periodo 1s (quindi



Si vuole generare questa onda quadra con un pin di output digitale di periodo 1s (quindi semiperiodo 0,5 s). In particolare scegliamo come output digitale uno dei led (uno qualunque anche tutti). Scopo veder "blinkare" ogni secondo uno o più led).

Scelta del timer indipendente. Se si sceglie timer 2 a 32 bit non si ha alcun problema di overflow perchè arriva a  $2^{32}$  che è un numero abbastanza grande e quindi si riesce a contare senza problemi 1s.

Ricordare inoltre che se non si fanno particolari impostazioni si lavora con il clock che viene dall'oscillatore con frequenza 8MHz.

$$\Rightarrow T_{\text{clk}} = \frac{1}{8} \cdot 10^{-9} \text{ s} = 1.25 \cdot 10^{-9} \text{ s}$$

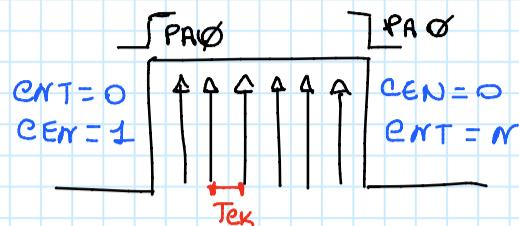
Stiamo lavorando con questo periodo, vogliamo contare 0,5 s, perchè quando sarà passato mezzo secondo, dobbiamo cambiare lo stato della linea collegata al led (Ogni mezzo secondo si deve accendere e spegnere un led).

# LABORATORIO 6

lunedì 7 novembre 2016

Scopo della lezione: adoperare il timer in modalità contatore e quindi adoperarlo per eseguire la misura di un intervallo di tempo.

Schema per la misura di un intervallo di tempo fatta con contatore numerico:



$$T_{ek} = 1.25 \text{ ms}$$

$$\bar{T} = N \cdot T_{ek} = \text{durata finestra temporale}$$

Per misurare la durata della finestra temporale, devono essere contati gli impulsi che si verificano al suo interno e andare a moltiplicare il valore ottenuto per il periodo di clock . Il risultato ottenuto da proprio il valore  $\bar{T}$ .

Per poter svolgere questo esercizio, bisogna per prima cosa stabilire la finestra temporale di cui si vuole misurare la durata. L'unico segnale proveniente dall'esterno visto finora è la tensione del pulsante USER, della linea PA0.

**ESERCIZIO:** Misurare il tempo durante il quale si tiene premuto il pulsante USER.

La finestra temporale da misurare comincia quando si verifica la transizione basso-alto della linea PA0 (qualcuno ha premuto il tasto) e termina quando si verifica la transizione alto-basso di PA0 (il tasto è stato rilasciato).

Quando il pulsante viene premuto, deve iniziare il conteggio e quindi bisogna alzare il bit CEN del registro di controllo.

Il contatore a partire da 0 conta gli impulsi ogni 125 ns (il periodo di clock, è dato dal clock di sistema che ha frequenza 8 MHz, quindi gli impulsi distano l'uno dall'altro 1/8MHz ovvero 125 ns. Questo significa che il registro CNT del Timer si incrementa ogni 125 ns). Quando il pulsante viene rilasciato, bisogna fermare il conteggio e si ha così il numero di impulsi contati nell'intervallo di tempo in cui il pulsante era premuto.

Il valore della finestra temporale in secondi sarà dato dal valore ricavato dal CNT moltiplicato per il periodo di clock.

## OSSERVAZIONI:

- 1) La risoluzione con la quale viene misurato l'intervallo di tempo è il periodo di clock (125 ns).
- 2) L'intervallo di tempo massimo da misurare dipende dal tipo di Timer con cui si sta lavorando: se è a 32 bit si può misurare un intervallo di tempo molto grande, circa 50 s; se invece è a 16 bit si può misurare solo qualche millisecondo. Questo è importante, perché se si sceglie di lavorare con un Timer a 16 bit, c'è il rischio che il valore nel CNT non sia corretto, ad esempio se vengono contati più di 8 ms, il counter arriva a  $2^{16}$ , si azzera e ricomincia di nuovo da 0, il numero di conteggi in CNT è quindi sbagliato, perché il contatore nell'intervallo va più volte in overflow e quindi si azzera più volte. Per risolvere questo problema ci sono due opzioni:

- Gestione dell'overflow, monitorando il bit UIF e incrementando una variabile ogni volta che si alza per contare il numero di volte che il timer è andato in overflow.

- Si adopera il prescaler, in modo tale da modificare il periodo di clock (vedi Lab.5).

Il prescaler è la soluzione più facile, tuttavia comporta un peggioramento della risoluzione.

# LABORATORIO 7

lunedì 14 novembre 2016 16:40

Alternative alla variabile stato del "Progetto 5 Lezione":

```
1) while (uif==0; quando uif diventa 1-> è passato mezzo secondo  
2) while(1){  
//altre operazioni  
if (uif==0).... se uif è diventato 1 fai qualcosa.  
//altre operazioni  
}
```

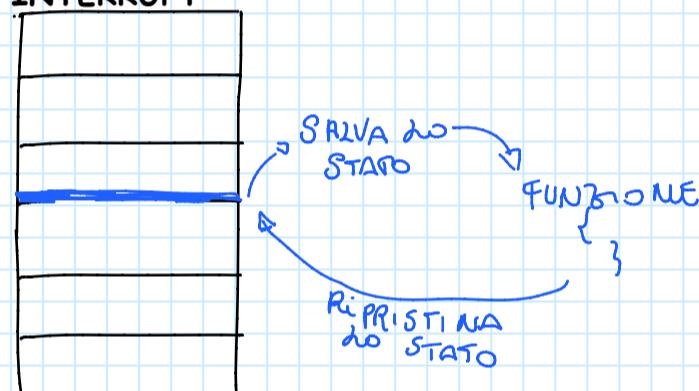
Questi due codici non sono ottimizzati, infatti lo scopo del programma è quello di accendere un led ogni mezzo secondo, quindi alla fine si vuole solo capire quando è passato mezzo secondo, nel frattempo si potrebbero fare tante altre cose. Tuttavia in entrambi i casi si hanno delle problematiche che non portano ad un utilizzo ottimizzato del microcontrollore

-Caso 1) Il microcontrollore resta nel while finchè non passa mezzo secondo, allora non fa altro che chiedersi se è passato mezzo secondo.

-Caso 2) Una volta che si entra nel while, prima che si alzi UIF potrebbero esserci altre operazioni in esecuzione. UIF si alza esattamente dopo mezzo secondo, tuttavia non è detto che la condizione dell'if sia verificata proprio quando passa un secondo, ma verrà verificata con un certo ritardo, a seconda del numero di operazioni che si stanno eseguendo prima dell'if.

Questo tipo di gestione del timer non va bene, il modo corretto sarebbe quello di eseguire le operazioni con il microcontrollore, finchè non scade il mezzo secondo, passato il mezzo secondo fare ciò che era stato richiesto (es. Accendere un led dopo mezzo secondo), per poi tornare a fare ciò che si stava facendo prima del mezzo secondo. In questo modo non vengono impiegate tutte le risorse ad aspettare che passi il mezzo secondo. Quest'operazione si può fare attraverso l'utilizzo degli Interrupt.

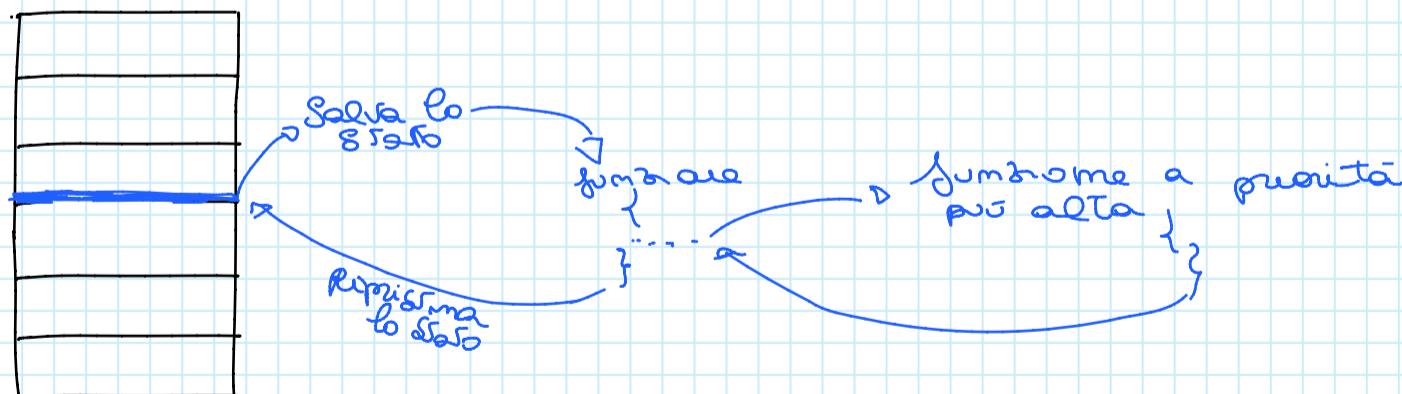
## INTERRUPT



Che cos'è un Interrupt?

Supponiamo che il microcontrollore stia eseguendo un'istruzione alla volta, il programma scritto nel main e che ad un certo punto si verifichi un evento definito come "Evento di Interrupt". Quando si verifica tale evento, il microcontrollore salva lo stato e salta ad eseguire una funzione, finchè non termina, per poi ripristinare lo stato e tornare al punto in cui stava prima dell'evento. Questo è un'esempio di funzione di Interrupt, detta anche ISR (Interrupt Service Routine).

Nei microcontrollori della ST, c'è un controllore a parte dedicato alla gestione degli interrupt e si chiama NVIC (Reference Manual pag. 183): Nested Vectored Interrupt Controller. Nested perché con i nostri microcontrollori è possibile fare degli interrupt innestati, nel senso che: supponiamo che viene eseguita una funzione di interrupt e che mentre è in esecuzione si verifichi una interrupt a priorità più alta -> allora si salta alla funzione di interrupt a priorità più alta per poi tornare alla prima funzione (prima dell'interrupt a priorità alta) e poi viene ripristinato lo stato.



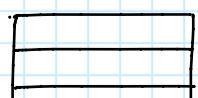
## Come funziona NVIC

In generale il microcontrollore viene programmato scrivendo una serie di istruzioni in memoria, scritte ad indirizzi consecutivi. Nei microcontrollori vecchi (ad esempio i PIC 16F che avevano istruzioni a 8 bit invece che a 32 bit) c'era un unico indirizzo, l'indirizzo 0x0004, che era riservato. Questo perchè il microcontrollore quando si verificava una causa di interrupt saltava direttamente all'indirizzo 0x0004.

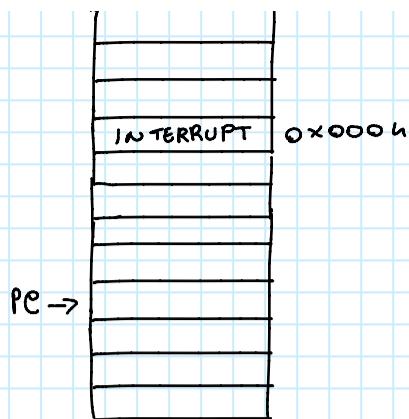
Che significa "saltare ad un istruzione"?

Esempio:

Memoria Programma

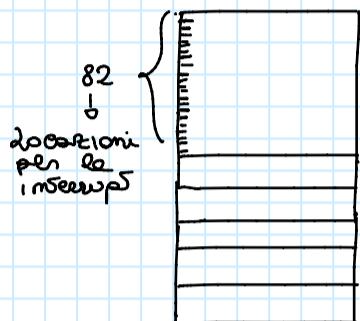


Consideriamo la memoria programma con istruzioni messe tutte in indirizzi consecutivi. Il



Consideriamo la memoria programmata con istruzioni messe tutte in indirizzi consecutivi. Il microcontrollore per capire quale istruzione deve eseguire si appoggia al registro Program Counter (PC) che contiene l'indirizzo dell'istruzione successiva. Il microcontrollore quindi esegue un'istruzione e incrementa il PC e fa questo tutte le volte.  
Se c'è un'istruzione di "salto" nel PC verrà scritto l'indirizzo su cui dovrà andare il microcontrollore (che non sarà sequenziale come gli altri), in particolare nei vecchi microcontrollori, quando si verificava una causa di interrupt, nel PC veniva scritto 0X0004 e il microcontrollore saltava a 0x0004.

Quindi il microcontrollore utilizza il registro PC per capire quale istruzione deve eseguire. A differenza dei vecchi microcontrollori, quelli attuali si sono evoluti a tal punto che, non appena si verifica un interrupt, non saltano più ad un indirizzo fisso, ma sono in grado di saltare ad un indirizzo particolare a seconda della causa che ha scatenato l'interrupt.



Se per esempio si potesse guardare la memoria del microcontrollore, si vedrebbe che le prime 82 locazioni sono occupate e servono per gli interrupt, nelle successive invece si può andare a scrivere il programma. Ci sono quindi 82 locazioni di memoria e a seconda della causa dell'interrupt, il microcontrollore salta alla i-esima locazione di memoria (con  $i=1, \dots, 82$ )

NVIC si occupa proprio della gestione delle cause di interrupt, in particolare deve capire perché si è verificata un interrupt e scrivere nel PC l'indirizzo al quale deve saltare il microcontrollore.

Che ne sa NVIC a capire chi ha generato la interrupt?

Ogni periferica se configurata opportunamente, in occasione di un dato evento, è in grado di mandare un segnale a NVIC, che si chiama Interrupt Request (IRQ) ed è proprio tramite questo segnale che NVIC riesce a capire la causa dell' interrupt e quindi a scrivere nel PC l'indirizzo opportuno a cui dovrà saltare il microcontrollore per poter gestire l' interrupt dovuto a quella particolare periferica.



#### Interrupts and Events (capitolo 11 Reference Manual).

A pag. 183 viene elencata una tabella, dove vengono mostrate tutte le possibili cause di interrupt che il microcontrollore sa gestire.

-Parte grigia: Cause che non possiamo abilitare né gestire perché sono interne al microcontrollore, in genere sono condizioni di default che il microcontrollore gestisce da solo.

-Parte bianca (pag. 184): Eventi di interrupt che noi possiamo abilitare e gestire programmando opportunamente il microcontrollore.

Scorrendo la tabella si può notare che ci sono cause di interrupt dovute al registro RCC; ci sono poi delle cause indicate con EXTI che sono cause di interrupt esterni, che si verificano quando I pin che interfacciano il microcontrollore con l'esterno hanno cambiato il loro livello (transizione: basso→alto || alto→basso); interrupt dovuti alla periferica DMA; convertitore analogico digitale (ADC)....

Analisi delle colonne:

Position	Priority	Type of priority	Acronym	Description	Address
				Reserved	0x0000 001C - 0x0000 002B
3	settable	SVCALL		System service call via SWI instruction	0x0000 002C
5	settable	PendSV		Pendable request for system service	0x0000 003B
6	settable	SysTick		System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog Interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line 16 detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper andTimeStamp Interrupts through the EXTI line 19	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup Interrupt through the EXTI line 20	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054

Indica la posizione nella quale dovrebbe andare il microcontrollore  
↓  
Es: se si verifica

COLONNA PRIORITÀ  
↓

Indica la priorità di ogni causa di interrupt  
(se non vengono fatte configurazioni particolari)

OSSERVAZIONE: È più prioritario il numero più basso  
↓

Es: se si verifica una causa di interrupt esterna (EXTI 0, una linea che si alza) il microcontrollore salta all'indirizzo 6; se si verifica una causa dovuta al canale 1 del DMA1 il microcontrollore salta a 11...

3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line 20	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2 and TSC	EXTI Line2 and Touch sensing interrupts	0x0000 0060
9	16	settable	EXTI3	EXTI Line3	0x0000 0064
10	17	settable	EXTI4	EXTI Line4	0x0000 0068
11	18	settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C
12	19	settable	DMA1_CH2	DMA1 channel 2 interrupt	0x0000 0070
13	20	settable	DMA1_CH3	DMA1 channel 3 interrupt	0x0000 0074
14	21	settable	DMA1_CH4	DMA1 channel 4 interrupt	0x0000 0078
15	22	settable	DMA1_CH5	DMA1 channel 5 interrupt	0x0000 007C
16	23	settable	DMA1_CH6	DMA1 channel 6 interrupt	0x0000 0080
17	24	settable	DMA1_CH7	DMA1 channel 7 interrupt	0x0000 0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000 0088
19 (1)	26	settable	USB_HP/CAN_TX	USB High Priority/CAN_TX Interrupts	0x0000 008C
20 (1)	27	settable	USB_LP/CAN_RXD	USB Low Priority/CAN_RXD Interrupts	0x0000 0090

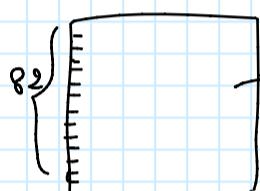
Il numero può 'bassso'

All'aumentare del numero diminuisce la priorità

la colonna Position va messa ogni volta che usciamo le interrupt

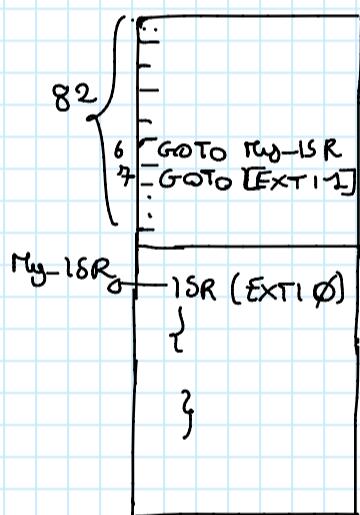
Ricapitolando, per ogni causa di interrupt c'è una locazione di memoria, in particolare si possono gestire da 0 a 81 cause (quindi 82). La funzione che si vuole far eseguire durante un interrupt, la ISR, sarà in realtà un insieme di istruzioni. Si può però osservare che nella parte di memoria riservata alle 82 cause di interrupt, non si può scrivere "un insieme di istruzioni per ogni causa", perché ad ogni causa è riservata una sola locazione di memoria da 32 bit. Se ad esempio scrivessi una funzione di quattro o cinque istruzioni, si andrebbe a scrivere sulle locazioni di memoria degli interrupt successivi.

Come si risolve questo problema?



In questa parte di memoria del microcontrollore, si hanno solo istruzioni di salto: un'unica istruzione che indica dove si trova la funzione (a più istruzioni) che deve essere eseguita al verificarsi di una particolare causa di interrupt.

ESEMPIO:



Supponiamo che si verifichi l'interrupt EXTI 0 e che al suo verificarsi si voglia eseguire una certa funzione, che chiamiamo "My\_ISR".

La funzione My\_ISR, che indica quindi cosa deve fare il microcontrollore se si verifica un interrupt dovuto a EXTI 0, andrà scritta in una zona qualunque della memoria del microcontrollore e non nella parte di memoria riservata alle 82 cause di interrupt, perché per ogni causa di interrupt si ha un'unica locazione di memoria. Sapendo però che al verificarsi di EXTI 0 il microcontrollore salta all'indirizzo 6, si può scrivere in tale indirizzo "goto My\_ISR".

Quindi non appena si verifica l'interrupt dovuto a EXTI 0, il microcontrollore salta all'indirizzo 6, dove trova un'unica istruzione che gli dice dove deve saltare per trovare la ISR che deve eseguire.

Analogamente all'indirizzo 7 si avrà "goto Funzione", che fa saltare il microcontrollore all'ISR per EXTI 1, visto che al 7 il microcontrollore va solo se si verifica l'interrupt dovuto a EXTI1.

E così per tutte le altre cause.

Per poter adoperare gli interrupt bisogna aggiungere al progetto un altro file: startup\_stm32f30x.s (dove il ".s" sta ad indicare che è un assembler). Il motivo per cui questo file è scritto in assembler è che attraverso l'assembler si ha un pieno controllo sulle istruzioni e in particolare sul modo in cui queste verranno inserite nella memoria flash del microcontrollore. Infatti dopo aver scritto un codice in C, il compilatore lo traduce poi in assembler ed infine il programma viene scaricato in linguaggio macchina nella flash del microcontrollore. Allora di un'istruzione scritta in C, non è possibile sapere come verrà tradotta, in quante istruzioni e dove verrà scritta. Scrivendo in assembler si ha invece il pieno controllo su queste cose, in particolare si avrà la certezza che ogni istruzione scritta in un certo ordine, verrà scritta in memoria secondo lo stesso ordine. Per capire cosa significa, basta guardare startup.s: essendo scritto in assembler, non ci sono dubbi che ad esempio EXTI 0 verrà scritta alla locazione di memoria 6, rispetto all'inizio dell'elenco (riga 79-> inizio).

Quindi questo pieno controllo si ha solo se si lavora in assembler.

Il file "startup\_stm32f30x.s" si trova nella cartella LIBRERIA ed è un file sorgente. Si aggiunge al progetto in modo analogo al main: tasto destro sul nome del progetto (in IAR)-> Add files->startup.s (dalla cartella LIBRERIA).

Analisi del file (visualizzandolo in IAR).

Dall'etichetta "\_vector\_table" (riga 60) in giù, si hanno tutte istruzioni di salto a particolari etichette.

Alla riga 79 iniziano gli "External Interrupts" che possiamo controllare.

Osservazione: se si confronta la tabella del Reference Manual a pag 183 con questo file.s, si vede che c'è una perfetta corrispondenza, ovvero in questo file vengono rispettate le posizioni elencate a pag. 183 del RM. Ad esempio nell'assembler EXTI

0 (riga 86), partendo dall'external interrupts (riga 80 posizione 0) è in posizione 6. In questo modo si trovano tutte gli altri interrupt.

Che cos'è DCC EXTIO\_IRQHandler? È un'istruzione di salto all'etichetta EXTIO\_IRQHandler. In pratica quando il microcontrollore legge questa istruzione, cerca una zona di memoria etichettata EXTIO\_IRQHandler e "salta lì".

Per ogni causa di interrupt si ha una stessa struttura (inizio riga 165) ed è costituita dalle seguenti parti.

-PUBWEAK (pubblica/debole): Sezione di memoria che si chiama esattamente come il nome dell'IRQHandler a cui fa riferimento.  
-B : Branch, ovvero "salta all'etichetta". Questa istruzione di default serve nel caso in cui non si riesce a trovare la ISR che si vuole far svolgere al microcontrollore nel caso di un interrupt, in pratica se non questa funzione non viene trovata, il microcontrollore resta alloopato in questa istruzione che inizia con B nome\_Handler. Se invece si trova una zona di memoria che ha lo stesso nome dell'etichetta indicata, allora quella zona sovrascrive questa istruzione con B e viene eseguita la funzione con le istruzioni che si vogliono far eseguire in caso di interrupt (questo è il significato di PUBWEAK).

Esempio con EXTIO:

```
254    EXTIO_IRQHandler   
255        B EXTIO_IRQHandler  
256  
257        PUBWEAK EXTI1_IRQHandler  
258        SECTION .text:CODE:REORDER(1)
```

Supponiamo di aver fatto saltare il microcontrollore alla posizione 6 e che lì abbia letto " B EXTIO\_IRQHANDLER " ovvero "salta a exti0 irq\_handler". Quando il microcontrollore arriva qui, eseguirà sempre la stessa istruzione "salta a exti0" restando quindi alloopata, oppure eseguirà una ISR precisa, con lo stesso nome scritto in  , se questa è presente/viene trovata, sovrascrivendo l'istruzione B EXTIO\_IRQHandler.

Questo è il motivo per cui questo file.s, ha per tutte le cause di interrupt la stessa porzione di codice, infatti tutto questo meccanismo infatti serve per evitare che, in assenza di una Interrupt Service Routine, si perda il controllo di quello che fa il microcontrollore,  
bloccandolo in un certo senso ad un'unica istruzione.

### Abilitazione degli interrupt

Per abilitare gli interrupt, bisogna effettuare due operazioni.

1) Dire alla periferica che in occorrenza di un dato evento deve inviare una Interrupt Request.

2) Dire a NVIC di servire la ISR comunicatagli dalla particolare periferica.

Mentre la prima va fatta tra I registri della periferica, la seconda va fatta adoperando NVIC.

Documentazione relativa a NVIC: - Programming manual cortex M4

- CortexM4TechRefMan (descrive più a fondo l'architettura del microcontrollore, non parla solo di NVIC. In particolare la parte su NVIC inizia a pag. 64, in questa lezione quando parlo dei registri di NVIC faccio riferimento a questo pdf a partire da questa pagina)

I registri di NVIC che servono ad abilitare un interrupt sono gli ISER (Interrupt Set-Enable Registers), otto registri da ISER 0 a ISER 7, dove ogni bit fa capo ad un interrupt. Perchè sono otto? Il motivo è che NVIC è stato realizzato per diverse famiglie di microcontrollori della st e quindi, sapendo che ogni registro ISER è a 32 bit, arriva a gestire  $8 \times 32$  cause di interrupt e non solo 82 come nel caso specifico del nostro microcontrollore.

Con queste premesse risulta ovvio che si adopererà:

- Iser 0-> per gestire 0,...,31 cause di interrupt
- Iser 1-> per gestire 32, ..., 63 cause di interrupt
- Iser2 -> per gestire 64,...81,...95 cause di interrupt, anche se nel nostro caso gli ultimi 14 bit sono inutilizzati, visto che il nostro microcontrollore può gestire solo 82 cause di interrupt diverse (quindi 0,...,81).

Per abilitare un interrupt, bisogna alzare un bit in particolare di questi registri, la scelta del bit è legata alla "Position" della tabella di pag. 183 (Reference Manual). I bit negli ISER seguono infatti lo stesso ordine delle "Position", per cui quello da alzare si troverà alla posizione corrispondente, indicata nella tabella.

Esempio:

- Per abilitare una interrupt dovuta a EXTI 0, sapendo che nella tabella ha posizione 6, bisogna alzare il bit 6 che si trova in iser 0.
- Per abilitare una interrupt dovuta a SPI1 (linea di comunicazione seriale),che ha posizione 35 nella tabella,bisogna alzare il bit 35 degli iser, in particolare si alzerà il bit 4 di ISER 1 (perchè il 35 ricade nell'intervallo: 32,...,63).

Prima di passare agli esercizi è bene notare che inserendo startup\_stm32f30x.s nel progetto, quando si compila si ha un errore, perchè non viene trovata la funzione "SystemInit", che evidentemente chiama l'assembler.

Bisogna aggiungere al progetto anche il file system\_stm32f30x.c, che sta sempre nella cartella Libreria e che contiene la funzione richiesta dall'assembler.

I codici di questo file e in particolare la funzione system init per lo più resettano il clock control su RCC, al fine di poter utilizzare la famiglia di microcontrollori stm32f30x al massimo delle loro potenzialità. Il nostro microcontrollore può eseguire fino a 72000000 di istruzioni al secondo, finora lo abbiamo utilizzato con un clock da 8MHz quindi, ricordando che il clock scandisce le istruzioni, lo abbiamo utilizzato facendogli eseguire 8000000 milioni di istruzioni al secondo.

La funzione system init introduce un pll che moltiplica il clock che viene dall'esterno (nel nostro caso quello dell'oscillatore con

frequenza 8MHz) per nove (vedere PLL Configuration, riga 342 del file system\_stm32f30x.c).

Allora la frequenza nuova, con queste modifiche, sarà data da  $(8 \times 9)$ MHz=72MHz, grazie alla quale il microcontrollore riesce ad eseguire le istruzioni più velocemente, ovvero 72000000 milioni di istruzioni al secondo e quindi potrà essere utilizzato al massimo delle sue capacità.

### ESERCIZIO 1.

Traccia, abilitare un interrupt dovuto a TIMER 2 , a valle dell'evento di update del Timer, e verificare che in assenza della ISR il programma si blocca all'istruzione B TIM2\_IRQHandler (riga 365 file startup\_stm32f30x.s, su cui mettere un breakpoint).

#### Indicazioni per la risoluzione

Il primo scopo dell'esercizio è quello di far generare un interrupt dall'evento di update di TIM2. Bisogna quindi lavorare su due fronti: Lato periferica e lato NVIC.

##### - Lato periferica.

Il registro che permette di configurare gli interrupt per I Timer è il DIER (Dma Interrupt Enable Register, di cui 16 bit vengono utilizzati per configurare I DMA, gli altri per configurare gli interrupt-> pag. 498)

In generale tutti bit che "lato periferica" servono ad abilitare un interrupt terminano in "ie", nel caso specifico del registro DIER, per abilitare l'interrupt sull'evento di update del timer bisogna alzare (quindi metterlo a 1) il bit UIE (Update Interrupt Enable).

Osservazione: dove è possibile si cerca di mantenere una coerenza posizionale (cioè la stessa posizione) dei bit, infatti il corrispondente flag del bit 0 del registro DIER (che serve per abilitare l'interrupt), è il bit zero dello Status Register.

##### -Lato NVIC.

Quindi alzando il bit UIE di DIER si sta dicendo al microcontrollore che in corrispondenza dell'update (ovvero quando il count arriva al valore in ARR), la periferica deve inviare una ISR.

Il bit che bisogna alzare per dire a NVIC che deve servire una ISR proveniente da TIM2 è il bit 28 di Iser0.

Infatti guardando la tabella di pag.185 del Reference Manual, si può leggere che Tim2 si trova in posizione 28, che rientra nell'intervallo

[0,31], per cui si considera Iser0 e in particolare bisogna mettere a 1 il bit 28.

Ricapitolando, per svolgere l'esercizio 1, bisogna seguire I seguenti punti:

- 1) Configurare ARR in modo che si conti mezzo secondo, ricordando che la nuova frequenza di clock è 72MHz.

$$T_{CK} = \text{nuovo valore del periodo di clock} = \frac{10}{72} \cdot 10^{-6}$$

$$N_{ARR} = \text{valore da mettere in ARR} = \frac{0,5}{T_{CK}} = 36\ 000\ 000 \rightarrow \text{con questo valore si può contare mezzo secondo con un clock a } 72 \text{ MHz}$$

- 2) Mettere UIE di DIER a 1 per abilitare l'interrupt sull'evento di update del timer.

- 3) Mettere il contatore a zero e abilitare il conteggio.

- 4) Abilitare l'interrupt su NVIC, alzando il bit 28 di Iser0.

Per questo esercizio non è richiesta anche la ISR, quindi se il microcontrollore è stato configurato bene per l'abilitazione dell'interrupt su Timer2, una volta raggiunto il valore in ARR si allooperà nell'istruzione "Salta a Tim2\_IRQHandler". Infatti mandando in esecuzione, quando Timer 2 conta mezzo secondo, chiede a NVIC di servire la Interrupt Service Routine, NVIC scrive quindi nel PC 28, il microcontrollore salta alla 29 posizione (posizione 28 partendo da zero) e trova "Salta a TIM2\_IRQHandler", ma la zona di memoria TIM2\_IRQHandler è questa qui, non ce ne sono altre e quindi resta bloccato su questa istruzione.

### ESERCIZIO 2.

Creare un contatore binario in modo tale che ogni secondo (in realtà era mezzo secondo) si incrementi una variabile contatore (cont).

Ad ogni conteggio deve accendersi un led diverso fino a quello legato a PE15, per poi ripartire da PE8. L'aggiornamento del conteggio (quindi della variabile cont) deve avvenire interrupt.

Quindi a differenza dell'esercizio 1, qui è richiesta anche la creazione di una ISR, una funzione che verrà eseguita ogni secondo, quando il programma va in interrupt, nella quale va incrementata la variabile cont eseguendo quest'operazione in maniera asincrona rispetto al main. che ogni secondo va in interrupt ad eseguire delle operazioni in maniera asincrona rispetto al main.

Quindi ogni secondo il programma va in interrupt e incrementa la variabile cont.

Per vedere un po' di dinamismo, si può fare un toggle: invece di accendere un led e di lasciarlo acceso, si può mettere un ciclo for per introdurre un certo ritardo, in modo tale che mentre il programma accende il led se ogni secondo, questo verrà poi spento con un certo ritardo impostato da noi.

Esempio : for (int i=0; i<10000000; i++);

Il valore massimo di I lo decidiamo noi in modo da riuscire a vedere ciò che accade e che non sia troppo veloce (scegliamo il numero più adatto affinché con l'occhio riusciamo a vedere questo led che si accende e si spegne).

NOTA: La ISR non va richiamata nel main, ma parte in automatico quando si verifica l'evento.

# LABORATORIO 8

lunedì 21 novembre 2016 16:37

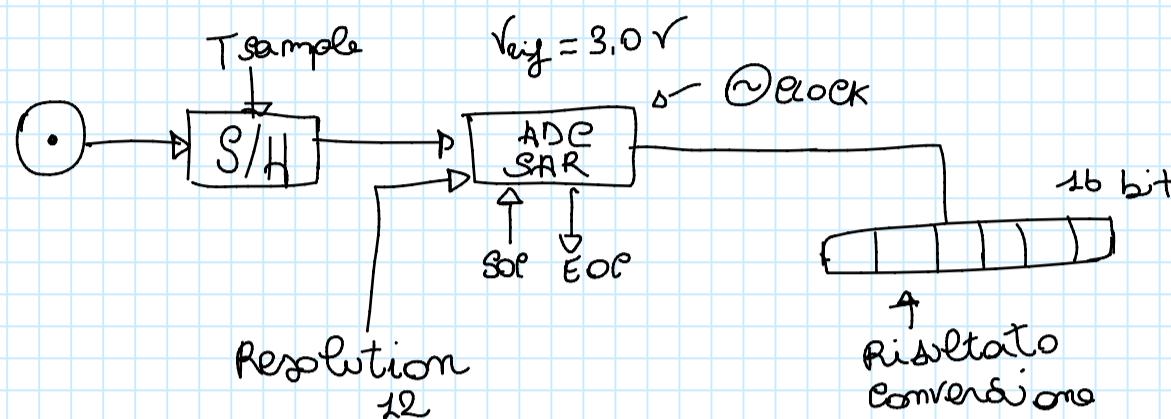
## CONVERTITORI ANALOGICO/DIGITALE

Gli ADC convertono I valori di tensione in ingresso nel numero corrispondente espresso in binario.

Nel nostro microcontrollore ci sono quattro convertitori analogico/digitale: ADC1, ADC2, ADC3, ADC4.

Poichè si ha la possibilità di far lavorare gli ADC a coppie (in particolare le coppie ADC1\_2 e ADC3\_4), nel senso che I due convertitori sono in qualche modo sincronizzati, per fare acquisizioni sincrone oppure per lavorare multiplexati (per raddoppiare la frequenza di campionamento equivalente), nei registri per l'abilitazione del clock si troverà un bit di abilitazione per la coppia ADC1\_2 e uno per la coppia ADC3\_4 (anche se si vuole usare un solo ADC, bisogna abilitare la coppia).

Schema del convertitore Analogico digitale.



Descrizione schema (lezione di riferimento teoria in cui viene spiegato tutto nei dettagli: ADC ad approssimazioni successive), con l'introduzione di ciò che andrà configurato.

Ognuno di questi quattro convertitori è un Sar a 12 bit.

Al suo interno entra una tensione da convertire, ovvero il segnale che si vuole convertire in digitale. Si sa dalla teoria che deve esserci un Sample&Hold per cui andrà configurato anche il tempo di sample, ricordando inoltre che nella fase di Hold il segnale costante viene inviato all'ADC.

Un convertitore Analogico /Digitale ad approssimazioni successive è costituito da un SAR e da un convertitore Digitale/Analogico, che ha una tensione di riferimento, che va abilitata. Le operazioni del SAR vengono scandite da un segnale di clock, che va configurato.

La conversione del segnale inizia quando l'ADC riceve un segnale che si chiama "Start Of Conversion" (SOC) , per cui va configurata la fonte di questo segnale (ovvero chi manda il SAC all'ADC). Al termine della conversione il convertitore risponde con un segnale "End Of Conversion" (EOC) e il risultato viene poi scritto in un registro a 16 bit.

Essendo un SAR è possibile configurare la risoluzione. Per come è fatto, un SAR determina un bit alla vota, dal più significativo al meno significativo.

Il convertitore per effettuare la conversione e quindi per determinare 12 bit, impiega 12 colpi di clock, in pratica determina un bit ad ogni colpo di clock. Se si vuole ad esempio velocizzare la conversione, perdendo però in risoluzione si può imporre al convertitore che dopo I primi 6 bit, deve fermare la conversione, in questo modo è come se avesse solo 6 bit, perchè determina solo I primi 6, e quindi il tempo di conversione vale 6 colpi di clock. In questo modo si dimezza la velocità ma si dimezza anche la risoluzione, a seconda dell'applicazione si sceglie poi quale prediligere.

Il succo di questa sorta di esempio è che solo con il SAR ( visto che è fatto in modo tale che ad ogni colpo di clock determina un bit) si ha la libertà di scegliere la risoluzione e quindi il tempo di conversione del convertitore, in base alle esigenze del programma. In particolare si può scegliere tra quattro valori: 6 bit, 8 bit, 10 bit, 12 bit.

CONVERTITORE ANALOGICO-DIGITALE-> Introduzione (Reference Manual pag. 200).

Come vanno configurate tutte queste caratteristiche di cui il convertitore ha bisogno per poter lavorare?

ADC1 e ADC2 sono strettamente accoppiati e possono operare in modalità duale (stesso discorso per ADC3 e ADC4).

Ogni ADC consiste di un ADC ad approssimazione successiva di 12 bit e ha (soggetto "Ogni ADC") fino a 19 canali multiplexati, nel senso che all'ADC arrivano 19 canali, ai quali possono essere collegati 19 segnali. I canali che si vogliono convertire (quelli a cui sono collegati I segnali che si vogliono convertire) vengono collegati tramite un multiplexer al convertitore analogico-digitale. Il risultato dell'ADC è conservato in un registro a 16 bit allineato a destra o a sinistra.

Perchè l'allineamento?

Alla massima risoluzione si hanno come risultato 12 bit che vengono scritti in un registro a 16 bit. Se si sceglie :

- Allineamento a Destra: I primi 4 bit del registro a 16 bit sono pari a zero, I 12 bit restanti sono quelli del risultato .
- Allineamento a Sinistra: I primi 12 bit sono quelli del risultato, I restanti 4 sono pari a 0.

Infine l'ADC è mappato sul Bus AHB.

CARATTERISTICHE DEGLI ADC (Reference Manual pag 201)

- 1) ADC1 è connesso a 10 canali esterni + 4 interni  
ADC2 è connesso a 12 canali esterni + 2 interni

- ADC3 è connesso a 15 canali esterni + 2 interni  
 ADC4 è connesso a 13 canali esterni + 2 interni
- 2) La risoluzione configurabile 12 o 10 o 8 o 6 bit
  - 3) Tempo di conversione minimo (sampling + conversione del tempo) : 0,19  $\mu$ s per I Fast Channels (canali etichettati come "Fast") che corrispondono a frequenza di 5.1 Ms/s. (dove MS sta per Mega Sample); 0,21  $\mu$ s per I Slow Channels che corrispondono a frequenza di 4.8 Ms/s.
  - 4) Canali Interni: canali collegati ad alcuni segnali interni al convertitore. In particolare si ha un canale di ADC1 collegato all'uscita di un sensore di temperatura. Quindi configurando opportunamente il convertitore si può misurare la temperatura perché viene convertito in digitale l'uscita di un sensore di temperatura. Un altro segnale interno, nel caso in cui si lavori con una batteria, è la tensione di batteria. Ancora un altro segnale interno è la tensione di riferimento. Altri segnali sono le uscite degli amplificatori operazionali: all'interno ci sono anche degli amplificatori operazionali, che servono per fare circuiti analogici di condizionamento (filtri, amplificatori differenziali, amplificatori di tensione,...).
  - 5) La Start of Conversion (pag.202) può essere inizializzata: via software sia per le conversioni regolari che injected oppure da un trigger hardware, di cui si può configurare la polarità (fronte di salita o fronte di discesa di un particolare segnale hardware).  
 Le conversioni injected, sono in pratica quelle innestate: Mentre si stanno convertendo dei canali si salta a convertire altri canali che si chiamano injected (cosa che ignoriamo-> noi ci occuperemo solo delle conversioni regolari).
  - 6) Modalità di conversione  
 Ogni convertitore può convertire un singolo canale o tramite un multiplexer effettuare una scansione di più canali (visto che ne ha fino a 19 collegati). Si distinguono varie modalità di conversione:
    - Single Mode: Il convertitore, al trigger, esegue una conversione e poi si ferma;
    - Continuos Mode: Ogni volta che viene letto il registro dove è contenuto il risultato della conversione, automaticamente il convertitore inizia una nuova conversione, non aspetta il trigger.

#### COSE CHE ANDREMO A CONFIGURARE, Schema pag.204 Reference Manual

- Input Selection & Scan Control: qui arrivano tutti I canali dell'AD e noi configuriamo questa parte dicendo quali canali vogliamo convertire. In questo punto entrano I canali ADC\_IN da 1 a 15, dopodichè entrano due fili VINP e VINV (P e N stanno per positivo e negativo). Il convertitore può effettuare infatti due tipi di misure:  
 Misura Single Ended-> è la misura della tensione sul canale rispetto alla massa.  
 Misura Differenziale-> è la misura della differenza di tensione tra due canali.
- Start & Stop Control ( chi controlla lo Start of Conversion).  
 Qui c'è il pin "software trigger", che ci consente di far partire lo Start of Conversion per via software, oppure si può fare in modo che il segnale arrivi da una circuiteria esterna, dove si hanno una serie di segnali esterni dei quali si può selezionare la polarità (lo start of conversion, il fronte di salita di discesa, entrambi I fronti,...)
- EXTEN [1;0]: Permette di abilitare Il trigger esterno, agendo sui bit 1 e 0 per poter scegliere la modalità del trigger esterno.
- EXTSEL [3; 0]: Da la possibilità di dire quale dei possibili segnali deve fare da trigger, utilizzando I quattro bit da 0 a 3 che si chiamano appunto extsel.
- SMP [2; 0]: Usando questi tre bit (dopo la selezione degli input) è possibile stabilire il tempo di sampling.
- Tensione di Riferimento: deve essere compresa tra 1.8 V e 3.6 V, nel nostro caso è 3 V.
- ADEN ADDIS: bit che serve ad abilitare l'ADC (ADEN) o a disabilitarla (ADDIS).
- ADCAL (bit).
- RDATA: Registro in cui viene scritto il risultato della conversione.
- Vari bit di abilitazione  
 OVRMOD permette di specificare cosa deve fare l'ADC se ha eseguito una nuova conversione ma noi non abbiamo letto ancora il vecchio risultato e si troverebbe a sovrascrivere il risultato precedente. Noi decidiamo come deve comportarsi.  
 ALIGN.: Da la possibilità di scegliere se si vuole allineare a destra o sinistra.  
 RES [1;0]: sono due bit con I quali si sceglie la risoluzione.  
 JOFFSET: riguarda le injected.

#### SCHEMA DI INGRESSO ADC 1 E ADC2 (Pag. 206 Reference Manual)

Se si vuole fare una misura single ended di ognuno dei canali che arrivano all'ADC, si manda al convertitore il singolo canale rispetto alla ( $V_{REF}$ ) che è la massa, o meglio la tensione di riferimento bassa. Se si vuole fare invece una misura differenziale non si può scegliere la differenza tra due canali a caso, ma I canali da scegliere devono essere consecutivi. Quindi si può fare una misura differenziale solo tra due canali consecutivi (es. IN1 e IN2, IN2 e IN3, IN3 e IN4,...). I primi cinque canali sono I Fast Channels, gli altri sono Slow. Al SAR arrivano VINP e VINV, dove VINV in modalità single ended è la massa. O Lo schema è lo stesso per tutti I convertitori.

#### ADC VOLTAGE REGULATOR -> ADVREGEN (Reference Manual pag. 208)

Il convertitore analogico digitale confronta le tensioni di ingresso con una di riferimento ( $V_{REF}$ ) che deve essere quanto più stabile possibile. A tale scopo, c'è un circuito che garantisce una tensione di uscita quanto più stabile possibile e questo circuito si chiama Regolatore di Tensione.

Il Regolatore di Tensione riceve la tensione dall'alimentazione e fornisce in uscita una tensione di 3 V quanto più stabile

possibile.

Per questioni di Power-Safe, cioè per non consumare troppo, questo regolatore di tensione di solito è spento, quindi se si vuole operare il convertitore analogico digitale, deve essere acceso ed è la prima cosa da fare.

#### Attivazione di ADVREGEN

Si attiva agendo su due bit ADVREGEN, che di base devono essere portati a 0x1 dal loro valore di reset 1x0

Per evitare che si attivi il regolatore di tensione involontariamente, non si può fare questo passaggio in un'unica istruzione, ma deve essere seguita una precisa sequenza.

Sequenza: da 1x0 a 0x0 e da 0x0 a 0x1.

Nota: per disabilitarla, sequenza inversa ovvero da 0x1 a 0x0, e da 0x0 a 1x0.

Il software (quindi il nostro programma) deve aspettare il tempo di start up del regolatore di tensione: appena si attiva il regolatore c'è un transitorio, per cui prima di avere I 3 V stabili bisogna aspettare la fine di questo transitorio. Nella peggiore delle Hp il tempo di start up è di 10  $\mu$ s.

Quindi il primo passo da fare è quello di abilitare il Regolatore di tensione e aspettare almeno 10  $\mu$ s, ovvero il tempo necessario che finisca quest'operazione.

$\mu$ s

SINGLE ENDED AND DIFFERENTIAL INPUT CHANNELS (Reference Manual pag. 208-209)  $\mu$ s

I canali possono essere configurati per una misura Single -Ended o Differential.

La selezione di una delle due modalità avviene con dei bit che si chiamano DIFSEL del registro ADC\_DIFSEL.

In modalità single ended viene convertita la tensione tra l'i-esimo canale e la  $V_{REF}$  -, cioè la massa, mentre in modalità differenziale viene convertita la tensione dal canale  $IN_i$  e il canale  $IN_{i+1}$ .

#### CALIBRAZIONE (Reference Manual pag. 209)

Una volta che si ha un riferimento di tensione stabile, si adopera l'Autocalibrazione: L'ADC che si sta utilizzando, in base alla  $V_{REF}$  effettua un'autocalibrazione.

L'autocalibrazione è un'operazione preliminaria, da fare prima di ogni operazione dell'ADC, se non si fa non si può fare niente.

Bit ADCALDIF: se messo a 0, vuol dire che si vuole calibrare l'ADC per effettuare misure Single ended, se messo a 1 si vuole calibrare l'ADC per misure differenziali. Il reset è 0.

La calibrazione è poi inizializzata dal software (la programmiamo noi) settando il bit ADCAL a 1, cioè viene fatta partire alzando il bit ADCAL. Attenzione, l'autocalibrazione può partire solo quando il bit ADEN è 0 (cioè solo se l'AD è disabilitato). Il bit ADCAL sta a 1 durante tutta la procedura di calibrazione, si azzera solo quando la procedura è terminata.

Ricapitolando ciò che abbiamo fatto finora: se si vuole utilizzare un AD bisogna attivare il regolatore di tensione e aspettare 10 microsecondi; Alzare ADCAL e aspettare che si abbassi.

Una volta che ADCAL è tornato a 0, se per curiosità di vogliono vedere I fattori di calibrazione, questi si trovano nei due registri CALFACT.

#### SCHEMA DELLA PROCEDURA SOFTWARE PER CALIBRARE ADC (pag 210 RM)

- Assicurarsi che ADVREGEN sia abilitato.
- Assicurarsi che ADEN sia 0.
- Selezionare se la calibrazione la si vuole single ended o differenziale.
- Mettere ADCAL a 1.
- Aspettare che ADCAL passi a 0.

Tutte queste operazioni (Regolatore di tensione e Calibrazione) sono fatte a convertitore disabilitato.

Fatte tutte queste operazioni preliminari si può quindi abilitare il convertitore (pag. 211 Reference Manual) e tale operazione è possibile utilizzando il bit ADEN che deve essere impostarlo a 1.

Al convertitore serve un certo tempo per l'abilitazione, in particolare sarà pronto alle conversioni quando alza un bit che si chiama ADRDY.

Quindi una volta alzato ADEN, bisogna aspettare che ADRDY passi a 1.

Osservazione : si può anche abilitare un interrupt sull'evento ADRDY, settando un bit che si chiama ADRDY Interrupt Enable. La conversione regolare

Si può anche abilitare un'interrupt quando ADRDY diventa a 1 ADRDY Interrupt enable

La conversione regolare può essere fatta partire sia settando un bit ADSTART (caso del trigger software) mettendolo a 1 (In pratica è lo start of conversion dato via software), sia quando occorre un evento di external trigger.

#### ADCCLOCK (Reference Manual pag 212).

Ci sono 4 possibili opzioni.

- 1) Sorgente di clock esterne : si fornisce il segnale di clock esternamente all'ADC, che sarà indipendente e asincrono rispetto al resto dello schedino, perchè viene fornito da fuori.
- 2) Si può derivare il segnale di clock dell'ADC dal bus AHB al quale è collegato. In pratica questo non è altro che il

segnalet dell'oscillatore a 72 MHz (usandolo al massimo delle sue potenzialità, vedi laboratorio 7).

Il segnale di clock può essere preso diviso per 1, diviso per 2 o diviso per 4.

La selezione la si fa con due bit che si chiamano CKMODE.

Perchè si ha la possibilità di scegliere il clock?

Se si rallenta il clock, la conversione è più lenta, tuttavia se il convertitore lavora più lentamente si risparmia in consumo di potenza.

NOTA: in un secondo momento leggere il paragrafo 12.4.9 (pag. 213), Vincoli quando scrivi nei control bit di ADC, perchè tutte queste cose fatte finora vanno fatte seguendo dei precisi vincoli.

#### SELEZIONE DEL CANALE (Reference Manual pag 214)

Visto che al singolo ADC sono collegati fino a 19 canali, come si fa a dire quali e quanti canali si vogliono convertire?

Utilizzando dei registri che si chiamano SQRx. SQR sta per "Registro della Sequenza" e si chiama così perchè l'insieme di canali che si vogliono convertire viene chiamato "SEQUENZA" di conversione.

In questo paragrafo vengono elencati tutti I canali, è bene memorizzarli, in particolare si ha:

- Il sensore di temperatura, è collegato a canale 16 di ADC1.
- La batteria è collegata al canale 17 di ADC1.
- La tensione di riferimento è collegata al canale 18 di tutti e quattro gli ADC.

Un gruppo regolare è composto da fino a 16 conversioni. I canali regolari e il loro ordine nella sequenza di conversione deve essere selezionata nei registri ADC\_SQRx. Il numero totale di conversioni nel gruppo deve essere scritto in 4 bit indicati con L, nel registro ADC\_SQR1.

ES.

Se voglio convertire I canali 3 5 e 8, significa che il multiplexer collega l'ADC in modo che vengano convertiti in sequenza 3 5 e 8 ad ogni evento di trigger. La lunghezza è 3, perchè sono 3 I canali da convertire, per cui in L devo scrivere qualcosa che indica tre.

Nello scrivere la sequenza dovrò scrivere proprio 3, 5 e 8 nell'ordine che si vogliono convertire.

#### TEMPO DI SAMPLING PROGRAMMABILE (Reference Manual pag 215).

Si va da un minimo che sono 1.5 colpi di clock, fino al massimo 601.5 colpi di clock.

Il ".5" tiene conto degli eventuali ritardi del circuito ed è fornito in colpi di clock per l'ADC in modo tale che è possibile stimare il tempo di conversione totale in maniera più semplice.

Se ad esempio fisso come tempo di Sampling il minimo possibile (1.5 colpi di clock), il tempo di conversione totale sarà dato dalla seguente espressione:

- Tempo conversione totale=  $(12.5 + T_{SMP})/72MHz$

Dove 12.5 deriva dal fatto che il SAR, se lavora a 12 bit, impiega appunto 12 bit (qui mette un ".5" per eventuali ritardi anche del SAR).

Diviso 72MHz solo se si lavora con il clock dell'oscillatore, altrimenti la frequenza del segnale di clock che si sta utilizzando.

Si ha la possibilità di cambiare il tempo di sampling perchè se il tempo di sampling è elevato allora il condensatore si carica al valore corretto di tensione, ma aumenta il tempo di conversione.

I registri nei quali si va a mettere un tempo di sampling si chiamano SMPR1 e SMPR2.

#### MODALITA' DI CONVERSIONE SINGOLA (Reference Manual pag 216)

Se si lascia un bit che si chiama CONT a 0 (che è il suo valore di reset), vuol dire che per ogni conversione si deve specificare il trigger (cioè se lo lascio a 0 per ogni conversione voglio dare io il trigger). Se metto questo bit a 1, dopo aver messo la prima volta il trigger l'ADC dopo ogni risultato, attacca una nuova conversione.

La conversione con CONT=0 è iniziata in due modi : settando il bit ADSTART del registro ADCCR (JADSTART sta per le injected) o con un evento di trigger esterno. All'interno della sequenza regolare di conversione, dopo che si è completata ogni conversione, il dato convertito viene registrato nel registro a 16 bit che si chiama ADC\_DR e settato un flag che si chiama EOC (End of conversion). Se si abilita un interrupt su EOC alzando EOCIE, si ha un'interrupt al termine della conversione. Quando invece si dà una sequenza, dopo ogni conversione si alza l'EOC ma alla fine di tutta la sequenza si alza EOS (End of Sequence). EOC si abbassa automaticamente dopo aver letto il valore nel DATA REGISTER.

#### TIMING (Reference Manual pag 218)

Formula sopra sul Tempo di conversione totale, a seconda delle configurazioni fatte.

#### ABILITAZIONE DEL TRIGGER HARDWARE (Reference Manual pag 219).

Si possono abilitare I trigger Hardware e sceglierne la polarità, con due bit che si chiamano EXTN.

La prima volta ADSTART deve comunque essere messo a 1 altrimenti I trigger hardware vengono ignorati, mettendo invece la prima volta ADSTART a 1, ogni volta che parte un trigger hardware inizia una conversione.

A pag 220 c'è una tabella in cui viene mostrata la corrispondenza tra la polarità e I due bit EXTN. Se li lascio a 00

voglio lavorare con un trigger software altrimenti, altrimenti con un trigger sul fronte di salita, di discesa o su entrambi. Se si usa un trigger esterno il segnale che farà da trigger deve essere specificato configurando quattro bit che si chiamano EXSEL.

A pag. 221 c'è una tabella di corrispondenza tra I quattro bit EXTEL e tutti I possibili segnali che si possono adoperare come trigger.

#### RISOLUZIONE PROGRAMMABILE (Reference Manual 235)

Due bit che si chiamano RES con I quali è possibile configurare la risoluzione : 6, 8, 10, 12 bit.

#### 3 FLAG (Reference Manual pag 235).

- End Of Conversion.
- End of Sampling (si alza quando termina la fase di sampling, non lo useremo mai, perchè noi vogliamo essere avvertiti quando è terminata tutta la conversione).
- End of Sequence (indica quando termina una sequenza di canale).

A ognuno di questi flag è possibile associare un interrupt.

#### GESTIONE DEI DATI (Reference Manual pag 237)

A pag. 239 sono presenti degli schemi che mostrano l'allineamento a destra e quello a sinistra.

Consiglio: Utilizzare sempre l'allineamento a destra, perchè se si adopera quello a sinistra si trova il risultato della conversione moltiplicato per 16 (I 12 bit e 0000).

#### ADC OVERRUN (Reference Manual pag. 241)

Con il bit OVRMOD, si configura come deve comportarsi l'ADC in caso di Overrun.

#### REGISTRI ADC

Nel Reference Manual I registri dell'ADC sono suddivisi in due sezioni: registri per configurare ogni ADC singolarmente ; registri comuni per gli ADC, cioè I registri che servono a configurare le coppie ADC1\_2 e ADC3\_4.

##### 1) Registri per configurare gli ADC singolarmente (Pag.266).

- Registro ISR (pag. 266): lo Status Register, in cui si trovano I flag.  
Bit significativi di questo registro sono I seguenti.
  - ADRDY (Bit 0) : che si alza quando è stato abilitato con ADEN l'ADC e l'ADC è pronto. In particolare dopo aver messo ADEN a 1, questo bit resta 0 se ADC non è ancora pronto per la conversione, altrimenti passa a 1.
  - EOSMP, EOC, EOS (rispettivamente bit 1, 2, 3): valore di reset 0, passano a 1 quando si verifica la fine di uno degli specifici eventi a cui si riferiscono.
  - OVR (bit 4): indica se si è verificato un overrun impostando il suo valore a 1 (dal valore di reset 0).  
Tutto il resto è injected. Si possono poi configurare delle finestre di watchdog analogico che avvertono quando il segnale è entrato o uscito da una particolare finestra di valori di tensione. I bit che riguardano questa funzionalità sono : AWD1, AWD2, AWD3.

##### • Registro IER (pag. 269): Interrupt Enable Register, permette di abilitare un interrupt su uno dei flag.

- ADRDYIE (bit 0): Impostandolo a 1, abilita l'interrupt sull'ADRDY.
- EOSMPIE, EOCIE, EOSIE (rispettivamente bit 1,2,3): messi a 1 abilitano l'interrupt alla fine di uno degli eventi.
- OVRIE (bit 4): Messo a 1 abilita l'interrupt sull'overrun.  
Da osservare la corrispondenza posizionale tra I due registri.

##### • ADC CONTROL REGISTER (CR->pag.271).

- ADCAL (bit 31) : Reset 0, deve essere alzato quando si vuole dare il via all'autocalibrazione
- ADCLDIF (bit 30) : Calibrazione differenziale (1) o single ended (0->Reset)
- ADVREGEN (bit 29 e bit 28): I due bit del regolatore di tensione che hanno valore di reset 10 quando ADC è disabilitato e che vanno impostati a 01 per poter abilitare AVC, seguendo la sequenza già indicata.  
Osservazione: i due bit sono a 00 solo nel passaggio intermedio per l'abilitazione; 11 è riservato.
- ADSTART (bit 2): trigger software, impostando a 1 il suo valore dal valore di reset 0, da inizio alla conversione
- ADDIS (bit 1): scrivendo 1 disabilita l'ADC
- ADEN (bit 0): scrivendo 1 abilita l'ADC.

##### • ADC CONFIGURATION REGISTER (CFGR-> pag. 274)

Sono presenti i bit di configurazione dell'ADC ,

- RES (bit 3 e bit 4): servono a settare la risoluzione della conversione  
00-> 12 bit  
01-> 10 bit

10-> 8 bit

11-> bit

- ALIGN (bit 5): per selezionare l'allineamento a destra (0->reset) o a sinistra (1).
- EXTSEL (bit 6, bit 7, bit 8, bit9): per la selezione del trigger esterno
- EXTN (bit 10, bit 11): selezione polarità del trigger esterno.
- OVRMOD (bit 12): come comportarsi in caso di Overrun
- CONT (bit 13): per settare la modalità continua (1) o singola (0)
- AUTDLY (bit 14): eventuale ritardo da applicare (mettendolo a 1) dopo il trigger

Ci sono poi le configurazioni delle finestre del watchdog analogico.

- REGISTRI SAMPLE TIME (SMPR-> pag. 278 e 279).

Ci sono due registri di Sample: Sample time 1 e Sample time 2.

Ogni registro è suddiviso in terne di bit che si chiamano SMPx con x=1,..,18.

In particolare se si vuole modificare il tempo di sampling del canale 1, allora si scriverà in SMP1 in SMPR1; se si vuole modificare il tempo di sampling del canale 2, si scriverà in SMP2 di SMPR2, e così via.

Solo che SMPR1 arriva fino a SMP9, quindi in SMPR1 si può modificare il tempo di sampling fino al canale 9 lasciando così fuori 10 canali. A tal scopo c'è quindi anche SMPR2, per poter modificare il tempo di sampling dei restanti canali. Per cui si scriverà in SMP10 in SMPR2 per il canale 9, in SMP17 in SMPR2 per il canale 17,....

E questo è il motivo per cui ci sono due registri per il tempo di sampling.

A prescindere dal registro o dalle terne di bit, I valori da scrivere in SMPx saranno I seguenti:

000 -> 1.5 ADC clock cycles

001 -> 2.5 ADC clock cycles

010 -> 4.5 ADC clock cycles

011 -> 7.5 ADC clock cycles

100 -> 19.5 ADC clock cycles

101 -> 61.5 ADC clock cycles

110 -> 181.5 ADC clock cycles

111 -> 601.5 ADC clock cycles

- REGISTRO PER LA SEQUENZA REGOLARE (Regular Sequence Register SQRx con x=1,..4 -> pag. 282 a pag. 286)

In questo registro va scritta la sequenza dei canali che si vogliono convertire. I quattro bit meno significativi (0,1,2,3) servono per la lunghezza della sequenza, in particolare si scriverà (in binario):

0 (0000) se si vuole effettuare 1 conversione, valore di reset;

1 (0001) per 2 conversioni;

2 (0010) per 3 conversioni;

....

15 (1111) per 15 conversioni.

Dopo questi primi 4 bit (presenti solo in SQR1), I registri sono suddivisi in bit denominati SQx, con x =1, .., 16.

In questi bit va scritto il numero del canale che si vuole convertire nella sequenza, ricordando che l'ordine è importante: viene convertito prima il canale scritto in SQ1, poi quello in SQ2, poi quello in SQ3, ....

ESEMPIO: Voglio convertire la sequenza costituita dai tre canali 3,8,5 nel seguente ordine.

Nei bit indicati con L scriverò 0010 , perché voglio convertire tre canali.

Poi scriverò in SQ1-> 00011 (3)

SQ2-> 01000 (8)

SQ3-> 00101 (5)

Il motivo per cui ci sono più registri SQR è simile a quello per il tempo di sampling. Poiché la sequenza può essere costituita da massimo 16 canali, queste sedici "posizioni" sono suddivise in tutti e quattro I registri. Se ad esempio volessi convertire una sequenza di 6 canali, I primi quattro canali dovrei scriverli in SQR1, poi però gli altri due, poiché è finito lo spazio in SQR1, devono essere scritti nei primi 10 bit meno significativi di SQR2.

Se voglio convertire una sequenza di 11 canali, I primi 4 canali andranno scritti in SQR1, I 5 canali successivi andranno In SQR2, e gli ultimi due canali andranno poi nei primi 10 bit meno significativi di SQR3.

Quindi si può fare una scansione fino a 16 canali, che devono essere scritti in SQRx

- DATA REGISTER (DR-> pag.287)

In questo registro è scritto il risultato della conversione (per questo è un registro a sola lettura), allineato a sinistra o a destra, a seconda di come abbiamo configurato.

## 2) REGISTRI COMUNI (Pag. 294)

- REGISTRO COMMON STATUS REGISTER (CSR->294)

Questo registro è la copia degli Status Register dei due convertitori della coppia (es. ADC1\_2 o ADC3\_4), visto che possono essere utilizzati in modalità duale e quindi possono essere utilizzati tutti e due insieme, per comodità vengono I due status register in questo unico registro.

- REGISTRO COMMON CONTROL REGISTER (CCR->pag 296)

Questo registro permette di abilitare il canale dove c'è la tensione di batteria, l'abilitazione del sensore di temperatura, l'abilitazione del canale dove c'è la tensione di riferimento (in pratica I canali interni che si possono misurare con l'ADC).

In particolare i bit indicati con CKMODE (bit 16 e 17), permettono di scegliere la frequenza del segnale di clock. Il motivo per cui si trova in questo registro è che visto che ADCx e ADCy possono lavorare in modo duale, devono condividere il clock.

17 e 16 clock mode, I due bit chon I quali sceglio la frequenza di clock. Perch+è li trovo qui? Perchè ad esempi AD1 I possibili valori di clock mode possibili sono:

-00-> clock esterno asincrono

-01-> clock del bus AHB, I 72 MHz

-10-> ((72)/2)MHz

-11-> ((72)/4)MHz

### ESERCIZIO DEL GIORNO

Realizzare un programma che permetta di misurare iterativamente la tensione del canale PA0, quello del pulsantino USER (l'unico segnale che siamo in grado di modificare). Quindi lo scopo è quello di convertire la tensione di PA0.

In particolare si avrà che quando il pulsante viene premuto il convertitore converte la massima tensione che può convertire (cioè I 3 V), quando viene rilasciato si ha come valore di tensione 0.

Collegamento tra I canali dell'ADC e I pin esterni.

Nel Data Sheet a pag. 36 c'è una tabella che indica ogni pin del microcontrollore a quale periferica è collegato.

Da questa si ricava facilmente che, fra tutte le periferiche a cui è collegata PA0, PA0 è anche IN1 di ADC1 (e quindi il canale 1 di ADC1).

Table 13. STM32F302xx/STM32F303xx pin definitions (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
LQF P100	LQF P64	LQF P48					Alternate functions	Additional functions
23	14	10	PA0	I/O	TTIa		USART2_CTS, TIM2_CH1_ETR, TIM8_BKIN <sup>[2]</sup> , TIM8_ETR <sup>[2]</sup> , TSC_G1_IO1, COMP1_OUT	ADC1_IN1, COMP1_INM, RTC_TAMP2, WKUP1, COMP7_INP <sup>[2]</sup>  ADC1_IN2, COMP1_INP,

### SCHEMA PER LA RISOLUZIONE DELL'ESERCIZIO

- Abilitazione del clock di ADC1\_2 (si abilita il clock della coppia-> quando si lavora con gli ADC, bisogna sempre abilitare il clock delle coppie ADC1\_2 e ADC3\_4, a seconda di quello che si deve utilizzare), alzando il bit 28 del registro RCC\_AHBENR.
- Abilitazione del clock di PA0.
- Settare PA0 come analogico digitale (quindi mettere 11 in Modero nel registro GPIOA\_MODER). In questo modo PA0 viene internamente collegato al convertitore analogico digitale.
- Abilitare il regolatore di tensione con la sequenza già indicata (scrivere quindi in ADVREGEN del CR prima 00 e poi 01), aspettare poi 10 microsecondi.

10 microsecondi rappresenta un transitorio che si deve estinguere, è un tempo minimo da aspettare e non deve essere necessariamente preciso, per cui si può mettere un ciclo for.

- Configurare il CKMODE del registro ADC1\_2\_RCC, con I valori 01 in modo tale che la frequenza di clock sia quella del bus AHB ovvero 72MHz.

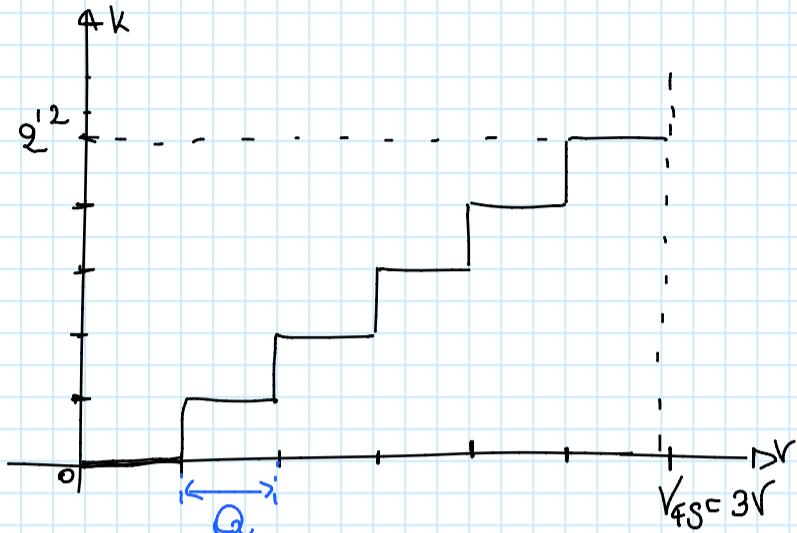
La configurazione del CKMODE va fatta prima dell'autocalibrazione, altrimenti non può essere eseguita.

- Calibrazione, quindi alzare ADCAL del CR e aspettare che passi a 0 e quindi che la calibrazione finisca.
- Abilitare ADC, impostando ADEN a 1 e aspettare che il convertitore sia pronto, quindi che ADRDY passi a 1.
- Configurazioni varie per I registri CFGR se diverse da quelle di reset (quindi impostare la risoluzione, l'allineamento).
- Impostare il registro SQR, lasciando la lunghezza a 0, visto che la sequenza da convertire nel nostro caso è costituita da un solo canale, e scrivendo 1 (in binario->0001) in SQR1, visto che PA0 è il canale 1 di ADC1.
- Impostare il tempo di sampling in SMPR, in modo che abbia un valore medio quindi 19.5 colpi di clock. Questo perchè anche se si ha a che fare con segnale costante (quello del pulsante), per cui non serve un grosso tempo di sampling, tuttavia nella transizione che si ha nel premere il pulsante e poi rilasciarlo, si ha l'apeggiore delle situazioni possibili, visto che il condensatore da 0 si carica poi a 3 V. Servirebbe quindi un tempo di sampling alto nella transizione e basso quando il segnale si mantiene costante. Per questo motivo si sceglie un tempo di sampling a metà delle due esigenze.
- While 1 ();

- Iniziare la conversione mettendo ADSTART a 1 e aspettare che EOC di ISR passi a 1.
- Il risultato della conversione sarà poi nel Data Register. Può essere scritto in una variabile "risultato", di cui si può vedere la variazione in live watch.

### OSSERVAZIONE SUL RISULTATO.

Caratteristica del convertitore analogico digitale (Gradinata):



①  $V_{FS}$  = tensione di fondovalo

tensione di riferimento fornita al convertitore: individua il massimo valore fornito in ingresso convertibile in binario

②  $K$ : codice in binario corrispondente al valore di tensione in ingresso.

Si possono avere  $2^m$  codici, nel massimo caso  $m=12$  perché il risultato occupa 12 bit del DR

$$③ Q = QUANTO = RISOLUZIONE = \frac{V_{FS}}{2^m}$$

$$④ V_{in} = \text{Tensione di ingresso} = KQ = K \frac{V_{FS}}{2^m}$$

In base a questi dati (spiegati nel dettaglio a "teoria"), il risultato della conversione andrà scritto in questo modo:

1) Dichiara una variabile float Risultato;

2) Alla fine della conversione: Risultato =  $\underbrace{(\text{ADC1} \rightarrow \text{DR})}_{K} \cdot Q$

$$\begin{aligned} &= (\text{ADC1} \rightarrow \text{DR}) \cdot \frac{V_{FS}}{2^m} \text{ con } m=12 \\ &= (\text{ADC1} \rightarrow \text{DR}) \cdot \frac{3.0}{4096.0} \end{aligned}$$

### OSSERVAZIONE

Mettendo ".0" alle costanti, queste vengono interpretate come float (Risultato è un float, esitiando il cast)

Per questo 3.0 è 4096.0

# LABORATORIO 9

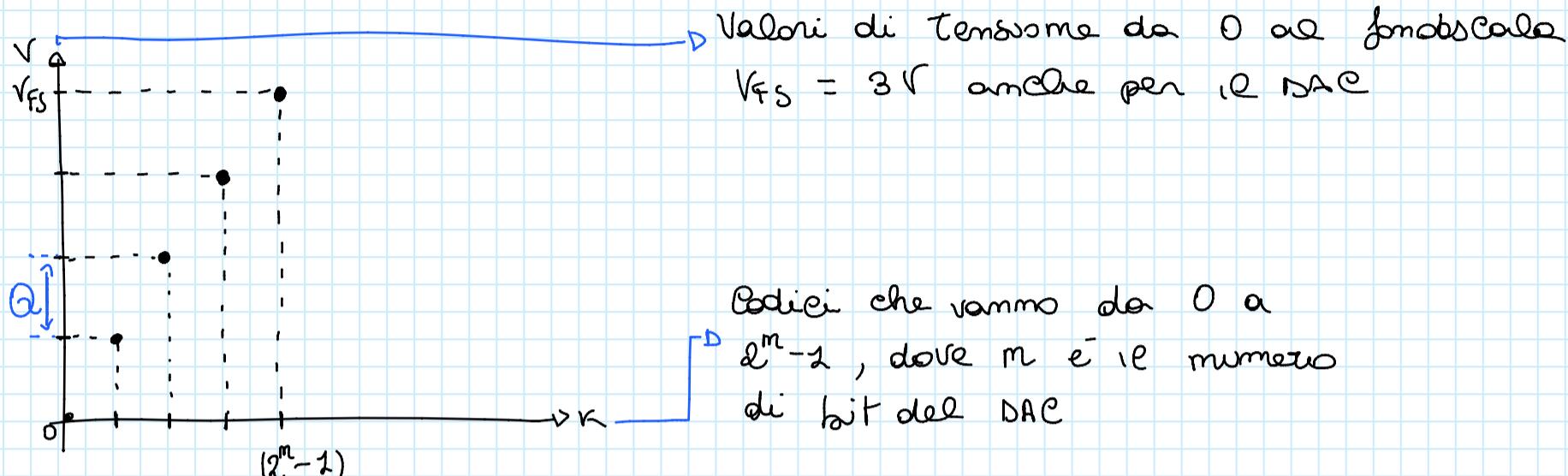
venerdì 25 novembre 2016 14:34

## CONVERTITORE DIGITALE/ANALOGICO (DAC)

Il convertitore digitale analogico è un dispositivo che riceve in ingresso un numero decimale codificato in binario, tramite  $N$  ingressi, e fornisce poi un'uscita analogica. Fa in pratica l'operazione inversa del convertitore analogico digitale: al DAC viene fornito un codice, che converte poi in una tensione analogica.

L'utilità nell'introdurre il DAC sta nel fatto che utilizzandolo, si avrà la possibilità di generare tutti i valori di tensione possibili, senza essere limitati a convertire continuamente e solo da 0 a 3V (e viceversa).

### CARATTERISTICA DEL DAC



A differenza dell'ADC, la cui caratteristica è una gradinata poiché ad un intervallo di tensione viene associato un codice, la caratteristica del DAC è una Punteggiata, perché ad ogni codice viene restituito un unico valore di tensione corrispondente a quel codice. In particolare la tensione generata è legata al codice fornito in input al DAC in questo modo: il fondoscala è suddiviso in  $2^m$  valori discreti di tensione e la distanza tra questi valori è detta Quanto (Q) tale che

$$Q = \frac{V_{FS}}{2^m}$$

Il valore di tensione generato dal DAC è dato dalla seguente equazione:

$$V_{DAC} = \underbrace{k_{DAC}}_{\text{codice fornito al DAC}} Q$$

Da questa equazione è possibile inoltre, dato un valore di tensione noto, ricavare il codice da dare al DAC.

$$k_{DAC} = \frac{V_{DAC}}{Q} \quad \text{con } V_{DAC} \in Q \text{ N.R.}$$

Il rapporto deve essere INTEGRATO, perché il codice che va inserito al DAC deve essere un intero

### INTRODUZIONE (Reference Manual pag 303)

Il modulo DAC è a 12 bit e può essere configurato in modalità 8 bit, nel caso in cui si voglia usare un solo byte, perdendo però risoluzione, oppure in modalità 12 bit, nel caso in cui si vogliano usare tutti i 12 bit, sapendo però che si dovranno gestire 2 byte e quindi 16 bit. Proprio per questo nella modalità a 12 bit, il dato può essere allineato a destra o a sinistra (stesso discorso per l'ADC, si hanno 16 bit a disposizione, se ne occupano 12 e quindi restano 4 bit vuoti, per cui va scelto un allineamento).

Il DAC può essere inoltre usato come DMA.

È infine disponibile un riferimento di tensione e l'output può essere bufferizzato.

Del DAC ci sono due canali di uscita, quindi è possibile generare due tensioni diverse sul canale 1 e sul canale 2, indicati con  $DAC1\_OUT1$  e  $DAC1\_OUT2$ . I due canali di uscita possono essere utilizzati separatamente (in modo che siano indipendenti) o simultaneamente.

### Schema a Blocchi (Reference Manual pag 304).

Il codice che si invia al DAC va inserito nel registro DHR (Data Holding Register).

La vera conversione digitale/analogico avviene nel blocco Digital to Analog Converter, dalla cui uscita si ha il pin OUT1/2 da cui uscirà il segnale analogico.

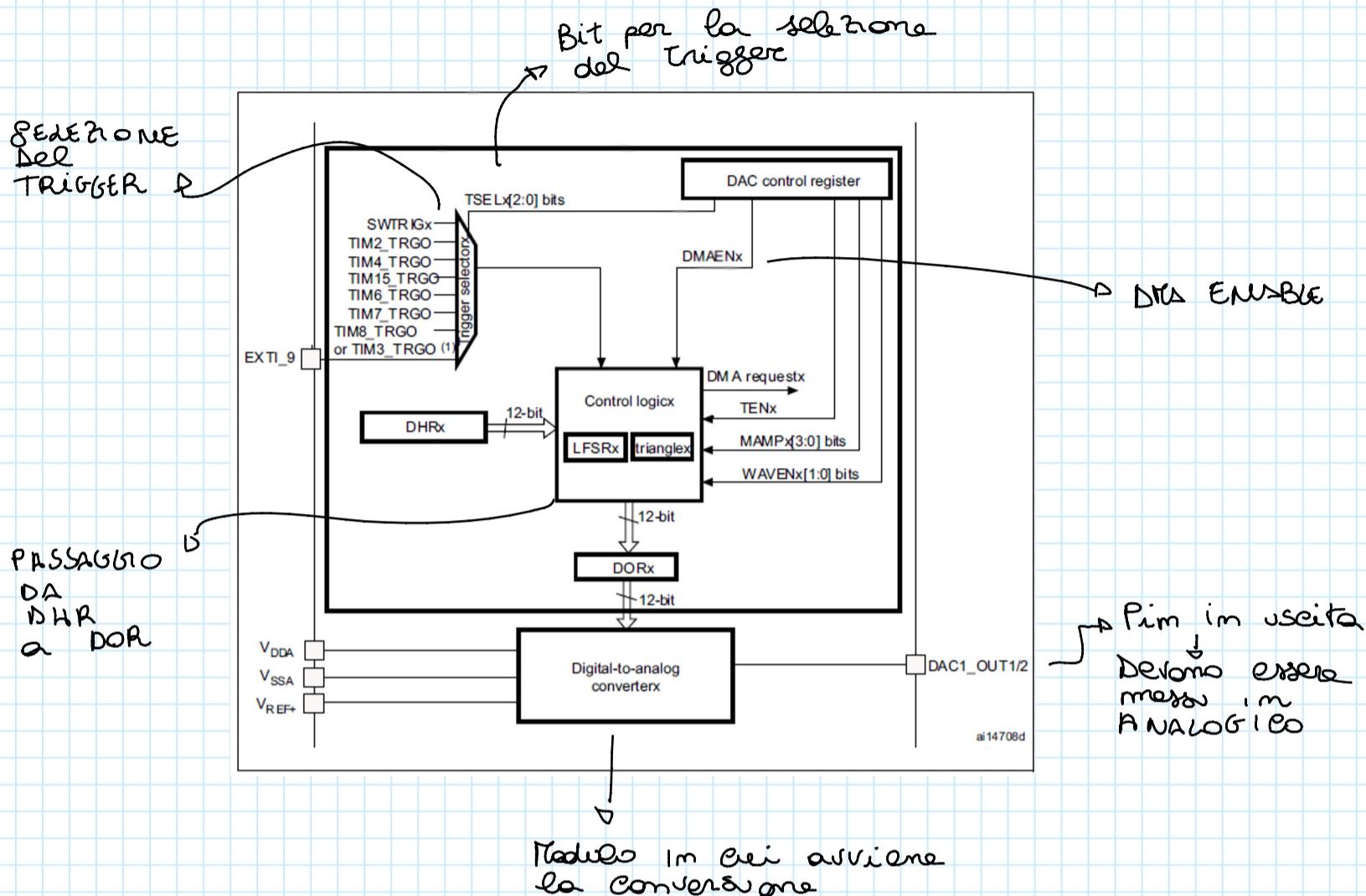
Dal registro DOR (Data Output Register) entra un codice a 12 bit. Visto che il DOR è in sola lettura, quindi non gli si può accedere, si scrive il codice in DHR (accessibile) e a seguito di un evento, che si chiama evento di trigger, il codice viene

poi trasferito dal DHR al DOR e infine viene generata la tensione analogica corrispondente.

La possibile selezione degli eventi di trigger avviene nel modulo "Trigger Selector". Dallo schema si può notare che ci sono, come possibili eventi di trigger: diversi Timer; un ingresso esterno (quindi il cambio di livello di una linea); un software trigger. Il trigger scelto si seleziona con 3 bit che si chiamano TSEL (da 2 a 0).

Il trasferimento tra DHR e DOR avviene nella Logica di Controllo, che ha già di suo immagazzinate due forme d'onda "Triangle e LFSR". Se viene scelta una di queste due forme d'onda, la generazione della tensione avviene da queste, altrimenti si dovrà dare al DAC il codice da generare scrivendo nel DHR.

È possibile infine inviare i dati al DAC attraverso il DMA (infatti c'è anche il DMA Enable).

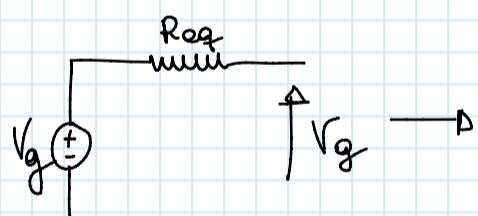


#### DESCRIZIONE MODALITA' SINGLE MODE (Reference Manual pag. 305)

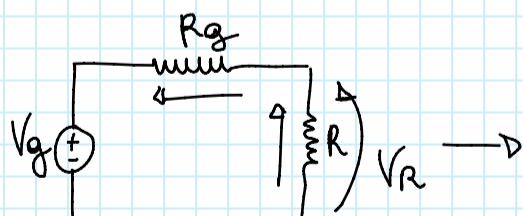
Il canale del DAC è alimentato settando il bit EN1 del registro di controllo (CR) e viene abilitato dopo un certo tempo  $t_{WAKEUP}$ .

È possibile inserire un buffer di uscita (un amplificatore operazionale) che serve a ridurre l'impedenza di uscita (per avere un sistema di generazione ad impedenza di uscita basso).

Significato impedenza di uscita



Modello generatore (sorgente di segnale), sfruttando Thevenin: si ha un generatore ideale di tensione, con resistenza in serie. A questo generatore deve essere caricato un carico, perché il segnale verrà generato applicato ad un circuito. Quindi a circuito aperto, si misura la  $V_g$  generata.



Chiudendo il generatore sul carico, su  $R$  non si vede più  $V_g$ , perché ora che circola corrente, di questa  $V_g$  generata, una parte cade sulla resistenza interna del generatore e una parte cade sul carico.  
Si vede quindi una  $V_R < V_g$  ed è tanto minore quanto più è alta la resistenza interna  $R_g$  e quanto più è alta la corrente che circola.

Un generatore è un buon generatore, se questa resistenza interna è molto bassa.

Infatti gli amplificatori operazionali, almeno quando li si considera come componenti ideali, hanno un'impedenza di uscita nulla. L'importanza di dire "impedenza di uscita nulla" sta proprio nel fatto che questi generatori possono generare corrente senza che la tensione scenda.

Si ha quindi la possibilità di collegare l'uscita del DAC ad un amplificatore operazionale.

Il motivo per cui l'inserire questo amplificatore è una possibilità e quello per cui non è pensato di default, è dovuto al all'ingresso dell'amplificatore, che ha un comportamento capacitivo, che rallenta la salita del segnale.

Perciò si ha la possibilità di inserirlo o meno questo buffer con un bit che si chiama BOFF1.

#### FORMATO DEI DATI (Reference Manual pag. 305)

Al DAC si può inviare il codice in 3 possibili formati:

- 8 bit -> si scrive il codice in un registro che si chiama DHR8R (Data Holding Register 8 bit allineati a destra).
- 12 bit -> si scrive il codice in un registro che si chiama DHR12L (12 bit allineati a sinistra).
- 12 bit -> si scrive il codice in un registro che si chiama DHR12R (12 bit allineati a destra).

Il DHR è dopo caricato nel DOR:

- Automaticamente: caso in cui si sta lavorando senza trigger, quindi appena viene scritto il codice nel DHR, questo viene spostato nel DOR e viene poi generata la tensione.
- Tramite Software Trigger: il trasferimento avviene con un trigger software da programma.
- Tramite un evento di trigger esterno.

### DAC CONVERSION (pag. 305 e pag 306)

A seconda dell'evento di trigger che si adopera, cambia il tempo che intercorre tra scrittura in DHR e la generazione della tensione analogica. In particolare il dato memorizzato in DHR è automaticamente trasferito nel DOR dopo un colpo di clock del bus se non è stato selezionato nessun trigger hardware (quindi se si lavora senza trigger, si scrive il codice in DHR, che dopo un colpo di clock viene spostato in DOR ed infine viene generata la tensione), altrimenti se è selezionato un hardware trigger il trasferimento avviene dopo 3 colpi di clock del bus.

### SELEZIONE DEL TRIGGER (pag. 306)

Il trigger, sia interno che esterno, si abilita con un bit che si chiama TEN e si sceglie il trigger con I 3 bit che si chiamano TSEL.

Tabella con I possibili trigger da selezionare:

Source	Type	TSEL[2:0]
Timer 6 TRGO event	Internal signal from on-chip timers	000
Timer 3 TRGO event or Timer 8 TRGO event <sup>(1)</sup>		001 <sup>(1)</sup>
Timer 7 TRGO event		010
Timer 15 TRGO event		011
Timer 2 TRGO event		100
Timer 4 TRGO event		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

UPDATE DEI TIMER

→ TRIGGER SOFTWARE

LINEA ESTERNA ↓

NOTA : Linea Esterna a EXTI9, significa che può essere una delle linee PA9, PB9, PC9, PD9, PE9, PF9

### DESCRIZIONE MODALITA' DUAL MODE (Reference Manual pag. 307)

Si adoperano I due canali di uscita insieme (stesso discorso che è stato fatto per gli ADC) in modalità sincrona o alternata.

### GENERAZIONE FORMA D'ONDA TRIANGOLARE E RUMORE BIANCO (pag. 309 e pag 310).

Questa parte la tralasciamo perchè noi genereremo forme d'onda, ma le generiamo noi, non adopereremo questa funzionalità del DAC, "dobbiamo soffrire" (cit.).

### REGISTRI (Reference Manual pag. 316)

#### • REGISTRO DI CONTROLLO (CR-> pag. 316)

Poichè ci sono 2 canali del DAC, 16 bit sono riservati al canale2 e 16 bit sono riservati al canale1, e sono gli stessi bit per entrambi I canali. In particolare:

- bit 29-28 per canale2 e bit 13-12 per canale1: servono per l'impiego del DMA.
- bit 27-24 per canale2 e bit 11-8 per canale1: consentono di selezionare l'ampiezza della forma d'onda triangolare (qualora la si voglia adoperare).
- bit WAVE (23-22 per canale2 e 7-6 per canale1): consentono di selezionare la forma d'onda.
- bit TSEL (21-19 per canale2 e 5-3 per canale1): per selezionare il trigger, vedere tabella sopra per I valori da inserire.
- bit TEN2 (18) e TEN1 (2): valore di default 0, messi a 1 abilitano il trigger su quel canale.
- bit BOFF2 (17) e BOFF1 (1): valore di default 0, messo a 1 disabilita il buffer (quindi per default il buffer è abilitato).
- bit EN2 (16) e EN1 (0): messi a 1, dal valore di default 0, abilitano il canale.

#### • SOFTWARE TRIGGER (SWTRIGR -> pag. 320)

Di questo registro si usano due soli bit: uno per il canale1 e l'altro per il canale2, e sono I due bit da alzare nel caso in cui si adopera il software trigger.

Scrivere 1 nel bit 0 (SWTRG1) o nel bit 1 (SWTRIG2) significa dare l'evento di trigger software al DAC, quindi dopo 3 colpi di clock il codice in DHR viene trascritto in DOR.  
Questo bit è poi cancellato dall'hardware.

- DHR12R1 e DHR12R2 (pag.320 e pag. 321)  
Registro in cui si va a scrivere il codice, a 12 bit allineati a destra, per il canale1 o per il canale 2 (rispettivamente).
- DHR12L1 e DHR12L2 (pag.320 e pag. 322)  
Registro in cui si va a scrivere il codice, a 12 bit allineati a sinistra, per il canale1 o per il canale2 (rispettivamente).
- DHR8R1 e DHR8R2 (pag.321 e pag. 322)  
Registro in cui si va a scrivere il codice, a 8 bit allineati a destra, per il canale1 o per il canale2 (rispettivamente).
- Nel caso in cui si adoperino I due canali in modalità duale, c'è un unico registro che consente di scrivere I codici dei due canali, nei 16 bit più significativi e nei meno significativi. A seconda dell'allineamento si hanno quindi I seguenti registri:  
 - DUAL DAC 12 BIT RIGHT ALIGNED DATA HOLDING REGISTER (DHR12RD → Pag. 322).  
 - DUAL DAC 12 BIT LEFT ALIGNED DATA HOLDING REGISTER (DHR12LD → Pag. 323).  
 - DUAL DAC 8 BIT RIGHT ALIGNED DATA HOLDING REGISTER (DHR8RD → Pag. 323).
- DATA OUTPUT REGISTER (DOR1 E DOR2 → Pag. 324)  
Registro in sola lettura, da leggere nel caso in cui si voglia la certezza che il codice è stato trasferito al DHR al DOR.
- STATUS REGISTER (SR→ Pag. 324 e 325)  
L'unico flag che viene riportato è un errore di Underrun del DMA (canale 1 e canale2).

## ESERCIZIO

Generare una tensione con il DAC e acquisirla con l'ADC.

Con l'ADC non si avrà lo stesso identico codice dato al DAC perchè ci sono in mezzo tutte le non linearità di questi due componenti.

Quindi provare a generare qualcosa con il DAC e collegare esternamente l'uscita del DAC con l'ingresso dell'ADC e vedere se si riesce ad acquisire con l'ADC.

Collegare esternamente due pin, significa fare l'aggancio tra I canali del DAC e I PIN della scheda.

Nella tabella del Data Sheet a pag. 36, si legge, scorrendo la colonna "Additional Functions ", che PA4 è il DAC1\_OUT1 e che PA5 è il DAC1\_OUT2.

Quindi i due canali, quando si abilita il DAC che genera la tensione, sono PA4 e PA5.

Per l'esercizio , adoperare solo il canale1 e quindi PA4, dal quale uscirà una tensione da convertire con l'ADC.

Per effettuare i collegamenti sulla scheda si usa uno dei jumper che stanno dietro lo schedino (ponticello metallico utilizzato per cortocircuitare dei contatti (pin) situati in punti prestabiliti di un circuito digitale o per attivare o disattivare funzioni diverse del circuito).

Si parte da PA4 e si cerca un PIN adiacente a PA4 che si può collegare a PA4 con il jumper con il quale effettuare la conversione analogico digitale.

Affiancati a PA4 ci sono PA2 canale3 di ADC1, PA6 canale3 di ADC2 e PA5 canale2 di ADC2 (informazioni ricavate dal Data Sheet a pag. 36).

Scegliamo di collegare PA4 e PA2, mettendo il jumper tra questi due PIN, ciò che genera il DAC è collegato fisicamente al canale 3 di ADC1.



## NOTE:

- Mettere PA2 e PA4 in analogico.
- Si può lavorare con il Software Trigger
- Il DAC genera una tensione analogica che acquisisce l'ADC, se passa poco tempo tra queste due istruzioni, l'ADC acquisisce la tensione durante il transitorio. Invece si dovrebbe dire al DAC di generare, aspettare con un for che il segnale sia stabile (tempo di SETTLING-> tempo di assestamento: 4 microsecondi) e poi acquisire con l'ADC.

# LABORATORIO 10

lunedì 28 novembre 2016 16:33

## EXTENDED INTERRUPTS AND EVENTS CONTROLLER (Reference Manual pag. 187)

Gli interrupt EXTI sono degli interrupt legati al cambio di stato di una linea, quindi ad una transizione alto-basso o basso-alto di una linea. L'EXTI permette di gestire 36 linee di evento di cui 8 sono interne e le restanti 28 sono esterne.

Come viene gestito l'evento che scatena l'interrupt EXTI?

Riferimento Schema-> Reference Manual pag. 188

In ingresso si ha una delle possibili 36 linee

Il circuito a cui è collegata la linea è l'EDGE DETECT CIRCUIT, che serve appunto per la rilevazione del fronte, di salita o di discesa a seconda di cosa è stato selezionato.

Successivamente viene generato un evento di interrupt, in particolare l'interrupt può essere generato via hardware, perché l'EDGE DETECT CIRCUIT si è accorto che c'è un fronte di salita o di discesa, ma può anche essere generato via software, scrivendo infatti su un opportuno registro (in pratica siamo noi in questo modo a generare l'interrupt software su una delle linee EXTI).

Quando si verifica un evento, se è stato smascherato quell'evento di interrupt, allora viene scatenato l'interrupt, in particolare viene settato un flag che si chiama PENDING REQUEST (to NVIC).

NVIC si accorge di questo flag settato a uno e gestisce l'interrupt.

Nota: Come viene gestito l'interrupt relativo al timer.

Anche nel caso del timer succede una cosa simile: c'è un flag che si alza in corrispondenza di un determinato evento, ovvero l'UPDATE INTERRUPT FLAG (UIF), ed NVIC appena questo flag si alza gestisce l'interrupt.

Per fare in modo che la periferica timer dica ad NVIC che si è generato un evento, bisogna andare a smascherare l'interrupt, cioè bisogna alzare il bit di abilitazione dell'interrupt che si chiama UIE.

Anche con EXTI avviene la stessa cosa: se è stato abilitato un interrupt su una particolare linea, allora l'interrupt è stato smascherato, si alza il relativo flag e NVIC gestisce l'interrupt.

La differenza con i Timer sta nella parte a monte del circuito che deve accorgersi che c'è stata una transizione della linea (o fronte di salita o fronte di discesa).

## Hardware Interrupt selection (Reference Manual pag 189).

Per configuire una linea come sorgente di interrupt bisogna seguire questa procedura:

- Configurare il corrispondente bit della maschera nel registro EXTI\_IMR
- Configurare la selezione del trigger (fronte di salita o di discesa) della linea di interrupt nei registri EXTI\_RTSR (per il fronte di salita) e EXTI\_FTSR (per il fronte di discesa).
- Configurare il bit che controlla il canale di NVIC IRQ, ovvero alzare l'opportuno bit ISER dei registri di ISER per fare in modo che NVIC gestisca l'interrupt.

## Collegamento delle linee tramite il multiplexer (Schema Reference Manual pag. 190)

Delle 28 linee esterne, le prime 16 si chiamano EXTI0, EXTI1, ..., EXTI15.

Tramite un multiplexer ogni linea EXTI può essere collegata fisicamente a tutte le linee corrispondenti di tutte le porte.

Ad esempio: - La linea EXTI0 può essere collegata fisicamente a tutte le linee 0 di tutte le porte PA0, PB0, PC0,

PDO, PEO.

- La linea EXTI1 può essere collegata fisicamente a tutte le linee 1 di tutte le porte PA1, PB1, PC1, PD1, PE1.

E così via fino a EXTI15, che può essere collegata fisicamente a tutte le linee 15 di tutte le porte PA15, PB15, PC15, PD15, PE15.

Per poter effettuare questo collegamento, il multiplexer va configurato, settando 16 bit EXTI (da 15 a 0) dei registri SYSCFG\_EXTICR<sub>i</sub>, con i=1,..,4 (Reference Manual pag. 174).

Ogni registro è organizzato in modo da poterconfigurare quattro linee EXTI, settando 4 bit con la seguente configurazione:

- x000 per la porta A;
- x001 per la porta B;
- x010 per la porta C;
- x011 per la porta D;
- x100 per la porta E.

Ad esempio settando i primi quattro bit da 3 a 0 di SYSCFG\_EXTICR1, si collega la linea EXTI0 alla linea 0 di una delle porte (in base alla configurazione scelta).

Altro esempio, settando I bit da 11 a 8 di SYSCFG\_EXTICR3, si collega la linea EXTI10 alla line 10 di una delle porte (sempre in base alla configurazione scelta).

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
EXTI11[3:0]	EXTI10[3:0]	EXTI9[3:0]	EXTI8[3:0]												
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Osservazione: SYSCFG è una periferica, quindi per poterla operare bisogna abilitare il clock, come per tutte le altre periferiche.

Osservazione 2: A pag. 191, c'è un elenco in cui viene mostrato tutte le altre linee, da EXTI16 in poi a cosa sono collegate (protocolli di comunicazione, gli USART, uscite dei comparatori...)

### REGISTRI DI INTERESSE (Reference Manual pag. 192)

Tutti i registri degli EXTI sono suddivisi in 32 bit e ogni bit corrisponde alla linea EXTI relativa (es. Bit 0-> EXTI0 oppure bit 12-> EXTI 12).

Poiché le linee EXTI sono 36, in 32 bit non è possibile gestirle tutte, per questo motivo per ogni tipologia di registro ci sono sempre due registri, di cui il secondo serve per gestire le linee: EXTI32, EXTI33, EXTI34, EXTI35.

Esempio: Si avrà IMR1 e IMR2, RTSR1 e RTSR2, ... e così via per tutti I registri.

- Registro IMR (Interrupt Mask Register -> Reference Manual pag. 192) :

Registro con I bit delle maschere.

Se un generico bit viene messo a 0 l'interrupt request della linea x relativa a quel bit è mascherato (disabilitato), se invece viene messo a 1 l'interrupt request di quella linea è smascherato e quindi abilitato.

Quindi se ad esempio si vuole utilizzare EXTI0, bisogna mettere 1 in MRO di IMR1.

- Registro RTSR (Rising Trigger Selection Register -> Reference Manual pag. 193)

Se si alza uno di questi bit si abilita il rising trigger e quindi l'interrupt sul fronte di salita (0 disabilitato, 1 abilitato). Il bit che si alza corrisponde alla linea che si sta considerando.

- Registro FTSR (Falling Trigger Selection Register -> Reference Manual pag 193)

Serve per abilitare l'interrupt sul fronte di discesa, mettendo ad 1 il bit relativo alla linea EXTI corrispondente.

- Registro SWIER (Software Interrupt Event Register -> Reference Manual pag. 194)

Si scatena l'interrupt software, agendo su questo registro e in particolare scrivendo 1 sul bit relativo alla linea EXTI che si sta considerando.

- Registro PR1 ( Pending Register -> Reference Manual pag. 194)

Registro dei flag che si alzano quando si verifica la causa di interrupt (ovviamente si alza il bit relativo alla linea EXTI utilizzata).

Osservazione: Ricordare di abbassare il flag (come si fa anche per I Timer).

Sotto ogni bit c'è una scritta che indica appunto come si manipolano I bit. In questo caso si legge rc\_w1, dove la r indica che questi bit sono in sola lettura, ma che è possibile cancellarli (c) scrivendo 1 (w1). Quindi per cancellare/azzerare un dato flag bisogna scrivere 1 sul bit di interesse.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR31	PR30	PR29	Res.	Res.	Res.	Res.	Res.	Res.	PR22	PR21	PR20	PR19	PR18	PR17	PR16
rc_w1	rc_w1	rc_w1							rc_w1						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1															

### ESERCIZIO

Scatenare un evento di interrupt dovuto alla pressione o al rilascio del pulsante USER sulla linea PA0.

Si ricorda che sulla pressione si ha una transizione da 0 a 3V (fronte di salita), mentre sul rilascio da 3V a 0 (fronte di discesa).

In particolare si vuole realizzare un programma in cui il DAC genera un valore analogico che l'ADC dovrà acquisire in interrupt, alla pressione o al rilascio del tasto.

Bisogna quindi configurare la parte di registri EXTI per abilitare l'interrupt su PA0 (quindi si sa già che si lavorerà con EXTI0).

Bisogna poi dire a NVIC di servire l'interrupt EXTI0: guardando la tabella di tutte le possibili cause di interrupt gestite da NVIC a pag 184 del Reference Manual, si legge che EXTI0 è in posizione 6. Allora bisogna alzare il bit 6 di ISER[0].

6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2 and TSC	EXTI Line2 and Touch sensing interrupts	0x0000 0060
9	16	settable	EXTI3	EXTI Line3	0x0000 0064
10	17	settable	EXTI4	EXTI Line4	0x0000 0068

Se tutto è andato a buon fine, quando si preme o si rilascia il pulsante, il microcontrollore salterà alla posizione 6 degli EXTERNAL INTERRUPTS, per cui se esiste nel progetto una funzione che si chiama EXTIO\_IRQ\_HANDLER il microcontrollore salterà a quella funzione.

Osservazione: Quando si definisce una funzione con quella particolare nomenclatura tipica di un interrupt, il compilatore già sa che deve prendere questa funzione e metterla in una particolare locazione di memoria dove salterà. Quindi quando arriva al primo interrupt salta a quella locazione di memoria, da lì fa un altro salto alla locazione di memoria dove è andato ad inserire tutte le istruzioni che si trovano nella routine di interrupt (Non dobbiamo chiamare noi questa funzione nel main, dovrebbe essere tutto automatico).

Nella funzione bisogna inserire le istruzioni relative a : cancellazione del flag, avvio conversione ADC (dare lo Start of Conversion), attesa della fine della conversazione, risultato della conversazione.

Nel main mettere un while 1, o qualunque cosa che servi a bloccare il microcontrollore.

NOTA 1: ricordare di mettere la linea PA0 come ingresso analogico.

NOTA 2: Migliorare la leggibilità dei programmi mettendo qualche funzione (es. Inizializza ADC, inizializza DAC, Conversione che dal SOC adpetta l'EOC, ...)

# LABORATORIO 11

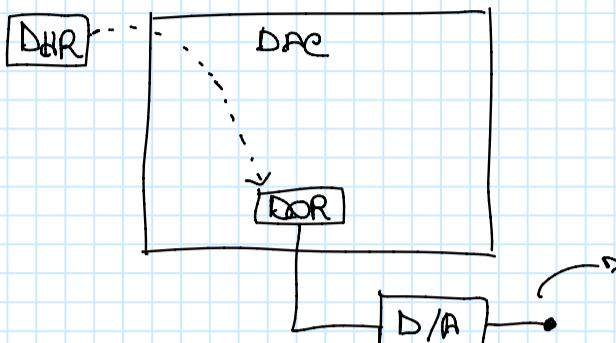
lunedì 5 dicembre 2016 16:39

## GENERAZIONE D'ONDA E DMA

### RIEPILOGO SUL DAC.

I codici vanno scritti nel registro DHR e all'occorrenza dell'evento di trigger il DAC trasferisce il dato contenuto del DHR nel registro DOR, il registro di output. Questo codice viene poi convertito in analogico, ottenendo una tensione che è pari al codice per il quanto, dove il quanto.

Finora si è lavorato con il trigger software perché generando una tensione continua non era di interesse l'istante di tempo in cui veniva generata questa tensione.



Un uscita fu ha una tensione analogica pari a:

$$V_{DAC} = KQ \quad \text{dove } Q = \text{quanto} = \frac{V_{FS}}{2^m}$$

## GENERAZIONE DI UNA FORMA D'ONDA COL DAC MONITORATO DAL TIMER

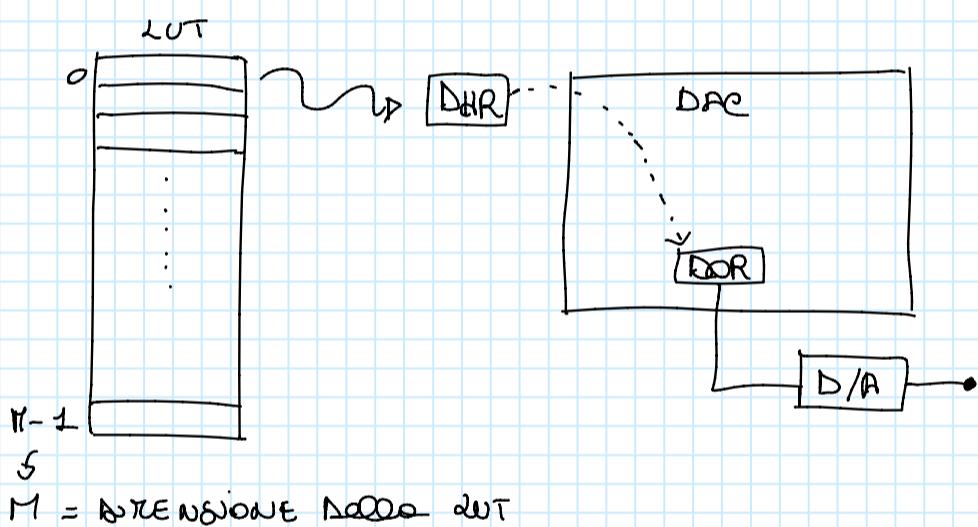
L'obiettivo del giorno è quello di generare una forma d'onda, in particolare un segnale periodico sinusoidale.

Si dovrà realizzare quindi una LUT (Look Up Table → un array/matrice di dati), ovvero una tabella contenente I valori di ampiezza che man mano devono essere passati al DAC, tali valori saranno I campioni della sinusoide.

La dimensione dell'array LUT (arbitrario), rappresenta il numero di punti del periodo della sinusoide che si vuole generare.

Nell'esercizio che faremo, si sceglierà come dimensione della LUT 100, quindi si rappresenterà con 100 campioni il periodo di una sinusoide.

Gli elementi della LUT Andranno scritti di volta in volta nel DHR



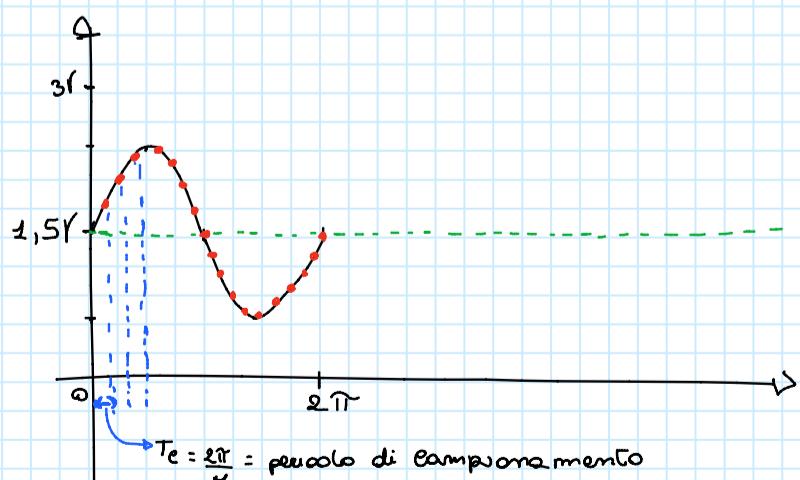
→ Alla prima occorrenza di trigger viene scritto  $LUT[0]$  nel DHR, alla seconda  $LUT[1]$ , poi  $LUT[2], \dots$

Il DHR riceve di volta in volta i campioni i-esimo (con  $i = 0, \dots, M-1$ ) dalla LUT, per trasferirlo poi al DOR.

## COME RIEMPIRE LA LUT.

Lo scopo è quello di riempire una tabella che contiene un periodo di una sinusoide, senza specificare la frequenza di questa sinusoide.

Avendo il DAC che può generare solo tensioni da 0 a 3V, bisogna realizzare una sinusoide che ha un offset di  $(3/2)V$  e la cui ampiezza è arbitraria, purché non superi  $(3/2)V$ . Questo periodo sarà descritto con un numero  $M$  di campioni, che rappresenta quindi il numero di campioni scelti per rappresentare il periodo del segnale, per cui il periodo di campionamento in radianti sarà  $2\pi/M$  (essendo il periodo in radianti  $2\pi$ ).



### CAMPIONI SELEZIONATI

#### Perché questo offset 1,5V?

→ DAC PUÒ CADUTARE SOLO TENSIONI DA 0 A 3V (non ci sono valori negativi)  
 $\Rightarrow 1,5$  È PROPRIO IL RETRO DI MODO TUTTI I PUNTI DELLA SINUOSIDE SI TROVERANNO NEL I° QUADRANTE, IN PARTICOLARE I PUNTI CHE DOVREBBERO ESSERE NEGATIVI SARANNO SOTTO AL DI SOTTO DELLE CENSORE 1,5 E SAPRANNO ZERO.

Stesso discorso per l'ADC

La scelta dei campioni ovviamente non è arbitraria, ma questi derivano dalla formula:

$$LUT[i] = \underbrace{1.5V}_{\text{OFFSET}} + \underbrace{A \sin \left( \frac{2\pi}{M} \cdot i \right)}_{\substack{\text{Ampiezza} \\ \text{espressa in Volt} \\ \rightarrow \text{NON DEVE SUPERARE} \\ \text{L'OFFSET}}}$$

I < 0, ..., M-1  
 Ram memo che i si incrementa, nello LUT  
 Verremo scritti i campioni spaziali  
 di  $\frac{2\pi}{M}$   
 Azzerato al campione M-1,  
 è finito, e periodo di  
 sinusoida.

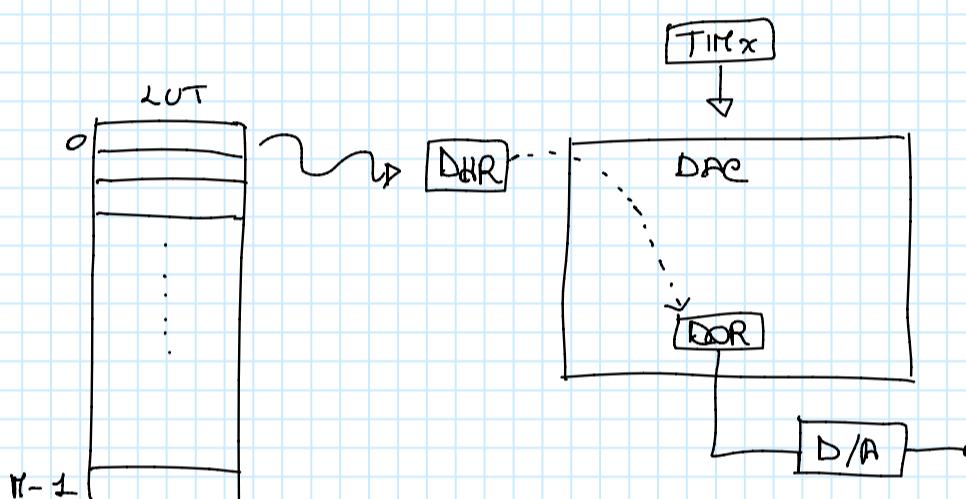
IN PONTE:  $1.5V = 2048$  e  $1 \leq A \leq 2047$

OSSERVAZIONE: Nella formula la sinusoide è stata espressa in Volt, ma nel DHR deve andare un codice da 0 a 4095, quindi conviene fare direttamente la formula di questa sinusoide in codice. A tal scopo si divide 1.5V per il Quanto che è pari a  $V_{FS}/2^m = 2048$ , e si esprime l'ampiezza direttamente in codice e sarà quindi un numero che varia da 1 a 2047 (quindi  $1 < A < 2047$ ).

Quindi in questo modo si costruisce la LUT che naturalmente deve essere un vettore di interi, perché i valori al suo interno corrispondono ai codici che si inviano in ingresso al DAC.

Il DAC genererà ad uno ad uno questi campioni, ma questa volta non si utilizzerà il software trigger (che diamo noi da codice), in quanto è aleatorio, nel senso che c'è un ritardo non prevedibile che dipende da ciò che è scritto nel codice. Per avere una generazione a tempo uniforme, si adopera come trigger un timer (uno qualunque).

Se si sceglie ad esempio Timer2, l'evento di trigger sarà l'update di Timer2 ed è questo l'aggancio con il tempo: si scrive un valore in ARR, il counter di Timer2 parte da 0 si incrementa e quando arriva ad ARR si verifica l'evento di update e riparte da 0. Se si fa in modo che l'evento di update sia il trigger, allora si ha la certezza che la generazione dei campioni avverrà in maniera uniforme nel tempo dato da  $ARR \cdot T_{CK}$  (dove  $T_{CK}$  è il periodo del clock -> 72MHz)



#### SELEZIONE DEL TRIGGER ESTERNO SULL'EVENTO DI UPDATE DEL TIMER

Questa informazione va data sia alla periferica (il DAC) che al Timer.

- LATO DAC.

Osservando lo schema a pag. 304 del Reference Manual, si può notare che attraverso i 3 bit TSEL, si seleziona la sorgente del trigger ed è possibile selezionare una linea "Trigger Out" di tutti i timer.

Per selezionare la sorgente del trigger, bisogna settare i bit TSEL (bit 5,4,3) del registro di controllo CR.

Scegliendo Timer2 bisogna scrivere 100 in questi bit (quindi mettere un uno sul cinque). In questo modo si sta dicendo al DAC che il trigger proverrà da una linea di Timer2.

- LATO TIMER

Bisogna dire al Timer che in occasione di un evento di update deve generare un segnale "Trigger out".

Reference Manual pag. 486, paragrafo 17.3.19

Ogni timer può essere in modalità Master Mode e comandare altre periferiche (o altri Timer) che si chiamano Slave, con un segnale che si chiama "Trigger Out" ( $TRO_{ia}$ ).

Il comportamento del Timer in modalità Master, dipende da un registro o meglio da un insieme di bit che si chiamano MMS.

Per capire il comportamento del Timer in modalità Master, si può vedere l'esempio fatto nel paragrafo: si adopera Timer3 come Prescaler di Timer2, cioè l'evento di update di Timer3 avvia l'incremento di Timer2. Timer3 viene quindi configurato in modalità master in modo tale che cacci un segnale di trigger periodico in corrispondenza di ogni evento di update. Quindi se si scrive 010 nei bit MMS del registro CR2 di Timer3, un fronte di salita è generato in output sul Trigger Out, ogni volta che viene generato un evento di update.

Allora risulta evidente che per fare in modo che in occasione di una update il timer generi un evento di trigger, bisogna agire sul registro Control Register 2 (CR2) -> Secondo registro di controllo del timer (Pag. 494 del Reference Manual)

In questo registro sono presenti i 3 bit MMS (Master Mode Selection), che consentono di selezionare l'informazione che deve essere inviata in Master Mode ad timer slave, per la sincronizzazione, le combinazioni sono le seguenti:

- 000: Il Trigger Out viene mandato in corrispondenza del reset del timer (si azzerà il conteggio e viene inviato un trigger out).
- 001: Il fronte di salita sul Trigger Out viene inviato in corrispondenza dell'Enable (cioè appena si alza il bit CEN)
- 010: Il Trigger Out viene mandato in corrispondenza di un evento di update.

Ci sono poi le configurazioni per gli "Output Compare" che non ci interessano.

Per l'esercizio di oggi useremo la combinazione 010: Il counter del timer conta fino al valore in ARR, arrivato al valore di ARR si azzerà, invia il trigger e poi comincia a contare e si ripete tutto. Questi segnali di trigger che sono cadenzati nel tempo, servono a sincronizzare il DAC.  
 La prima cosa da fare è settare questi bit MMS sul Timer e poi agire sul lato DAC.

## ESERCIZIO 1

Generare una forma d'onda, utilizzando per ogni periodo 100 campioni, scritti in una LUT e inviare tali campioni al DAC ad ogni evento di update del timer.

Step da seguire:

1. Popolamento della LUT
2. Configurare Timer2 e DAC, in modo che il DAC generi I campioni In corrispondenza dell'evento di trigger che viene dal Timer2
3. Il trasferimento del campione dalla LUT al DHR deve essere fatto da codice. Il valore LUT[i] va scritto in DHR, dopo che il DAC ha effettuato il trasferimento da DHR a DOR e quindi il DHR è libero ed è possibile scriverci dentro, in particolare il DHR sarà libero quando il timer va in update. Si hanno dunque due modi per sapere quando il DHR è libero:
  - Si monitora il bit UIF (while UIF==0): quando UIF passa a 1 si può scrivere nel DHR l'elemento successivo della LUT, visto che sicuramente il DAC ha effettuato il trasferimento, essendogli arrivato l'evento di trigger (UIF==1-> il contatore è arrivato ad ARR-> evento di update).

Osservazione: La variabile I che scorre la LUT deve avere modulo M, per cui una volta che arriva a M-1, deve ripartire da zero per poi generare un nuovo periodo

## DMA (Direct Memory Access Controller -> Reference Manual pag. 152)

È una periferica del microcontrollore che effettua trasferimenti di memoria e li fa in maniera autonoma dalla CPU: una volta configurato opportunamente e abilitato, il DMA, definita una sorgente da cui prelevare I dati e una destinazione, effettua questa sorgente di memoria dalla sorgente alla destinazione. Nel caso dell'esercizio 1 si può vedere benissimo che il DMA può essere adoperato per effettuare I trasferimenti dalla LUT al DHR.

Ci sono due controllori DMA (DMA1 e DMA2) che insieme possono gestire 12 canali, cioè 12 trasferimenti diversi di memoria, in particolare DMA1 gestisce 7 canali, DMA2 ne gestisce 5.

Il trasferimento del DMA consiste nelle seguenti operazioni (Reference Manual pag. 154):

- Si definisce un indirizzo di periferica in un registro che si chiama CPAR e un indirizzo di memoria in un registro che si chiama CMAR.
- Il DMA esegue trasferimenti di memoria in entrambe le direzioni dalla memoria alla periferica e dalla periferica alla memoria.
- Si fornisce inoltre un conteggio nel registro CNDTR, che indica il numero di trasferimenti che deve eseguire il DMA.
- Ogni canale del DMA gestisce trasferimenti tra un registro di periferica che si trova ad indirizzo fisso e un indirizzo di memoria (e viceversa).
- Si devono infine dimensionare I dati da trasferire, in particolare la dimensione di questi dati deve essere espressa in bit, nel senso che è possibile effettuare trasferimenti a 8 bit, 16 bit, 32 bit. Si deve impostare sia la lunghezza in bit dei dati da trasferire sia nella sorgente che nella destinazione e va impostata tramite dei bit che si chiamano PSIZE (size della periferica) ed MSIZE (size di memoria) nel registro DMACCR).

## PROCEDURA DI CONFIGURAZIONE DEL CANALE (Reference Manual pag 155)

- Settare l'indirizzo di periferica nel registro DMACPAR
- Settare l'indirizzo di memoria nel registro DMACMAR
- Configurare il numero di dati da trasferire in CNDTR
- Configurare la priorità del canale con I bit PL, poichè si possono gestire fino a 12 canali per il trasferimento di memoria, qualora si usassero più canali contemporaneamente è bene settare una priorità, perché accedono tutti al bus.
- Configurare la direzione del trasferimento : Memoria->Periferica o Periferica->Memoria
- Modalità circolare, se si vuole che una volta effettuati gli N trasferimenti, il DMA ricominci da capo e che quindi non si fermi).
- Incremento di periferica o memoria.
- Interrupt legati all'avvenuto trasferimento: si ha la possibilità di impostare un interrupt quando è stato trasferito metà del conteggio inserito in CNDTR o un interrupt sul trasferimento completo.
- Abilitazione del canale nel registro CCR.

## TABELLA REFERENCE MANUAL pag. 156

In questa tabella viene mostrato come viene gestito il trasferimento nel caso in cui la size della sorgente sia diversa dalla size della memoria.

## INTERRUPTS E RELATIVI FLAG (Reference Manual pag. 157)

Si può mettere un interrupt su :

- HALF TRANSFER : quando si arriva a metà del numero di trasferimento
- TRANSFER COMPLETE: quando il trasferimento è stato completato
- TRANSFER ERROR: possibilità di mettere un interrupt anche nel caso in cui ci sia stato un errore di trasferimento

## DMA1 REQUEST MAPPING (Reference Manual pag. 158)

Per adoperare il DMA per trasferire dati ad un'altra periferica, non si può usare uno qualunque dei 12 canali, ma ogni canale riceve richieste da particolari periferiche.

Ad esempio il Canale 1 del DMA1 riceve richieste dall'ADC1, da Timer1, Timer2, Timer4,..

## REGISTRI (Reference Manual pag 163)

### 1. STATUS REGISTER (RM pag. 163)-> Registro di flag.

Questo registro è suddiviso in gruppi di bit per ogni periferica (in particolare per il DMA1 si hanno 7 gruppi di flag perchè può gestire 7 canali, per il DMA2 ci sono 5 gruppi perchè ne può gestire 5) ed ogni gruppo è costituita da :

- GIF (Global Interrupt Flag): Se è a 1 si è verificata una qualunque delle cause per cui di alza il flag del DMA (Half transfer, Transfer Complete e Transfer error).
- TCIF (Transfer Complete Flag): è 1 se sono stati trasferiti tutti I campioni
- HTIF (Half Transfer Flag): è 1 se sono stati trasferiti metà dei campioni
- TEIF (Transfer Error Flag): è 1 se si è verificato un errore di trasferimento)

Tutti questi bit sono in sola lettura, sono settati dall'hardware, perciò se si vuole abbassare uno di questi flag da software, bisogna scrivere 1 nei corrispondenti bit del registro DMA\_IFCR.

### 2. IFCR (Interrupt Flag Clear Register ->Reference Manual pag. 164)

Mettendo uno di questi bit a 1, cancella il corrispondente flag dello Status Register.

Es. Si alza il TCIF del canale 3, per abbassarlo, bisogna scrivere 1 nel corrispondente bit del registro IFCR, ovvero nel CTCIF (bit 9)

CCR (Channel Configuration Register-> RM pag. 165)

- MEM2MEM (bit 14): va alzato nel caso in cui si voglia fare un trasferimento da una zona di memoria ad un'altra zona di memoria

- PL (bit 13, 12): sono due bit che servono a settare la priorità del canale

- MSIZE e PSIZE (bit 11 a bit 8): servono a impostare la dimensione in bit dei dati della periferica e della memoria in particolare :

00: 8 bit;

01: 16 bit;

10: 32 bit;

11: Riservato.

- MINC e PINC (bit 7 e 6)-> Memory increment e Peripheral increment: se uno di questi due bit viene messo a 1 (ovviamente possono essere messi a 1 anche entrambi), si incrementa ad ogni trasferimento l'indirizzo di memoria (MINC=1) o l'indirizzo di periferica (PINC=1).

- CIRC (bit 5) : Messo a 1 abilita la Circular Mode, quindi trasferisce continuamente gli N elementi.

- DIR (bit 4) : Direzione-> se lasciato a 0 trasferisce da periferica a memoria, messo a 1 trasferisce da memoria a periferica.

- bit da 3 a 1: sono gli interrupt enable su uno degli eventi Half transfer, transfer complete e transfer error (si abilitano mettendo a 1 uno di questi bit)

- EN (bit 0): Enable del canale

### 3. CNDTR (Channel Number of Data Register -> RM pag. 166)

In questo registro si scrive direttamente (in intero ) il numero di trasferimenti che si vuole effettuare

### 4. CPAR (Channel Peripheral Address Register-> RM pag. 167)

Si scrive l'indirizzo della periferica .

### 5. CMAR (Channel Memory Address->RM pag.167)

In questo registro va l'indirizzo della memoria

OSSERVAZIONE: Quando si utilizzano I DMA, bisogna settare anche nella periferica che partecipa al trasferimento, il legame con il DMA. Ad esempio nel caso del DAC bisogna alzare il bit DMA ENABLE.

## ESERCIZIO 2

Trasferire dati dalla LUT alla periferica , quindi l'indirizzo della Lut è l'indirizzo di memoria, l'indirizzo di DHR è l'indirizzo di periferica ed effettuare 100 trasferimenti (Lunghezza della LUT).

ATTENZIONE: Il DHR è a 16 bit, perciò si sa che sicuramente la destinazione è a 16 bit. Per non avere problemi conviene che il DMA effettui trasferimenti da 16 bit a 16 bit, perchè se la sorgente fosse a 32 bit, il DMA dovendo scrivere in 16 bit, prende prima I 16 bit più significativi, poi I 16 meno significativi e si fa casino. Meglio un trasferimento da 16 bit a 16 bit. Questa informazione va scritta in PSIZE ed MSIZE ma anche deve essere data anche alla sorgente : in pratica la LUT deve essere un array di 16 bit, perciò non può essere un array di interi perchè l'intero è a 32bit, ma va dichiarato come short int.

Ad ogni trasferimento bisogna incrementare l'indirizzo della memoria (non quello della periferica)

### POINTER INCREMENTATION (Reference Manual pag. 155)

Quindi Il primo elemento della LUT deve essere l'indirizzo di memoria e l'indirizzo di DHR è l'indirizzo della periferica. Tuttavia poi la si deve scorrere la LUT , per cui man mano bisogna dire al DMA che l'indirizzo di memoria ad ogni trasferimento va incrementato. L'indirizzo di periferica no, perchè la sorgente di periferica è sempre la stessa, il DHR.

Per quanto riguarda il collegamento tra il DAC e il DMA, bisogna cercare il canale del DMA che può ricevere richieste dal DAC.

Osservando la tabella a pag 158 si scopre che questo canale è il canale 3 di DMA1, DAC\_CH1 (comunicano canale1 del DAC e canale3 di DMA1).

Tuttavia le richieste del DMA sono mappate su questo canale, solo se il corrispondente bit di remapping è stato settato in CYSCFG\_CFG1. Se si vuole adoperare il canale 3 di DMA1 bisogna fare un'operazione di remapping in SYSCFG.

Scorrendo lo schema, si scopre però che anche DMA2 ha un canale che può ricevere richieste dal DAC ed è il canale 3 (che manda richieste al canale 4 del DAC).

Quindi o si adopera il DMA2 canale 3 oppure si può adoperare DMA1 canale 3 ma bisogna fare il remapping in SYSCFG.

NOTA: Nel settare l'indirizzo di periferica o si cerca l'indirizzo del DHR nel file.h oppure si mette un &DAC->DHR. Nel settare invece l'indirizzo di memoria, che corrisponde all'indirizzo da cui parte la LUT o si scrive direttamente "LUT" oppure "&LUT[0]".

### OSSERVAZIONE FINALE SULL'ESERCIZIO.

Stiamo facendo una catena che opera in maniera indipendente dalla CPU, perchè il trasferimento di memoria lo esegue il DMA e il trigger lo fornisce timer 2 noi non dobbiamo fare niente.

Quando il DMA prende il campione successivo e lo scrive in DHR?

È il DAC che appena effettua il trasferimento da DHR a DOR manda una richiesta al DMA, per dirgli che il DHR è vuoto e che può effettuare un nuovo trasferimento.

Per fare in modo che il DAC invii una richiesta al DMA, bisogna settare il bit 12 (DMA ENABLE) del Registro di Controllo del DAC: alzando questo bit infatti si dice al DAC che il trasferimento lo fa il DMA e che quindi quando si svuota deve inviare richiesta al DMA.

Alla fine il programma farà questo: il timer invia il trigger out al DAC, il DAC prende quello che c'è in DHR e lo trasferisce in DOR, genera la tensione e manda una richiesta al DMA, che incrementa l'indirizzo di memoria e prende il secondo elemento della LUT per poi scriverlo in DHR e il ciclo si ripete. Noi da programma non facciamo niente, è tutto in Hardware.

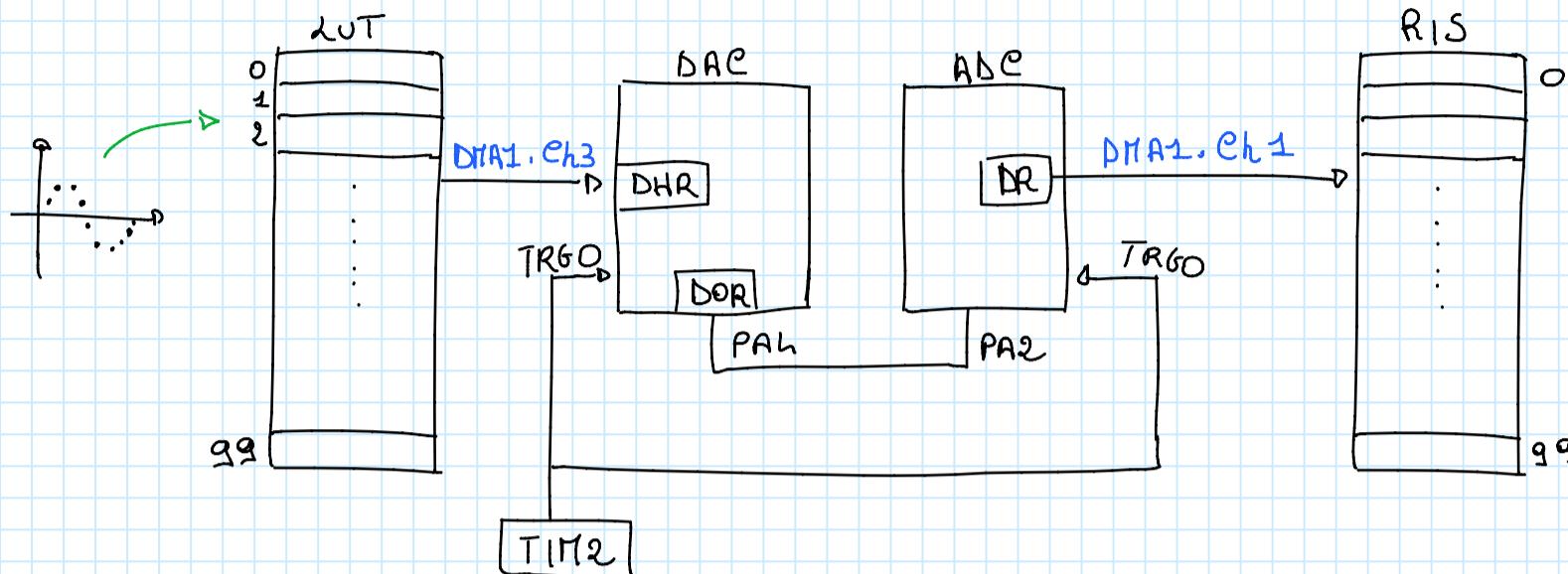
# LABORATORIO 12

mercoledì 7 dicembre 2016

## ESERCIZIO SU DMA CON DAC E ADC

Sotto controllo del TIM2, il DMA deve trasferire, da un vettore LUT, 100 campioni al DAC. La tensione in uscita dal DAC deve essere poi trasferita all'ADC, che convertirà gli elementi uno ad uno e li trasferirà tramite il DMA in un altro vettore in memoria.

Note: la generazione dei campioni (DMA->DAC) deve avvenire in modalità circolare, l'acquisizione deve essere singola.



Per questo esercizio è richiesto che il DAC e l'ADC lavorino in modo sincrono e quindi entrambi innescati dallo stesso Timer.

CONFIGURAZIONI PER LA PARTE RELATIVA A DMA E ADC, LA PRIMA PARTE E' GIA' STATA FATTA NEL LABORATORIO 11.

### Lato DMA

Il canale del DMA che può ricevere richieste da ADC1 è il canale 1 del DMA1. Allora bisogna configurare questo canale per il trasferimento.

CPAR: indirizzo della periferica->indirizzo di ADC1.

CMAR : indirizzo del vettore RIS (che ha dimensione 100), in cui voglio che I campioni convertiti dall'ADC vengano allocati .

PINC: l'indirizzo di periferica non va incrementato

MINC: l'indirizzo di memoria va incrementato

Non va messa la modalità circolare, visto che si vuole che l'ADC converta una sola finestra di questo segnale.

COUNT: 100, il numero di campioni che si vogliono acquisire.

PSIZE = MSIZE = 16bit

### Lato ADC

Configurazione standard.

Anche in questo esercizio da programma non dobbiamo fare niente, a parte aspettare che sia finita l'acquisizione.

Per capire se l'acquisizione è finita, si monitora il flag del Transfer Complete.

Allora nel main, dopo aver fatto tutte le configurazioni e aver abilitato tutto, deve esserci un while che aspetta che il transfer complete si alzi.

### Programma corretto dall'ingegner Tocchi

### FUNZIONI

```
#include "libreria.h"
```

```
void RCC_EN (){
    RCC->APB1ENR |= 1; //Abilito il clock per la periferica Timer2
    RCC->APB1ENR |= 1<<29; //Abilito il clock per il DAC
    RCC->AHBENR |= 1<<1; //Abilito DMA2
    RCC->AHBENR |= 1; //Abilito DMA1
    RCC->AHBENR |= (1<<28); //Alzo il bit 28 per abilitare il clock di ADC1
    RCC->AHBENR |= (1<<17); //Alzo il bit 17 per abilitare il clock di GPIOA
}
```

```
//Per il collegamento tra PA4 e PA2 di ADC e DAC
```

```
void Configurazione_GPIOA(){
    GPIOA->MODER |= 0x00000300; //PA4 Analogico
    GPIOA->MODER |= 0x00000030; //PA2 Analogico
}
```

```

void Configurazione_TIM2(){
    TIM2->CR2 |= 1<<5; //Setto i bit MMS per abilitare il trigger out sull'evento di update del timer
    TIM2->ARR=12000000; //conto 0.5 secondi con il clok a 72 MHz
    TIM2->CNT=0x0; //azzero il conteggio
}

void Configurazione_DAC(){
    DAC->CR |= 1<<5; //Sto dicendo al dac che deve arrivargli un trigger out dal TIM2
    DAC->CR |= 1<<2; //Abilito il trigger out (tim2) sul DAC
    DAC->CR |=1<<12; //Abilito il DMA sul canale 1 del DAC
    DAC->CR |=1; //Abilitazione canale 1 del DAC
}

void Popolamento_LUT (vettore v, short int n){
    short int A =1023;
    short int Kdac=2048;

    for(int i=0; i<M; i++){ v[i]=(int)(Kdac+A*sin(2*Pi*(float)i/(float)M));
}

void Configurazione_DMA_DAC(vettore v){

DMA2_Channel3->CCR |=1<<10; // Tasferimento 16 a 16
DMA2_Channel3->CCR |=1<<8;//Tasferimento 16 a 16
DMA2_Channel3->CCR |=1<<7; //Impostato che la memoria deve incrementarsi
DMA2_Channel3->CCR |=1<<5;//impostato la modalità circolare
DMA2_Channel3->CCR |=1<<4;//Trasferimento da memoria a periferica
DMA2_Channel3->CNDTR=M; //Devono essere effettuati 100 trasferimenti
DMA2_Channel3->CMAR=(uint32_t)v;
DMA2_Channel3->CPAR=(uint32_t)&DAC->DHR12R1;
DMA2_Channel3->CCR |=1;//abilito il canale

}

void Configurazione_ADC(){

//REGOLATORE DI TENSIONE DI ADC1
ADC1->CR &=~(1<<29); //Passo da 1x0 a 0x0 per il regolatore di tensione ADVREGEN
ADC1->CR |= (1<<28); //PAsso da 0x0 a 1x0
for (int i=0; i<6000000; i++);
//CLOCK
ADC1_2->CCR |= (1<<16); // clock è 72MHz
//CALIBRAZIONE
ADC1->CR |=(1<<31); //ADCA1 a 1 inizio calibrazione
while((ADC1->CR&(1<<31))==(1<<31)); //Aspetto che adcal passi a 0
//Abilitazione del DMA
ADC1->CFGGR |= 1;

//Abilito il trigger esterno
ADC1->CFGGR |= 0xB << 6;
ADC1->CFGGR |= ADC_CFGGR_EXTEN_0;
//TEMPO DI SAMPLING E CANALE DA CONVERTIRE
ADC1->SQR1 |= 0x000000C0; //Scrivo in SQR1 il numero del canale da convertire (3=0011), ricordando che L=0000 (perchè ho una sequenza di un solo canale)
ADC1->SMPR1=1<<11; //Imposto il tempo di sampling a 19.5
//ABILITO LA CONVERSIONE
ADC1->CR |= 0x0001; //Abilito conversione
while((ADC1->ISR&1)==0); //Aspetto che si alzi ADRDY
}

/*Configurazioni simili al DMA_DAC solo che non imposto la modalità circolare e quindi converto un solo periodo della sinusoide, inoltre il trasferimento in questo caso va da periferica a memoria */
void Configurazione_DMA_ADC(vettore v1){

DMA1_Channel1->CCR |= 1<<10;
DMA1_Channel1->CCR |= 1<<8;
DMA1_Channel1->CCR |= 1<<7;
DMA1_Channel1->CNDTR=M;
DMA1_Channel1->CMAR=(uint32_t)&(v1[0]);
DMA1_Channel1->CPAR=(uint32_t)&(ADC1->DR);
DMA1_Channel1->CCR |= 1;

}


```

## MAIN

```
#include "libreria.h"
```

```

vettore LUT,Risultato;

short int i;

void main (){
    Popolamento_LUT(LUT,M);
    RCC_EN(); //Abilito i clock
    Configurazione_GPIOA();
    Configurazione_TIM2();
    Configurazione_DAC();
    Configurazione_DMA_DAC(LUT);
    Configurazione_ADC();
    Configurazione_DMA_ADC(Risultato);
    TIM2->CR1 |=1;//Inizio conversione
    ADC1->CR |=0x00000004; //Start of Conversion data una sola volta perchè c'è il timer che gestisce la conversione
    while((DMA1->ISR&DMA_ISR_TCIF1)==0); //Aspetta che siano terminati i campioni convertiti da ADC (si alza TCIF1 di DMA1)

    while(1);
}

```

### **LIBRERIA.H**

```

#include <stm32f30x.h>
#include <math.h>

#define M 100
#define Pi 3.1415

typedef short int vettore [M];

void RCC_EN ();
void Configurazione_DAC();
void Popolamento_LUT (vettore, short int);
void Configurazione_DMA_DAC(vettore);
void Configurazione_GPIOA();
void Configurazione_ADC();
void Configurazione_TIM2();
void Configurazione_DMA_ADC();
void Configurazione_DMA_ADC(vettore);

```

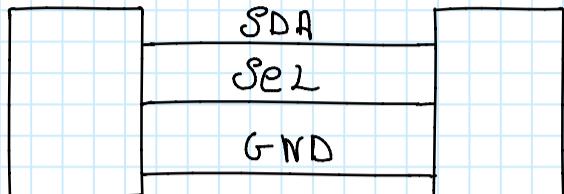
**NOTA:** all'esame tutto su un unico file !

# LABORATORIO 13

lunedì 12 dicembre 2016

## I<sup>2</sup>C (INTER-INTEGRATED CIRCUIT)

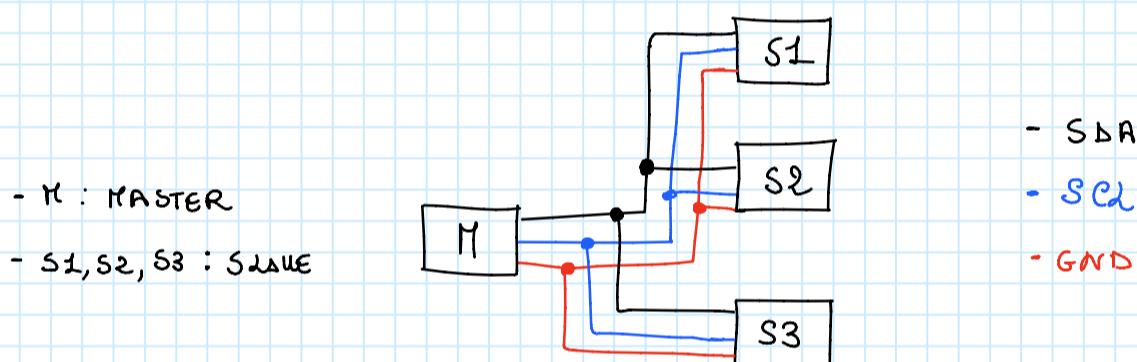
Protocollo di comunicazione seriale sincrono: seriale nel senso che viene trasmesso un bit alla volta, partendo dal più significativo al meno significativo; sincrono nel senso che la comunicazione è temporizzata tramite una linea hardware che si chiama "linea del clock". La comunicazione I<sup>2</sup>C tra due periferiche si realizza con tre fili:



- SDA (SERIAL DATA): linea dei dati, dove viaggiano i bit
- SCL o SCK: linea del clock, serve a temporizzare la comunicazione.
- GND (GROUND): linea rispetto alla quale lo stato delle linee dati e della linea clock viene determinato o alto o basso  
In pratica il GND è il riferimento.

La linea di clock, in particolare la sua transizione basso-alto, avverte la periferica quando c'è un bit da prelevare sulla linea dati, per questo si chiama seriale sincrono: le due periferiche si sincronizzano tra di loro tramite questa linea di clock. Si differenzia proprio per questo dal protocollo di comunicazione asincrono, in cui le due periferiche si mettono d'accordo sulla velocità con la quale devono trasmettersi I bit: ognuno delle due periferiche separatamente conta il tempo e trascorso un certo tempo va a prelevare il valore (il bit) dalla linea dati.

I<sup>2</sup>C al contrario dei protocolli seriali asincroni consente di realizzare un bus, dove viaggiano gli stessi dati e dove possono essere collegate diverse periferiche. In particolare è previsto che sul bus ci sia un "master" che gestisce la comunicazione, tutte le altre periferiche si chiamano "slave"



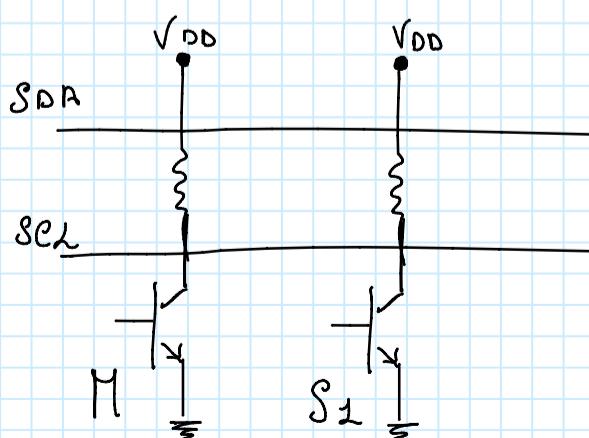
Il master stabilisce a quale slave vuole scrivere o da quale slave vuole leggere e gestisce inoltre la linea di clock, gli slave invece rispondono quando vengono interrogati dal master.

Essendo un bus dove I dati sono accessibili a tutti gli slave, è necessaria una procedura di indirizzamento: ognuno degli slave è identificato da un indirizzo univoco a 7 bit, per cui quando il master vuole comunicare con uno slave, invia l'indirizzo e la periferica che lo riconosce come proprio, invia un segnale di acknowledge (per dire "ho capito che vuoi parlare con me"), mentre gli altri slave mettono le linee in alta impedenza e ignorano la comunicazione.

## LIVELLO FISICO DELLE LINEE DI DATO E DI CLOCK

Dal punto di vista elettronico queste linee sono gestite in Open Drain.

La linea di clock è il drain di un mosfet che può essere mandato in conduzione o in interdizione con un segnale di tensione sulla gate. Quando il mosfet conduce il livello della linea SCL è bassa, quando è interdetto il livello della linea sarebbe indefinito ed è questo il motivo per cui viene messo un resistore di pull-up verso 3 volt, per cui quella linea o vale 3 V o vale 0 V a seconda della tensione applicata alla gate. Tutti I dispositivi del bus I<sup>2</sup>C, sono collegati in questo modo:

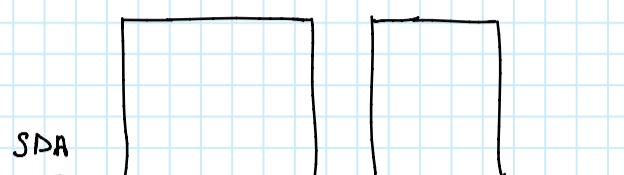


Se nessuno pilota la linea (parla di quella di clock) il suo stato è alto, quindi tutti I mosfet collegati a quella linea sono interdetti. Basta che uno solo dei dispositivi mandi in conduzione il mosfet che la linea si porta al valore basso, per tutti-> tutti la vedono bassa. Questo modo di gestire le linee digitali si chiama "Open Collector" o anche "Hardware And", perchè è una and fatta in hardware: basta che uno metta a zero che tutti vedono zero.

La linea dati è gestita allo stesso modo solo che, mentre la linea clock è gestita solo dal master, quest'ultima è pilotata dal master quando il master vuole scrivere su una periferica ed è pilotata dallo slave quando il master vuole ricevere da una periferica.

Se non c'è attività sul bus I<sup>2</sup>C, la linea dato e la linea clock sono alte, quindi sono a 3 V.

## GESTIONE DELLE LINEE DURANTE UNA TRASMISSIONE (esempio teorico)



- da slave muove con il mosfet la linea clock

- Per mandare un byte vengono inviati 8 segnali di clock

- da linea dati a seconda di chi scrive, ha un andamento di questo tipo

→ • BYTE TRASMESSO IN QUESTO CASO: 11101100  
1 → per ogni clock che ricade nella linea "alta"

Per quanto riguarda la velocità della linea di clock, esistono 3 velocità standardizzate: 100 kHz, 400 kHz, 1MHz.

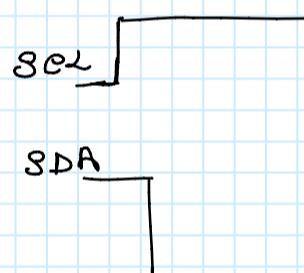
Queste velocità non devono essere rispettate con estrema precisione, perché I<sup>2</sup>C è un protocollo seriale sincrono: se ad esempio consideriamo 100 kHz, allora il periodo del segnale di clock deve durare 10 µs, se invece dovesse durare 11 µs non sarebbe un problema proprio perché il protocollo è seriale sincrono, quindi il bit successivo viene prelevato in corrispondenza del fronte di salita, anche se non sono esattamente 10 µs.

Il fatto che il fronte attivo sia quello di salita, significa che la linea dati può essere cambiata solo quando il clock è basso: quando il clock è basso viene posizionato il dato, in modo che quando si alza (e questo significa che c'è un dato valido sulla linea dati) il bit sulla linea dati sia stabile, perché in questa fase la periferica che ascolta preleva il dato. Quindi quando la periferica si accorge del fronte di salita del clock, preleva il dato e proprio per questo motivo, il dato deve essere stabile sul fronte di salita.

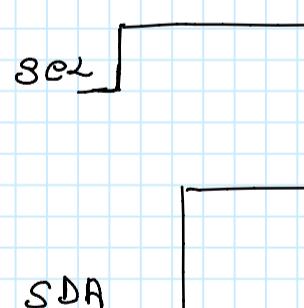
Ci sono solo due casi in cui a clock alto viene mossa la linea dati. Questi due casi sono delle anomalie del protocollo e proprio per questo sono classificati come:

1) Evento di start della comunicazione: quando il master vuole dare il via ad una comunicazione, manda un segnale di start.

Questo segnale è l'abbassamento della linea dati a clock alto (è un'anomalia, perché la linea dati può essere mossa solo quando il clock è basso). Questa anomalia fa capire alle periferiche collegate al bus, che il master ha dato il via ad una nuova comunicazione.



2) Evento di stop della comunicazione: quando il master vuole terminare una comunicazione, invia un segnale di stop. Questo segnale è la transizione basso-alto della linea dati a clock alto.



Osservazione.

Dopo che è stato trasmesso un byte, la periferica che scrive deve ricevere un "acknowledge" da parte di chi sta ascoltando. L'acknowledge è un segnale che serve per far capire alla periferica che la comunicazione è andata a buon fine e viene trasmesso su un altro colpo di clock, per cui per ogni byte trasmesso i colpi di clock sono 9: 8 per il byte e 1 per l'acknowledge. Questo segnale è dato dalla periferica che mantiene la linea dati bassa, mandando in conduzione il suo mosfet.

**DESCRIZIONE DEL PROTOCOLLO** → Cosa devono dirsi il master e lo slave durante la comunicazione

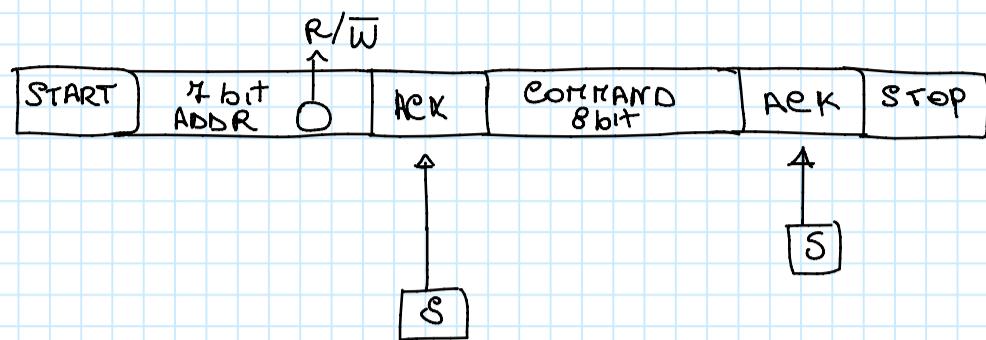
**CASO 1:** Caso in cui il master vuole inviare un comando ad uno slave (Master trasmette/Slave riceve)

Le due linee sono inizialmente alte, finché il master non abbassa la linea dati generando l'evento di start. A questo punto tutti gli slave monitorano la linea clock, fino al momento in cui il master manda l'indirizzo dello slave con il quale vuole comunicare. Insieme all'indirizzo, che è a 7 bit, il master manda un altro bit (per chiudere il byte) che si chiama "Read Not Write", che sta ad indicare cosa vuole fare il master: se vale 0 significa che il master vuole scrivere qualcosa allo slave; se vale 1 vuole dire che il master vuole ricevere qualcosa dallo slave (in questo caso ovviamente è 0).

Lo slave che riconosce l'indirizzo inviato dal master come proprio e manda l'acknowledge (tenendo bassa la linea dati), mentre

tutti gli altri lasciano la linea e si disinteressano della comunicazione fino al prossimo start. Successivamente il master (Nota: lei dice lo slave, ma secondo me non ha senso credo che si sia confusa) invia il comando a 8 bit e aspetta l'acknowledge. Cosa fa lo slave a seguito di questo comando dipende dal particolare dispositivo che è lo slave, ma comunque risponde sempre dando l'acknowledge ad ogni byte che riceve. Termina la comunicazione: il master (Nota: stessa cosa di prima, lei dice lo slave) invia il segnale di stop alzando la linea dati (quindi entrambe le linee tornano alte) e così finisce la comunicazione.

SCHEMA: MASTER TRASMETTE / SLAVE (S) RICEVE

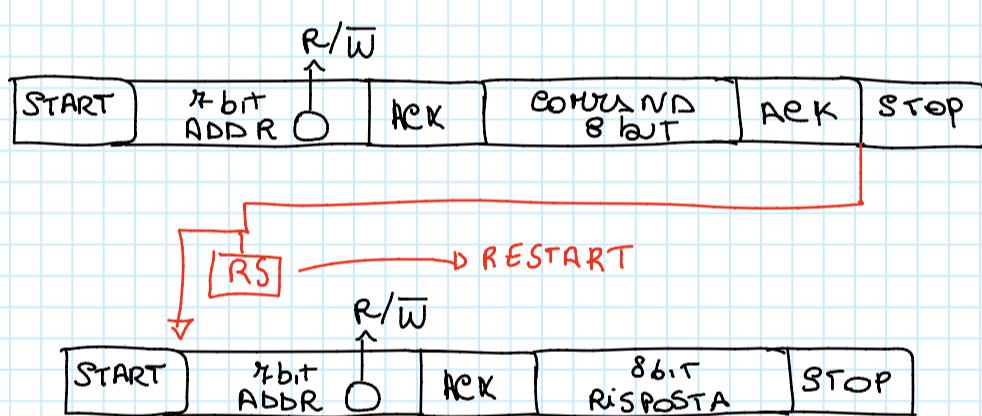


### CASO 2: Master riceve/Slave trasmette

Partendo dal "Caso 1", supponiamo che il master chieda allo slave, con quel comando, di "eseguire una misura e di restituirla poi il risultato", allora lo slave che riceve questo comando, prima di dare l'acknowledge, esegue la misura richiesta. Alla comunicazione del Caso 1 seguirà quindi una nuova comunicazione, per fare in modo che il master possa leggere il risultato trascritto dalla periferica.

Il master manda prima lo start poi l'indirizzo della periferica da cui vuole ricevere. L'indirizzo è lo stesso, perchè la periferica da cui il master vuole ricevere è la stessa che ha eseguito la misura, solo che questa volta l'ultimo bit viene messo a 1 perchè il master vuole leggere da quello slave. Segue quindi un acknowledge da parte dello slave, in seguito al quale il master lascia la linea dati (gestendo così solo il clock), che verrà gestita dallo slave che deve scrivere il risultato della misurazione. A questo punto il master dovrebbe dare l'acknowledge perchè "lo slave ha scritto e vuole sapere se è stato ricevuto ciò che ha scritto". Vale però che se la trasmissione è di un unico byte, Il master può dare anche direttamente lo stop senza dare l'acknowledge, visto che tutte le periferiche quando vedono l'evento di stop lasciano la comunicazione, fino al successivo start.

SCHEMA: MASTER RICEVE / SLAVE TRASMETTE



Questo esempio (caso 1 e 2 insieme) è stato fatto perchè sullo schedino sono presenti un accellerometro e un magnetometro che comunicano tramite I<sup>2</sup> C. Useremo in particolare l'accellerometro considerando una comunicazione in cui il microcontrollore è il master e l'accellerometro è lo slave di cui si dovrà capire l'indirizzo e a cui si dovrà chiedere l'accellerazione lungo uno degli assi (si ha un'accellerazione lungo l'asse x, y, z), il cui valore dovrà poi essere letto dal microcontrollore.

Osservazione.

Invece di dare lo stop e subito dopo lo start, si può adoperare un ulteriore evento che si chiama restart. Quindi se in seguito all'ultimo acknowledge, prima dello stop, il master volesse comunicare con la stessa periferica, allora potrebbe dare l'evento di restart.

Differenza tra start/stop e restart

Dando il restart dal punto di vista delle linee non cambia niente, perchè la linea dati deve comunque alzarsi ed abbassarsi, tuttavia in questo caso si ha che il master non lascia le linee di dato e di clock.

Il primo caso (start/stop) serve in particolare nei bus in cui è prevista la modalità multimaster: c'è un master che gestisce il clock, finchè non termina la comunicazione, da lo stop e un'altra periferica comincia a fare il master (una nuova periferica può fare il master, quando il vecchio master da lo stop) e ciò è possibile perchè il vecchio master mette le linee alte e le lascia, mettendo i mosfet ad alta impedenza, in modo tale che qualche altra periferica possa usarle.

Se invece si da il restart, ciò non è possibile, perchè il master non lascia le linee e quindi rimane sempre lo stesso, non può cambiare.

### RIFERIMENTI SUL REFERENCE MANUAL

## IMPLEMENTATION (pag. 654)

Ci sono due periferiche I<sup>2</sup>C: I<sup>2</sup>C1 e I<sup>2</sup>C2.

## FUNCTIONAL DESCRIPTION (pag. 654) E DIAGRAMMA A BLOCCHI (pag. 655)

Le due linee che si adoperano come dato e clock sono due pin del microcontrollore, quindi bisogna configurare il relativo modulo di GPIO.

I<sup>2</sup>C per funzionare ha bisogno di un segnale di clock, con il quale misurare il tempo con cui pilotare la linea di clock, questo segnale di clock può essere il clock di sistema (72 MHz) oppure quello esterno a 8 MHz. Se non si fa nessuna configurazione e si lascia tutto di default il clock è quello a 8MHz.

## MODE SELECTION (pag. 656)

A seconda di come è programmato, il microcontrollore può fare sia da master che da slave.

## COMMUNICATION FLOW (pag. 656)

Describe il flusso di comunicazione di cui già abbiamo parlato.

## I<sup>2</sup>C INITIALIZATION (pag. 657)

Per abilitare la periferica I<sup>2</sup>C, bisogna abilitare per prima cosa il relativo clock nei registri RCC, e bisogna poi settare un bit (metterlo a 1) che si chiama PE (Peripheral Enable) nel registro I<sup>2</sup>C\_CR1.

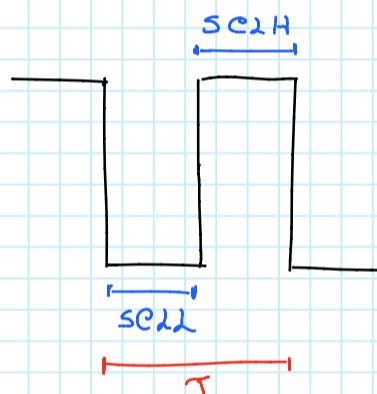
Si ha inoltre la possibilità di adoperare dei filtri sulle linee di dato e clock, nel caso in cui questi segnali presentassero rumore, visto che nella comunicazione digitale è importante riconoscere in maniera pulita i fronti (non è il caso nostro).

## I<sup>2</sup>C TIMINGS (pag. 659, pag. 660, pag. 661)

Come fare in modo che il segnale di clock vari con la frequenza definita dallo standard?

L'accelerometro comunica con una frequenza di 100 kHz, quindi bisogna configurare il microcontrollore in modo che comunichi secondo un I<sup>2</sup>C a 100 kHz.

Sul microcontrollore si ha una periferica che gestisce la comunicazione I<sup>2</sup>C, alla quale bisogna dare alcune informazioni: quanto tempo la linea di clock rimane bassa (SCLL), quanto tempo la linea di clock rimane alta (SCLH) ed infine due ritardi indicati con SCLDEL e SDADEL.



Il ritardo SDADEL, è il tempo che si aspetta, dopo aver abbassato la linea di clock (quindi si ha la possibilità di mettere un nuovo dato), per avere la certezza che questa sia stabilmente bassa. In pratica è il tempo che si aspetta da quando si abbassa la linea di clock a quando si posiziona il dato sulla linea.

Il ritardo SCLDEL, è il tempo che si aspetta, dopo aver posizionato il dato sulla linea, per avere la certezza che il dato sia stabile, prima di alzare la linea di clock.

SCLL, SCLH, SDADEL ed SCLDEL, sono fondamentalmente i quattro parametri che stabiliscono a che velocità deve andare la periferica I<sup>2</sup>C e devono essere forniti alla periferica, scrivendo in un registro che si chiama "TIMINGR". In questo registro, non vanno messi i tempi in secondi, perché devono andare degli interi, ma devono essere espressi secondo il numero di colpi di clock, per questo è fondamentale sapere con quale clock si sta lavorando.

Ad esempio se si sta lavorando con un clock ad 8 MHz, per un tempo di clock di 1 s, bisogna scrivere 8000000.

Nel caso in cui il clock è molto alto e si ottiene un numero che, essendo troppo alto, non entra nel registro TIMINGR, si ha inoltre la possibilità di adoperare un prescaler.

Quindi nel registro TIMINGR dovranno essere indicati:

- Prescaler
- SCLL
- SCLH
- SDADEL e SCLDEL

## DATA TRANSFER (pag. 662)

In questo paragrafo vengono mostrati degli esempi di comunicazione.

### CASO 1 : Ricezione

La periferica I<sup>2</sup>C, si occupa di ricevere un bit alla volta (serializzando la ricezione dei bit), per fornire direttamente tutto il byte che è stato ricevuto. Questo byte viene scritto in un registro che si chiama RXDR (in lettura).

### CASO 2: Trasmissione

In questo caso, il byte va scritto direttamente in un registro che si chiama TXDR, ed è la periferica che si occupa della trasmissione, con un bit alla volta, serializzando il trasferimento dei bit

## CLOCK STRETCHING (NOTA)

Il clock è gestito solo dal master. Supponiamo che il master voglia una misura lungo l'asse x e che poi voglia leggerla. Ce l'ha fatta in questo tempo lo slave ad eseguire la misura? Visto che lo slave non gestisce la linea di clock, il tempo è dettato dal master, come si può avere la certezza che quando il master invia un certo numero di colpi di clock, perchè vuole leggere la risposta, lo slave è pronto a scriverla?

Lo slave ha un'unica arma a suo vantaggio, che si chiama "clock stretching": manda in conduzione il suo mosfet sulla linea clock mantenendola bassa (in modo che il master non riesce a pilotarla) fino a quando non è pronto a rispondere (cioè ha scritto nel registro TXDR), per cui rilascia la linea e quindi il master può mandare I colpi di clock.

#### MODALITA' MASTER MODE (inizio pag. 672).

A pag. 674 c'è una tabella in cui vengono mostrati I tempi minimi che sono previsti dallo standard I<sup>2</sup>C.

In particolare: Comunicazione Standard->100 kHz

Fast Mode-> 400 kHz

Fast Mode Plus -> 1 MHz

Per ogni modalità vengono quindi mostrati I tempi minimi dei vari parametri visti in precedenza.

Ad esempio nella comunicazione standard, il periodo basso del segnale di clock deve essere almeno 4.7  $\mu$ s, il segnale alto del periodo di clock deve essere almeno 4  $\mu$ s, tempo di salita delle linee deve essere al massimo 1000 ns, mentre il tempo di discese deve essere al massimo 300 ns.

I valori da mettere nel timingr dipendono da questa tabella.

#### MASTER COMMUNICATION INITIALIZATION (pag. 674)

Inizializzazione della comunicazione-> Fase di indirizzamento.

Bisogna lavorare sul registro CR2. In particolare in questo registro bisogna:

- Inserire l'indirizzo dello slave a 7 bit (anche se il protocollo I<sup>2</sup>C prevede anche un indirizzamento a 10 bit, ma noi non lo adopereremo).
- Impostare la direzione della trasmissione, scrivendo in un bit che si chiama RD\_WRN
- Specificare quanti byte si vogliono scrivere allo slave, infatti se non si da lo stop la comunicazione è continua, dando invece il numero di byte che si vogliono trasmettere, allora provvede direttamente la periferica alla trasmissione, fermandosi non appena è terminato il numero di byte indicato.
- Dare il via alla comunicazione con l'evento di start, alzando un bit che si chiama START.

Una volta fatte queste configurazioni, dopo aver manda to lo start, il master aspetta l'acknowledge per avere la certezza che l'indirizzo sia stato recepito dalla periferica giusta. Successivamente abbassa il bit di START.

Allora il bit di start viene alzato via software (siamo noi ad impostarlo) e viene abbassato in hardware in questo punto.

Quindi se dopo aver mandato l'indirizzo, si vuole scrivere nella periferica, bisogna aspettare che lo start passi a 0.

NOTA: Si ha la stessa procedura per il restart, si alza il restart e si aspetta che questo torni a 0.

#### MASTER TRANSMITTER (pag. 676)

Supponiamo ora che, dopo tutta la fase iniziale di indirizzamento, appena si abbassa lo start, si voglia scrivere un byte.

Quando può essere scritto questo byte ?

Bisogna monitorare un flag di controllo che si chiama TXIE che vale

-1 quando il registro di trasmissione è vuoto.

-0 quando il registro di trasmissione è pieno.

Quindi prima di scrivere un byte, bisogna controllare che il registro di trasmissione sia vuoto, altrimenti si commette un errore, si sovrascrive un dato che è in fase di trasmissione.

Dopo aver controllato il TXIE, se il registro è vuoto, si può scrivere il byte nel registro di trasmissione TXDR.

Da questo momento il byte verrà inviato un bit alla volta. La trasmissione termina quando TXIE passa a 0.

Si ha inoltre la possibilità di dare automaticamente lo stop, tramite un bit che si chiama AUTOEND.

Se questo bit è a 1, se è stato specificato il numero di byte da trasferire, una volta trasferito tutto il numero di byte configurato, allora la periferica da automaticamente lo stop.

Mettendo invece AUTOEND a 0, per fermare la trasmissione, bisogna dare lo stop via software, settando un bit (a 1) che si chiama STOP.

La sequenza di Stop termina quando torna a 0 (come nel caso dello start).

Oltre a TXIE, come flag per capire se si può scrivere, è possibile monitorare anche un altro bit che si chiama TXIS, che si alza alla fine di ogni trasmissione.

#### MASTER RECEIVER (pag. 680)

Quando il master deve ricevere, la parte dell'indirizzamento è la stessa: si configura in CR2 l'indirizzo della periferica, si scrive 1 in RD\_WRN e il numero di byte che ci si aspetta di leggere ed infine si alza il bit START e si aspetta che questo si azzeri.

Il master dopo deve aspettare che lo slave gli scriva, per fare questo bisogna monitorare un altro bit che si chiama RXNE che quando è uguale a 1 vuol dire che il registro di ricezione non è vuoto.

Quindi quando RXNE è pari a 1, si può andare a leggere quello che ha scritto lo slave che si trova nel registro RXDR, per poi dare lo stop.

Anche in questo caso se è impostato l'AUTOEND, dopo aver letto il numero di byte impostato, automaticamente viene dato lo stop, altrimenti va dato via software.

A pag. 684 ci sono diverse tabelle di esempio a seconda della frequenza di clock con cui opera I<sup>2</sup>C, nel nostro caso il clock dell'I<sup>2</sup>C (e solo di questo) è 8 MHz, quindi consideriamo la prima tabella (che prenderemo come riferimento).

#### 24.4.9 I2Cx\_TIMINGR register configuration examples

Table 84. Examples of timings settings for  $f_{I2CCLK} = 8 \text{ MHz}$

Parameter	Standard mode		Fast Mode	Fast Mode Plus
	10 kHz	100 kHz	400 kHz	600 kHz
PRESC	1	1	0	0
SCLL	0x07	0x13	0x9	0x6
t <sub>SCLL</sub>	200x250 ns = 50 $\mu\text{s}$	20x250 ns = 5.0 $\mu\text{s}$	10x125 ns = 1250 ns	7x125 ns = 875 ns
SCLH	0x03	0xF	0x3	0x3
t <sub>SCLH</sub>	196x250 ns = 49 $\mu\text{s}$	16x250 ns = 4.0 $\mu\text{s}$	4x125 ns = 500 ns	4x125 ns = 500 ns
t <sub>SD</sub> <sup>(1)</sup>	$\sim 100 \mu\text{s}$ <sup>(2)</sup>	$\sim 10 \mu\text{s}$ <sup>(2)</sup>	$\sim 2500 \text{ ns}$ <sup>(3)</sup>	$\sim 2000 \text{ ns}$ <sup>(4)</sup>
SDADEL	0x2	0x2	0x1	0x1
t <sub>SDADEL</sub>	2x250 ns = 500 ns	2x250 ns = 500 ns	1x125 ns = 125 ns	1x125 ns = 125 ns
SCLDEL	0x4	0x4	0x3	0x1
t <sub>SCLDEL</sub>	5x250 ns = 1250 ns	5x250 ns = 1250 ns	4x125 ns = 500 ns	2x125 ns = 250 ns

1. SCL period t<sub>SCL</sub> is greater than t<sub>SCLL</sub> + t<sub>SCLH</sub> due to SCL internal detection delay. Values provided for t<sub>SCL</sub> are examples only.
2. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 1000 ns
3. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 750 ns
4. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 655 ns

In caso di 8 MHz e comunicazione a 100 kHz, I conteggi che bisogna mettere per avere I tempi di pag. 674 sono quelli indicati in questi tabella:

- PRESC=1
- SCLL=0x13
- SCLH=0xF
- SDADEL=0x2
- SCLDEL=0x4

Fa anche vedere questi numeri a che tempi corrispondono, considerando che il periodo è 250 ns.

#### ERROR CONDITION (pag. 699)

Sono presenti dei bit di errore nei registri di status per capire se si è presentato un errore di comunicazione.

- Flag di Bus Error
- Flag di Arbitration Lost
- Overrun/Underrun : si possono verificare ad esempio quando si scrive nel TXDR, quando questo era già pieno.

#### DMA REQUEST (pag. 701)

Nel caso in cui si volessero inviare/ricevere tanti byte tramite I<sup>2</sup>C si ha la possibilità di far gestire il trasferimento/ricezione ad un DMA.

#### I<sup>2</sup>C INTERRUPTS (pag. 702)

Possono infine essere generati degli eventi di interrupt in corrispondenza di alcuni eventi su I<sup>2</sup>C, ad esempio su RXNE (si va in interrupt quando si riceve qualcosa), oppure quando si trasmette qualcosa (TXIE) , sull'evento di stop (STOPF), sull'evento di start, su un errore...

NOTA: l'interrupt legato all'I<sup>2</sup>C è uno solo, globale dell'I<sup>2</sup>C, quindi c'è un'unica ISR che deve capire quale degli eventi dell'I<sup>2</sup>C ha fatto scattare l' interrupt.

#### REGISTRI

- CONTROL REGISTER 1 (CR1-> pag.703)
  - Tutti i bit che terminano in IE riguardano gli interrupt enable in corrispondenza di un particolare evento
  - DNF: bit che permette di inserire un filtro sulla linea del clock
  - RXDMA e TXDMA: consentono l'abilitazione del DMA in ricezione (RX) o in trasmissione (TX).
  - PE: messo a 1 abilita la periferica (unico di interesse in questo registro)
- CONTROL REGISTER 2 (CR2-> pag.707)
  - PECBYTE: normalmente lasciato a 0.
  - AUTOEND: messo a 1 manda automaticamente lo stop dopo che sono stati trasmessi tutti I byte indicati.
  - RELOAD: mettendolo a 1 permette di mandare più volte lo stesso byte allo slave
  - NBYTE: il numero di byte da inviare allo slave, bisogna impostarlo prima di iniziare la comunicazione
  - NACK: l'acknowledge inviato dal master (quando lo slave scrive al master ed è il master a dover dare l'acknowledge), si chiama "Not Acknowledge" ovvero NACK. Se si vuole dare un not acknowledge bisogna alzare questo bit.
  - STOP e START: già visti
  - ADD10: lo si alza se l'indirizzamento è a 10 bit
  - RD\_WRN-> Bit Read not Write: 0 se il master vuole scrivere, 1 se vuole leggere.
  - SADD: indirizzo dello slave con il quale il master vuole parlare. Ci sono 10 bit, di cui si adoperano solo I 7 meno significativi .
- OWN ADDRESS REGISTER (OAR-> pag.709 e 710)
 

Registro in cui si scrive l'indirizzo del microcontrollore, nel caso in cui sia lo slave.
- REGISTRO DI TIMING (TIMINGR-> pag.711)

Già descritto. Da configurare prima dell'inizio della comunicazione.

- **TIMEOUT REGISTER** (TIMEOUTR-> pag.712)

Per impostare un tempo di timeout.

- **INTERRUPT AND STATUS REGISTER** (ISR-> pag.713)

In questo registro ci sono i flag, in sola lettura, a seconda dell'evento che si è verificato. A parte quelli già descritti, ci sono :

- bit **BUSY**: segnala se il bus è occupato o libero, in particolare diventa 1 se c'è attività sul bus
- bit **TC**: si alza quando sono stati inviati tutti gli NBYTE scritti nel CR2, ad esempio se NBYTE=3 ad ogni byte si alza il bit TXIE, ma TC si alza solo quando sono stati inviati tutti e 3 I byte.
- bit **DIR** (direzione è in lettura o in scrittura?) e il bit **ADDR** (è stato mandato un indirizzo o un comando?): sono adoperati solo in slave mode quando lo slave vuole capire se gli è stato inviato un indirizzo o un comando e se il master vuole leggere dallo slave o vuole che sia lo slave a scrivere.

- **INTERRUPT CLEAR REGISTER** (ICR-> pag. 716)

Registro che dà la possibilità di azzerare I flag, scrivendo 1 nel relativo bit, visto che non è possibile agire direttamente sui bit dello Status Register per cancellarli.

- **RXDR** (pag. 717)

Registro da cui si legge. I bit significativi sono solo 8, perché si può leggere solo un byte alla volta

- **TXDR** (pag. 718)

Registro in cui si scrivono I byte da inviare.

### ESERCIZIO.

Scrivere nel CTRL\_REG1\_A (Control Register 1\_A) dell'accelerometro le configurazioni per l'attivazione delle componenti dell'accelerazione lungo I tre assi (x,y,z) e verificare con una lettura che la scrittura sia andata a buon fine.

Per poter svolgere questo esercizio, avendo già descritto I<sup>2</sup>C, che deve essere adoperato per il trasferimento e per la lettura, bisogna ora vedere le caratteristiche dell'accelerometro.

NOTA: Microcontrollore → Master / Accelerometro → Slave

### DATASHEET ACCELEROMETRO (axis accelerometer- magnetometer)

L'accelerometro può comunicare tramite I<sup>2</sup>C a 100 kHz .

Il suo indirizzo è mostrato nella tabella seguente (che è quella di pag. 20 ) ed è quindi il valore da scrivere nel campo "Address", relativo allo slave, ogni volta che si vuole comunicare con l'accelerometro.

Table 14. SAD+Read/Write patterns

Command	SAD[7:1]	R/W	SAD+R/W
Read	0011001	1	00110011 (33h)
Write	0011001	0	00110010 (32h)

In particolare questa tabella mostra anche l'indirizzo nei due casi di lettura e scrittura, considerando il bit Read Not Write, ma nel nostro caso è inutile perché della realizzazione degli 8 byte se ne occupa la periferica I<sup>2</sup>C.

Allora l'indirizzo dell'accelerometro è semplicemente 0011001, a cui corrisponde il valore in esadecimale 0x19.

### REGISTER MAPPING pag. 22

Le funzioni dell'accelerometro sono organizzate in registri, per cui il "comando" da dare allo slave in realtà è un indirizzo.

L'accelerazione è su 12 bit (per tutti e tre gli assi) quindi l'accelerazione ha due registri per ogni asse: quello del byte basso e quello del byte alto. Ad esempio il byte basso dell'accelerazione lungo x è conservato all'indirizzo 0x28 e se si vuole leggere questo valore, bisogna scrivere all'accelerometro 0x28, perché nel manuale dell'accelerometro "il comando" è concepito come l'indirizzo che si vuole andare a leggere.

### CTRL\_REG1\_A (pag. 24)

Registro suddiviso in 8 bit, di cui I primi 4 servono a definire l'output data rate (ODR), che indica la frequenza con la quale si misura l'accelerazione (quanto tempo deve essere misurata l'accelerazione). I valori da mettere in questi quattro bit sono I seguenti:

Table 20. Data rate configuration

ODR3	ODR2	ODR1	ODR0	Power mode selection
0	0	0	0	Power-down mode
0	0	0	1	Normal / low-power mode (1 Hz)
0	0	1	0	Normal / low-power mode (10 Hz)
0	0	1	1	Normal / low-power mode (25 Hz)
0	1	0	0	Normal / low-power mode (50 Hz)
0	1	0	1	Normal / low-power mode (100 Hz)
0	1	1	0	Normal / low-power mode (200 Hz)
0	1	1	1	Normal / low-power mode (400 Hz)

In particolare il valore 0000 significa che l'accelerometro è spento, quindi bisogna mettere un valore diverso.

Ad esempio se si sceglie 0001, allora il valore dell'accelerazione sui tre assi viene aggiornato ogni secondo (questo significa 1 Hz), se invece si sceglie 0010, il valore dell'accelerazione sui tre assi viene aggiornato ogni 10 ms (10 Hz), se si sceglie 0011, viene aggiornato ogni 100 ms (100 Hz),....

NOTA: Per l'esercizio scegliere un qualsiasi valore tra questi purchè sia diverso da 0.

I restanti 4 bit sono:

- LPEN: low power mode enable -> Può essere lasciata a 0 senza problemi.
- XEN YEN ZEN : Consentono l'abilitazione dell'accelerometro sui tre assi, perciò se si vuole fare una misura di accelerazione, questi tre bit devono essere abilitati.

#### CONFIGURAZIONE DI CTRL\_REG1\_A PER LA RISOLUZIONE DELL'ESERCIZIO

Quindi lo scopo è quello di scrivere ad uno slave il cui indirizzo è 0x19 che "all'indirizzo 0x20, deve scrivere 0x17", dove 0x17 corrisponde in binario a 00010111 (0001->bit ODR / 0111->abilitazione degli assi), e verificare poi, con una lettura, che nel registro sia stato effettivamente scritto 0x17.

Configurazione della comunicazione:

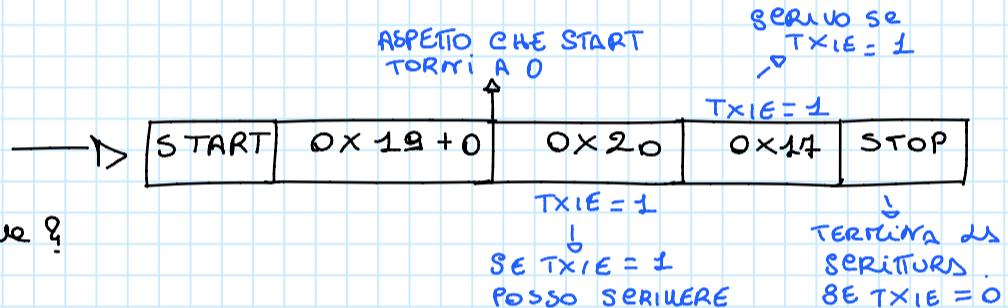
INDIRIZZO SLAVE: 0x19

R/W = 0 perché voglio scrivere

NBYTE = 2  
↓

Im particolare quali byte scrivo allo slave?

- INDIRIZZO CTRL\_REG1\_A: 0x20
- COSA SERIVO IN CTRL\_REG1\_A: 0x17

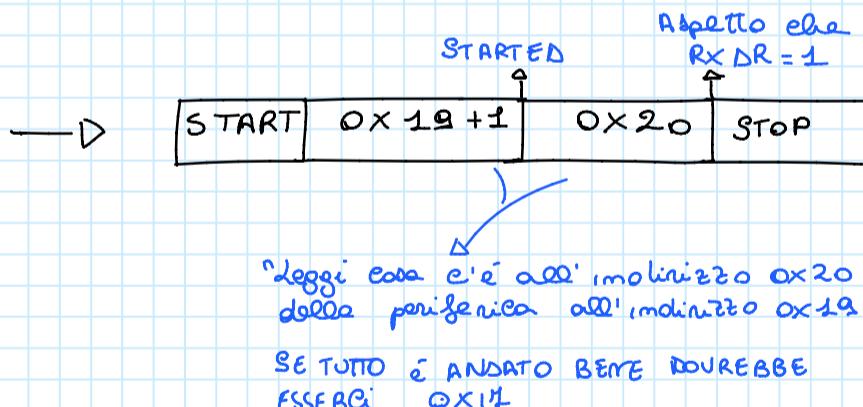


Lettura -> leggere cosa c'è all'indirizzo 0x20:

SADD : 0x19 (sempre lo stesso)

NBYTE = 1 => Voglio leggere 1 byte che c'è all'indirizzo 0x20

R/W = 1 => perché voglio leggere.



#### NOTE:

- 1) Tutte le attese vanno fatte con cicli while, perchè è tutto temporizzato.
- 2) I<sup>2</sup>C non può essere debuggato con uno step alla volta, perchè si rischia di avere un errore di arbitration lost, visto che la sincronizzazione è fondamentale.



# LABORATORIO 14

(Continuo Laboratorio 13).

venerdì 16 dicembre 2016

## CONFIGURAZIONE GPIO PER COMUNICARE VIA I<sup>2</sup>C

Per I<sup>2</sup>C si adoperano due linee (dato e clock) che sono fisicamente (via Hardware) collegate alle linee di dato e di clock dell'accelerometro sullo schedino. Bisogna quindi trovare l'aggancio tra i pin del microcontrollore e le periferiche interne, per poter configurare opportunamente la relativa periferica GPIO.

DataSheet → Pinouts and pin description → La tabella pag. 35, indica ogni pin a quale periferica interna al microcontrollore è collegato.

In questa tabella vanno cercate la linea SDA e SCL dell'I<sup>2</sup>C1.

A pag. 40 si scopre, nella colonna delle Alternate Function, che queste linee sono PB6 e PB7.

92	58	42	PB6	I/O	FTf		I2C1_SCL, USART1_TX, TIM16_CH1N, TIM4_CH1, TIM8_CH1 <sup>(3)</sup> , TSC_G5_IO3, TIM8_ETR, TIM8_BKIN2 <sup>(3)</sup>	
93	59	43	PB7	I/O	FTf		I2C1_SDA, USART1_RX, TIM3_CH4, TIM4_CH2, TIM17_CH1N, TIM8_BKIN, TSC_G5_IO4	
94	60	44	BOOT0	I	R			Root memory selector

Quindi la periferica che si dovrà configurare (preventivamente, prima di fare una qualsiasi comunicazione I<sup>2</sup>C) è la GPIOB. Come va configurata?

Dopo aver abilitato il clock di GPIOB, bisogna dire che PB6 e PB7 funzionano come linea clock e dato di I<sup>2</sup>C.

L'informazione relativa alle due linee è stata trovata nella colonna Alternate Function della tabella a pag. 35, per cui nel MODER si dovranno impostare queste due linee in modalità "alternate".

Visto che ogni linea può fare più di un'alternate, dopo aver settato i relativi bit nel MODER, bisogna anche specificare quale alternate si sta considerando.

La tabella di pag. 35 è immediatamente seguita dalle tabelle delle Alternate Function per ogni porta (A, B, C,...), in particolare la tabella relativa alla porta B si trova a pag. 43, dalla quale si legge che in corrispondenza di PB6 e PB7, si ha che I<sup>2</sup>C1\_SCL e I<sup>2</sup>C1\_SDA sono l'alternate function 4 (AF4).

Table 15. Alternate functions for port B

AF n°	Port & Pin Name	AF0	AF1	AF2	AF3	AF4
5	PB0			TIM3_CH3	TSC_G3_IO2	TIM8_CH2N
6	PB1			TIM3_CH4	TSC_G3_IO3	TIM8_CH3N
2	PB2				TSC_G3_IO4	
10	PB3	JTDO-TRACE SWO	TIM2_CH2	TIM4_ETR	TSC_G5_IO1	TIM8_CH1N
10	PB4	NJTRST	TIM16_CH1	TIM3_CH1	TSC_G5_IO2	TIM8_CH2N
9	PB5		TIM16_BKIN	TIM3_CH2	TIM8_CH3N	I2C1_SMBA
9	PB6		TIM16_CH1N	TIM4_CH1	TSC_G5_IO3	I2C1_SCL
8	PB7		TIM17_CH1N	TIM4_CH2	TSC_G5_IO4	I2C1_SDA

Questa informazione, va impostata nel registro AFR della periferica GPIO. Il registro AFR è organizzato in modo tale che per ogni linea si hanno 4 bit con i quali si configura il tipo di alternate function.

Poichè le linee in questione sono PB6 e PB7 e visto che l'alternate function in questione è la AF4, si dovrà scrivere in AFRL6 e in AFRL7 il valore 4.

Osservazione: Essendo 15 le possibili linee di una periferica GPIO, il registro AFR, dove si impostano le alternate function, viene suddiviso in un registro AFR parte bassa e AFR parte alta, per poter lavorare su tutte le linee.

## ELENCO PUNTATO SU COME DEVE ESSERE FATTO IL CODICE DEL LABORATORIO 13 (Esercizio 1)

- Abilitazione dei clock (I<sup>2</sup>C e GPIOB)
- Configurazione PB6 e PB7 con AF4 (in MODER6 e MODER7 andrà scritto 10 e in AFR6 e AFR7 4)
- Configurazione Timing Register con i valori mostrati nella la tabella di pag 684 del Reference Manual (prima di abilitare la periferica).
- Abilitazione della periferica I<sup>2</sup>C alzando il bit PE nel registro CR1

- **Configurazione del registro CR2:**
  - Indirizzo dello slave al quale si vuole scrivere -> bit SADD. Per questo indirizzo si hanno a disposizione 10 bit, perchè si ha anche la possibilità di effettuare un indirizzamento a 10 bit. Quando si adopera l'indirizzamento a 7 bit, vengono usati solo I bit che vanno da 7 a 1 del campo SADD, I bit 9 e 0 vengono adoperati solo per quello a 10 bit. Per questo motivo in bit l'indirizzo dello slave è **shiftato di 1**.  
Allora nel nostro caso lo slave è l'accelerometro, il cui indirizzo in esadecimale è 0x19, che andrà scritto in SADD shiftato di 1.
  - R/W = 0 (poichè si vuole scrivere)
  - NBYTES = 2.  
Per abilitare I tre sensori sui tre assi, bisogna fare una configurazione che va fatta (in generale) scrivendo l'indirizzo del registro in cui si vuole scrivere e cosa si vuole scrivere (per questo 2 byte).
- Dare lo Start alzando il bit start del registro CR2, che si abbassa al nono colpo di clock, quando la periferica I<sup>2</sup>C ha trasmesso Il byte relativo a Indirizzo+R/W e ha anche ricevuto l'acknowledge. Allora dopo aver dato lo start bisogna aspettare che sia finita questa trasmissione, quindi bisogna **aspettare che lo start si abbassi**.
- Verificare che non ci sono trasmissioni in corso, aspettando che il flag TXIE (indica se il registro di trasmissione è libero) diventi 1 e scrivere poi nel registro di trasmissione TXDR 0x20, ovvero l'indirizzo del registro di controllo 1 dell'accelerometro.  
**Bisogna poi aspettare che la trasmissione**, un bit alla volta, **sia finita** e quindi aspettare che TXIE diventi di nuovo 1, per poter inviare il secondo byte.
- Nuova scrittura in TXDR, per mandare il secondo byte. Il valore da scrivere questa volta è 0x17.
  - La prima cifra esadecimale è praticamente l'ODR, ovvero la frequenza con cui l'accelerometro deve aggiornare le misure di accelerazione, lasciandola al valore di default (0000) il modulo è spento. Bisogna quindi mettere un qualsiasi valore che sia diverso da 0, ad esempio 0x1 (<-> 0001).
  - La seconda cifra esadecimale è 0x7, in binario 0111, e serve ad abilitare I tre sensori sui tre assi "z enable, y enable, x enable".
- Aspettare che anche questa trasmissione sia conclusa e quindi dare lo Stop.  
Anche lo Stop ha un suo tempo per di esecuzione, ma allora bisogna aspettare che la procedura di stop termini e quindi bisogna **aspettare che il bit STOPF** (che si alza se è stata rilevata una procedura di stop sul bus) del registro ISR diventi 1.
- Per dare il via ad una nuova comunicazione, bisogna abbassare il flag STOPF, altrimenti la linea viene vista in stop e la periferica I<sup>2</sup>C non può cominciare una nuova comunicazione. Per poter cancellare questo flag (e tutti gli altri) bisogna scrivere 1 nel bit corrispondente (in questo caso il 5) del registro ICR.

Alla fine di questa sequenza di istruzioni, l'accelerometro è acceso.

#### ELENCO PER LA LETTURA DALL'ACCELEROMETRO (Esercizio 2)

La lettura avviene attraverso una fase di scrittura, in cui si scrive all'accelerometro da quale registro si vuole leggere, e una successiva fase di lettura, in cui l'accelerometro scrive il contenuto del registro.

Per dare il via ad una nuova trasmissione si può partire direttamente dal riempimento del registro CR2, dato che tutto il resto è già stato configurato.

#### CONFIGURAZIONE DEL REGISTRO CR2 PER LA LETTURA

- L'indirizzo dello slave è sempre lo stesso, quindi è **inutile riscriverlo**.
- R/W = 0 (prima fase di scrittura per scrivere l'indirizzo da cui si vuole leggere)
- **NBYTES=1**  
Osservazione: in questo caso non va fatta una "or", perchè il valore di NBYTES deve cambiare e passare quindi da 2 (valore scritto per la trasmissione precedente) a 1. Dovrebbe in realtà essere prima azzerato.
- Dare lo Start e aspettare che il bit start passi a 0.
- Aspettare che TXIE passi a 1 e scrivere in TXDR l'indirizzo del registro di cui si vuole leggere il contenuto.  
Per vedere dove l'accelerometro conserva l'accelerazione, bisogna scorrere la tabella di pag. 22 del manuale axis accelerometer- magnetometer (Data Sheet dell'accelerometro). Essendo un accelerometro triassiale e poichè ognuna delle accelerazioni è formata da 12 bit, si hanno 6 registri che sono: byte basso dell'accelerazione lungo x, byte alto dell'accelerazione lungo x, byte basso y, byte alto y, basso z, alto z. Quindi bisognerà leggere da questi 6 registri. Supponiamo per ora di leggere solo da un registro, in particolare dal primo, quello che ha indirizzo 0x28. In TXDR andrà scritto 0x28 (per ora).

- Aspettare che TXIE=1.

- Fine della fase di scrittura.

Secondo il protocollo I<sup>2</sup>C si dovrebbe dare uno stop e cominciare una nuova trasmissione in lettura dando lo start.

Altra opzione è quella di utilizzare la procedura di restart, settando il bit AUTOEND del CR2 e dando direttamente un nuovo start.

Allora alla fine della fase di scrittura si hanno due opzioni:

- 1) Dare lo stop, aspettare lo stop flag, cancellare il flag, dare lo start.
- 2) Mettere a 1 AUTOEND e dare direttamente lo start.

Prima di dare lo start in entrambi i casi bisogna di nuovo configurare CR2, per la fase di lettura.

- Indirizzo dello slave è sempre lo stesso.

• R/W = 1 (lettura)

- NBYTES=1 (perchè ogni registro, dei sei, contiene un solo byte)

- START (aspettare che passi a 0).

- Aspettare che il bit RXNE diventi 1.

Questo flag passa a 1 quando il registro di ricezione è pieno e serve per capire se è stato ricevuto tutto il byte dall'accelerometro. Solo quando RXNE=1 si può segnare il risultato in una variabile RISULTATO (ad esempio).

-> RISULTATO = RXDR

- Stop

Tutto questo se si vuole leggere solo da un indirizzo tra i sei in cui l'accelerometro conserva tutto il risultato.

Per poter leggere da tutti e sei i registri (i cui indirizzi sono 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D), si dovrebbe ripetere tutta questa procedura per ogni indirizzo.

Il costruttore dell'accelerometro ha messo a disposizione una funzionalità molto comoda, ovvero l'autoincremento dell'indirizzo, che avviene mandando l'indirizzo con il bit più significativo alto.

In pratica :

- 0x28=10101000 (in binario), l'indirizzo del primo registro in cui viene conservata l'accelerazione
- Alzando il bit più significativo (0x28->0xA8) e mandandolo all'accelerometro, questo capisce che deve incrementare gli indirizzi. In questo modo si sta dicendo all'accelerometro che non si vuole leggere un solo byte ma sei.

Allora l'accelerometro manda il byte dell'indirizzo 0x28, incrementa l'indirizzo e manda il secondo byte dell'indirizzo 0x29, incrementa di nuovo l'indirizzo e manda il terzo byte contenuto in 0x2A,..., fino al contenuto di 0x2D.

### LETTURA DA TUTTI E SEI GLI INDIRIZZI DELL'ACCELEROMETRO (Esercizio 3).

Dall'esercizio 2 si dovrebbero quindi ricevere 6 byte che sono basso asse x, alto asse x,..., alto asse z.

Per il risultato, bisognerà quindi creare un vettore di 6 elementi (di interi ad 8 bit senza segno, quindi di tipo char o unsigned short int), dove ognuno di questi elementi è il contenuto dei registri 0x28, 0x29,...,0x2D.

I byte dei sei registri, prima di essere scritti nel vettore, vanno elaborati.

### ELABORAZIONE DEI BYTE

A pag. 9 del Data Sheet dell'accelerometro c'è la tabella delle caratteristiche del sensore.

Si ha la possibilità di settare il fondoscalo del sensore attraverso uno dei registri di controllo. Lasciandolo al valore di default il FS è più o meno 2g, 4 g=4000 mg in totale. L'uscita che restituisce è a 12 bit (corrispondente a 4095 codici).

Dato che il FS è 4000 mg e dividendo per 4095 codici (che è l'uscita), la sensibilità è circa 1mg per codice.

Allora su 12 bit si ha una lettura dell'accellerazione in mg.

L'accelerometro organizza questi 12 bit in due registri consecutivi di 1 byte ciascuno.

Ad esempio per quanto riguarda l'asse x:

0x28  → PARTE BASSA DI x

0x29  → PARTE ALTA DI x

I 12 bit che esprimono l'accellerazione lungo x, l'accelerometro li memorizza in questo modo:



Quindi si legge prima quello che è contenuto in 0x28 e poi quello contenuto in 0x29, per cui l'accellerazione va ricostruita.

Visto che si hanno in totale 16 bit (8 bit di 0x28 e 0x29), l'accellerazione lungo x su 16 bit si ottiene :

ACCX = parte\_alta (\* 256 oppure shiftata di 8)+parte\_bassa

NOTA: questi byte sono degli interi senza segno, allora bisogna effettuare prima un cast (unsigned short int) altrimenti non si può moltiplicare la parte alta per 256.

## ACCELERAZIONE SUI TRE ASSI:

- Si dichiara un vettore di char RES, da cui si ricevono I 6 byte.
- Si dichiara poi un secondo vettore (tipo short int) RES2, in cui ci sono le tre accelerazioni a 16 bit.  
In particolare:

$$\textcircled{1} \text{ RES2[0]} = (\text{unsigned short int}) \text{RES[1]} \times 256 + \text{RES[0]}$$

↓  
Accelerazione lungo x

Conversione di RES[1] (parte alta di x) per 256

parte bassa di x

$$\textcircled{2} \text{ RES2[1]} = (\text{unsigned short int}) \text{RES[3]} \times 256 + \text{RES[2]}$$

↓  
accelerazione lungo y

Parte alta di y

parte bassa di y

$$\textcircled{3} \text{ RES2[2]} = (\text{unsigned short int}) \text{RES[5]} \times 256 + \text{RES[4]}$$

↓  
accelerazione lungo z

parte alta di z

parte bassa di z

In RES2 si hanno I 16 bit che esprimono l'accelerazione lungo x, lungo y e lungo z.

Nella descrizione dell'accelerometro, si ricavano due informazioni importanti:

- 1) L'accelerazione viene fornita in mg con una rappresentazione a complemento di 2, bisogna quindi effettuare un altro cast -> questi interi senza segno a 16 bit devono essere trasformati in interi con segno.

Si dichiara un nuovo vettore RES3 (tipo short int) e lo si pone uguale a RES2.

$\text{RES3[0]} = \text{RES2[0]}$

$\text{RES3[1]} = \text{RES2[1]}$

$\text{RES3[2]} = \text{RES2[2]}$

Passaggio dal 16 bit senza segno al 16 bit con segno.

- 2) L'accelerometro fornisce 12 bit in uscita su 16 allineati a sinistra. Le cifre dell'accelerazione sono le prime 12, allora per ottenere l'accelerazione in mg, bisogna spostare a destra di 4 bit il valore dell'accelerazione (dividendo per 16 o shiftando di 4).

Ricapitolando

- La comunicazione I<sup>2</sup>C fa avere 8 bit senza segno
- Questi bit devono essere passati in un vettore di sei elementi (ogni accelerazione occupa 2 byte) di interi senza segno
- Si crea un altro vettore in cui c'è la combinazione degli elementi del primo vettore a due alla volta (parte alta e bassa ) per ottenere l'accelerazione su 16 bit per ogni asse
- Convertire questi valori in interi con segno
- Allineare a destra per avere l'accelerazione in mg.