

Corso di Laurea Magistrale in Ingegneria Meccatronica
Microcontrollori e DSP

Controllo velocità di rotazione di una ventola con scheda KL25Z

Matricola	Cognome	Nome
1197747	Braceschi	Marco
1197740	Rigon	Saverio
1205388	Zanrosso	Luca

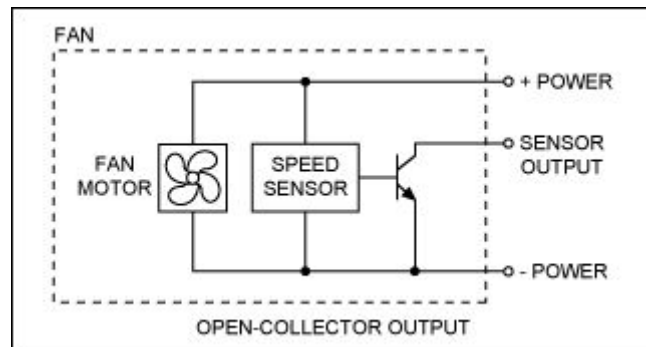
Materiale fornito per l'esercitazione:

- Scheda KL25Z
- Ventola
- Circuito di interfaccia per la ventola
- Adattatore USB per la trasmissione dati tramite UART

Obiettivi del progetto:

- Generazione di un segnale PWM che permetta di regolare l'alimentazione della ventola e perciò la velocità di rotazione.
- Implementazione, mediante timer in modalità «input capture», di un sistema di misura della velocità di rotazione della ventola.
- Interfacciamento del microcontrollore, mediante periferica seriale, con il PC.
- Caratterizzazione statica della ventola.
- Analisi della risposta al gradino della ventola. Sulla base di questo transitorio la ventola dovrà essere modellata come un sistema del primo ordine.
- Implementazione di un filtro a media mobile. Il filtro dovrà essere progettato sulla base del sistema da controllare e del rumore eventualmente osservato sperimentalmente.

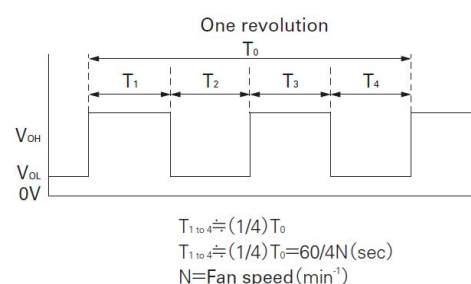
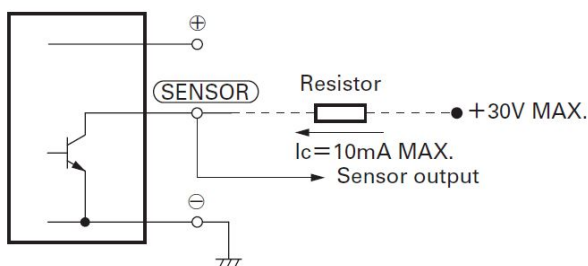
Ventola



La ventola è alimentata in DC. La velocità di rotazione è in funzione della tensione di alimentazione. I dettagli tecnici della ventola sono descritti in tabella.

Model No.	Rated Voltage [V]	Operating Voltage Range [V]	Rated Current [A]	Rated Input [W]	Rated Speed [min ⁻¹]	Max. Airflow [m ³ /min]	Max. Static Pressure [Pa]	SPL [dB(A)]	Operating Temperature [°C]	Expected Life [h]
9S0912L401	12	6.0 to 13.8	0.07	0.084	1750	0.83	13.1	17	-10 to +70	40000

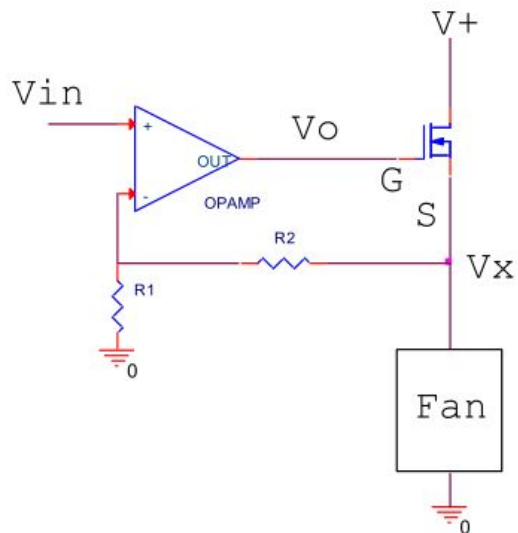
Per la misura di velocità di rotazione è presente un encoder ottico di tipo incrementale. Il segnale digitale generato è proporzionale alla frequenza di rotazione. Il circuito del sensore e del segnale generato sono qui raffigurati.



A seconda della posizione angolare della ventola viene generato un segnale alto o basso. In ogni rotazione vengono generati complessivamente 2 segnali alti e due bassi. Contando quindi l'intervallo temporale trascorso tra il segnale alto i e il segnale alto $i + 2$ è possibile determinare la frequenza di rotazione della ventola.

La tensione massima in uscita dall'encoder deve essere di 3.3V poiché andrà in ingresso ad una porta della scheda. La tensione di alimentazione della ventola deve essere di 12V e la corrente di 70mA. Serve quindi un circuito di amplificazione.

Circuito di interfaccia per la ventola



Considerando il circuito rappresentato sopra valgono le seguenti equazioni:

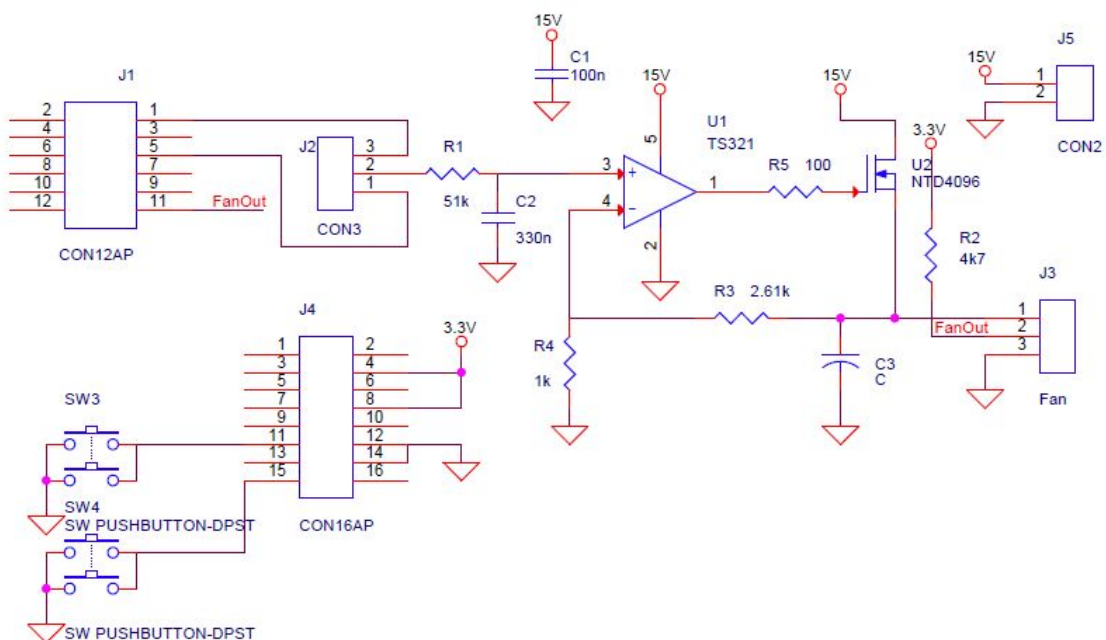
$$V_x = \left(1 + \frac{R_1}{R_2}\right)$$

$$V_o = V_x + V_{GS} > V_x + V_T$$

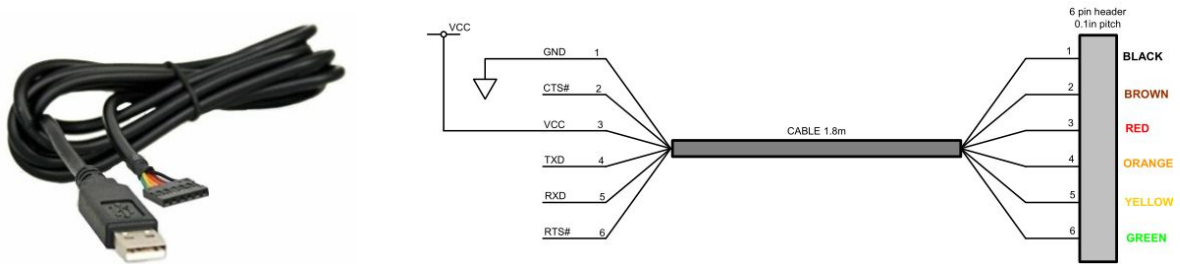
$$V_T = 3V \quad (\text{tensione di soglia del mosfet})$$

$$V_+ = V_{xmax} + V_T = 15V$$

La tensione di alimentazione del circuito deve essere quindi maggiore di 15V. Si riporta per completezza il circuito completo.



Adattatore USB per la trasmissione dati tramite UART



L'adattatore USB (TTL-232R) serve per la trasmissione seriale dei dati dalla scheda al pc e viceversa. I pin interessati sono quelli arancio per la scrittura (da pc alla scheda) e giallo per la lettura (da pc alla scheda) oltre ovviamente al pin nero della massa. Per leggere i dati trasmessi da seriale si utilizza il programma HyperTerminal.

Parte 1

Generazione di un segnale PWM per regolare la velocità della ventola

Per la configurazione del segnale PWM si utilizza il canale TPM1 (Timer/PWM Module). Per prima cosa si abilita il clock su TMP1 e sulla porta E, poiché il segnale deve uscire dalla porta PTE20.

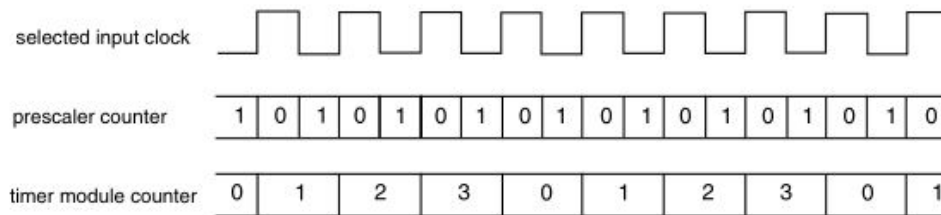
Si modifica inoltre il registro SC in modo da abilitare gli interrupt di TOF (Timer Overflow Flag) e impostare l'incremento del contatore ad ogni clock dell'LPTPM. L'overflow si verifica quando il contatore ha raggiunto lo stesso valore del MOD.

La frequenza del segnale PWM non deve essere troppo bassa altrimenti la tensione che arriva alla ventola (che dovrebbe essere continua) presenterebbe un ripple evidente. La costante di tempo del filtro del circuito di interfaccia è $\tau = R_1 \cdot C_2 = 0.01683s$. La frequenza di taglio perciò è $1/\tau = 59.4Hz$. La frequenza della PWM per essere ben filtrata deve stare almeno a una decade al di sopra della frequenza di taglio. A tal proposito per avere un ripple di corrente inferiore del 2% si è scelta una frequenza superiore ai 1000Hz. Dal momento che vale la relazione

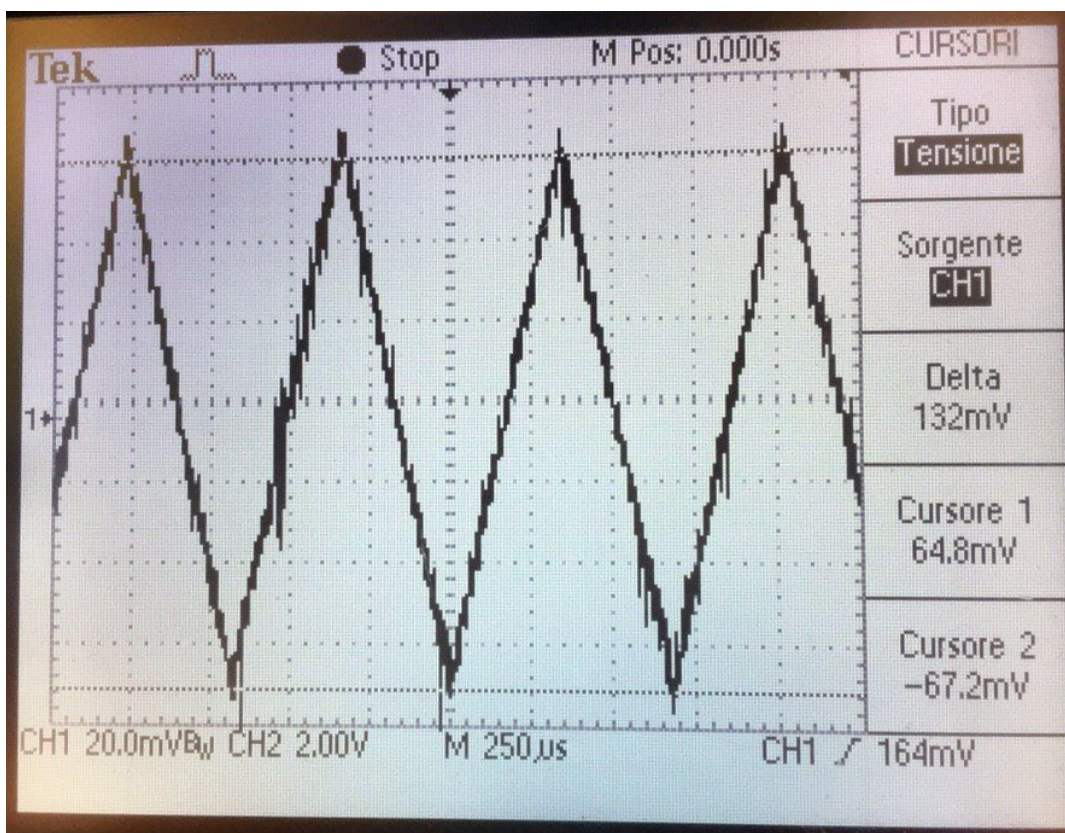
$$f = \frac{f_{clock}}{prescaler \cdot MOD}$$

dove:

- f è la frequenza della PWM
- f_{clock} è la frequenza del microprocessore (48MHz)
- $prescaler$ è un fattore che permette, in questo caso, di diminuire la frequenza della PWM.
- MOD è il valore massimo del contatore (LPTPM counter). Quando il contatore raggiunge il valore massimo viene settato il flag di overflow e il valore successivo del counter dipende dal prescaler. L'immagine è un esempio di un contatore con $prescaler = 2$ e $MOD = 3$.



Per avere una frequenza superiore a 1000Hz si è scelto un prescaler di 1 e un MOD di 32678. La frequenza della PWM diventa quindi di 1465Hz. **Si riporta la verifica all'oscilloscopio.**



Per utilizzare la porta PTE20 bisogna configurare il canale 0 di TMP1. È necessario innanzitutto modificare il registro PCR della porta E scegliendo l'alternativa 3 e in particolare agendo sui bit 8, 9 e 10 dedicati al MUX. Successivamente si modifica il registro CnSC, canale 0, di TMP1 andando a settare i bit MSA e ELSA in modo da generare un segnale PWM edge-aligned.

Infine si inizializza a 0 l'uscita della PWM.

Nelle pagine seguenti vengono riportati gli schemi dei registri menzionati.

TPMx_SC field descriptions

Field	Description
31–9 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
8 DMA	DMA Enable Enables DMA transfers for the overflow flag. 0 Disables DMA transfers. 1 Enables DMA transfers.
7 TOF	Timer Overflow Flag Set by hardware when the LPTPM counter equals the value in the MOD register and increments. The TOF bit is cleared by writing a 1 to TOF bit. Writing a 0 to TOF has no effect. If another LPTPM overflow occurs between the flag setting and the flag clearing, the write operation has no effect; therefore, TOF remains set indicating another overflow has occurred. In this case a TOF interrupt request is not lost due to a delay in clearing the previous TOF.

TPMx_SC field descriptions (continued)

Field	Description
	0 LPTPM counter has not overflowed. 1 LPTPM counter has overflowed.
6 TOIE	Timer Overflow Interrupt Enable Enables LPTPM overflow interrupts. 0 Disable TOF interrupts. Use software polling or DMA request. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.
5 CPWMS	Center-aligned PWM Select Selects CPWM mode. This mode configures the LPTPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 LPTPM counter operates in up counting mode. 1 LPTPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the LPTPM counter clock modes. When disabling the counter, this field remain set until acknowledged in the LPTPM clock domain. 00 LPTPM counter is disabled 01 LPTPM counter increments on every LPTPM counter clock 10 LPTPM counter increments on rising edge of LPTPM_EXTCLK synchronized to the LPTPM counter clock 11 Reserved
2–0 PS	Prescale Factor Selection Selects one of 8 division factors for the clock mode selected by CMOD. This field is write protected. It can be written only when the counter is disabled. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128

80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
3	—	—	—	PTE2	DISABLED		PTE2	SP11_SCK					
4	—	—	—	PTE3	DISABLED		PTE3	SP11_MISO			SP11_MOSI		
5	—	—	—	PTE4	DISABLED		PTE4	SP11_PCS0					
6	—	—	—	PTE5	DISABLED		PTE5						
7	3	1	—	VDD	VDD	VDD							
8	4	2	2	VSS	VSS	VSS							
9	5	3	3	USB0_DP	USB0_DP	USB0_DP							
10	6	4	4	USB0_DM	USB0_DM	USB0_DM							
11	7	5	5	YOUT33	YOUT33	YOUT33							
12	8	6	6	VREGIN	VREGIN	VREGIN							
13	9	7	—	PTE20	ADC0_DP0/ ADC0_SE0	ADC0_DP0/ ADC0_SE0	PTE20		TPM1_CH0	UART0_TX			

PORTx_PCRn field descriptions (continued)

Field	Description
23–20 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
19–16 IRQC	Interrupt Configuration This field is read only for pins that do not support interrupt generation. The pin interrupt configuration is valid in all digital pin muxing modes. The corresponding pin is configured to generate interrupt/DMA request as follows: 0000 Interrupt/DMA request disabled. 0001 DMA request on rising edge. 0010 DMA request on falling edge. 0011 DMA request on either edge. 1000 Interrupt when logic zero. 1001 Interrupt on rising edge. 1010 Interrupt on falling edge. 1011 Interrupt on either edge. 1100 Interrupt when logic one. Others Reserved.
15–11 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).

TPMx_CnSC field descriptions

Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 CHF	Channel Flag Set by hardware when an event occurs on the channel. CHF is cleared by writing a 1 to the CHF bit. Writing a 0 to CHF has no effect. If another event occurs between the CHF sets and the write operation, the write operation has no effect; therefore, CHF remains set indicating another event has occurred. In this case a CHF interrupt request is not lost due to the delay in clearing the previous CHF. 0 No channel event has occurred. 1 A channel event has occurred.
6 CHIE	Channel Interrupt Enable Enables channel interrupts. 0 Disable channel interrupts. 1 Enable channel interrupts.
5 MSB	Channel Mode Select Used for further selections in the channel logic. Its functionality is dependent on the channel mode. When a channel is disabled, this bit will not change state until acknowledged in the LPTPM counter clock domain.
4 MSA	Channel Mode Select Used for further selections in the channel logic. Its functionality is dependent on the channel mode. When a channel is disabled, this bit will not change state until acknowledged in the LPTPM counter clock domain.
3 ELSB	Edge or Level Select The functionality of ELSB and ELSA depends on the channel mode. When a channel is disabled, this bit will not change state until acknowledged in the LPTPM counter clock domain.
2 ELSA	Edge or Level Select The functionality of ELSB and ELSA depends on the channel mode. When a channel is disabled, this bit will not change state until acknowledged in the LPTPM counter clock domain.
1 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
0 DMA	DMA Enable Enables DMA transfers for the channel. 0 Disable DMA transfers. 1 Enable DMA transfers.

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01/10/11	00	Software compare	Pin not used for LPTPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		X1		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		X1		Low-true pulses (set Output on match-up, clear Output on match-down)

Si riporta il codice completo riguardante l'abilitazione e la routine della PWM:

```
void TPM_init(void) {
    // Abilito clock sul TPM1 e TPM0
    SIM->SCGC6 |= (1UL<<SIM_SCGC6_TPM1_SHIFT) |
(1UL<<SIM_SCGC6_TPM0_SHIFT);

    // bit 6: Abilito gli interrupt. Un interrupt è generato
    quando il flag TOF è uguale a 1.
    // bit 3: LPTPM counter increments on every LPTPM counter
    clock
    // Prescaler impostato a 1 e MOD = 2^15 in modo da avere una
    freq di 1460 Hz e quindi ripple di corrente inferiore a 2%.
    TPM1->SC = (1UL<<3) | (1UL<<6);
    TPM1->MOD = 32768;

    // Utilizzo il canale 0 del modulo come uscita PWM
    edge-aligned con funzione di output reset on compare
    TPM1->CONTROLS[0].CnSC = (1UL<<5) | (1UL<<3);

    // Inizializzazione dell'uscita della PWM
    TPM1->CONTROLS[0].CnV = 0;
    // bit 10-8: Pin Mux Control, alternativa 3
    PORTE->PCR[20] = (1UL<<8) | (1UL<<9);
}
```

All'interno del main viene abilitata la routine legata al timer TMP1 con la seguente riga di codice:

```
// Abilito routine TMP1
NVIC_EnableIRQ(TPM1_IRQn);
```

Infine la routine di TMP1 è stata così implementata:

```
int TPM1routine(void) {

    TPM1->SC |= (1UL << 7);

    // Imposto un valore del duty cycle al 50%
    TPM1->CONTROLS[0].CnV = duty;

    // LED toggle routine per vedere visivamente che la PWM stia
    funzionando
    FPTD->PTOR = (1UL<<1);
```

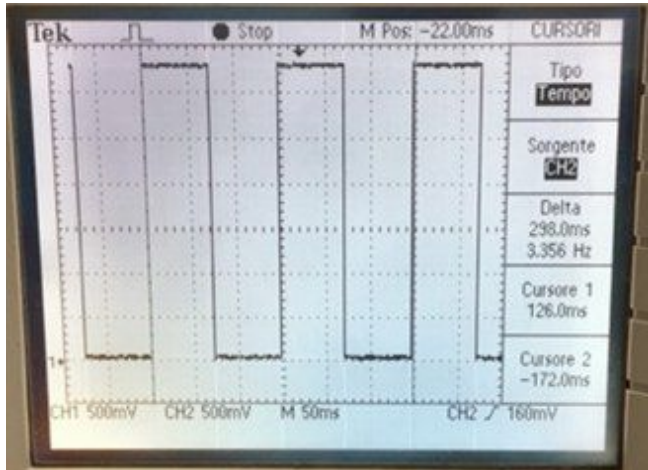
```

FPTB->PTOR = (1UL<<18);
}

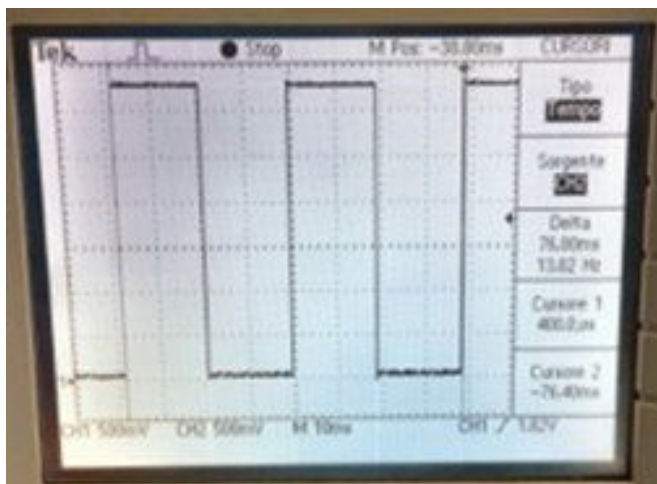
```

Per individuare il range di frequenza in cui opera la ventola si effettua una caratterizzazione statica all'oscilloscopio. Questa analisi servirà per la realizzazione della routine di input capture. Si riportano le immagini del segnale in uscita dall'encoder a diverse tensioni applicate

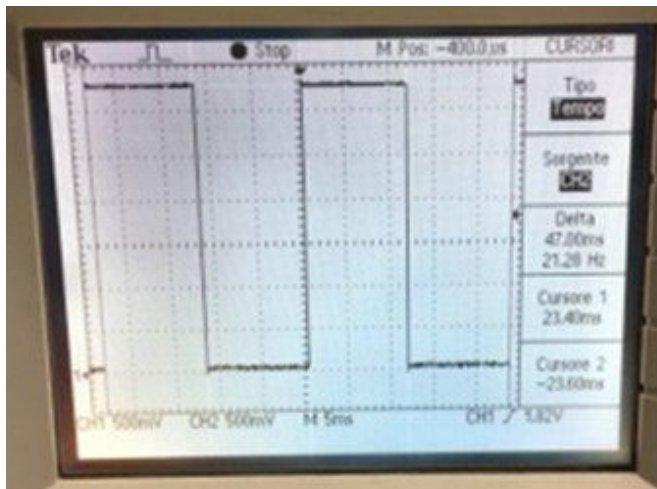
- 3V, periodo 298ms, frequenza 3.36Hz



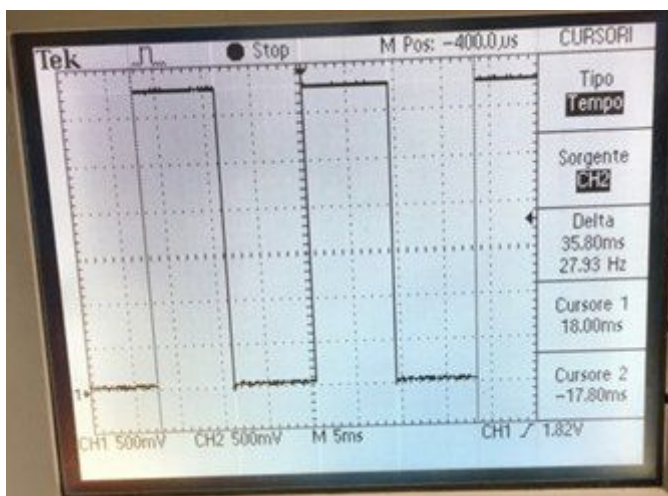
- 6V, periodo 76.8ms, frequenza 13.82Hz



- 9V, periodo 47,00ms, frequenza 21.28Hz



- 12V, periodo 35.80ms, frequenza 27.93Hz



Si nota come per tensioni superiori a 6V la velocità rimanga pressoché proporzionale alla tensione applicata fino a raggiungere quasi la velocità nominale (29,2Hz). Per tensioni poco al di sopra della tensione minima di accensione la velocità presenta delle forti non linearità. Si analizzerà in seguito l'isteresi presente nell'accensione/spegnimento della ventola.

Parte 2

Implementazione di un sistema di misura della velocità di rotazione della ventola

La misura della velocità avviene tramite la modalità input capture. Volendo usare la porta PTE30 e seguendo l'alternativa 3 come fatto in precedenza bisogna utilizzare il timer TMP0, canale 3. Quindi in modo simile a quanto fatto precedentemente si abilitano i clock su TMP0 e sulla porta E e si specifica l'alternativa 3 nella porta PTE30.

La differenza riguarda l'impostazione del timer come input capture e come rappresentare la frequenza in modo da avere la migliore risoluzione possibile sfruttando tutti i byte del microprocessore.

Scegliendo una rappresentazione in 16 bit, in modo da non avere problemi di overflow in caso di moltiplicazione, bisogna capire quanti bit servono per la parte intera e quanti bit si possono usare per la parte decimale, in modo da aumentare il più possibile la precisione della misura. Considerando che dall'analisi della frequenza non viene mai superato il valore di 31Hz allora posso utilizzare 5 bit per la parte intera. Togliendo il bit per il segno allora si possono dedicare 10 bit alla parte decimale.

Si consideri l'uguaglianza:

$$f_{FAN} = \frac{f_{clock}}{(cnt_2 - cnt_0) \cdot prescaler} 2^M$$

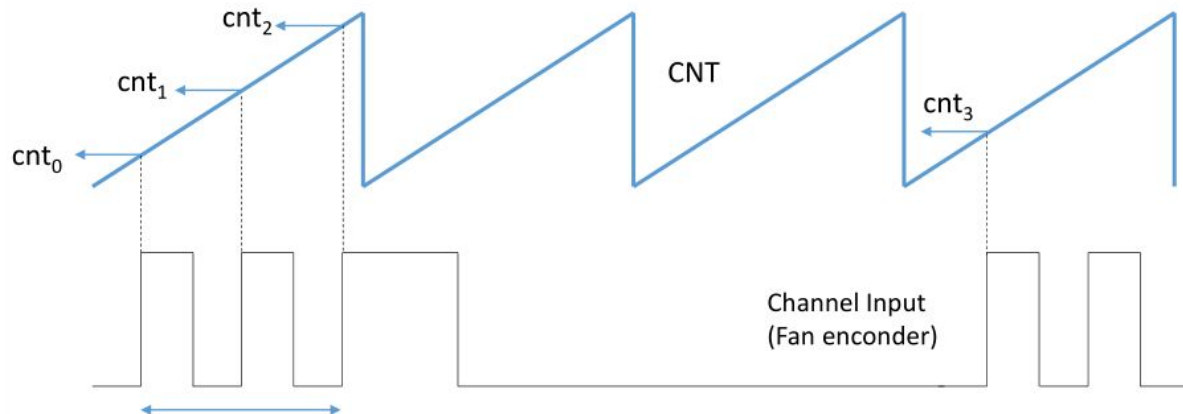
dove:

- f_{FAN}
- cnt_i è il valore del conteggio del timer
- M è il fattore di scala, impostato a 10 per le motivazioni esposte sopra

Escludendo i valori di cnt_i , i rimanenti termini sono costanti, perciò il calcolo viene fatto una sola volta all'inizio. Rimane la scelta del valore del prescaler che dipende appunto dal valore massimo rappresentabile da un int, che è $2^{32} - 1 = 4.294.967.295$. Con una f_{clock} di 48MHz la scelta del prescaler è di 16 in modo che la costante calcolata abbia il valore di 3.072.000.000.

Resta da abilitare il timer in modalità input capture, agendo sul registro CnSC. Si setta anche il bit 6 per abilitare l'interrupt nel canale.

All'interno della routine di servizio bisogna differenziare il caso in cui la routine stessa venga chiamata per un evento di interrupt o per overflow.



Osservando la figura, la routine deve rispettare tutti i tre seguenti casi:

- All'interno della stessa rampa si verificano tre "input capture" (ad esempio cnt_2 , cnt_1 e cnt_0). In questo caso è sufficiente calcolare la differenza ($cnt_2 - cnt_0$, perché in un giro avvengono due eventi di input capture) e utilizzare il risultato nella formula di f_{FAN} .
- Due eventi di "input capture" si verificano a cavallo di uno o più overflow del contatore. Sarà necessario correggere la differenza sommando un valore proporzionale al numero di overflow intercorsi tra i due counter.
- Se la ventola è ferma, bisogna impostare un numero massimo di overflow dopo il quale la ventola si considera ferma (f_{FAN} viene fissata a zero). Il numero massimo di overflow deve essere ovviamente legato alla frequenza minima della ventola.

Il periodo di un overflow è uguale all'inverso della formula

$$f = \frac{f_{clock}}{prescaler \cdot MOD}$$

quindi con un prescaler = 16 e un MOD = 2^{16} , $T_{overflow} = 0.021845s$. Dalla caratterizzazione statica è emerso che nel caso peggiore, ovvero quando la velocità è molto bassa, il periodo di rotazione $T_{rotazione}$ è di circa 300ms. Dividendo $T_{rotazione}$ per $T_{overflow}$ si ottiene 13,7 overflow. Per sicurezza si approssima il valore a 15, quindi trascorsi 15 overflow l'algoritmo considererà la velocità della ventola nulla.

Si riporta il codice riguardante il timer TMP0.

```
#define constant 3072000000;
int counter = 0;
int counter2 = 0;
int overflow = 0;
int diff = 0;
unsigned int ffan = 0;
bool toCapture = true;
```

Inizializzazione:

```
// bit 2: Prescaler impostato a 16. Sfrutto tutti i 32 bit
della costante Fcf * 2^M / PS
TPM0->SC = (1UL<<6) | (1UL<<2) | (1UL<<3);

// bit 6: CHIE, Enable channel interrupts
TPM0->CONTROLS[3].CnSC = (1UL<<6) | (1UL<<2);

// bit 10-8: Pin Mux Control, alternativa 3
PORTE->PCR[30] = (1UL<<8) | (1UL<<9);
```

Abilitazione:

```
NVIC_EnableIRQ(TPM0_IRQn);
```

Routine:

```
int TPM0routine(void) {

    //Overflow
    if(TPM0->STATUS&TPM_STATUS_TOF_MASK){
        overflow++;
        TPM0->STATUS=TPM_STATUS_TOF_MASK;
    }

    //Input capture
    if(TPM0->STATUS&TPM_STATUS_CH3F_MASK){
        if (toCapture) {
            counter2=TPM0->CONTROLS[3].CnV;
            diff = counter2 - counter + (overflow << 16);
            ffan = constant / diff;
            counter = counter2;

            overflow = 0;
        }
        toCapture = !toCapture;
        TPM0->STATUS=TPM_STATUS_CH3F_MASK;
    }

    if(overflow > 15) {
        ffan = 0;
    }

    return 1;
}
```

Il codice riguardante l'input capture non è corretto al 100%. Infatti la prima misura acquisita appena la ventola parte non sarà corretta, perché il valore di counter inizialmente è 0. Ad

ogni modo con questo codice tutte le misure successive alla prima sono corrette. Per sistemare l'algoritmo bisognerebbe quindi introdurre una nuova variabile booleana messa inizialmente a true e un controllo che, dopo avere svolto l'operazione corretta per la prima misura, imposti la variabile a false. La variabile verrà rimessa a true quando la velocità della ventola ritorna a 0. Viene riportato il codice appena esposto, che però non è stato testato sulla scheda.

```
#define constant 3072000000;
int counter = 0;
int counter2 = 0;
int overflow = 0;
int diff = 0;
unsigned int ffan = 0;
bool toCapture = false;
bool firstMeasure = true;

int TPM0routine(void) {

    //Overflow
    if(TPM0->STATUS&TPM_STATUS_TOF_MASK){
        overflow++;
        TPM0->STATUS=TPM_STATUS_TOF_MASK;
    }

    //Input capture
    if(TPM0->STATUS&TPM_STATUS_CH3F_MASK){
        if (firstMeasure)
            counter = TPM0->CONTROLS[3].CnV;
        else {
            if (toCapture) {
                counter2=TPM0->CONTROLS[3].CnV;
                diff = counter2 - counter + (overflow << 16);
                ffan = constant / diff;
                counter = counter2;

                overflow = 0;
            }
            toCapture = !toCapture;
            firstMeasure = false;
            TPM0->STATUS=TPM_STATUS_CH3F_MASK;
        }
    }

    if(overflow > 15) {
        ffan = 0;
        firstMeasure = true;
    }
}
```

```
        toCapture = false;
    }
    return 1;
}
```

Parte 3

Interfacciamento del microcontrollore con il PC mediante periferica seriale

La comunicazione seriale tra microcontrollore e PC avverrà tramite UART attraverso un adattatore USB.

Innanzitutto bisogna abilitare il clock per la UART agendo sul registro SIM_SCGC4, bit 11. Si abilitano quindi le porte dedicate alla trasmissione (PTE0, TX) e ricezione (PTE1, RX) come fatto in precedenza.

Per stabilire il baud rate, ovvero la velocità di trasmissione dei dati, si agisce sui registri UART1_BDH e UART1_BDL.

UARTx_BDH field descriptions

Field	Description
7 LBKDIE	LIN Break Detect Interrupt Enable (for LBKDIF) 0 Hardware interrupts from UART_S2[LBKDIF] disabled (use polling). 1 Hardware interrupt requested when UART_S2[LBKDIF] flag is 1.
6 RXEDGIE	RxD Input Active Edge Interrupt Enable (for RXEDGIF) 0 Hardware interrupts from UART_S2[RXEDGIF] disabled (use polling). 1 Hardware interrupt requested when UART_S2[RXEDGIF] flag is 1.
5 SBNS	Stop Bit Number Select SBNS determines whether data characters are one or two stop bits. 0 One stop bit. 1 Two stop bit.
4-0 SBR	Baud Rate Modulo Divisor. The 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the UART baud rate generator. When BR is cleared, the UART baud rate generator is disabled to reduce supply current. When BR is 1 - 8191, the UART baud rate equals $BUSCLK/(16 \times BR)$.

UARTx_BDL field descriptions

Field	Description
7-0 SBR	Baud Rate Modulo Divisor These 13 bits in SBR[12:0] are referred to collectively as BR. They set the modulo divide rate for the UART baud rate generator. When BR is cleared, the UART baud rate generator is disabled to reduce supply current. When BR is 1 - 8191, the UART baud rate equals $BUSCLK/(16 \times BR)$.

Vale l'uguaglianza

$$divisore = \frac{busclock}{16 \cdot baud\ rate}$$

dove:

- divisore è il valore da inserire nei registri sopra menzionati
- busclock è la frequenza del bus, cioè 24MHz

Dal momento che divisore è un numero intero, la frazione deve essere il più vicina possibile ad un valore intero. Scegliendo un baud rate di 115200, si ottiene che divisore = 13.02. È sufficiente quindi impostare il valore di 13 solo sul registro UART1_BDL.

Si abilita infine la trasmissione e la ricezione dei dati agendo sul registro UART1_C2.

UARTx_C2 field descriptions

Field	Description
7 TIE	Transmit Interrupt Enable for TDRE 0 Hardware interrupts from TDRE disabled; use polling. 1 Hardware interrupt requested when TDRE flag is 1.
6 TCIE	Transmission Complete Interrupt Enable for TC 0 Hardware interrupts from TC disabled; use polling. 1 Hardware interrupt requested when TC flag is 1.
5 RIE	Receiver Interrupt Enable for RDRF 0 Hardware interrupts from RDRF disabled; use polling. 1 Hardware interrupt requested when RDRF flag is 1.
4 ILIE	Idle Line Interrupt Enable for IDLE 0 Hardware interrupts from IDLE disabled; use polling. 1 Hardware interrupt requested when IDLE flag is 1.
3 TE	Transmitter Enable TE must be 1 to use the UART transmitter. When TE is set, the UART forces the TxD pin to act as an output for the UART system. When the UART is configured for single-wire operation (LOOPS = RSRC = 1), TXDIR controls the direction of traffic on the single UART communication line (TxD pin). TE can also queue an idle character by clearing TE then setting TE while a transmission is in progress. When TE is written to 0, the transmitter keeps control of the port TxD pin until any data, queued idle, or queued break character finishes transmitting before allowing the pin to revert to a general-purpose I/O pin. 0 Transmitter off. 1 Transmitter on.
2 RE	Receiver Enable When the UART receiver is off, the RxD pin reverts to being a general-purpose port I/O pin. If LOOPS is set the RxD pin reverts to being a general-purpose I/O pin even if RE is set. 0 Receiver off. 1 Receiver on.

La trasmissione dei dati avviene all'interno della routine di TMP0. In particolare si è scelto di trasmettere i dati ad ogni overflow in modo che l'invio avvenga ad una cadenza fissa. Alcuni dati, soprattutto quando la ventola gira piano, saranno ritrasmessi anche se invariati ma questa scelta faciliterà l'analisi della risposta al gradino.

Il dato da trasmettere deve essere copiato nel registro UART1_D. Si predispone perciò un array contenente il dato e due caratteri ascii necessari per andare a capo e a sinistra. Dal momento che il dato utile ha un massimo di 5 cifre, dopo il settimo dato o comunque quando il valore è '\n' la trasmissione viene interrotta. Si inserisce anche un ciclo while per aspettare che la trasmissione del dato successivo avvenga solo quando il buffer è libero.

Si riporta il codice per l'inizializzazione e la trasmissione:

```
unsigned int constant = 3072000000;

void UART_init(void) {

    // bit 11: Abilito clock su UART
    SIM->SCGC4 |= (1UL<<11);

    // bit 10-8: Pin Mux Control, alternativa 3
    PORTE->PCR[0] = (1UL<<8) | (1UL<<9);
    PORTE->PCR[1] = (1UL<<8) | (1UL<<9);

    // Imposto divisore a 13 = 1101
    UART1_BDH = 0;
    UART1_BDL = (1UL<<3) | (1UL<<2) | (1UL<<0);

    // Abilito trasmettitore e ricevitore
    UART1_C2 = (1UL<<3) | (1UL<<2);
}

int TPM0routine(void) {

    //Overflow
    if(TPM0->STATUS&TPM_STATUS_TOF_MASK){
        overflow++;

        TPM0->STATUS=TPM_STATUS_TOF_MASK;

        sprintf(data, "%u\r\n", ffan);
        int i = 0;
        while( i != 7 && data[i] != 10) {
            while(!(UART1->S1 & UART_S1_TDRE_MASK));
            UART1->D = data[i];
            i++;
        }
        while(!(UART1->S1 & UART_S1_TDRE_MASK));
        UART1->D = '\n';
    }
}
```

}

Utilizzando la modalità debug si è variato il duty cycle in runtime e si è verificata la corretta trasmissione dei dati

- duty al 50%

Watch 1		
<div><div></div><div></div></div>		
Name	Value	Type
duty	0x00003FDD	int
overflow	3	int
ffan	14073	unsigned int
output	50	int
data2	0x1FFFF02C data2[] "50\r\n"	char[5]
[0]	53 '5'	char
[1]	48 '0'	char
[2]	13	char
[3]	10	char
[4]	0	char

13983

13986

13987

13989

13991

13993

13993

13994

13996

13998

14000

14002

14005

14003

14002

14001

14000

13999

13998

13998












13998

13998

13999

—

- duty al 100%

Watch 1		
<div>▼</div>		
Name	Value	Type
 duty	0x00007FBB	int
 overflow	1	int
 ffan	29072	unsigned int
 output	100	int
  data2	0x1FFFF02C data2[] "100\r"	char[5]
 [0]	49 '1'	char
 [1]	48 '0'	char
 [2]	48 '0'	char
 [3]	13	char
 [4]	0	char

28850

28848

28844

28842

28840

28839

28842

28847

28851

28854

28855

28858

28859

28863

28868

28872

28874

28879

28881

28883












28885

28886

28888

-

- duty al 25%

Watch 1		
<div><div></div></div>		
Name	Value	Type
 duty	0x00001FEE	int
 overflow	5	int
 ffan	3500	unsigned int
 output	25	int
  data2	0x1FFFF02C data2[] "25\r\r"	char[5]
 [0]	50 '2'	char
 [1]	53 '5'	char
 [2]	13	char
 [3]	13	char
 [4]	0	char
<Enter expression>		

4697

4537

4388

4262

4156

4064

3986

3914

3857

3808

3768

3730

3694

3666

3641

3622

3605

3589

3581

3576

3569

3564

3562

-

- duty al 0%

Watch 1			
Name	Value	Type	
duty	0x0000198B	int	0
overflow	4060	int	0
ffan	0	unsigned int	0
output	20	int	0
data2	0x1FFFF02C data2[] "20\r\r"	char[5]	0
[0]	50 '2'	char	0
[1]	48 '0'	char	0
[2]	13	char	0
[3]	13	char	0
[4]	0	char	0

La ricezione dei dati avviene invece all'interno del main. Si suppone che il dato ricevuto sia in forma percentuale e rappresenti il duty cycle da applicare alla ventola. A parte il fatto che ora i dati vengono ricevuti, la logica rimane simile a quanto visto per la trasmissione.

Si riporta il codice:

```
int constant2 = 32767/100;

int main(void) {

    ...

    int i = 0;
    while(1) {
        if((UART1->S1&UART_S1_RDRF_MASK)) {
            data2[i] = UART1->D;
            if (data2[i] == '\r') {
                sscanf(data2, "%u", &output);
                duty = output * constant2;
                i = 0;
            }
            else {
                i++;
            }
        }
    }
}
```

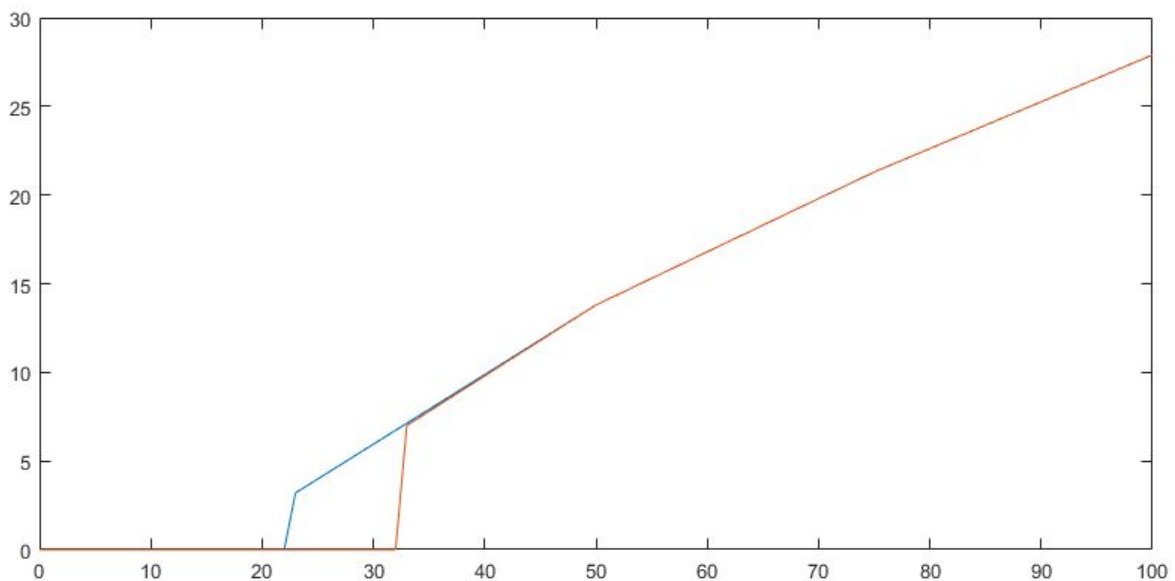
Parte 4

Caratterizzazione statica della ventola

Parte della caratterizzazione statica è già stata affrontata nella parte 1. Ora l'analisi si concentra nello studio del ciclo di isteresi della ventola. Ci si può aspettare, e così accade, che la tensione necessaria per accendere la ventola quando è ferma sia maggiore rispetto alla tensione minima applicabile quando la ventola sta già girando. Il fenomeno si verifica perché la ventola, se ferma, deve vincere una coppia di spunto per partire. Una volta accesa la coppia di spunto viene a mancare perciò è possibile raggiungere velocità più basse.

Sfruttando la comunicazione seriale si applicano diversi duty cycle crescenti partendo da ventola spenta. La ventola si accende quando il duty cycle è al 33% ovvero quando la tensione è a 4V. Se invece la ventola è accesa e si applica un duty cycle decrescente allora la ventola ruota fino a quando il duty cycle è al 23%, cioè 2.8V.

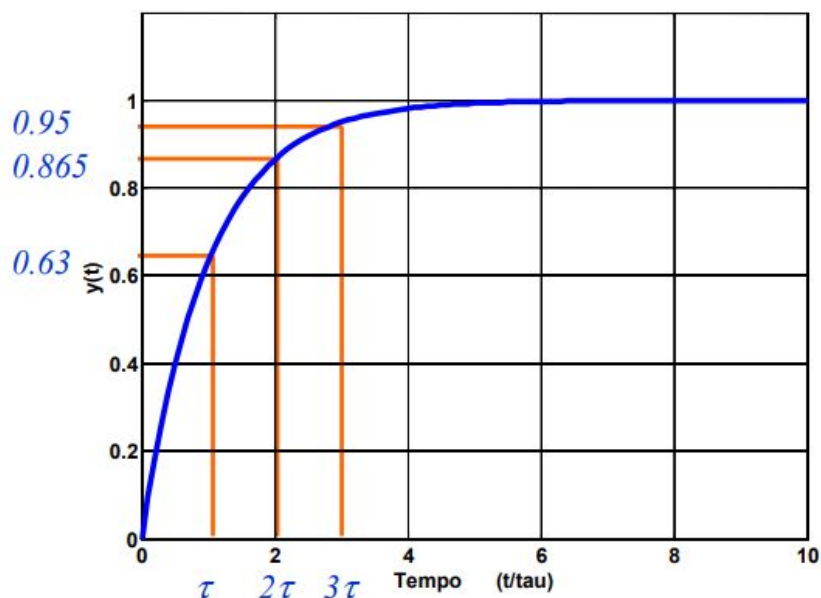
Tramite matlab è possibile tracciare con un'interpolazione lineare la curva con duty cycle crescente (arancio) e duty cycle decrescente (blu). Si nota chiaramente il fenomeno dell'isteresi nell'intervallo 23% - 33%.



Parte 5

Analisi della risposta al gradino e modellizzazione di un sistema del primo ordine

Per modellizzare la ventola come sistema del primo ordine si è analizzata la risposta a tre diversi gradini di tensione. In un sistema del primo ordine si raggiunge il 63,2% del valore del gradino nel tempo τ , che rappresenta la costante di tempo del sistema.

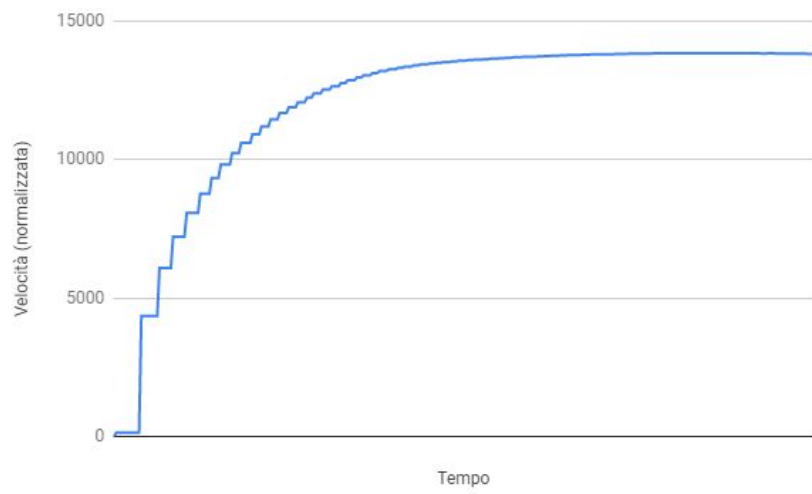


La scelta fatta nella parte 3, ovvero mandare i dati in intervalli temporali costanti facilita il calcolo di τ . Infatti, trovato l'intervallo temporale tra un overflow e il successivo, sarà sufficiente moltiplicarlo per il numero di intervalli fino al raggiungimento del 63,2% del gradino.

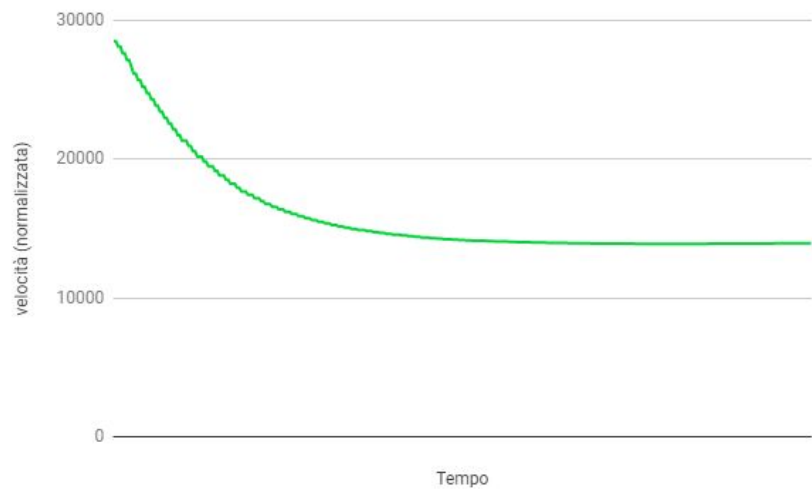
Con $f_{\text{clock}} = 48 \text{ MHz}$, prescaler = 16 e $\text{MOD} = 2^{16}$, allora l'intervallo temporale tra un overflow e il successivo è $T_{\text{overflow}} = \frac{\text{prescaler} \cdot \text{MOD}}{f_{\text{clock}}} = 0.0218453\text{s}$

I gradini di tensione applicati sono:

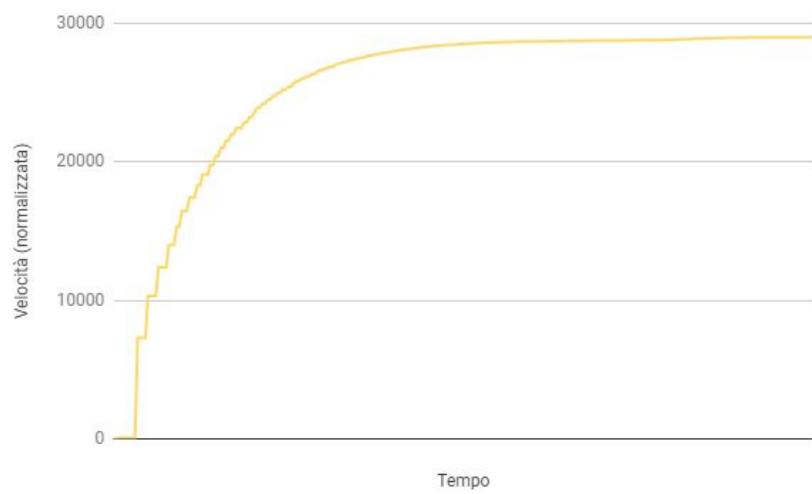
- 0 - 50%



- 100 - 50%

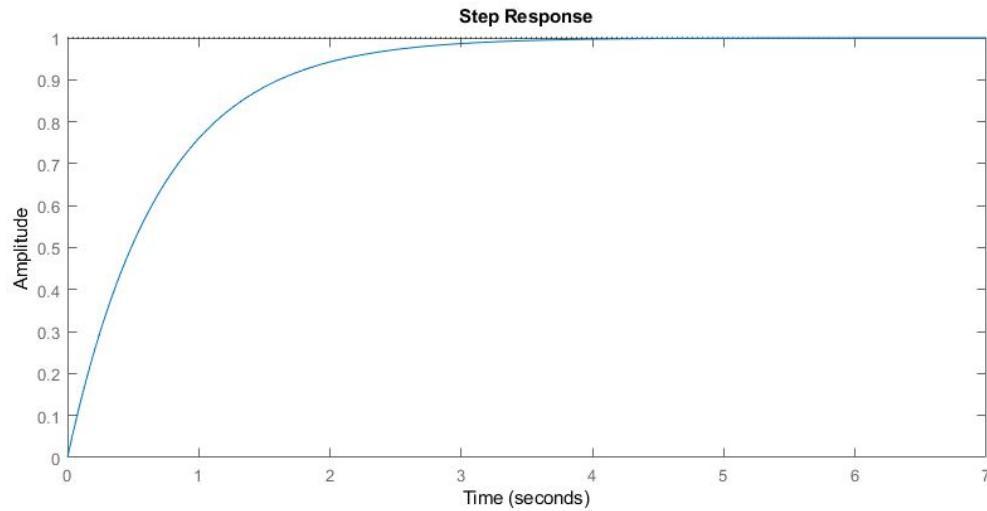


- 0 - 100%



Si è osservato che nei casi 0 - 50% e 100 - 50% per raggiungere il 63,2% del valore del gradino sono necessari 43 intervalli, quindi $\tau = 0.94$. Nel caso 0 - 100% ne servono 33, quindi $\tau = 0.72$. Si è quindi deciso di approssimare il sistema con una costante di tempo $\tau = 0.85$ come via di mezzo tra i risultati trovati sperimentalmente.

Viene riportata per completezza la risposta del sistema al gradino unitario.



Parte 6

Implementazione di un filtro a media mobile (FIR)

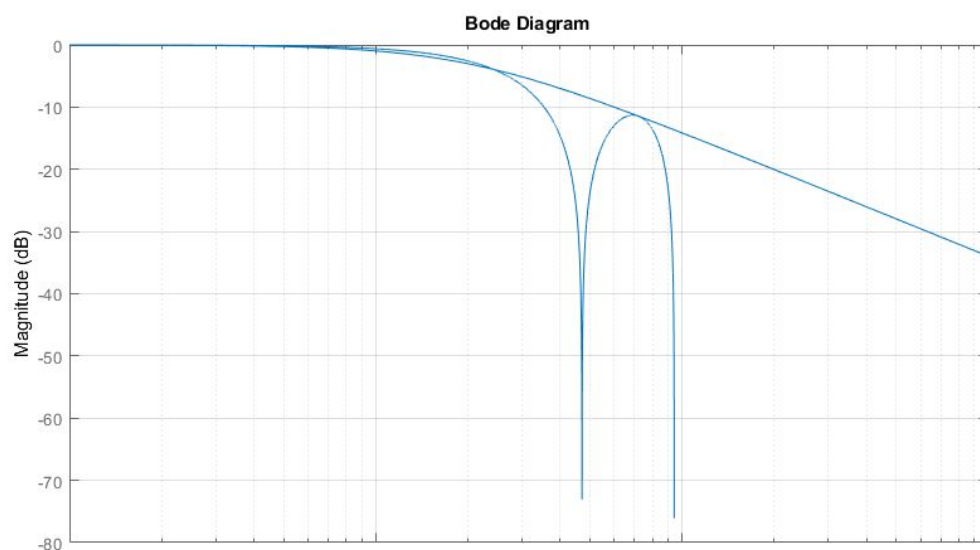
Per la progettazione del filtro è necessario considerare la frequenza minima alla quale si verifica il disturbo e la frequenza del polo dominante del sistema.

La frequenza del disturbo dipende proprio dalla frequenza di campionamento ed è uguale ad essa. Bisogna considerare il caso peggiore ovvero quando la frequenza è minima. Nelle precedenti parti si è visto che applicando il duty cycle minimo la velocità scende fino a quasi 3Hz. La frequenza di taglio del filtro dovrà quindi essere inferiore ai 3Hz.

Allo stesso tempo però non bisogna ritardare la dinamica del sistema, perciò la frequenza di taglio del filtro deve essere maggiore del polo dominante del sistema. Nella parte 5 si è modellizzato un sistema del primo ordine con $\tau = 0.85s$ perciò $\tau_{\text{filtro}} < \tau$.

La scelta cade su una pulsazione di taglio di 2rad/s, quindi $\tau = 0.5s$. Utilizzando la funzione freqz di matlab si trova che il valore dei coefficienti adatto per la realizzazione del filtro fir è di $\frac{1}{4}$. Dal momento che si stanno utilizzando numeri interi, si moltiplica il coefficiente per 2^{16} in modo da avere una precisione migliore.

Si riporta il grafico del filtro ideale e il fir.



Il calcolo deve essere effettuato dove viene raccolto il campione, ovvero durante l'input capture. Si riporta a tal proposito il codice:

```

#define c 16384
#define N 4;
int r0 = 0;
int values[N];
int k = 0;
int ffan2 = 0;

//Input capture
if (TPM0->STATUS & TPM_STATUS_CH3F_MASK) {
    if (toCapture) {
        counter2 = TPM0->CONTROLS[3].CnV;
        diff = counter2 - counter + (overflow << 16);
        ffan = constant / diff;
        counter = counter2;
        overflow = 0;

        r0 = c * ffan;
        r0 = r0 >> 16;
        ffan2 -= values[k];
        ffan2 += r0;
        values[k] = r0;
        k = (k + 1) % N;
    }
    toCapture = !toCapture;

    TPM0->STATUS = TPM_STATUS_CH3F_MASK;
}

```

A causa di un errore fatto in laboratorio i dati raccolti non sono corretti. Pertanto non si può avere una verifica sperimentale del funzionamento del filtro.

Codice completo

```
#include "MKL25Z4.h"
#include <stdbool.h>
#include <stdio.h>

int duty = 0x8000;

#define constant 3072000000;
int counter = 0;
int counter2 = 0;
int overflow = 0;
int diff = 0;
unsigned int ffan = 0;
bool toCapture = true;
char data[8];

int constant2 = 32767/100;
char data2[5];
int output = 0;

#define c 16384
#define N 4
int value = 0;
ina acc = 0;
int ffan2 = 0;
int values[N];
int k = 0;

void TPM_init(void) {
    SIM->SCGC6 |=
(1UL<<SIM_SCGC6_TPM1_SHIFT) | (1UL<<SIM_SCGC6_TPM0_SHIFT);
    SIM->SCGC5 |= (1UL<<13);

    TPM1->SC = (1UL<<3) | (1UL<<6);
    TPM1->MOD = 32768;
    TPM1->CONTROLS[0].CnSC = (1UL<<5) | (1UL<<3);
    TPM1->CONTROLS[0].CnV = 0;

    TPM0->SC = (1UL<<6) | (1UL<<2) | (1UL<<3);
    TPM0->CONTROLS[3].CnSC = (1UL<<6) | (1UL<<2);

    PORTE->PCR[20] = (1UL<<8) | (1UL<<9);
    PORTE->PCR[30] = (1UL<<8) | (1UL<<9);
}
```

```

void UART_init(void) {

    SIM->SCGC4 |= (1UL<<11);

    PORTE->PCR[0] = (1UL<<8) | (1UL<<9);
    PORTE->PCR[1] = (1UL<<8) | (1UL<<9);

    UART1_BDH = 0;
    UART1_BDL = (1UL<<3) | (1UL<<2) | (1UL<<0);

    UART1_C2 = (1UL<<3) | (1UL<<2);
}

void LED_init(void){
    SIM->SCGC5 |= (1UL<<12) | (1UL<<10);
    PORTD->PCR[1] = (1UL<<8);
    PORTB->PCR[18] = (1UL<<8);

    FPTD->PDDR = (1UL<<1);
    FPTB->PDDR = (1UL<<18);

    FPTD->PCOR = (1UL<<1);
    FPTB->PSOR = (1UL<<18);
}

int main(void){
    SystemCoreClockUpdate();//routine per aggiornare il clock

    TPM_init();
    UART_init();
    LED_init();

    NVIC_EnableIRQ(TPM1_IRQn);
    NVIC_EnableIRQ(TPM0_IRQn);

    int i = 0;
    while(1){
        if((UART1->S1&UART_S1_RDRF_MASK)){
            data2[i] = UART1->D;
            if (data2[i] == '\r'){
                sscanf(data2, "%u", &output);
                duty = output * constant2;
                i = 0;
            }
            else {

```

```

        i++;
    }
}

}

int TPM1routine(void) {
    TPM1->SC |= (1UL << 7);

    TPM1->CONTROLS[0].CnV = duty;

    FPTD->PTOR = (1UL<<1);
    FPTB->PTOR = (1UL<<18);

    return 1;
}

int TPM0routine(void) {

    //Overflow
    if (TPM0->STATUS&TPM_STATUS_TOF_MASK) {
        overflow++;

        TPM0->STATUS=TPM_STATUS_TOF_MASK;

        sprintf(data, "%u\r\n", out);
        int i = 0;
        while( i != 7 && data[i] != 10) {
            while(!(UART1->S1 & UART_S1_TDRE_MASK));
            UART1->D = data[i];
            i++;
        }
        while(!(UART1->S1 & UART_S1_TDRE_MASK));
        UART1->D = '\n';
    }

    //Input capture
    if (TPM0->STATUS&TPM_STATUS_CH3F_MASK) {
        if (toCapture) {
            counter2 = TPM0->CONTROLS[3].CnV;
            diff = counter2 - counter + (overflow <<16);
            ffan = constant / diff;
            counter = counter2;
            overflow = 0;

            value = c * ffan;

```



```

        acc -= values[k];
        acc += value;
        values[k] = value;
        k = (k + 1) % N;
        ffan2 = acc>>16;

    }
    toCapture = !toCapture;

    TPM0->STATUS = TPM_STATUS_CH3F_MASK;
}

if(overflow > 15) {
    ffan = 0;
}

return 1;
}

```