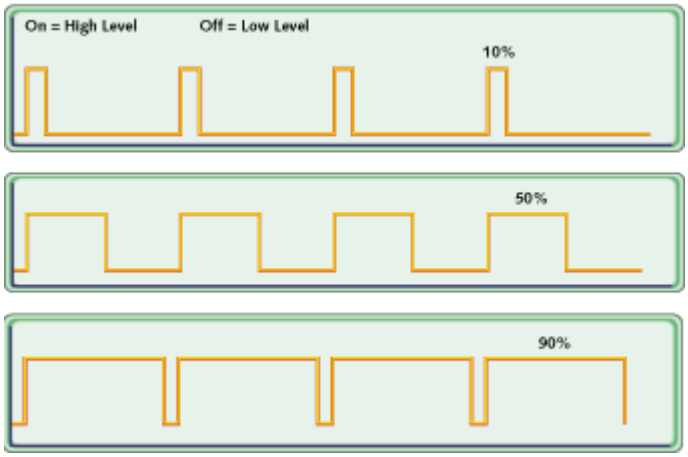
PWM ed oltre
ovvero prendiamo confidenza con il PWM di Arduino Duemilanove

La modulazione di larghezza di impulso (Pulse Width Modulation) può essere usata da Arduino in diversi modi. In questo tutorial cercherò di spiegare l'uso di PWM "semplici" , così come come utilizzare il PWM ed i registri che lo regolano per un maggiore controllo sul ciclo di lavoro e di frequenza.

Un segnale PWM è un'onda quadra (meglio rettangolare) che varia da 0V a VCC (tensione di uscita massima) , dove la frequenza è costante , e cambia la frazione di tempo in cui il segnale è attivo a VCC (detto ciclo di lavoro utile o dutycycle) può essere variata tra 0 e 100 %.



Il PWM ha diversi utilizzi :

* Variare la luminosità di un led LED (tramite un filtro rc), fornendo una tensione analogica tra lo 0% e il 100%.
* Generare di frequenze udibili e quindi audio
* Controllare la velocità dei motori
* Controllare servomotori da modellismo
*Puo essere usato come generatore di segnale modulante
*Ed altro ancora

Semplice PWM con l'uso del comando analogWrite
Il linguaggio di programmazione Arduino, rende il PWM facile da usare , è sufficiente chiamare analogWrite(Pin , dutyCycle), Dove dutyCycle è un valore da 0 a 255, e Pin è uno dei pin PWM (3, 5 , 6, 9 , 10, o 11). Il comando analogWrite fornisce una semplice interfaccia per il PWM hardware, ma non fornisce alcun controllo sulla frequenza.

Ma andiamo un pò più in profondità... ci sono altre opzioni che forniscono più flessibilità :)

Il PWM manuale su tutti i pin
È possibile creare "manualmente" un PWM su ogni pin semplicemente "spegnendo" e "accendendo" un pin in uscita. ad esempio:

```
void setup ()
{
  pinMode (13, OUTPUT) ;
}

void loop ()
{
  digitalWrite (13 , HIGH) ;   / / Lascio il pin "acceso" (per 100ms)
  delayMicroseconds (100); / / Circa il 10 %  di duty cycle (sarebbe 1KHz)
  digitalWrite (13, LOW);     / / Spengo il pin ed aspetto il prossimo ciclo
  delayMicroseconds (900) ;/ / Attesa prima del prossimo ciclo (per 900ms)
```

}

Facendo così, si ha il vantaggio di poter utilizzare qualsiasi pin di uscita digitale . Inoltre, avete il pieno controllo del duty cycle e della frequenza. Uno dei principali svantaggi è che ogni interrupt inciderà negativamente sui tempi , a meno che non si disattivino gli interrupt. Un altro svantaggio è che non si può lasciare girare il processo mentre il processore fa qualcos'altro. Infine , è difficile determinare le costanti appropriate per un ciclo di funzionamento particolare e la frequenza richiesta, a meno che non si calcoli attentamente dai cicli di clock.

Uso del PWM parlando con i registri dell'atmega di Arduino

Il chip ATmega168P/328P ha tre timer PWM , che controllano le 6 uscite PWM. Manipolando il timer del chip direttamente tramite i registri , è possibile ottenere un maggiore controllo rispetto al analogWrite per "piegarlo" alle nostre necessità.

Il datasheet dell'ATmega328P fornisce una descrizione dettagliata dei timer PWM, ma ati può essere difficile da capire, a causa delle molteplici modalità di uso dei timer.
Proviamo a capirci qualcosa. (datasheet alla mano ;))

Il ATmega328P ha tre timer conosciuti come Timer 0, Timer 1 e Timer 2 . Ogni timer ha due output con dei registri a comparazione che controllano la larghezza PWM per il timer di due uscite : quando il timer raggiunge il valore di comparazione impostato nel registro , l'uscita corrispondente è attivata . Le due uscite per ogni timer normalmente hanno la stessa frequenza , ma possono avere cicli di lavoro differenti (a seconda dei rispettivi valori dei registri a comparazione).

Ciascuno dei timer ha un prescaler che genera il suo clock dividendo il clock di sistema per un fattore di Prescale come 1, 8, 64 , 256 o 1024. Arduino duemilanove ha un clock di sistema di 16MHz e la frequenza di clock del timer sarà la frequenza di clock di sistema diviso per il fattore Prescale . Attenzione il Timer 2 ha un diverso insieme di valori Prescale rispetto agli altri timer .

I timers sono complicati da diverse modalità di uso e controllo. Le modalità PWM principali sono " PWM Fast "e" PWM a correzzione di fase ". Il timer può essere pilotato con valori 0-255 (8 bit) , o da un valore fisso, se serve. (Il Timer 1 è dotato di modalità supplementari per supportare valori fino a 16 bit.) Ogni uscita può anche essere invertita.

Il timer può anche generare interrupt con overflow , ma questo và oltre le mie possibilità esplicative (per adesso).

Timers e Registri
Per controllare ogni timer vengono utilizzati vari registri. I Timer / Counter dei Registri di controllo TCCRnA e TCCRnB contengono il bit di controllo principale per i timers.Questi registri possono contenere diversi gruppi di bit :

* Bit di Generazione di forme d'onda (WGM): questa e la modalità di controllo globale del timer. (Questi bit sono divisi tra TCCRnA e TCCRnB .)
* bit Clock Select (CS): questo serve a controllare il prescaler sul clock
* bit di comparazione uscita A (COMnA): questo serve ad attivare / disattivare / l'output invertito per l'uscita A
* bit di comparazione uscita B (COMnB): questo serve ad attivare / disattivare / l'output invertito l'uscita B

Le uscite di comparazione dei Registri OCRnA e OCRnB servono ad impostare i livelli a cui le uscite A e B saranno "azionate". Quando il valore del timer corrisponde al valore di registro , l'uscita corrispondente viene modificata come specificato dalla modalità.

I bit sono leggermente diversi per ogni timer , (vedi datasheet per i dettagli).Come dicevamo prima, il Timer 1 è un timer a 16 bit e dispone di modalità aggiuntive e Timer 2 ha valori diversi di prescaler .

Fast PWM
Nel "semplice" PWM , il timer conta ripetutamente 0-255 . L'uscita si accende quando il timer

è a 0, e si spegne quando il timer corrisponde a quanto impostato nell'apposito registro. Più alto è il valore nel registro di uscita , maggiore è il dutycycle. Questa modalità è nota come Fast PWM Mode.
L'esempio seguente illustra come usare le uscite per due valori particolari di OCRnA e OCRnB . Da notare che entrambe le uscite hanno la stessa frequenza, ed usano la frequenza di un ciclo completo di clock .

Fast PWM Mode

L'esempio di codice seguente imposta il PWM sui pin 3 e 11, (Timer 2). Per riassumere le impostazioni di registro , Il bit di generazzione di forma d'onda WGM lo impostiamo a 011 (selezione fast PWM mode). Impostiamo i bit COM2A e COM2B a 10 (PWM non invertito per le uscite A e B). l'impostazione del bit CS la mettiamo a 100 (quindi dividiamo il clock di sistema per 64). (Poiché i bit sono diversi per i diversi timer , consultare il datasheet .)
Le uscite dei registri di comparazione sono impostate arbitrariamente a 180 e 50 per il controllo del dutycycle del PWM per le uscite A e B. (Naturalmente, si può anche modificare i registri direttamente invece di utilizzare pinMode, ma è necessario impostare il pin di uscita.)

```
pinMode (3, OUTPUT) ;
pinMode (11, OUTPUT) ;
TCCR2A = _BV ( COM2A1 ) | _BV ( COM2B1 ) | _BV ( WGM21 ) | _BV ( WGM20 );
TCCR2B = _BV ( CS22 );
OCR2A = 180 ;
OCR2B = 50;
```

Su Arduino Duemilanove , abbiamo questi valori :

* Frequenza dell'uscita A: 16 MHz / 64 / 256 = 976.5625Hz
* Uscita A dutycycle: (180 +1) / 256 = 70,7%
* Frequenza dell'uscita B: 16 MHz / 64 / 256 = 976.5625Hz
* Uscita B dutycycle: (50 +1) / 256 = 19,9%

La frequenza di 16MHz (frequenza del clock di sistema), diviso per il valore prescaler (64) , diviso per i 256 cicli necessari per un ciclo del timer.

PWM a correzione di fase

La seconda modalità di PWM è chiamata phase-correct PWM. In questa modalità , il timer conta da 0 a 255 e poi di nuovo giù a 0. L'uscita si spegne, appena il timer raggiunge il valore del registro di comparazione di uscita durante la salita , e si gira nuovamente appena il timer raggiunge il valore del registro di comparazione di uscita durante la discesa . Il risultato è un uscita più simmetrica . La frequenza di uscita sarà pari a circa la metà del valore per la modalità fast PWM, perché avverrà allo scadere del tempo , sia in su che in giù (un pò dome le memorie DDR dei pc).
L'esempio seguente imposta il PWM a correzione di fase sui pin 3 e 11, (Timer 2). I bit di forma d'onda WGM sono impostati su a 001 (fase -correct PWM).
Gli altri bit sono uguali allèesempio del fast PWM .

```
pinMode (3, OUTPUT) ;
pinMode (11, OUTPUT) ;
TCCR2A = _BV ( COM2A1 ) | _BV ( COM2B1 ) | _BV ( WGM20 );
TCCR2B = _BV ( CS22 );
OCR2A = 180 ;
OCR2B = 50;
```

Su Arduino Duemilanove , abbiamo questi valori:

* Frequenza uscita A : 16 MHz / 64 / 255 / 2 = 490.196Hz
* Dutycycle uscita A: 180/255 = 70,6%
* Frequenza uscita B : 16 MHz / 64 / 255 / 2 = 490.196Hz
* Dutycycle uscita B : 50 / 255 = 19,6%

Come vedete dai valori, il PWM phase correct , divide la frequenza in due rispetto al fast PWM, perché il timer scatta sia a monte che a valle . Sorpresa delle sorprese, la frequenza è divisa per 255 invece di 256, e ai calcoli del Ciclo non aggiunge uno come per il fastPWM.

Oltre il limite : fast PWM
Entrambi le modalità di PWM dispongono di un modo aggiuntivo che permette di controllare la frequenza di uscita . In questa modalità , il timer conta da 0 a OCRA, invece che da 0 a 255. Questo dà un controllo sulla frequenza di uscita più preciso rispetto alle modalità precedenti. (Per un controllo di frequenza ancora più preciso, usa il timer 1 a 16 bit).

Si noti però,che in questa modalità, solo uscita B può essere utilizzata per il PWM ; OCRA non può essere utilizzato in due modi contemporaneamente. Tuttavia, vi è una modalità speciale per caso "Toggle Ocna on Match "che cambia lo stato di uscita A alla fine di ogni ciclo , generando un duty cycle fisso del 50% e la frequenza di un mezzo in questo caso.

Nel diagramma seguente, il timer viene azzerato quando cicla OCRnA , producendo una frequenza di uscita più veloce per OCnB che negli esempi precedenti. Nota come Ocna cambia una volta per ogni reset del timer.
Fast PWM con cimatura di OCRA.
Il codice seguente imposta PWM fast sui pin 3 e 11, (Timer 2), utilizzando OCR2A come valore di limite superiore del timer. Il bit di forma d'onda e generazione WGM sono impostati su 111 per PWM fast ma ora OCRA ha il controllo sul limite superiore. Il limite superiore di OCR2A lo fissiamo arbitrariamente a 180 , e OCR2B lo fissiamo arbitrariamente a 50. OCR2A adesso e in modalità " Attiva quando corrisponde", imposiamo il bit COM2A a 01 .

```
pinMode (3, OUTPUT) ;
pinMode (11, OUTPUT) ;
TCCR2A = _BV ( COM2A0 ) | _BV ( COM2B1 ) | _BV ( WGM21 ) | _BV ( WGM20 );
TCCR2B = _BV ( WGM22 ) | _BV ( CS22 );
OCR2A = 180 ;
OCR2B = 50;
```

Sul Arduino Duemilanove , abbiamo questi valori :

* Frequenza di uscita A: 16 MHz / 64 / (180 +1) / 2 = 690.6Hz
* Dutycycle uscita A : 50%
* Frequenza di uscita B : 16 MHz / 64 / (180 +1) = 1381.2Hz
* Dutycycle uscita B: (50 +1) / (180 +1) = 28,2%

Si noti che in questo esempio, il timer va da 0 a 180, che però prende 181 cicli di clock , per cui la frequenza di uscita è divisa per 181. L'uscita A ha la metà della frequenza dell'uscita B perché la modalità di confronto cambia l'uscita A una volta per ogni ciclo completo del timer .

Oltre ogni limite : ma per il phase-correct PWM
Allo stesso modo , il timer può essere configurato per il phase -correct PWM e ciclare quando raggiunge OCRnA .
L'esempio seguente imposta PWM phase correct sui pin 3 e 11, (con il Timer 2), utilizzando OCR2A come valore superiore per il timer. I bit del WGM sono impostati su a 101 per PWM phase correct con il controllo del limite superiore. Il limite superiore OCR2A è fissato arbitrariamente a 180 , e la OCR2B (registro di confronto) è fissato arbitrariamente a 50. OCR2A modalità è impostato su " Attiva su corrispondenza al confronto", impostando i bit COM2A a 01 .

```
pinMode (3, OUTPUT) ;
pinMode (11, OUTPUT) ;
TCCR2A = _BV ( COM2A0 ) | _BV ( COM2B1 ) | _BV ( WGM20 );
TCCR2B = _BV ( WGM22 ) | _BV ( CS22 );
OCR2A = 180 ;
OCR2B = 50;
```

Sul Arduino Duemilanove , abbiamo questi valori :

* Frequenza dell'uscita A: 16 MHz / 64 / 180 / 2 / 2 = 347.2Hz
* Dutycycle uscita A : 50%
* Frequenza dell'uscita B: 16 MHz / 64 / 180 / 2 = 694.4Hz
* Dutycycle uscita B: 50 / 180 = 27,8%

Si noti che in questo esempio, il timer va da 0 a 180 e di nuovo a 0 , il che prende 360 cicli di clock. Così, tutto è diviso per 180 o 360, a differenza del caso PWM fast, che invece ha diviso tutto con per 181 (vedi sotto per i dettagli).
L'errore "Off- by-one".
Avrete notato che PWM fast e PWM phase correct sembrano essere off- by-one l'uno rispetto all'altro , dividendo per 256 invece che per 255 e aggiungendo uno in vari luoghi. Supponiamo che il timer sia impostato in modalità PWM fast è impostata a contare fino a un valore di OCRnA di 3. Il timer assumerà i valori 012301230123 ... Si noti che ci sono 4 cicli di clock per ogni ciclo del timer. Così, la frequenza sarà divisa per 4 e non 3. Il dutycycle sarà un multiplo di 25%, dato che il segnale sara "alto" a 0, 1, 2 , 3 o 4 . Allo stesso modo, se il timer conta fino a 255 , ci saranno 256 cicli di clock in ogni ciclo del timer , e il dutycycle sarà multiplo di 1 / 256 . Quindi, PWM fast divide per N +1 , dove N è il valore del timer massimo (sia OCRnA o 255).
Ora consideriamo la modalità PWM phase correct con il timer conta fino ad un valore di OCRnA 3. I valori di timer sarà 012321012321 ... Ci sono 6 cicli di clock per ogni ciclo del timer (012.321). Così la frequenza sarà diviso per 6. Il ciclo di lavoro sarà un multiplo di 33%, dato che il segnale sarà "alto" per 0, 2 , 4 o 6 dei 6 cicli . Allo stesso modo, se il timer conta fino a 255 e ritorno verso il basso , ci saranno 510 cicli di clock in ogni ciclo del timer , e il dutycycle sarà un multiplo di 1 / 255 . Quindi, PWM phase correct divide per 2N , dove N è il valore del timer massimo.
La seconda differenza importante è che il PWM fast tiene l'output ad alto per un ciclo più lungo del valore del registro di comparazione. La motivazione di ciò è che PWM fast conta fino a 255 , il dutycycle può essere 0-256 cicli, ma il registro può contenere solo un valore da 0 a 255. Cosa succede al valore mancante ? La modalità PWM fast mantiene l' alta l'uscita per cicli di N +1 quando il registro di comparazione è impostato su N così un registro con valore di 255 è 100 % del dutycycle, ma un registro con valore 0 è non dutycycle 0 %, ma 1 / 256 del dutycycle. Questo a differenza del PWM phase correct, in cui un valore di registro di 255 è 100 % del dutycycle e un valore di 0 è un dutycycle dello 0%.

Timer e Arduino

Il Arduino supporta PWM su alcuni dei pin di uscita. Può non essere immediatamente evidente che c'è in timer che ne controlla l'utilizzo , ma la tabella seguente chiarirà la situazione. Essa fornisce, per ogni uscita del temporizzatore, il pin di uscita sul Arduino (cioè l'etichetta serigrafata sulla scheda) , il pin sul chip ATmega , e il nome e poco di porta di uscita . Per esempio Timer 0 OC0A uscita è collegata all'uscita Arduino pin 6 , utilizza chip pin 12 che è anche conosciuto come PD6 .

Timer di uscita	Uscita su Arduino	Pin sul chip	Nome pin
OC0A	6	12	PD6
OC0B	5	11	PD5
OC1A	9	15	PB1
OC1B	10	16	PB2
OC2A	11	17	PB3
OC2B	3	5	PD3

Arduino inizializza il prescaler su tutti e tre i timer per dividere il clock per 64. Timer 0 è inizializzato in PWM fast, mentre i Timer 1 e Timer 2 vengono impostati come PWM phase correct .

Il Arduino utilizza il Timer 0 interno per i comandi Millis () e delay () , quindi sappiate che modificando la frequenza di questo timer quei comandi danno segni di pazzia. Usare le uscite PWM, invece, è sicuro se non si cambiamo le frequenze.

Con il comando analogWrite (pin , duty_cycle) si imposta il pin appropriato PWM e l'output appropriato nel registro di comparazione (Con il caso particolare di dutycycle del Timer 0 su 0). Il comndo digitalWrite () disattiva l'uscita PWM se viene chiamato su un pin timer.

Se si utilizza analogWrite (5, 0) si ottiene un dutycycle dello 0% , anche se il pin 5 del timer (Timer 0) utilizza PWM fast. Come è possibile , quando un valore di PWM fast di 0 produce un ciclo di 1 / 256 , come spiegato sopra? La risposta è che analogWrite " bara" , ha un codice per i "casi speciali" per spegnere il pin quando si usa il Timer 0 con un dutycycle pari a 0. Come conseguenza , il ciclo che doveva essere 1 / 256 non è disponibile quando si utilizza analogWrite su Timer0 , e vi è un salto nel ciclo di lavoro reale tra i valori di 0 e 1.

Spero che questo piccolo tutorial messo insieme studianodo varii documenti reperibili sulla

rete ed il datasheet dell'atmega 328, sia utile a tutti voi così come è stato utile per mè.
Ciao
Brainbooster