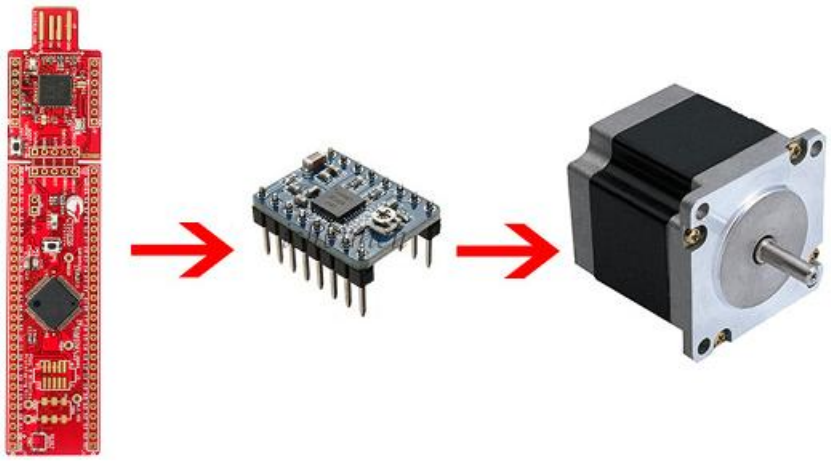
pusillus

# CYPRESS PSOC 4: PRIMI PASSI

13 May 2016

## Serviamoci di un PSoC 4 per pilotare un driver per motori stepper bipolari



*PSoC_A4988.JPG*

Qualche mese fa mi sono imbattuto in questi micro e sono rimasto colpito dalle potenzialità che offrono rispetto ad una normale MCU.

Ho acquistato il CY8CKIT-043 al costo di 15€ compresa spedizione.
Il kit comprende un processore PSoC 4200M (CY8C4247AZI-M485) e un kit di programmazione compresivo di debugger hardware. C'è anche un kit più economico CY8CKIT-049 ma non ha il debugger ed è necessario un bootloader per utilizzarlo.

Per il dettaglio delle caratteristiche di questi micro e un introduzione all'ambiente di sviluppo vi rimando algli esausitivi videocorsi sul sito della Cypress. Consiglio anche di leggere l'articolo di Boiler.

Questo progettino si prefigge di pilotare un driver per motori stepper tipo EasyDriver ed in questo caso Pololu A4988. Ho cercato in rete ma non ho trovato nulla di simile per il PSoC. Quindi sono partito da zero. Probabilmente ci sarà qualche soluzione migliore da quella implememtata da me!

Il driver per motori stepper bipolari A4988 (allegromicro.com) necessita essenzialmente di due soli controlli: pin "Step" riceve un impulso e fa muovere il motore di un passo. Pin "Dir" per il senso di rotazione.

Ci sono anche altri controlli come i pin per impostare il microstepping, reset, enable, sleep, ma non saranno utilizzati nel circuito.

Quello che serve per pilotare il driver è sostanzialmente un treno di impulsi che faccia muovere il motore ed un controllo sul pin "dir" per il senso di rotazione.

Il treno di impulsi sarà generato direttamente dai blocchi hardware del PSoC senza interessare il processore. Questo è un bel vantaggio perchè, durante il movimento del motore, il processore può eseguire altri compiti.

In questo articolo verrà realizzato un circuito che, alla pressione dello switch presente sul kit, fara ruotare di un giro il motore in senso orario per poi riportarlo indietro di un altro giro. Ad ogni giro la velocita di rotazione sarà incrementata di 5 giri al minuto.

## REQUISITI HARDWARE:

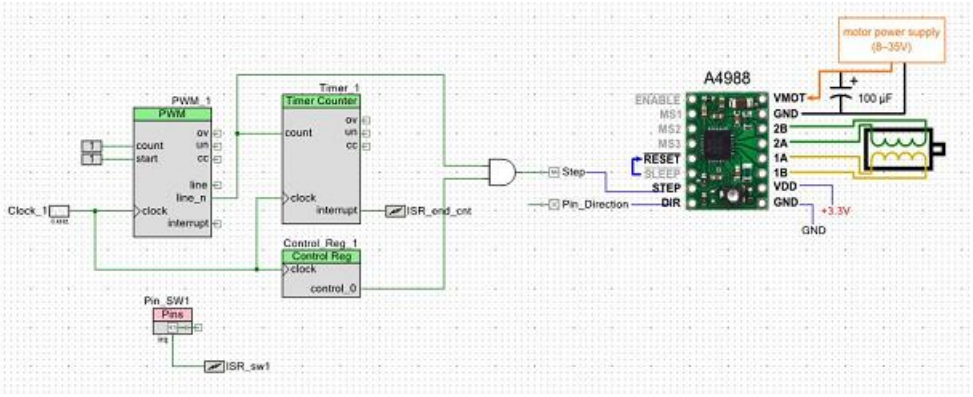
- Bread Board
- Kit CY8CKIT-043 Cypress (E' possibile utilizzare altri kit Cypress, con le opportune modifiche al progetto)
- Stepper driver Pololu A4988, EasyDriver o simili.
- Motore Stepper bipolare
- Alimentatore 12V
- Regolatore 3.3V . Io ne ho utilizzato uno da collegare direttamente al bread board (Vedi video).

**IMPORTANTE:** Il kit Cypress ed anche il Pololu A4988 necessitano di alimentazione regolata da 3.3V a 5V.

Lo schema è stato realizzato con L'ide della Cypress: PSoC Creator. Oltre alla rappresentazione grafica, il circuito verrà realmente realizzato, dal tool, all'interno del chip. Come per un PLD. Sul sito della Cypress ci sono parecchi videotutorial riguardanti l'IDE e l'utilizzo/configurazione dei vari componenti.

Per il Pin "VMOT" del A4988 utlizziamo direttamente la 12V dell'alimentatore. non ci dimentichiamo del condensatore da 100uF visibile nello schema.

Per alimetare il kit, una volta programmato, si deve utilizzare il pin VDD.




*schema_1.JPG*

## BLOCCHI HARDWARE:

La frequenza di Clock_1 viene divisa da PWM_1 in funzione della velocità di rotazione impostata. Sul terminale "line_n" avremo un onda quadra con la frequenza desiderata con deauty cycle al 50%.

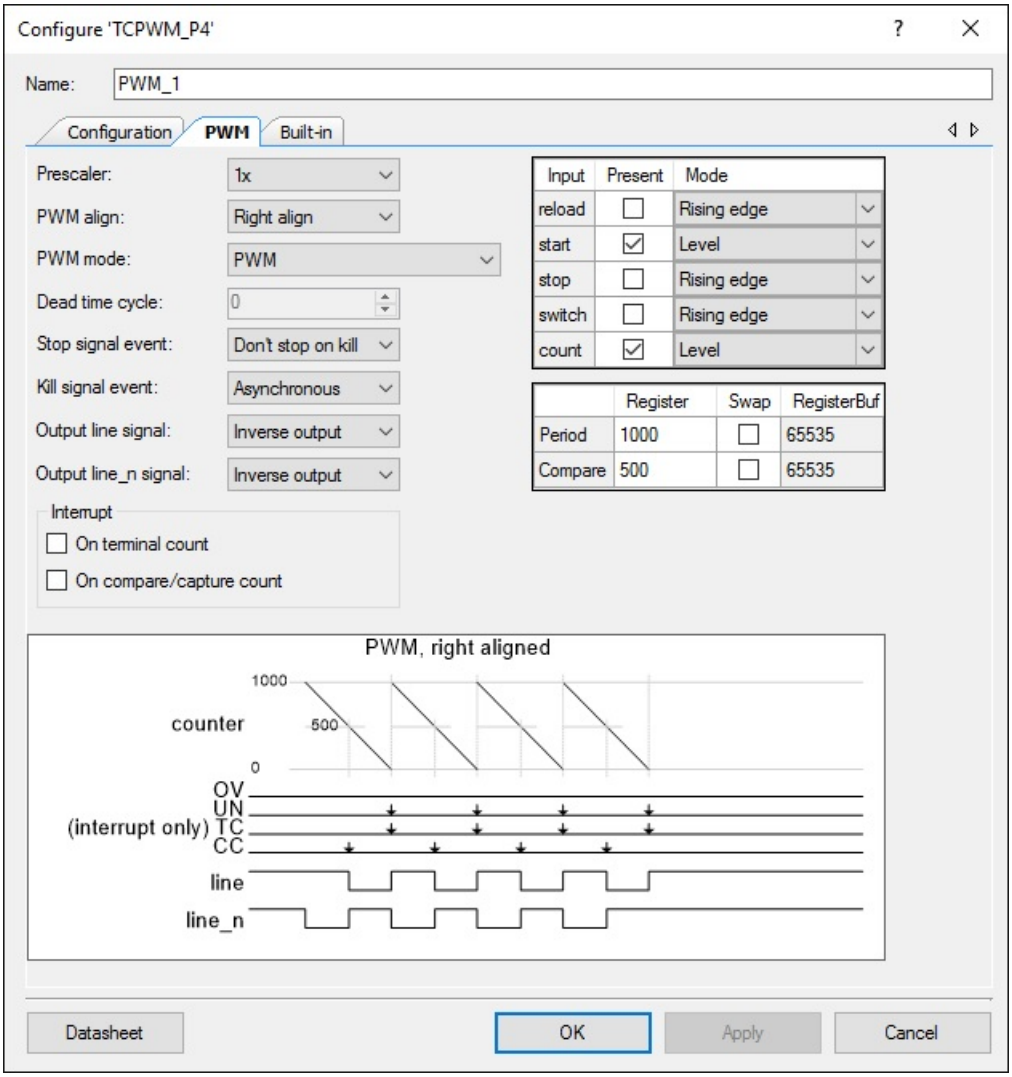
TIMER_1 si occupa di contare il nr di impulsi in uscita dal pwm e di generare un interrupt (ISR_end_cnt) raggiunto il numero impostato.

Control_Reg_1 si occupa di abilitare o no, tramite la porta AND, gli impulsi generati dal PWM sull pin “Step” quindi verso il driver A4988.

Clock_1 è collegato a tutti i componenti del circuito per la loro sincronizzazione.

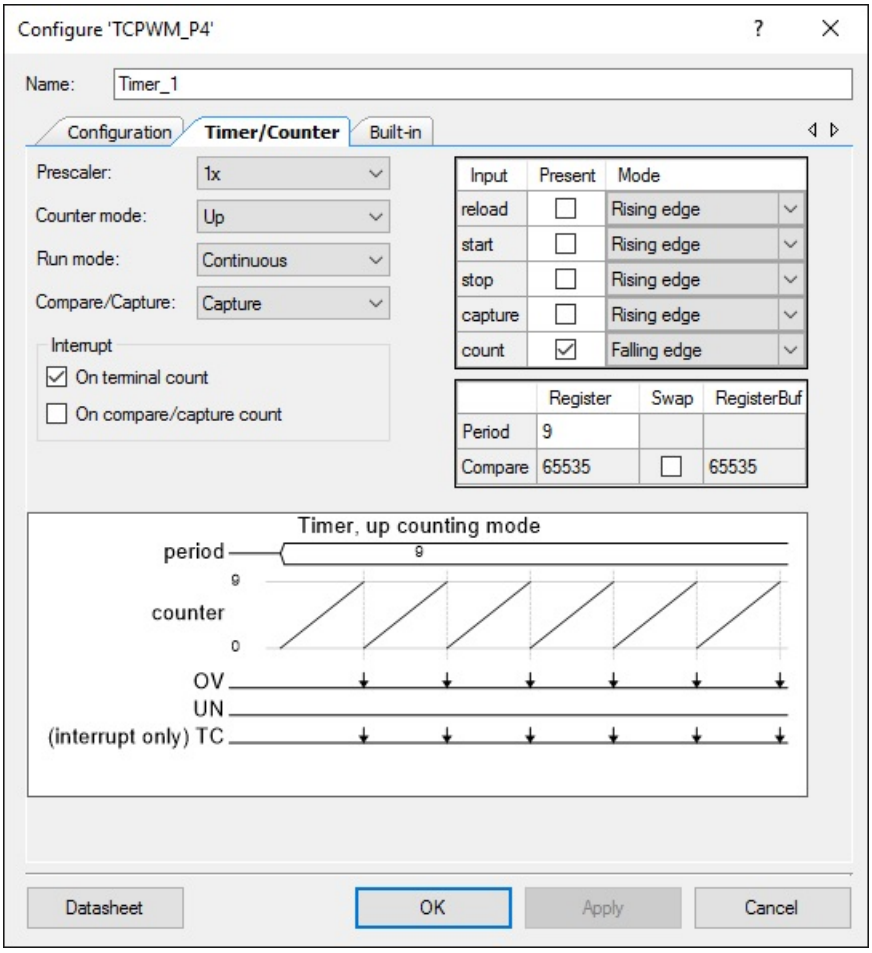
Il tasto Pin_SW1 tramite un altro interrupt (ISR_sw1) fa partire il motore.

## CONFIGURAZIONE DEI COMPONENTI:



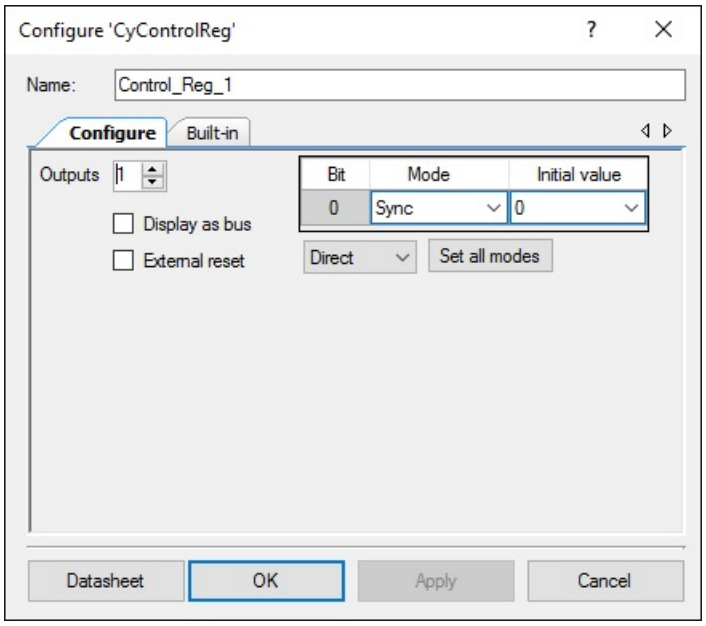

*PWM_1.jpg*

I valori di "period" e "compare" sono ininfluenti perchè verranno modificati dal firmware.



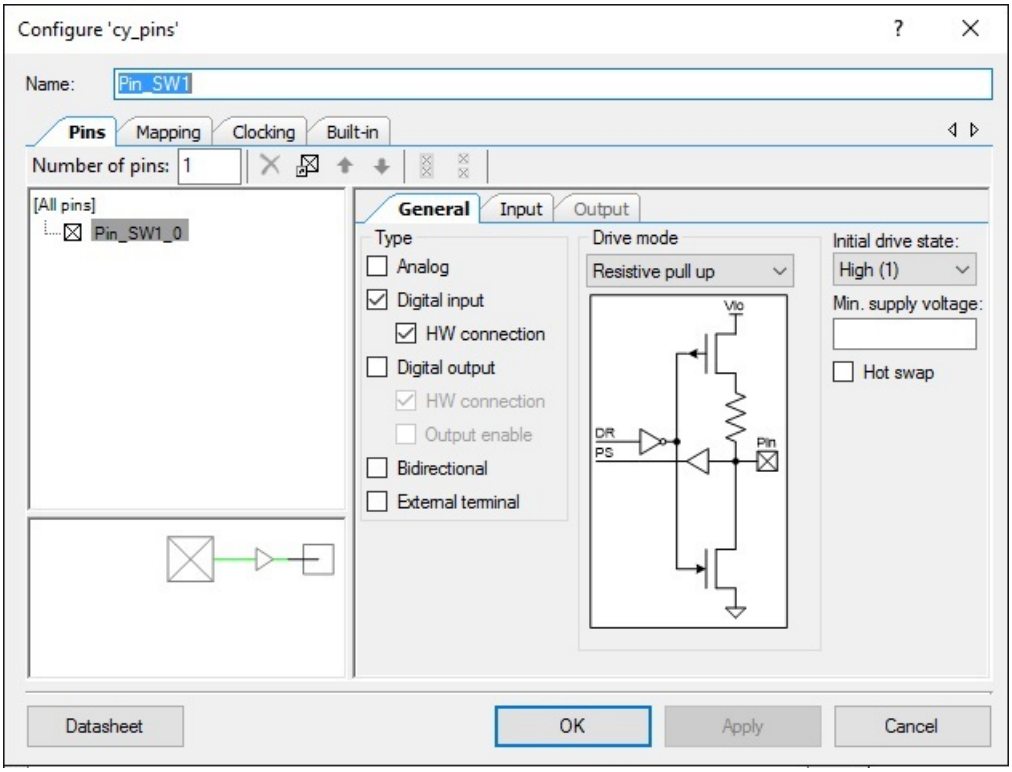

*TIMER_1.jpg*

Anche qui il valore di "Period" verrà modificato dal firmware.



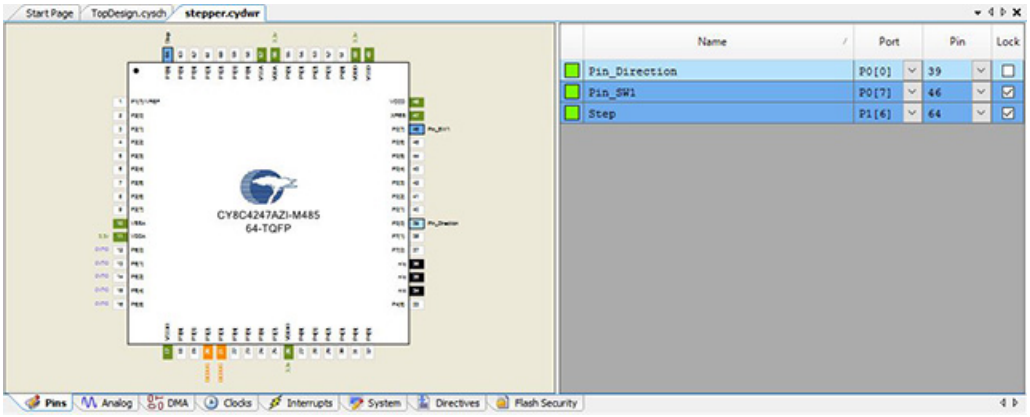

*Control_reg.jpg*

Per questo progetto è necessario configurare un solo output.




*Pin_SW1.jpg*

Importante Impostare il resistive pull up ed gli altri valori come in figura.




*pins.JPG*

Con il tool per la gestione dei pin si assegna a Pin_SW1 la porta P0.7 che corrisponde allo switch presente sul kit. Al pin “Step” ho vuluto assegnare la porta P1.6 che corrisponde al led del kit in modo che possa fungere da monitor. Il Pin_Direction è stato assegnato automaticamente dal tool alla porta P0.0 e non l’ho modificato.

Per gli altri componenti non ci sono particolarità di rilievo. Comunque in allegato c'è il file di progetto.

## IL FIRMWARE:

Nel corpo "main" vengono associate le routine di interrupt ed inizializzati i componenti. All'interno del ciclo for infinito viene fatto ruotare il motore per due volte nei due sensi di rotazione. Da notare che l'interrupt per il pulsante SW1 viene disabilitato durante la rotazione e riabilitato alla fine dei due giri. Questo si rende necessario perchè la rotazione del motore è indipendente dalla CPU. sarà l'interrupt generato dal contatore Timer_1 ad informare il firmware che il motore si è fermato, con il flag "IsRunning".

```
int main()
{
    CyGlobalIntEnable;                  /* Enable global interrupts. */
    ISR_end_cnt_StartEx(My_end_cnt_Handler);
    ISR_sw1_StartEx(My_sw1_Handler);

    Clock_1_Start();
    PWM_1_Start();
    Timer_1_Start();

    for (;;)
    {
        if (button == 1)
        {
```

```
            if (turns == 2)
            {
                turns = 0;
                ISR_sw1_Enable();         // riabilita interrupt pressione tasto
                button = 0;
            }
            else if (isRunning == 0)
            {
                CyDelay(500);
                ISR_sw1_Disable();         // disabilita interrupt pressione tasto
                runStepper(199, direction, speed);
                direction = (direction == 0) ? 1 : 0;
                turns += 1;
                speed += 5;                 //incremento di 5 rpm la velocità
            }
        }
    }
}
```

Come accennato l'interrupt del Timer_1 (My_end_cnt_Handler) abbassa il flag IsRunning e disabilita il pin "Step", tramite il Control register.

```
CY_ISR_PROTO(My_end_cnt_Handler);
CY_ISR(My_end_cnt_Handler)
{
    // routine di gestione interrupt fine treno impulsi
    Control_Reg_1_Write(0);     // disabilita il pin 'step' tramite la porta and
    isRunning = 0;
    Timer_1_ClearInterrupt(Timer_1_INTR_MASK_TC);
};
```

L'interrupt generato dal tasto SW1 (My_sw1_Handler) non fa altro che alzare il flag "button", in modo che il programma possa gestire l'evento. E' infatti buona regola far eseguire all'interno delle routine di interrupt funzioni poco complesse.

```
CY_ISR_PROTO(My_sw1_Handler);
CY_ISR(My_sw1_Handler)
{
    button = 1;
    Pin_SW1_ClearInterrupt();
};
```

La funzione "RunStepper" si occupa di impostare i giusti parametri per il PWM ed il Timer. Imposta il senso di rotazione. Abilita il pin "Step" per mezzo del control register ed alza il flag “isRunning”.

```
void runStepper(uint16 numberOfSteps, uint8 dir, uint16 motorSpeed)
{
    isRunning = 1;
    uint16 tempvar = PWM_Period(motorSpeed);
    Pin_Direction_Write(dir);
    Control_Reg_1_Write(1);       // porta and: abilitazione del pin 'step'
    PWM_1_WritePeriod(tempvar);   // questa funzione deve essere usata dopo lo start
                                  // altrimenti'period' si imposta al valore immesso co
                                  // tool di configurazione
    PWM_1_WriteCompare(tempvar / 2);    // deauty cycle al 50%
    PWM_1_WriteCounter(0);              // necessario resettare a 0 il contatore
    Timer_1_WritePeriod(numberOfSteps); // imposta nr di impulsi sul timer
    Timer_1_WriteCounter(0);            // necessario resettare a 0 il contatore: vedi
                                        // Cypress KBA88172
}
```

Il calcolo del periodo con il quale impostare il PWM è demandato alla funzione PWM_Period. Con clock è impostato a 6khz, il periodo più alto per l'onda quadra generata dal PWM sarà uguale a 1/6k x 65535 = 10.9secondi, mentre il massimo valore sarà di 3khz. Sarebbe possibile aumentare il range lavorando anche con il prescaler del PWM e variando la frequenza del clock. Non ho implementato questa possibiltà per semplificare il codice.

```
uint16 PWM_Period (float rpm)
{
    /*  calcolo del periodo con il quale il PWM divide
        la frequenza del clock in funzione dei giri al minuto
        che si richiedono allo stepper */
    uint16 period;
    period = (MAX_FREQ /  ((rpm * MOTOR_STEPS) / 60) );
    if (period > 65535) period = 65535;  //massimo valore di divisione del PWM
    else if (period < 2) period = 2;     //minimo valore di divisone del PWM
    return period;
};
```

Ho inserito diversi commenti nel codice per facilitarne la comprensione.

## DIFFICOLTA' INCONTRATE:

Premetto che sono un hobbista. Ho un diploma di elettronica industriale. Non sono a digiuno di processori e programmazione ma non mi occupo di questa roba tutti i giorni. I problemi incontrati sono quelli che possono capitare ad un principiante.

I blocchi hardware vanno configurati a dovere, basta un piccolo errore e si ottengono dei comportamenti anomali del sistema. Ad esempio, avevo dimenticato di impostare il resistore di pull up sullo switch ed il circuito sembrava impazzito. Bastava che sfiorassi il pulsante per innescare diverse chiamate di interrupt. Ogni componente ha il suo datasheet. Dimentichiamoci di utilizzarli senza studiarseli.

I blocchi hardware eseguono le loro funzioni in maniera indipendente. Bisogna far eseguire al firmware gli opportuni controlli, tramite le API dei componenti. Oppure, anche in questo caso, il sistema si comporterà in maniera anomala.

Ho utilizzato poco il debugger: Avevo dimenticato di dichiarare "volatile" la variabile "button" ed anche in questo caso il sistema era instabile.

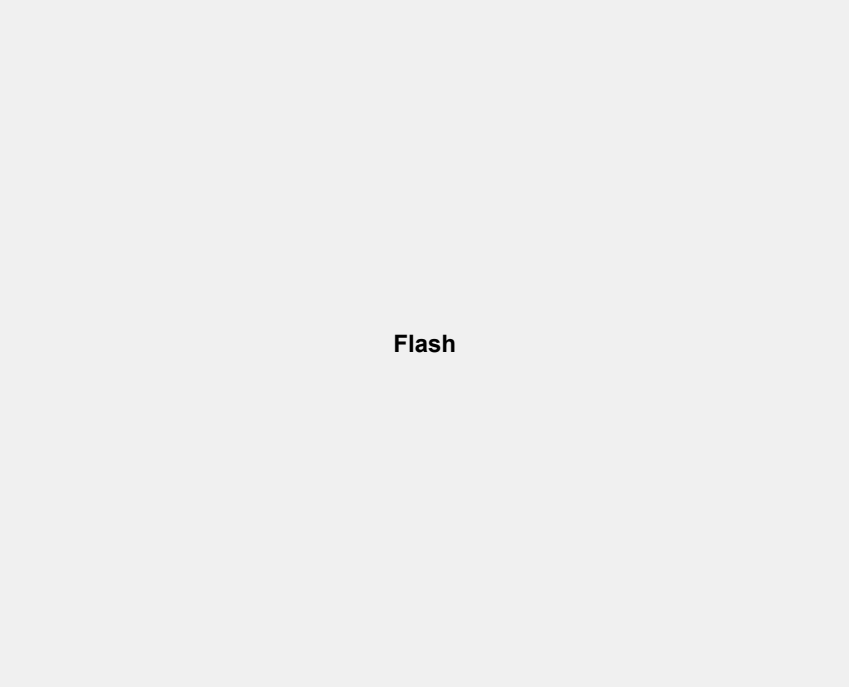
## CONCLUSIONI:

Questo PSoC è davvero pieno di potenzialità. Oltre ai blocchi digitali dispone di blocchi analogici. E' possibile creare i propri componenti, oltre che utilizzare quelli preconfezionati, servendosi dei blocchi (UDB). Addirittura si può implementare l'hardware in verilog . Il Creator è gratuito e senza limiti per tutti.

Ma per iniziare a lavorarci non basta installare il software e smanettarci. Ho seguito le lezioni sui video disponibili sul sito Cypress e poi ho iniziato a fare i primi esperimenti. Ci sono molti driver per diverse periferiche inclusi nel Creator ma non sono numerosi come quelli per Arduino. Ho trovato il forum della Cypress parecchio confusionario e non ci sono molte altre risorse in rete. Ho scovato il forum psocdeveloper.com che sembra organizzato meglio ma pare abbandonato da mesi. Dimentichiamoci di trovare info in italiano.

Spero, con questo mio piccolo contributo, di avervi incuriosito ad una alternativa ad Arduino non poi così difficoltosa anche per un hobbista. Se ci sto riuscendo io, credo possa essere alla portata di molti iniziarci a lavorare.

Files di progetto: stepper_drv.cywrk.Archive01.zip




Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Pusillus:cypress-psoc-4-primi-passi"