**MOSFET**

Il [Mosfet di potenza](https://it.emcelettronica.com/mosfet-e-igbt-per-il-controllo-motori) della terza generazione IRL540 sembra stato creato apposta per il pilotaggio di grossi carichi da parte dei microcontrollori. La lettera L nella sua sigla sta, infatti, per Logic-Level Gate Drive. È molto veloce, costa poco e ha una limitata Rdson che si traduce in una dissipazione quasi nulla e temperature basse di funzionamento. Inoltre il suo pilotaggio avviene senza problemi anche con una tensione di "gate" pari a 3.3 V. In figura 1 si può osservare il transistor (in contenitore TO220AB), assieme alla sua piedinatura. Le sue caratteristiche elettriche sono estremamente interessanti: la VDS massima è di 100 V, e la corrente massima di drain è pari a 28 A, per una totale dissipazione di ben 150 W. Un piccolo cavallo insomma.

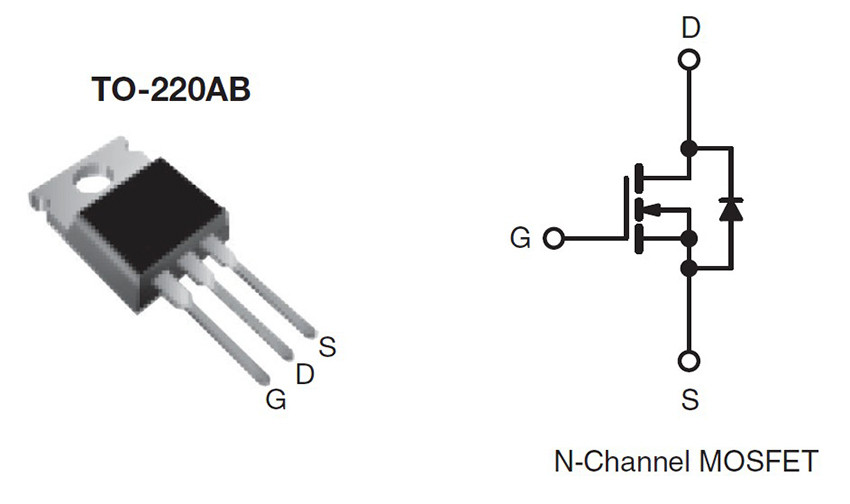
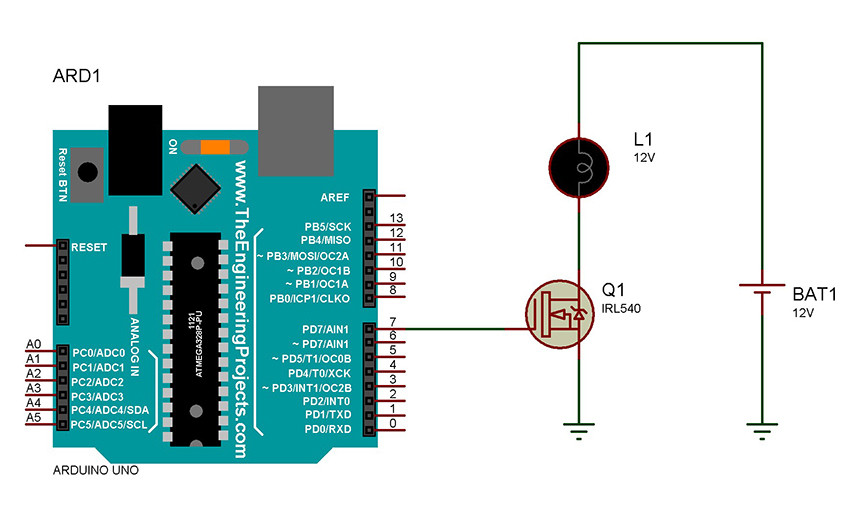


Figura 1: Il Mosfet IRL540

## Lampeggiatore ad alta potenza

Lo schema della figura 2 mostra **Arduino collegato a una lampada di grossa potenza**, caratterizzata da una tensione di alimentazione di 12 V, una resistenza di 1 ohm, una corrente di assorbimento di ben 12 A e, quindi, una dissipazione di circa 144 W. Per illuminarla occorre una batteria molto grossa.

Figura 2: Schema elettrico con una lampada e un MOSFET

La lampada è collegata ad Arduino sulla porta 7. In caso di livello logico alto, il gate del Mosfet si trova ovviamente alla tensione di 5 V (ma non c’è praticamente passaggio di corrente, data la sua altissima impedenza d’ingresso) e il drain è attraversato dalla massima corrente (circa 12 A). In tale condizione l’IRL540 dissipa una potenza di circa 7 W, non male per pilotare un carico di quella portata. Se il transistor dovesse scaldare si dovrebbe prevedere un’adeguata aletta di raffreddamento.

## Lo sketch del lampeggiatore

La figura 3 riporta il semplicissimo listato del lampeggiatore. Il pin 7 è configurato come uscita digitale, all’interno della funzione setup(). La funzione loop(), invece, attiva e disattiva, ciclicamente, tale porta, alla frequenza di un Hertz.

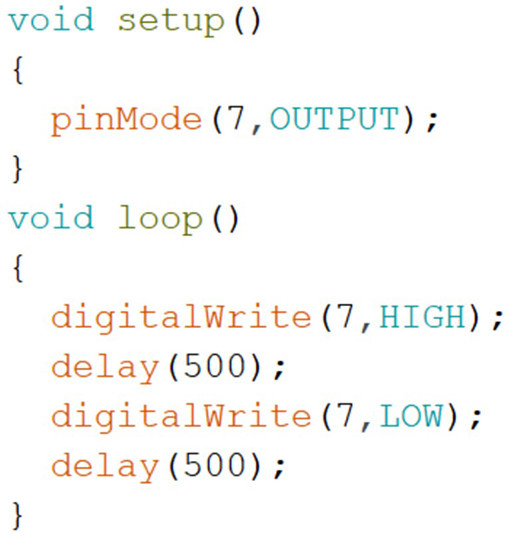


Figura 3: Lo sketch del lampeggiatore ad alta potenza

In figura 4 è riportato il grafico della corrente che attraversa la lampada, nel dominio del tempo, durante il funzionamento del prototipo.

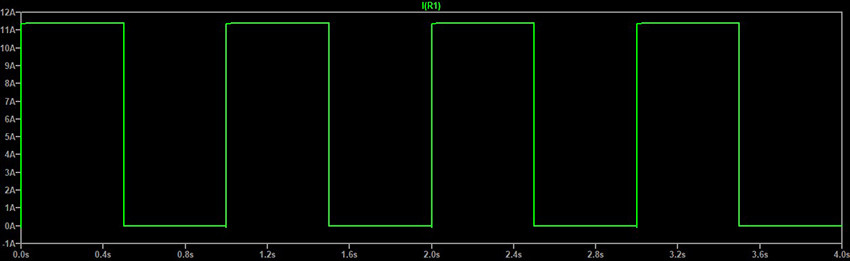
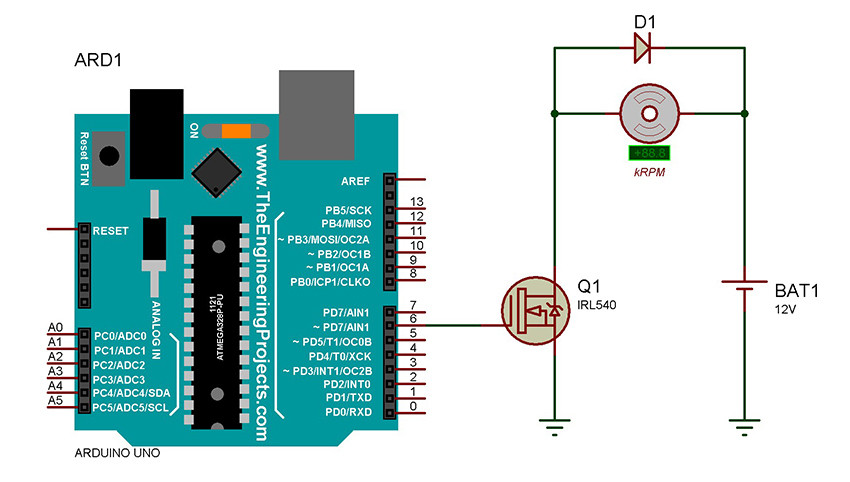


Figura 4: Il grafico della corrente che transita sulla lampada

## Motore in continua

Con tale componente è possibile anche pilotare un motore in corrente continua. Tramite la tecnica del PWM si può variare la sua velocità senza influire, peraltro, sulla coppia. La figura 5 mostra lo schema elettrico. Un diodo di ricircolo è montato in "antiparallelo" al carico, per eliminare i pericolosi picchi di tensione, dovuti agli avvolgimenti del motore.

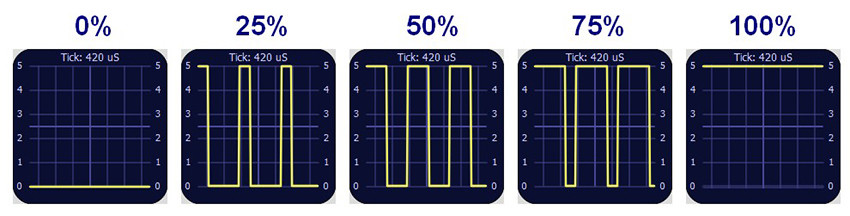
Figura 5: Schema elettrico per il pilotaggio del motore in C/C

## Lo sketch

Il listato proposto in figura 6 ha lo scopo di pilotare il motore, secondo le seguenti diverse temporizzazioni e potenze:

* 5 secondi allo 0%;
* 10 secondi al 25%;
* 10 secondi al 50%;
* 10 secondi al 75%;
* 10 secondi al 100%.

Il motore aumenterà la propria velocità di rotazione a ogni variazione del duty cycle del segnale PWM, come evidenziato dagli oscillogrammi di cui in figura 6.

Figura 6: I vari duty cycles per le diverse velocità del motore

In figura 7 è riportato il semplice listato. Si ricorda che per adottare la tecnica del PWM occorre utilizzare solamente i pin di Arduino predisposti a tale tipologia di lavoro.

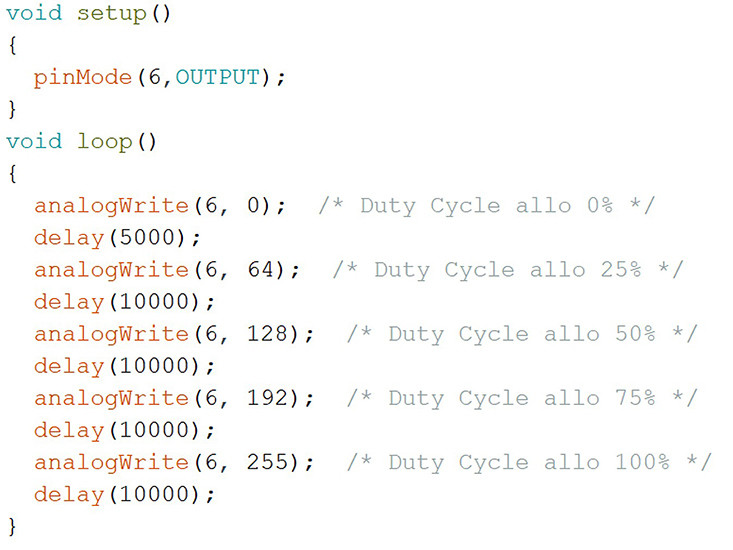


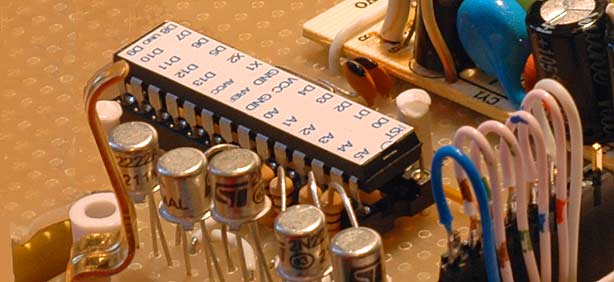
Figura 7: Lo sketch

If you are driving the MOS FET with 0-5 volts you need a logic version IRL540.

IRF540 requires 10 volts Vgs to get .077ohms Rds  
IRL540 requires  5 volts Vgs to get .077ohms  
  
Also, you need a 220 ohm resistor in series with the gate and I would put a 10K from the gate to ground.

# Pilotare dispositivi di potenza con Arduino

Qualche tempo fa ho cominciato a pensare di utilizzare Arduino per gestire tramite i propri pin di I/O dei dispositivi esterni. Mi sono quasi subito scontrato con un fatto del quale tutti i progettisti esperti sono sicuramente al corrente ma che per un maker alle prime armi non è per nulla scontato: per funzionare, qualsiasi dispositivo assorbe corrente.

[](https://www.0ex.it/wp-content/uploads/2014/02/circuito-di-potenza-atmega328-2n2222.jpg)

Il secondo importantissimo fatto è che i pin di Arduino possono fornire solo una certa corrente in uscita, se il carico, ovvero il dispositivo che voglio alimentare, richiede più corrente di quella erogabile, il circuito relativo al pin in questione si danneggia. Inoltre, sempre parlando di correnti, la somma delle correnti in uscita dai pin di Arduino non deve superare un determinato valore pena, anche qui, il danneggiamento del microcontrollore che costituisce il cuore della nostra scheda preferita.

Ultimo ma altrettanto degno di nota è il fatto che le uscite hanno una tensione fissa di 5V che non sempre corrisponde alla tensione nominale di funzionamento del dispositivi che vogliamo controllare.

# Resistenze, tensioni, correnti e potenze

Prima di poter continuare ho dovuto fare i conti con i parametri fondamentali che descrivono elettricamente un dispositivo e alcune regole dell’elettrotecnica che mettono in relazione questi parametri. Non voglio entrare nei dettagli  ma giusto annotare un paio di importanti concetti di base che devono sempre essere tenuti presenti quando si progetta un sistema elettrico/elettronico. (Non me ne vogliano gli esperti se per semplicità ho trascurato qualche particolare/dettaglio in favore della comprensibilità)

## Legge di Ohm

E’ una legge fondamentale che nella sua semplicità racchiude tutto ciò che accade in un qualsiasi bipolo passivo lineare ideale in corrente continua (Lo so, a questo bipolo sono riferiti un sacco di aggettivi strani, un giorno, se a qualcuno interesserà ne parleremo). Si enuncia così:

V = R \* I

Dove V è la tensione ai capi del bipolo, I la corrente che lo attraversa e R la sua resistenza.

Solitamente, nella pratica, R è data e dipende  dalle caratteristiche costruttive del bipolo che stiamo alimentando (nel nostro caso il dispositivo che vogliamo controllare, anche detto carico). Altrettanto solitamente la tensione nominale di alimentazione V è data. Questo ci porta a dire che la corrente I che scorrerà nel nostro dispositivo quando alimentato alla sua tensione nominale di alimentazione sarà:

I = V / R

Da questa seconda equazione si deduce anche un’altra importantissima cosa: la tensione (o differenza di potenziale) è la causa, la corrente è l’effetto. Per i nostri ragionamenti vale quindi questa regola: imposta un differenza di potenziale tra due punti tra i quali esiste una certa resistenza, scorrerà una certa corrente.

## Potenza in corrente continua

La seconda legge che serve per capire cosa succede quando si collegano i carichi a delle sorgenti di alimentazione è quella che consente di calcolare le potenze. Si enuncia così:

P = V \* I

Cioè la potenza P dissipata/erogata da un bipolo è pari al prodotto della tensione ai suoi capi per la corrente che lo attraversa.

Sempre semplificando un po’, a seconda del tipo di dispositivo, parte della potenza può essere trasformata per compiere il compito per cui il dispositivo è stato concepito e parte verrà dispersa in calore (secondo un parametro detto rendimento). La parte che viene dissipata in calore è quella che, se non correttamente gestita, determina in vario modo e misura guasti ai dispositivi. Non per niente si dice che un certo dispositivo è bruciato!!!

# Superare i limiti operativi di Arduino

La prima cosa da capire è quindi quali siano le correnti massime prelevabili dai singoli piedini e quale sia il limite massimo di corrente assorbibile da tutti i piedini di output contemporaneamente attivi. I datasheets del ATmega328 ci offrono molte informazioni tra cui anche quelle che stiamo cercando:

* Corrente massima in uscita da un piedino di output: 40 mA
* Somma delle correnti in uscita massima: 200 mA

Sembrano correnti veramente basse! Ed in effetti facendo ricorso alle equazioni viste sopra: a una tensione di 5V significa una potenza massima di 0,2W per pin e 1W totali. Giusto per fare qualche esempio un piccolo motore DC alimentato a 5V assorbe facilmente 250mA o la bobina di un relay tipo lo Zettler AZ8222-2C-5DSE assorbe 40 mA.

Tradotto in altri termini questo significa che non possiamo collegare ad Arduino carichi che assorbano potenze superiori a quelle massime erogabili da ciascun pin e da tutta la scheda pena la distruzione del microcontrollore.

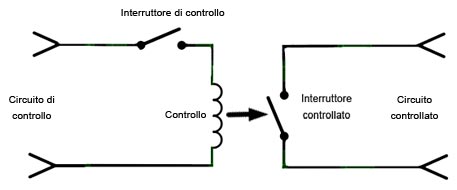
[](https://www.0ex.it/wp-content/uploads/2014/02/interruttore-controllato.jpg)

Figura 1: interruttore controllato

Per superare questi limiti si utilizzano le uscite come segnali di controllo per pilotare degli “interruttori controllati” o  “comandati” posti tra l’alimentazione principale e il carico da controllare. Quando l’uscita è alta, cioè +5V, l’interruttore è chiuso e collega il carico alla alimentazione (acceso); Quando l’uscita è bassa, cioè 0V, l’interruttore è aperto ed il carico è scollegato dall’alimentazione (spento).

# Il transistor come interruttore

Il modo più semplice di realizzare in elettronica un “interruttore comandato” è quello di utilizzare un transistor in commutazione ovvero un transistor che lavori nelle zone di saturazione/interdizione. In queste zone infatti il dispositivo si comporta in modo molto simile ad un interruttore facendo passare tutta la corrente possibile o bloccandola completamente tra due dei sui tre piedini in funzione del segnale applicato tra il terzo piedino ed un piedino comune.

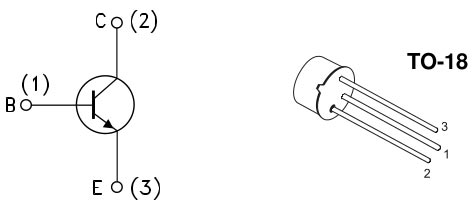
[](https://www.0ex.it/wp-content/uploads/2014/02/transistor_NPN.jpg)

Figura 2: Transitor con base B, collettore C e emettitore E  
(fonte 2N2222A datasheet – STMicroelectronics)

Con riferimento alla figura, il piedino comune è detto emettitore ed è collegato a massa, il piedino al quale applicare il segnale proveniente dal pin di controllo è detto base, il pedino a cui collegare un terminale del carico (nello schema rappresentato da Rc) è detto collettore (l’altro terminale del carico sarà collegato alla alimentazione).

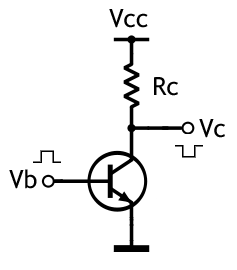
[](https://www.0ex.it/wp-content/uploads/2014/05/TransistorInterruttore.png)

Figura 3: Transistor come interruttore

Quando il segnale di controllo applicato alla base è positivo (HIGH logico), passerà tra base e emettitore una corrente positiva che, se sufficientemente grande, porterà il transistor in saturazione. Quando il transistor si trova in questo stato la tensione tra collettore e emettitore tende a zero, e tutta la tensione di alimentazione è applicata al carico.

Viceversa quando il segnale di controllo applicato alla base è zero (LOW logico), tra base e emettitore non scorrerà corrente e il transistor lavorerà in interdizione. Quando il transistor si trova in questo stato tutta la tensione di alimentazione “cade” tra collettore e emettitore e il carico risulta non alimentato.

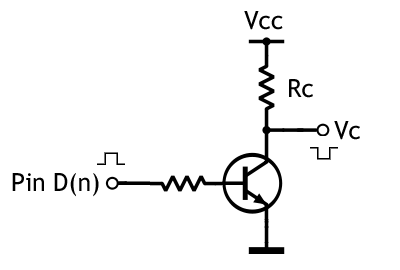
[](https://www.0ex.it/wp-content/uploads/2014/05/TransistorCollegatoArduino.png)

Figura 4: Transistor collegato ad Arduino con Rb

Affinché la corrente che passa nella base del transistor sia sufficiente per portare il dispositivo in saturazione ma non troppo intensa da danneggiarlo è necessario porre tra il pin di controllo e il terminale di base un resistenza. Il suo calcolo è abbastanza semplice.

## Calcolo Rb

Per calcolare la resistenza di base si procede a ritroso partendo dalla analisi della parte di circuito relativa al carico. Innanzitutto si deve determinare la corrente che scorrerà attraverso il carico, e quindi attraverso il collettore, quando il transistor sarà in saturazione:

Ic = (Vcc - Vcesat) / Rc

La corrente di collettore sarà pari alla tensione presente sul carico (Vcc – Vcesat dove Vcc è la tensione di alimentazione e Vcesat è la tensione tra collettore e emettitore del transistor quando si trova in saturazione – dato disponibile sui datasheets del componente) diviso la resistenza del carico.

Una volta nota la corrente di collettore possiamo calcolare la corrente di base sapendo che Ic = hfe \* Ib (dove Ib è la corrente di base e hfe il guadagno del transistor come da datasheets) quindi:

Ib = Ic / hfe

A questo punto si può analizzare il circuito di ingresso considerando la tensione del segnale di controllo, nel nostro caso 5V (valore alto di una uscita digitale di Arduino) la tensione tra emettitore e base Vbe (che si trova sui datasheets) e la corrente Ib desiderata e calcolata al passo precedente. Dato che:

Ib = (Vpin - Vbe ) / Rb

avremo che:

Rb = (Vpin - Vbe) / Ib

Questa è la resistenza massima da utilizzare sulla base del transistor per assicurare che vada in saturazione (cioè quella che garantisce una Ib sufficiente alla saturazione). Valori di resistenza minori comportano Ib maggiori e quindi una maggiore sicurezza di far lavorare il transistor nella zona corretta.

Esiste però un limite inferiore dettato da due parametri fondamentali:

* La corrente massima erogabile da un pin di Arduino
* La corrente massima assorbibile dalla basse del transistor

Chiamando il valore minore tra i due Ib(max) avremo che:

Rb(min) = (Vpin - Vbe) / Ib(max) dove Ib(max) = Ic(max) / hfe

Il valore finale di Rb  dovrà essere compreso tra Rb e Rb(min) e scelto in funzione della serie di resistori a disposizione (ad esempio E12).

## Esempio di calcolo

Vediamo concretamente come calcolare la resistenza di base nel caso in cui vogliamo alimentare un piccolo motore DC la cui tensione di alimentazione è 5V e la resistenza interna è di 20 Ohm con un transistor 2N2222 usato come interruttore.

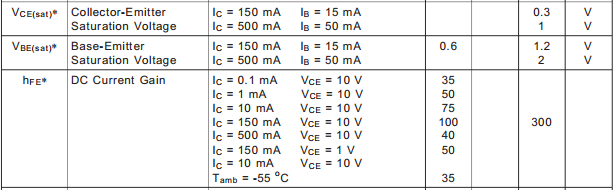
[](https://www.0ex.it/wp-content/uploads/2014/05/2a2222-DC-Values.png)

Tabella: transistor 2N2222, estratto datasheets  
(fonte 2N2222A datasheet – STMicroelectronics)

I dati sul transistor che ci interessano e che sono disponibili sui datasheets del componente sono i seguenti:

* Vcesat = 0.3 V
* Vbesat = 0.6 V
* hfe = 100
* Ic(max) = 600 mA

I dati del motore necessari per il calcolo sono invece:

* Ic = 250 mA

Quindi:

Ib = 250 mA / 100 = 2,5 mA

da cui:

Rb = (5 V - 0.6 V) / 2,5 mA = 1760 Ohm

**Nota sull’esempio:**  
I motori sono carichi induttivi che richiedono alcuni accorgimenti ulteriori per evitare che correnti di scarica possano circolare in modo incontrollato nel transistor e causarne la rottura. Per evitare queste correnti anche dette di fly-back è necessario posizionare un diodo ai capi del carico induttivo con il catodo collegato al terminale connesso all’alimentazione e l’anodo collegato al terminale connesso al collettore del transistor. Questo vale anche per gli esempi successivi in cui si utilizzano relay (che al loro interno hanno un elettromagnete, cioè un induttanza) per aumentare la potenza del carico comandato.

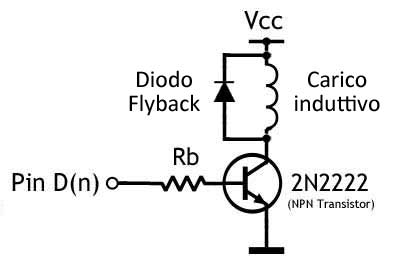
[](https://www.0ex.it/wp-content/uploads/2014/05/diodo_flyback_antiparallelo.jpg)

Figura 5: Diodo Flyback (antiparallelo) su carico induttivo

# Transistor a doppio stadio

Nel caso in cui le correnti di collettore siano tali da non poter utilizzare un transistor BJT semplice come il 2N2222 è possibile passare a componenti capaci di sopportare correnti più elevate. Il problema dei transistor più potenti è che necessitano di correnti di base maggiori di quelle erogabili da Arduino. La soluzione è utilizzare uno schema a due stadi collegati come in Figura 6.

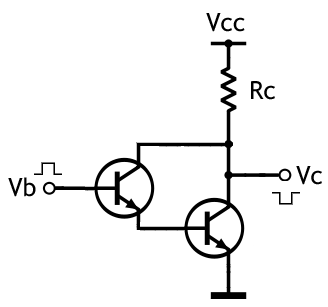
[](https://www.0ex.it/wp-content/uploads/2014/05/TransistorConfigurazioneDarlington.png)

Figura 6: Schema doppio stadio, configurazione darlington

Esistono componenti tipo i TIP120: coppia di transistor in configurazione darlington, che realizzano questa configurazione in un unico dispositivo.

# MOSFET, una alternative ai BJT

Se i transistor bipolari (BJT) sono una ottima scelta quando le correnti richieste dal carico rimangono nell’ordine delle centinaia di mA (600mA/800mA ) e le tensioni di alimentazione intorno alle decine di volts (20V/30V), per correnti nell’ordine degli A (1A/10A) e tensioni introno al centinaio di V iniziano ad essere interessanti i MOSFET.

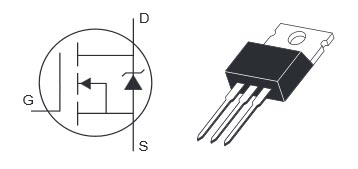
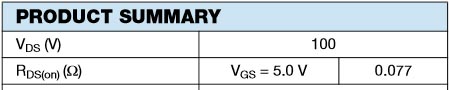
[](https://www.0ex.it/wp-content/uploads/2014/05/mosfet.jpg)

Figura 7: simblo e aspetto di un MOSFET

 A differenza dei BJT che si comandano in corrente, i MOSFET sono comandati in tensione e questo nel campo dei microcontrollori TTL può costituire un problema dato che solitamente la tensione necessaria per farli andare in conduzione è dell’ordine dei 10V, di gran lunga maggiore dei 5V a cui lavora Arduino.

[](https://www.0ex.it/wp-content/uploads/2014/05/ResistenzaDrainSource-IRL54.jpg)  
Figura 8: Dati su resistenza Drain-Surce a Vgs=5V  
(Fonte: datasheet Vishay)

Esistono fortunatamente dei MOSFET tipo l’IRL540 che sono appositamente progettati per andare in ‘OnState’ con una tensione Vgs  in ingresso di 5V.

L’altra grande differenza riguarda la frequenza alla quale far lavorare il componente: mentre un BJT può essere spinto senza troppi accorgimenti a frequenze dell’ordine dei 100kHz, un MOSFET già a qualche kHz necessita di particolari accorgimenti. Solitamente però, nella applicazioni a micocontrollore, questo aspetto non è un problema.

### MOSFET in parallelo

A differenza dei BJT i MOSFET possono essere collegati in parallelo e quindi permettono una soluzione ‘scalabile’ che consente cioè di gestire correnti maggiori semplicemente aumentando il numero di dispositivi impigati.

# Uno sguardo alle applicazione in AC

Fino ad ora abbiamo considerato applicazioni in corrente continua. Esistono però numerose applicazioni in cui il carico di potenza è un dispositivo che funziona in corrente alternata.

Una buona soluzione per gestire carichi in corrente alternata tramite un microcontrollore è l’utilizzo di un TRIAC.

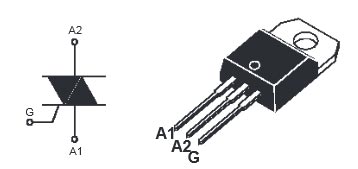
[](https://www.0ex.it/wp-content/uploads/2014/05/triac.jpg)

Figura 9: Simbolo e aspetto di un TRIAC

# Per potenze ancora maggiori

Il passo successivo è quello di utilizzare al posto di un secondo stadio a semiconduttore come nel caso del tipo120, un relay.

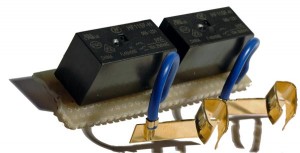
[](https://www.0ex.it/wp-content/uploads/2014/05/relay-HF115F-H.jpg)

Figura 10: relay

In questa configurazione il transistor garantisce la potenza necessaria per comandare la bobina del relay la quale comanda a sua volta dei contatti meccanici. Per quanto riguarda il carico, in questo caso, l’interruttore comandato è a tutti gli effetti un interruttore.

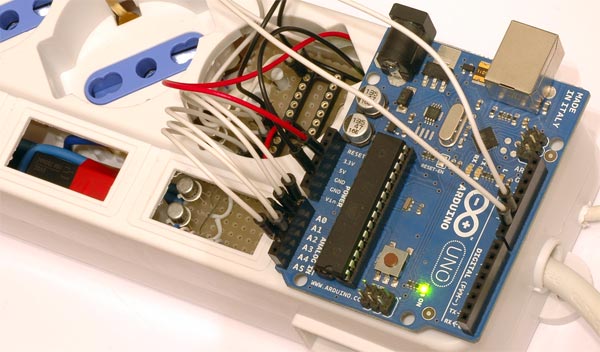
[](https://www.0ex.it/wp-content/uploads/2014/05/ArduinoComandaRelays.jpg)

Figura 11: Arduino comanda relay

Se non è ancora abbastanza si potrebbe pensare di utilizzare un relay a 5V per comandare un contattore, tipo l’Hager ES224 o ES424 al quale collegare il carico finale.

Una tipica applicazione in cui applicare un modello di questo tipo è il controllo di sistemi di riscaldamento/raffrescamento degli ambienti.

**Nota sulle applicazioni in alta tensione**

Gli ultimi esempi sono stati inclusi in questo articolo per completezza ma sono volutamente generici perchè riguardano applicazioni che richiedono conoscenze specifiche nella manutenzione degli impianti elettrici in alta tensione. Intervenire su impianti in alta tensione senza adottare le necessarie precauzioni comporta rischio di folgoramento e di morte, quindi attenzione!

# Trade off potenza/velocita’/costo

Le tre soluzioni proposte sopra: transistor, transistor doppio stadio e relay hanno ovviamente ambiti di applicazione molto diversi identificati soprattutto dai fattori: potenza, velocità di commutazione e costo.

Per quanto riguarda la potenza gestibile abbiamo visto che cresce passando dal transistor semplice, al doppio stadio al relay.

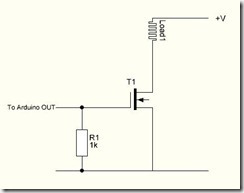
Anche per quanto riguarda il costo dei componenti la progressione segue lo stesso ordine: la soluzione transistor+relay è la più costosa, soprattutto per la presenza del relay, il doppio stadio una via intermedia e il transistor semplice la più economica.

Per quanto riguarda la velocità di commutazione l’ordine è invece l’inverso: il più veloce è il transistor singolo. Segue il doppio stadio. Chiude la classifica il relay che a causa delle componenti meccaniche ha tempi di commutazione limitati dalla fisicità dei contatti che si devono muovere.

**Using MOSFETS with TTL levels (5 Volt and 3.3 Volt)**

Various Arduino projects that need to switch a high DC load are using MOSFET’s to do this, according to the circuit at the right (R1 is optional and may be necessary to switch off the FET if the pin goes low.

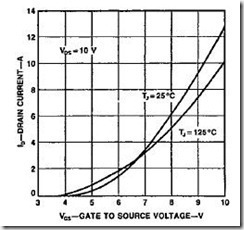
Popular MOSFET’s that are used are the IRF510 and IRF 520

[](https://arduinodiy.files.wordpress.com/2012/05/fet.jpg)

|  |
| --- |
| [irf510](https://arduinodiy.files.wordpress.com/2012/05/irf510ivgs.jpg) IRF510 |
| [irf520](https://arduinodiy.files.wordpress.com/2012/05/irf520vgs.jpg) IRF520 |

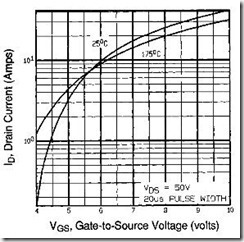
Looking at those graphs one can see that at a gate to source level of 5V (Arduino levels) the IRF510 is only capable of delivering 1 Amp, whereas it is specified for 5,6 Amps continuous current. The 520 is somewhat better: at 5 V it delivers 3 Amps from its max of 9.2. This is because these FET’s are designed to pass the max current at gate voltages of around 10 Volts and that is beyond what most microcontrollers can deliver.

For the IRF522 it is even worse.

[](https://arduinodiy.files.wordpress.com/2012/05/irf522ivgs.jpg)

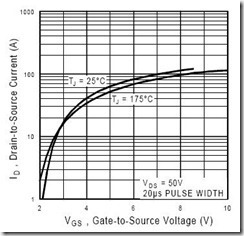
Looking at the curve, at a gate to source voltage of 5V the IRF522 is hardly turned on. You are limited to a current of about 200mA.

Much better to use a cheap Darlington transistor then.

The IRF530 is a better choice:  
[](https://arduinodiy.files.wordpress.com/2012/05/irf530ivgs.jpg)

At 5V volt on the gate, the IRF530 will pass something around 4.5Amps.

If you are shopping for a MOSFET for the Arduino consider the IRL540 The L shows that is a logic level mosfet. A logic level mosfet means that it is designed to turn on fully from the logic level of a microprocessor. The standard mosfet (IRF series etc) is designed to run from 10V.  
Here is the curve for the IRL540:

[](https://arduinodiy.files.wordpress.com/2012/05/irl540ivgs.jpg)

Now at 5V you are out of the linear region and the MOSFET can already deliver its specified 28 Amps continuous current.

You may also consider the IRLZ44.

The IRLZ44 data-sheet says that with a 3V TTL-level drive the FET drops less than 0.15V at 4A (at 25C, Rds(on) is about 0.04 ohms), and under 0.25V at 175C (Rds(on) < 0.063 ohms).  
So we know the FET’s I^2-R ohmic dissipation will be under 1 watt, and that’s good. If we use Vgs = 4V, specified for AVR chip outputs, the dissipation should be about 0.4W at 25C (0.8W for Tj = 175C).

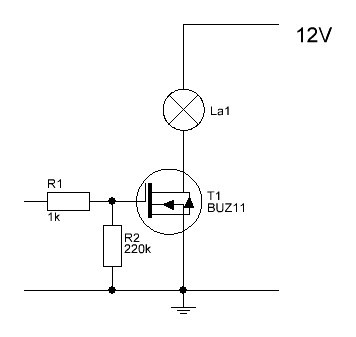
Wether a MOSFET is a standard MOSFET or a ‘logic’FET becomes clear from the Datasheet. If for instance you look at the Datasheet  of the IRFZ44N at the Rds(on), This lists the ‘on-resistance’ under the condition that Vgs=10V (and Id=25A). If there is no rating for Rds(on) when Vgs=5V (or 4.5V), then it is not a logic-level MOSFET. A logic level MOSFET will have Rds(on) specified for Vgs=5V or 4.5V.  If its only specified for Vgs=10V, its not logic-level.

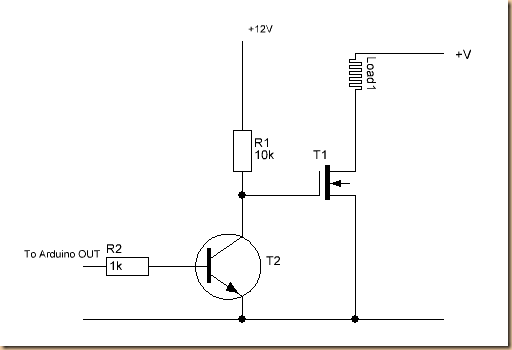
Another thing to beware of in datasheets is Vthresh (threshhold voltage).  This is not the gate voltage to turn the device on, its the gate voltage at which it switches fully off (less than a few uA of current, typically).  If Vthresh is given as 2..4V range, it cannot be a logic level MOSFET (Vthresh is usually 0.5 to 1V for logic-level MOSFETs).

When designing with MOSFETs be aware that instead of having a Vsat like a bipolar transistor, a fully-saturated MOSFET acts as a low-value linear resistor.  If for instance you want to switch 5A in a 12V circuit and you only want to waste 0.5V across the MOSFET, then its on-resistance (Vds(on)) should be <= 0.1 ohms (0.5V / 5A)

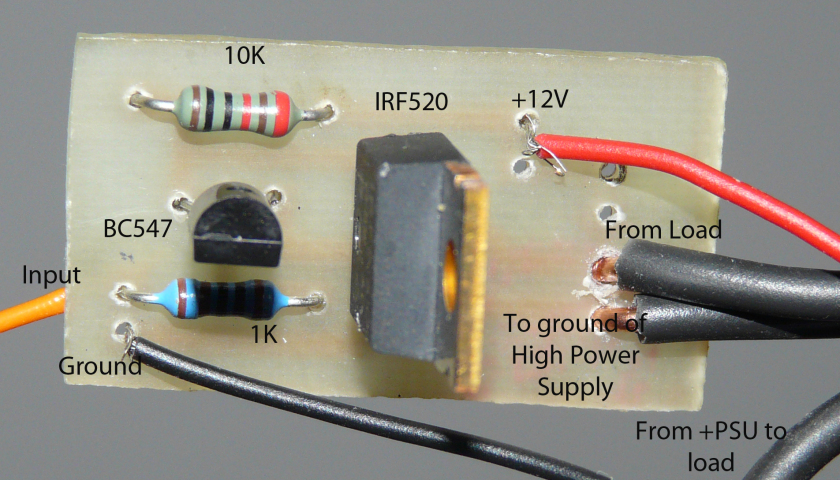
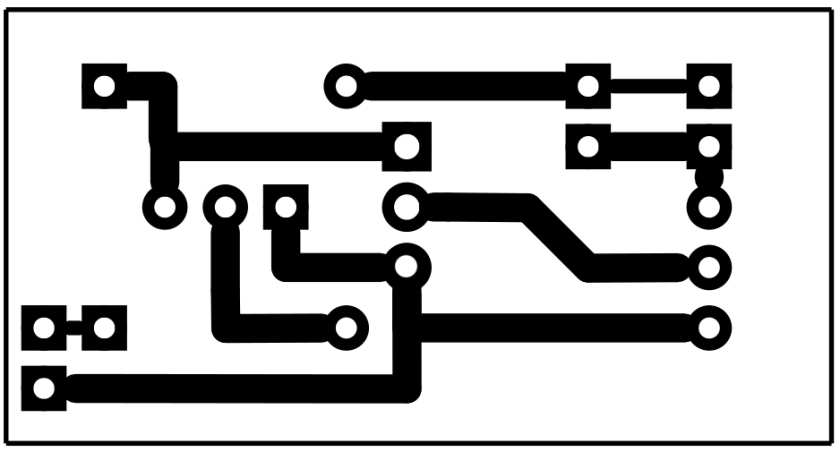
The dissipation then is 5x5x0.1=2.5 Watt. But suppose the FET you choose has 0.05ohm Vds(on) and carries 10A then it will dissipate I^2R watts, ie 10x10x0.05 = 5W.  This will need a good heatsink if the load is on for more than a second or two, but it is no issue if it gets millisecond pulses every few seconds.  ‘ON-resistance’ of 0.2 to 0.001 ohms are available (though less than 0.005ohms gets expensive).

The relatively cheap BUZ11 is also an option. Although it is no Logic level MOSFET, it will go into saturation with a 5 Volt gate voltage  at around 7 amps and a VDS of about 0.5 to 1 Volt.

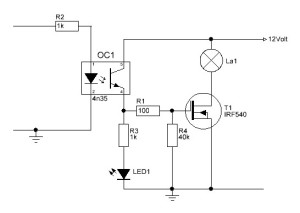
But  it’s RDS(on) will be far from ideal and you will lose 3.5-7 Watts in the FET:  
[](https://arduinodiy.files.wordpress.com/2012/05/buz11.jpg)

Should you however be stuck with a FET like the IRF522 that really needs a high voltage to switch efficiently then use the following circuit:  
[](https://arduinodiy.files.wordpress.com/2012/05/fet2.gif)

Realise though that it is an inverting circuit. A HIGH on the Output of the Arduino will switch the Load Off. Also the 520 and 510 will be more efficient with this circuit.

[](https://arduinodiy.files.wordpress.com/2012/05/mosfet-p1040304.jpg)  
Remember to use a heatsink for the MOSFET if you are using any serious loads  
[](https://arduinodiy.files.wordpress.com/2012/05/mosfet4.jpg)

If you are using this circuit to switch any serious loads, then it is wise to solder some thick wire over the tracks coming from the MOSFET. You will find the print design [here](http://dl.dropbox.com/u/52513692/mosfet4_etch.pdf). This is for direct transfer so it is already mirrorred the right way.

If you have a non logic FET (like the IRF series and need some galvanic separation from your  microcontroller circuit, the following circuit comes in handy:  
[](https://arduinodiy.files.wordpress.com/2012/05/fet520.jpg)

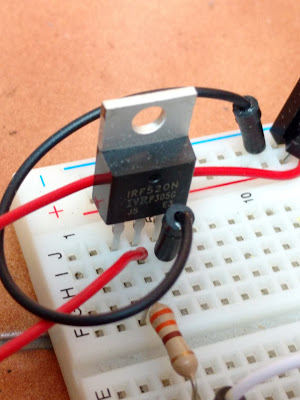
**3.3 Volt levels**

For a long time ‘TTL” meant 5 volt. Nowadays more and more 3.3 Volt boards are available as well in the Arduino series  as in the popular ESP8266 and the raspberry Pi.

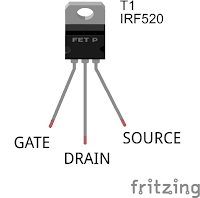
On these boards the [STN4NF03L](http://www.st.com/content/ccc/resource/technical/document/datasheet/04/34/f3/84/3e/6f/4c/eb/CD00002146.pdf/files/CD00002146.pdf/jcr:content/translations/en.CD00002146.pdf) can be a good choice. Not an ideal choice, but a good one. Check section 2.1 figure 4 in the datasheet.

### I Mosfet e Arduino, come pilotare carichi

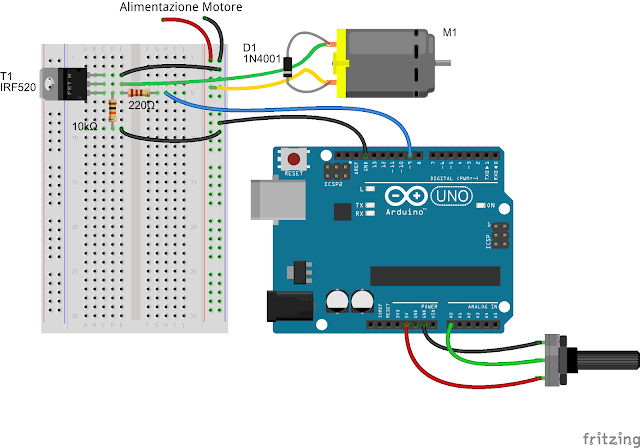
Questo tutorial si pone l'obiettivo di illustrare, in modo semplice, come utilizzare un MOSFET per pilotare carichi con Arduino.  
Infatti grazie a questo componente potremo controllare dispositivi che necessitano una (relativamente) alta quantità di corrente, come strisce led, motori, riscaldatori, solenoidi, e molto altro.

[](https://1.bp.blogspot.com/-chXaPXhuYLo/VyjX8NWlbMI/AAAAAAAACW0/jgA6vP5jtfcIR7MGcrV1PlYXKdxSNQtIwCKgB/s1600/IMG_20160503_185411477.jpg)

Veniamo subito al dunque, il mosfet che ho utilizzato in questo caso è un [IRF520](http://www.vishay.com/docs/91017/91017.pdf), dispositivo molto diffuso, infatti è presente anche nello starter kit ufficiale di Arduino.  
Questo è un mosfet N-CHANNEL di tipo ENANCHMENT, in pratica significa che permette il passaggio di una corrente tra i pin DRAIN e SOURCE solo se applico una tensione adeguata sul pin GATE.

[](https://1.bp.blogspot.com/-Nok0At9lKNE/VypfxU5xo0I/AAAAAAAACX8/Rv6h7dMAFdcmr1vwzxKF0_SQmlYJSG5nACLcB/s1600/MOSFET+PINOUT_bb.png)

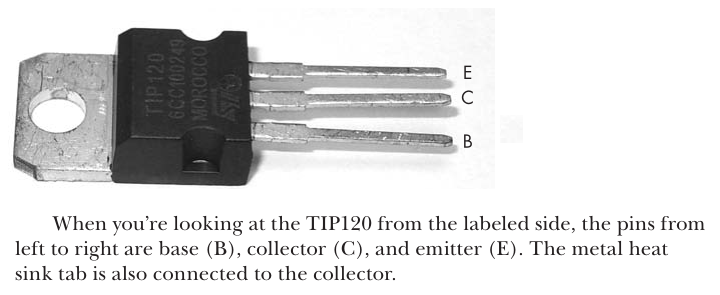
Il massimo passaggio di corrente (9.2 Ampere a 25°C) potrà avvenire quando applicherò sul GATE una tensione di circa 10V (nel caso dell'IRF520).  
Questo significa che se piloterò il gate direttamente con un'uscita digitale di Arduino (5 Volt) il mosfet ammetterà un passaggio di corrente tra drain e source di massimo 2 Ampere (sempre a 25°C, vedi figura 1 di [questo datasheet](http://www.vishay.com/docs/91017/91017.pdf)).  
  
Fortunatamente esistono anche mosfet pensati per lavorare con una tensione sul gate di 5v, sono quelli di tipo LOGIC LEVEL come ad esempio il RFL540.  
  
Un altro fattore da considerare è la frequenza con la quale il nostro Arduino farà commutare il nostro mosfet.  
Infatti per pilotare il nostro mosfet non applicherò sul gate semplicemente 5 Volt o 0 Volt con la funzione "digitalWrite", ma utilizzerò la funzione "analogWrite" e quindi il PWM.  
Qui potrebbe sorgere un problema però, perchè la frequenza del PWM di Arduino sul pin scelto potrebbe non essere la più adatta per la mia applicazione.  
  
Supponiamo di avere una frequenza di PWM estremamente bassa, per ipotesi 1 hertz, dando il comando "analogWrite (pin, 127)", quindi al 50% (il range è 0-255), avrò per mezzo secondo 5 volt e per mezzo secondo 0 volt, capite che se piloterò un motore riuscirò in qualche modo a variarne la velocità, ma con un risultato pessimo.  
  
In Arduino i pin 3,9,10,11 hanno una frequenza di circa 488hz e i pin 5, 6 di 976hz.  
488hz potrebbe essere un valore accettabile, ma dato che Arduino ce lo permette, proviamo ad aumentarla.  
Per aumentare la frequenza del PWM agirò (nello sketch) sui timer del mio microcontrollore portando la frequenza del PWM sul pin 9 a 3906 HZ.  
Ovviamente bisogna considerare che anche il mosfet impiegherà un certo tempo a passare da uno stato all'altro (nel caso del IRF520 30ns per chiudersi e 20ns per aprirsi) ma nella maggior parte dei casi è possibile trascurare la cosa.  
  
Ecco lo schema per collegare il nostro Arduino ad un piccolo motore DC, come potrete notare è presente un diodo sui poli del motore, questo diodo si vede tipicamente quando voglio pilotare una bobina (ad esempio quello delle bobine dei relè),  
La funzione di questo diodo è di limitare la sovratensione provocata dall'induttanza del motore nel momento dell'apertura del mosfet (comportamento tipico dei carichi induttivi).  
Questa sovratensione potrebbe danneggiare il vostro mosfet perchè potrebbe addirittura superare la massima Vds (tensione tra drain e source), mettendo un diodo invece la sovratensione si scaricherà sulla stessa bobina del motore.  
Un bell'approfondimento su questo argomento lo potete trovare a [questo link](http://www.vincenzov.net/tutorial/motoridc/driver.htm).  
  
Ultima cosa da notare è che qualche progettista inserisce in serie una resistenza tra l'uscita di Arduino e il Gate del mosfet, questa dovrebbe servire a proteggere la porta di Arduino da eventuali ritorni che potrebbero danneggiarla.  
Ho messo una resistenza in serie tra l'uscita di Arduino e il mosfet per limitare la corrente sull'uscita in fase di commutazione del mosfet. Chiedo venia per non averci pensato prima! ;).

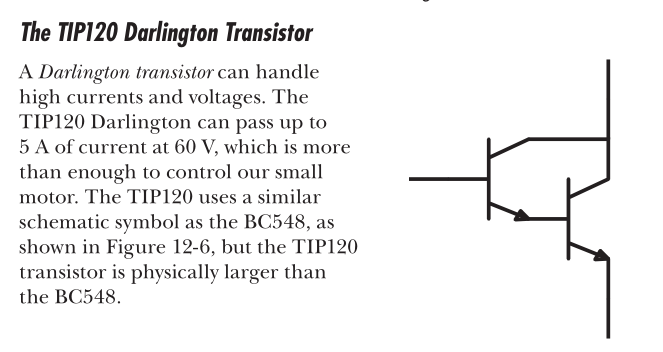
[](https://2.bp.blogspot.com/-1DyrO9W673o/WLXUVykTqII/AAAAAAAADIU/l68756ZoZsQ2PH9y5O7bAjdmId2YFm3VQCLcB/s1600/Mosfet_bb.png)

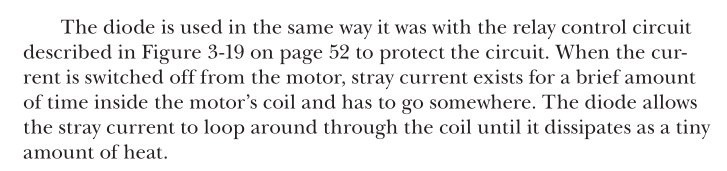
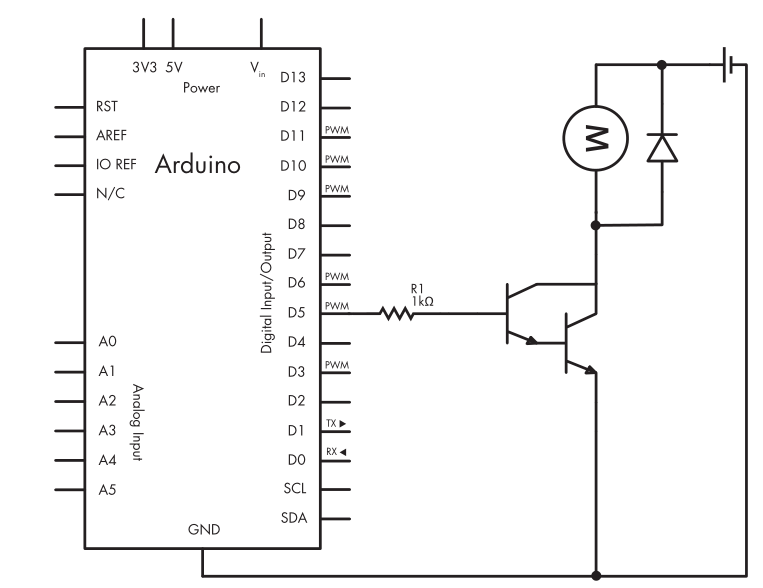
Ed ecco lo sketch che leggerà il valore in ingresso analogico sul pin A0, scalerà il valore e lo porterà in uscita sul mosfet.  
alla riga 8 potete vedere la funzione che serve ad impostare la frequenza di PWM di 3906 hz sui pin 9 e 10

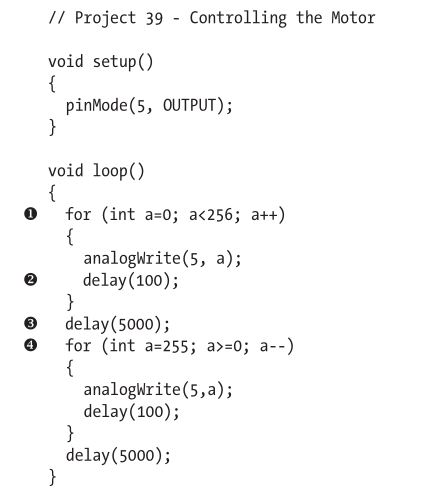
1. int trimmer\_pin = A0;
2. int trimmer\_val;
3. int mosfet\_pin = 9;
4. int mosfet\_val;
5. void setup()
6. {
7. TCCR1B = TCCR1B & 0b11111000 | 0x02;      *//imposta la frequenza del PWM sui pin 9 e 10 a 31250/8 = 3906 hz*
8. Serial.begin(9600);
9. pinMode(trimmer\_pin, INPUT);
10. pinMode(mosfet\_pin, OUTPUT);
11. }
12. void loop()
13. {
14. trimmer\_val = analogRead (trimmer\_pin);
15. Serial.print ("valore ingresso: ");
16. Serial.println (trimmer\_val);
17. mosfet\_val = map (trimmer\_val, 0, 1023, 0, 255);
18. analogWrite (mosfet\_pin, mosfet\_val);
19. Serial.print ("valore pwm uscita: ");
20. Serial.println (mosfet\_val);
21. }

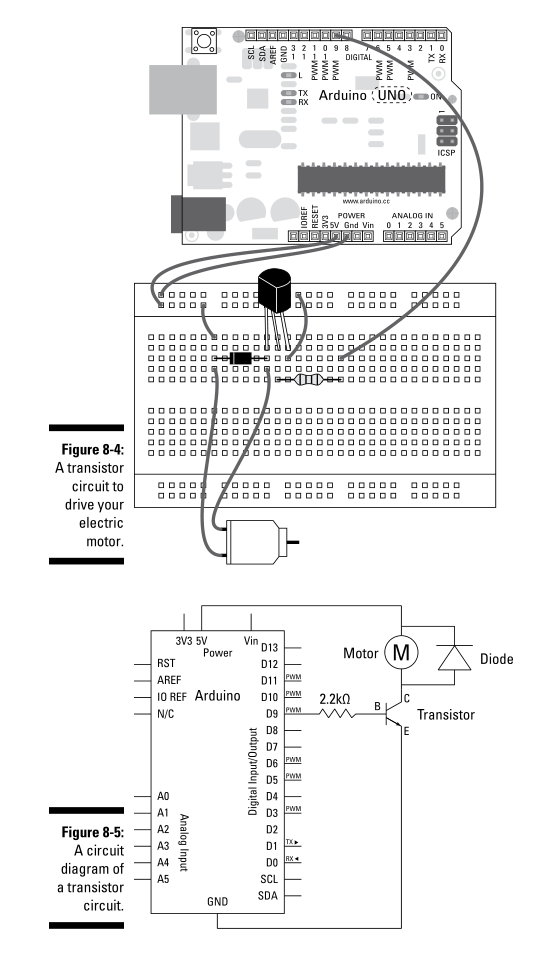
**TIP 120**

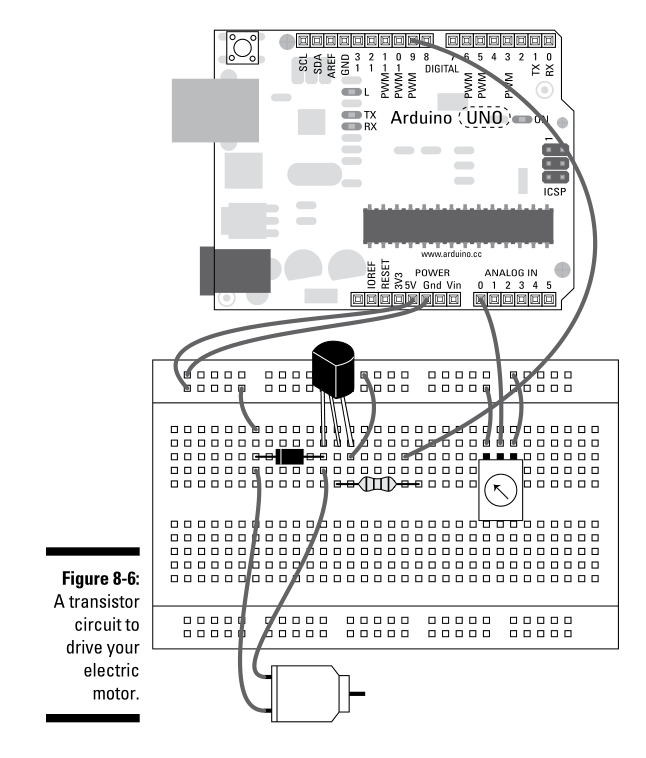
****

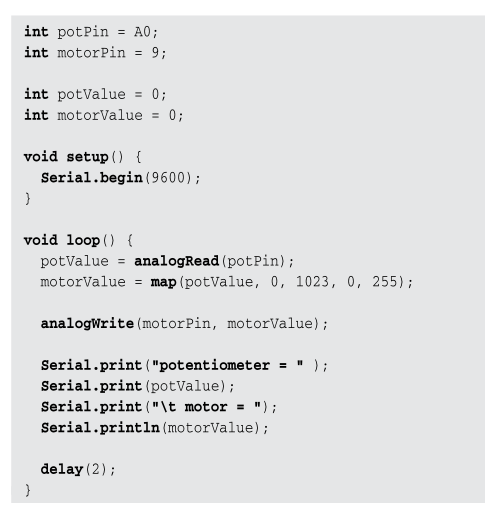
****











# USING A MOSFET TO CONTROL A DC MOTOR

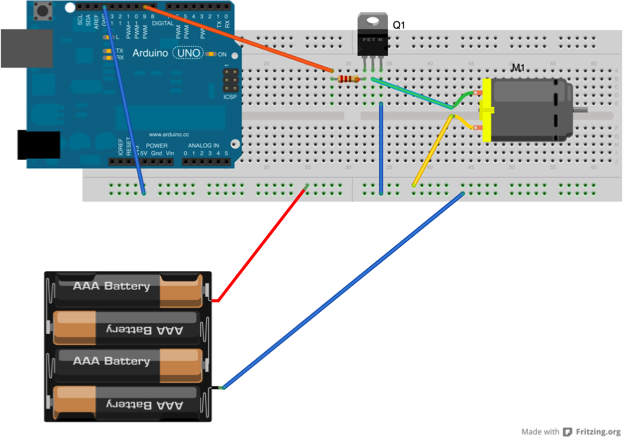
There is a current limit of 40ma for an Arduino pin. So how can we control circuits that require larger currents such as motors or even mains circuits? There are three key ways of doing this and they each have their own advantages and disadvantages:

1. Relays – mechanical or solid-state. This is the easiest way of controlling mains circuits – where simple on/off control is required. Solid-sate relays are typically used where safe operating conditions preclude mechanical action that may create sparks. Opto-isolators are often used to further isolate the mains circuit from the low-level circuit.
2. Where mains circuits are being varied (for example a light dimmer) then a triac can be used often in conjunction with an opto-isolator. They can be used for on-off switching too.
3. The simplest circuit for motor control is the use of a transistor. It’s possible to use a transistor (BJT or a MOSFET) but the MOSFET applies infinitesimal load on the driving circuit so it has a clear advantage.

### Using the supplied MOSFETs to drive a DC motor

Without going into the complexities of the MOSFET we can treat it as a switch. If we apply a potential to the Gate it will switch the current through the Source/Drain junction. So we connect the gate to the Arduino through a 220 ohm current limiting resistor to protect the Arduino. Then we connect the motor to the external power supply and the Drain of the MOSFET. We complete the circuit by connecting up the grounds. In this case the MOSFET controlling the motor is driven from Arduino Pin 9.

### Circuit



### Code

/\*

Speed up the motor

This example shows how to control the speed of a DC motor an LED on pin 9 using the analogWrite() function. This example based on the Arduino Example Fade sketch but modified to use timing instead of the delay() function

\*/

int turns = 0; // how fast the motor runs

int turnAmount = 1; // how many turns the motor makes

unsigned long currentTime;

unsigned long loopTime;

void setup() {

// declare pin 9 to be an output:

   pinMode(9, OUTPUT);

   currentTime = millis();

   loopTime = currentTime;

}

void loop() {

   currentTime = millis();

   if(currentTime >= (loopTime + 20)){

       // set the speed of pin 9:

       analogWrite(9, turns);

       // change the turnings for next time through the loop:

       turns = turns + turnAmount;

       // speed up or slow down the motor

       if (turns == 0 || turns == 255) {

           turnAmount = -turnAmount ;

       }

       if (turns == 0) {

           delay(5000);

       }

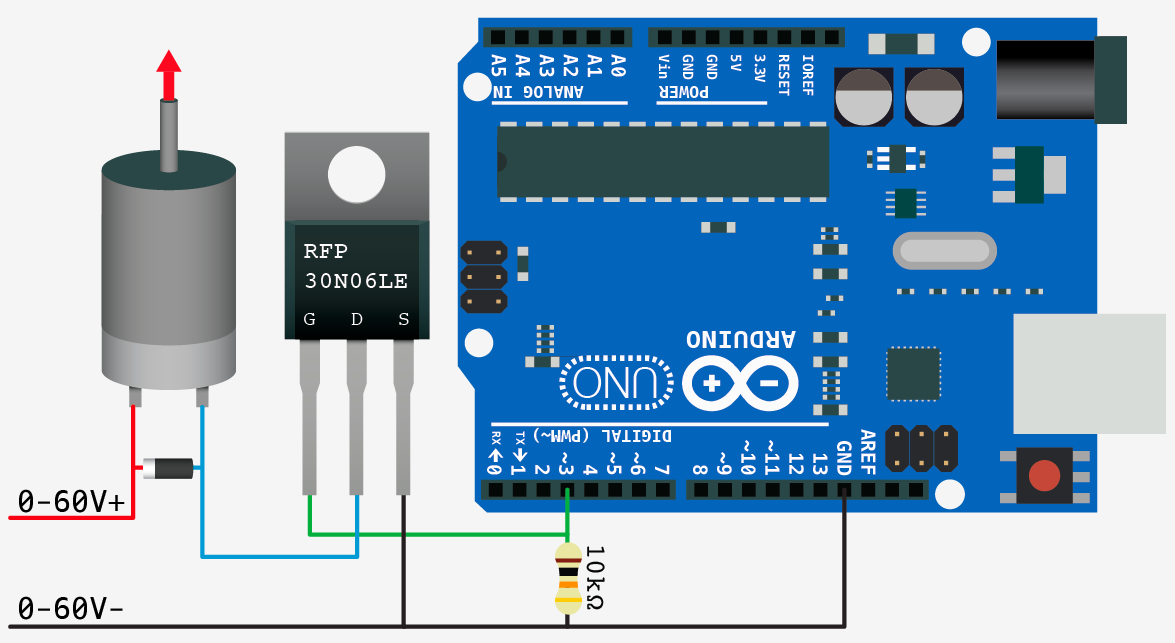
       loopTime = currentTime; // Updates loopTime

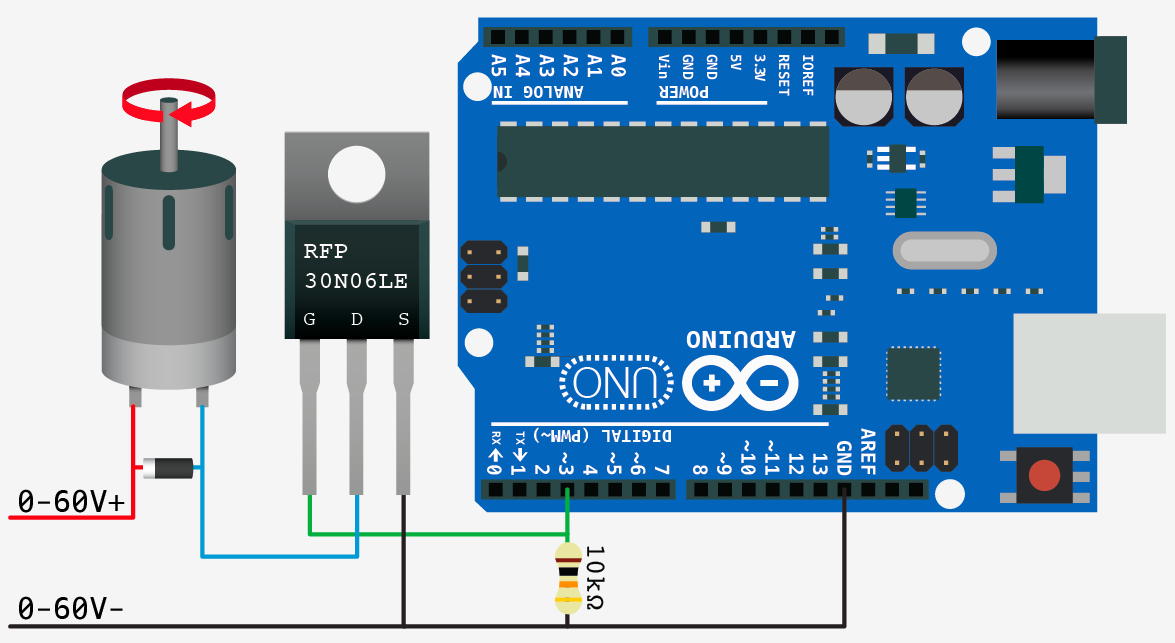
   }

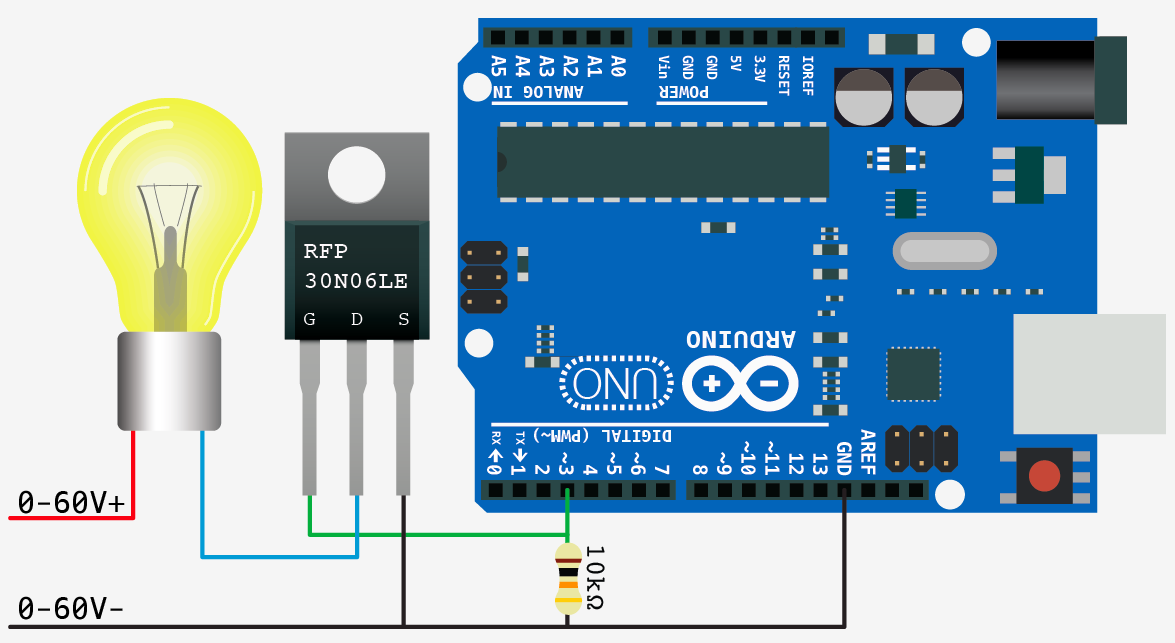
   // Other processing can be done here

}

You need a pull down resistor on the pin that drives the gate. Look at the 10K resistor in the pictures below:







I quote from KyranF:

The resistor holds the gate low when the arduino does not send a high signal. This is here incase the arduino comes loose, or the wiring is bad it will default to off. You don’t want this pin to ever be floating as it will trigger on and off.

Floating MOSFETs are bad, and because the "on" control of the FET is essentially just a capacitor with very low capacitance, it is quite easy for it to float up and turn itself on.

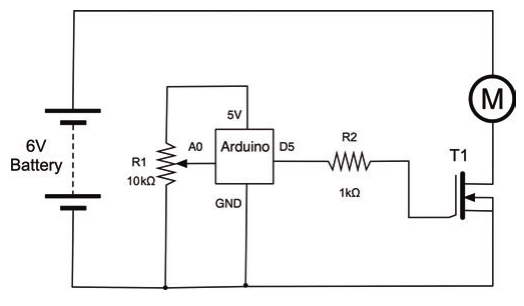
This situation will only really happen in your arduino program if you make the output pin an Input by mistake, or during power off/on/restart states. The ATMEGA328P on the Uno makes all it's pins go into high impedance state during power cycle, which is a prime opportunity for the gate of that FET to float high.

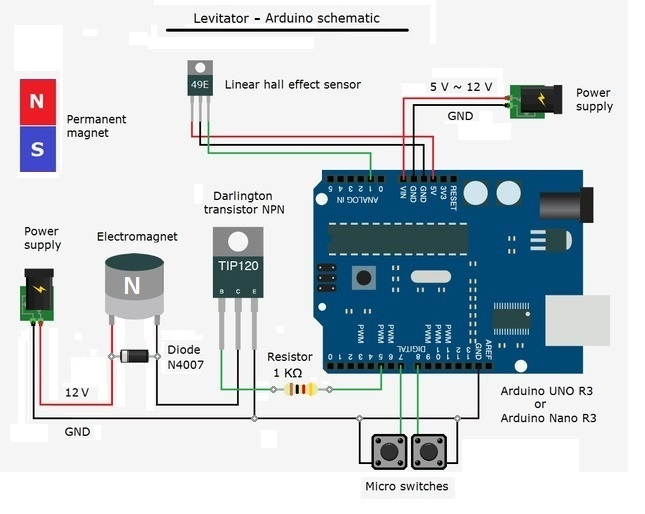
The resistor ensures there is always a known state, and only an active output HIGH from the Arduino will cause it to actually turn on.

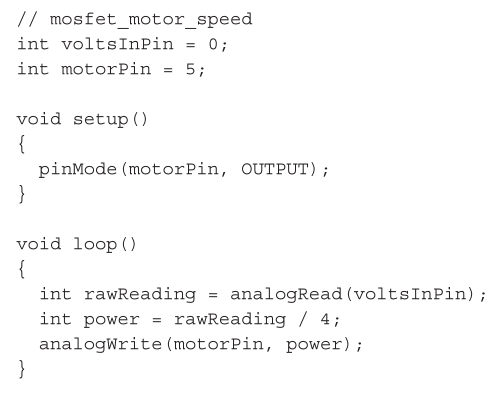
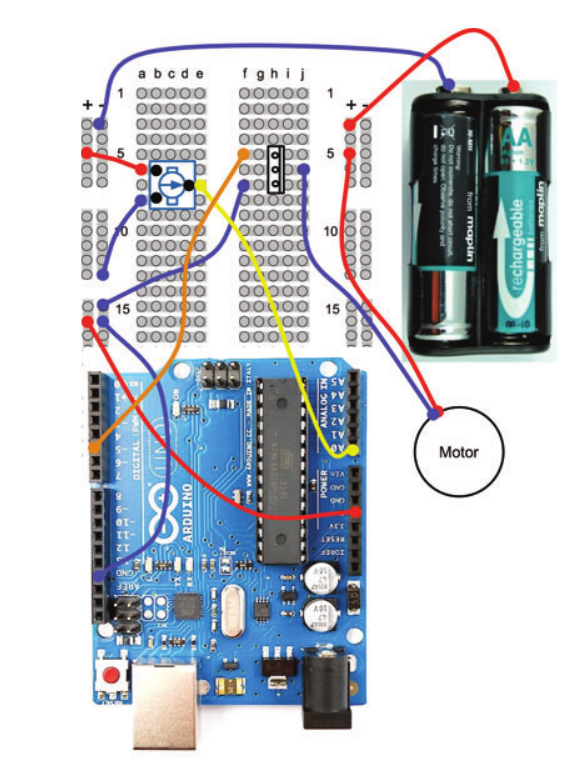
For your third question - MOSFET gates only "use" current for a brief time during the ON period, to charge the gate capacitor. The Arduino's 40mA output maximum per pin is not going to be an issue. It WOULD be an issue if the FET was a BJT Transistor instead, as those will constantly draw current into the base in order to operate. MOSFETs work differently, and do not consume current to have them be "on" constantly.

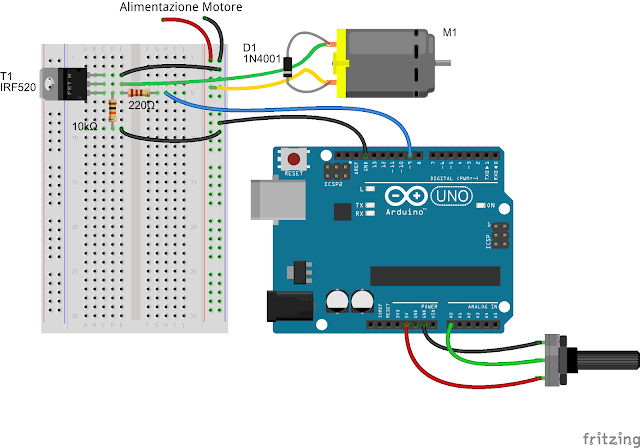
Putting a 10K ohm resistor is also too high in general, it will slow down the ON/OFF time of the FET considerably, and cause major switching losses if you are doing any reasonable frequency PWM. Use something like 100 Ohms if you want to put a resistor there. Putting a resistor there may not be needed for a MOSFET, but it IS recommended to reduce the possibility of inductive feedback into the microcontroller and other forms of dirty business related to switching an inductive load like a motor.

Add the diode as a flyback protection or you are going to have to add a new MOSFET + diode very soon!!!!



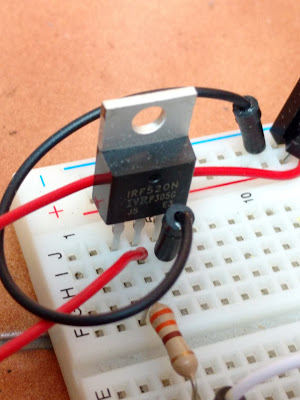




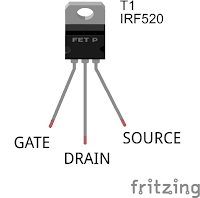


**I Mosfet e Arduino, come pilotare carichi**

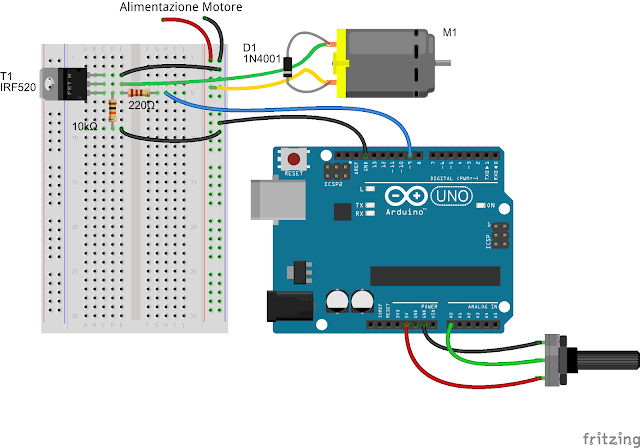
Ciao!  
  
Questo tutorial si pone l'obiettivo di illustrare, in modo semplice, come utilizzare un MOSFET per pilotare carichi con Arduino.  
Infatti grazie a questo componente potremo controllare dispositivi che necessitano una (relativamente) alta quantità di corrente, come strisce led, motori, riscaldatori, solenoidi, e molto altro.  
  
Non starò qui a spiegarvi la [teoria che sta dietro ai transistor di tipo MOSFET](https://it.wikipedia.org/wiki/MOSFET), primo perché non sono la persona più adatta, e secondo perché lo scopo di questo post è solo di avere un primo approccio pratico con questo componente. Vedremo solo alcune dritte pratiche ed alcune caratteristiche con cui dovremo interfacciarsi già dai primi usi.

[](https://1.bp.blogspot.com/-chXaPXhuYLo/VyjX8NWlbMI/AAAAAAAACW0/jgA6vP5jtfcIR7MGcrV1PlYXKdxSNQtIwCKgB/s1600/IMG_20160503_185411477.jpg)

Veniamo subito al dunque, il mosfet che ho utilizzato in questo caso è un [IRF520](http://www.vishay.com/docs/91017/91017.pdf), dispositivo molto diffuso, infatti è presente anche nello starter kit ufficiale di Arduino.  
Questo è un mosfet N-CHANNEL di tipo ENANCHMENT, in pratica significa che permette il passaggio di una corrente tra i pin DRAIN e SOURCE solo se applico una tensione adeguata sul pin GATE.

[](https://1.bp.blogspot.com/-Nok0At9lKNE/VypfxU5xo0I/AAAAAAAACX8/Rv6h7dMAFdcmr1vwzxKF0_SQmlYJSG5nACLcB/s1600/MOSFET+PINOUT_bb.png)

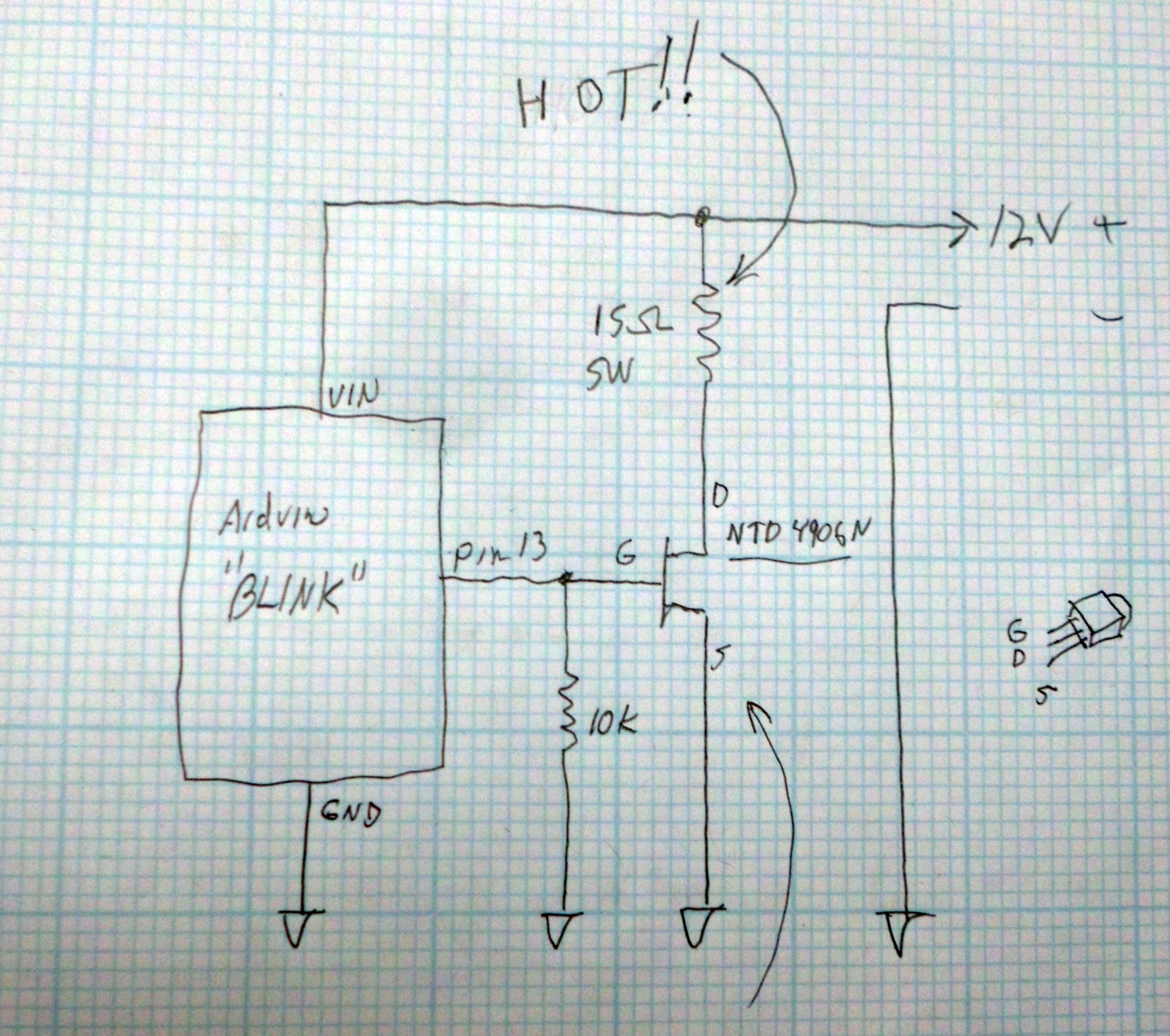
Il massimo passaggio di corrente (9.2 Ampere a 25°C) potrà avvenire quando applicherò sul GATE una tensione di circa 10V (nel caso dell'IRF520).  
Questo significa che se piloterò il gate direttamente con un'uscita digitale di Arduino (5 Volt) il mosfet ammetterà un passaggio di corrente tra drain e source di massimo 2 Ampere (sempre a 25°C, vedi figura 1 di [questo datasheet](http://www.vishay.com/docs/91017/91017.pdf)).  
  
Fortunatamente esistono anche mosfet pensati per lavorare con una tensione sul gate di 5v, sono quelli di tipo LOGIC LEVEL come ad esempio i [IRLZ44N](http://www.irf.com/product-info/datasheets/data/irlz44n.pdf), i [FQP30N06L](https://www.fairchildsemi.com/datasheets/FQ/FQP30N06L.pdf) oppure i [STP36NE06](http://pdf.datasheetcatalog.com/datasheet/SGSThomsonMicroelectronics/mXyytux.pdf).  
  
  
  
Un altro fattore da considerare è la frequenza con la quale il nostro Arduino farà commutare il nostro mosfet.  
Infatti per pilotare il nostro mosfet non applicherò sul gate semplicemente 5 Volt o 0 Volt con la funzione "digitalWrite", ma utilizzerò la funzione "analogWrite" e quindi il PWM.  
Qui potrebbe sorgere un problema però, perchè la frequenza del PWM di Arduino sul pin scelto potrebbe non essere la più adatta per la mia applicazione.  
  
Supponiamo di avere una frequenza di PWM estremamente bassa, per ipotesi 1 hertz, dando il comando "analogWrite (pin, 127)", quindi al 50% (il range è 0-255), avrò per mezzo secondo 5 volt e per mezzo secondo 0 volt, capite che se piloterò un motore riuscirò in qualche modo a variarne la velocità, ma con un risultato pessimo.  
  
In Arduino i pin 3,9,10,11 hanno una frequenza di circa 488hz e i pin 5, 6 di 976hz.  
488hz potrebbe essere un valore accettabile, ma dato che Arduino ce lo permette, proviamo ad aumentarla.  
Per aumentare la frequenza del PWM agirò (nello sketch) sui timer del mio microcontrollore portando la frequenza del PWM sul pin 9 a 3906 HZ.  
Ovviamente bisogna considerare che anche il mosfet impiegherà un certo tempo a passare da uno stato all'altro (nel caso del IRF520 30ns per chiudersi e 20ns per aprirsi) ma nella maggior parte dei casi è possibile trascurare la cosa.  
  
  
Ecco lo schema per collegare il nostro Arduino ad un piccolo motore DC, come potrete notare è presente un diodo sui poli del motore, questo diodo si vede tipicamente quando voglio pilotare una bobina (ad esempio quello delle bobine dei relè),  
La funzione di questo diodo è di limitare la sovratensione provocata dall'induttanza del motore nel momento dell'apertura del mosfet (comportamento tipico dei carichi induttivi).  
Questa sovratensione potrebbe danneggiare il vostro mosfet perchè potrebbe addirittura superare la massima Vds (tensione tra drain e source), mettendo un diodo invece la sovratensione si scaricherà sulla stessa bobina del motore.  
Un bell'approfondimento su questo argomento lo potete trovare a [questo link](http://www.vincenzov.net/tutorial/motoridc/driver.htm).  
  
Ultima cosa da notare è che qualche progettista inserisce in serie una resistenza tra l'uscita di Arduino e il Gate del mosfet, questa dovrebbe servire a proteggere la porta di Arduino da eventuali ritorni che potrebbero danneggiarla.  
Personalmente, pilotando piccoli carichi, non ho notato questa necessità, ma se volete potete mettere una resistenza da un centinaio di ohm.  
  
EDIT:  
come consigliato da un lettore nei commenti, ho messo una resistenza in serie tra l'uscita di Arduino e il mosfet per limitare la corrente sull'uscita in fase di commutazione del mosfet. Chiedo venia per non averci pensato prima! ;).

[](https://2.bp.blogspot.com/-1DyrO9W673o/WLXUVykTqII/AAAAAAAADIU/l68756ZoZsQ2PH9y5O7bAjdmId2YFm3VQCLcB/s1600/Mosfet_bb.png)

La resistenza in serie serve. I mosfet dell'AVR, come tutti i fet, si comportano come resistenze, si scaldano in funzione della corrente che ci passa. Il mosfet esterno, come tutti i mosfet, assorbe (meglio, immagazzina) corrente durante la commutazione. più veloce commuti, più corrente ci passa e più calore si svilupperà nel mosfet dell'AVR, distruggendolo. D'altro canto più sarà alta la resistenza più tempo ci metterà il mosfet esterno a commutare e più si scalderà

Ed ecco lo sketch che leggerà il valore in ingresso analogico sul pin A0, scalerà il valore e lo porterà in uscita sul mosfet.  
alla riga 8 potete vedere la funzione che serve ad impostare la frequenza di PWM di 3906 hz sui pin 9 e 10

1. int trimmer\_pin = A0;
2. int trimmer\_val;
3. int mosfet\_pin = 9;
4. int mosfet\_val;
5. void setup()
6. {
7. TCCR1B = TCCR1B & 0b11111000 | 0x02;      *//imposta la frequenza del PWM sui pin 9 e 10 a 31250/8 = 3906 hz*
8. Serial.begin(9600);
9. pinMode(trimmer\_pin, INPUT);
10. pinMode(mosfet\_pin, OUTPUT);
11. }
12. void loop()
13. {
14. trimmer\_val = analogRead (trimmer\_pin);
15. Serial.print ("valore ingresso: ");
16. Serial.println (trimmer\_val);
17. mosfet\_val = map (trimmer\_val, 0, 1023, 0, 255);
18. analogWrite (mosfet\_pin, mosfet\_val);
19. Serial.print ("valore pwm uscita: ");
20. Serial.println (mosfet\_val);
21. }



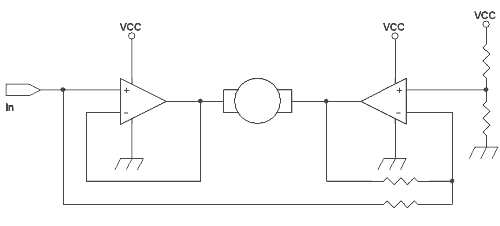
## *Il pilotaggio lineare*

Il pilotaggio lineare di un motore si ottiene quando l'alimentazione è collegato all'uscita di un circuito elettronico che genera una tensione continua, per esempio un amplificatore operazionale di potenza adeguata oppure un regolatore di tensione.

In prima approssimazione si può supporre che tanto più è grande la tensione che alimenta il motore, tanto più il motore ruota velocemente: è quindi sufficiente cambiare la tensione in ingresso all'amplificatore per cambiare la velocità di rotazione.

Qualora l'amplificatore possa erogare tensioni positive e negative, collegando il motore tra uscita dell'amplificatore e massa è anche possibile ottenere l'inversione del verso di rotazione. Da notare che per fare questo è necessario disporre di due sorgenti di alimentazione, una positiva ed una negativa, a volte difficili da ottenere quando le potenze in gioco sono rilevanti.

In alternativa all'uso di una alimentazione duale è possibile usare due amplificatori lineari nella cosiddetta configurazione a ponte. I fili di alimentazione del motore devono essere collegati alle uscite degli amplificatori a loro volta connessi in modo tale che all'aumento della tensione su una uscita corrisponde la diminuzione dell'altra.



L'esempio, solo di principio, mostra l'amplificatore di sinistra configurato come inseguitore di tensione (Vout(1) = Vin) e quello di destra come differenziale (se i quattro resistori sono uguali, Vout(2) =  Vcc - Vin).

|  |  |  |  |
| --- | --- | --- | --- |
| **Vin** | **Vout(1)** | **Vout(2)** | **Vm** |
| 0 | 0 | Vcc | -Vcc |
| Vcc/2 | Vcc/2 | Vcc/2 | 0 |
| Vcc | Vcc | 0 | +Vcc |
| 1V | 1V | (Vcc - 1) V | 2V |

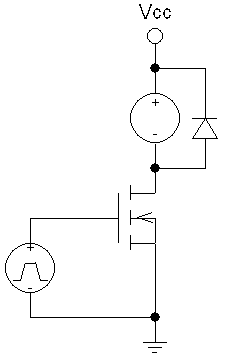
La tensione Vm effettivamente applicata al motore è quella riportata nella tabella ed è quindi possibile impostare modulo e verso della Vm pur disponendo solo di alimentazione positiva.

Il difetto del pilotaggio lineare è legato alla notevole dissipazione di potenza che lo rende difficoltoso da usarsi quando la potenza del motore supera i pochi watt: infatti una parte considerevole della tensione di alimentazione cade sullo stadio di uscita dell'amplificatore. Nella configurazione ad un solo amplificatore la situazione peggiore si ha quando la tensione di uscita è pari al 50% di quella di alimentazione, metà dell'energia viene infatti dissipata sotto forma di calore dall'amplificatore.

Da notare che è necessario utilizzare i [diodi di protezione](https://www.vincenzov.net/tutorial/motoridc/driver.htm#diodi) (non indicati nello schema), a causa delle sovratensioni causate dalla commutazione delle spazzole sul collettore.

## *Il pilotaggio on/off*

Il pilotaggio on/off è realizzato attraverso un transistor che lavora in commutazione cioè o in conduzione o come circuito aperto, come un interruttore. Nello schema ho inserito anche un diodo la cui funzione è descritta nel [prossimo paragrafo](https://www.vincenzov.net/tutorial/motoridc/driver.htm#diodi).



In questa configurazione la potenza dissipata sul transistor (uguale al prodotto di corrente per tensione ai suoi capi) è sempre minima e tutta l'energia prelevata dall'alimentazione è effettivamente usata dal motore. Infatti:

* quando il transistor conduce la tensione ai suoi capi è prossima a 0 volt mentre quella ai capi del motore è praticamente pari alla Vcc. Il motore quindi ruota alla massima velocità
* quando il transistor è aperto la corrente nel MOS è evidentemente nulla e quindi nulla la potenza dissipata dal MOS (e anche fornita a motore);

Il limite di questa tecnica di pilotaggio è evidente: il motore è fermo oppure ruota alla massima velocità.

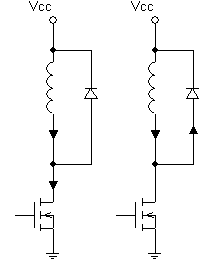
## *Perché si usano sempre tanti diodi ?*

L' avvolgimento del motore è sostanzialmente in induttore, cioè un oggetto che tende a mantenere costante la corrente che in esso scorre.

Quando il transistor rappresentato nello schema precedente  si chiude, la corrente raggiunge il valore a regime in un certo tempo, secondo una curva esponenziale dipendente dal rapporto di La ed Ra del circuito equivalente. Questo fatto non causa, in prima approssimazione, particolari problemi.

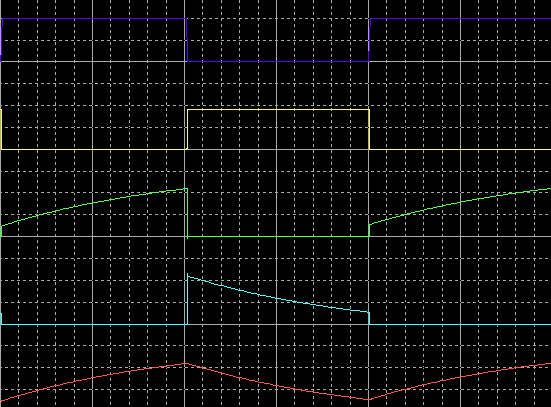
Quando un transistor si apre, la corrente istantaneamente dovrebbe andare a zero; l'induttore tende però ad impedire questo repentina diminuzione e per fare questo tende a far salire la tensione sul collettore del transistor (immaginate il transistor che, improvvisamente, sia diventato una resistenza molto elevata in cui l'induttore tenta di far passare una corrente: per la legge di ohm, la tensione deve salire). La tensione arriva facilmente a centinaia di volt, danneggiando il transistor stesso. Tale tensione è spesso chiamata "di fly-back".

Per evitare questo fenomeno distruttivo viene inserito in parallelo alla bobina del motore un diodo che fornisce alla corrente una via alternativa a quella del transistor nel momento in cui questo si apre.



Il catodo va connesso alla tensione di alimentazione: in pratica la corrente va "in salita". Nello schema è rappresentata a sinistra la situazione in cui il transistor è in conduzione (la corrente attraversa l'avvolgimento del motore, rappresentato dal solo induttore, ed il transistor; nel diodo non passa corrente in quanto polarizzato inversamente). A destra invece l'andamento della corrente subito dopo l'apertura del transistor: la stessa corrente che prima attraversava il transistor ora passa nel diodo.

Ovviamente quest'ultima situazione si esaurisce abbastanza rapidamente, mancando generatori in grado di mantenere nel tempo il passaggio di corrente. Questo tempo è legato al rapporto tra l'induttanza e la resistenza equivalente del motore e, in situazioni tipiche, è dell'ordine delle decine di millisecondi.



Il grafico mostra l'andamento idealizzato e non in scala delle tensioni e delle correnti nel caso in cui la velocità del motore si costante; quello rappresentato è un ciclo formato da accensione, spegnimento e accensione. Dall'alto troviamo:

* (in blu) La tensione (o la corrente) di comando: un segnale alto significa motore acceso, un segnale basso significa motore spento
* (In giallo) La tensione presente di drain del MOS (o sul collettore del transistor). Ovviamente ha un andamento opposto a quello della tensione di comando. Non ho indicato tempi di ritardo o di salita in quanto praticamente irrilevante in questi casi.
* (in verde) La corrente di drain o di collettore. Si noti il tipico andamento esponenziale di carica di un induttore. Quando il transistor è spento ovviamente la corrente è nulla
* (in azzurro) La corrente nel diodo di ricircolo, nulla quando il transistor è on, decrescente secondo una curva esponenziale
* (in rosso) La corrente nel motore, ovviamente pari alla somma delle due precedenti e caratterizzata dall'assenza di discontinuità, essendo schematizzabile come un induttore

I diodi da utilizzare i queste applicazioni devono avere due caratteristiche fondamentali:

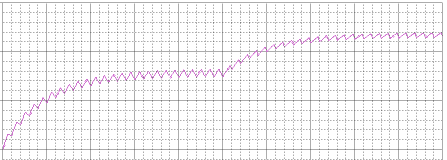
* Essere veloci, cioè essere capaci di passare in poco tempo dalla condizione di non passaggio di corrente a quella di conduzione e viceversa. In genere sono richiesti tempi di intervento dell'ordine delle decine di nanosecondi.
* Essere capaci di gestire correnti elevate in quanto al momento dello spegnimento tutta la corrente del motore, in genere ampere, passa nei diodi.

Per l'hobbista non è sempre facile recuperare diodi adatti ma, nel caso, meglio un 1N4001 o un 1N4148 che nulla.

## *Il pilotaggio pwm*

Questa tecnica permette di modificare la velocità pur assicurando un rendimento energetico elevato.

Il circuito è lo stesso utilizzato nel funzionamento on/off, già descritto nel [precedente paragrafo](https://www.vincenzov.net/tutorial/motoridc/driver.htm#onoff). L'idea è però quella di pilotare la base del transistor con un'onda quadra: se la commutazione è piuttosto frequente (qualche KHz) a causa della presenza di un induttore la corrente media è sostanzialmente costante e proporzionale al duty cycle del segnale sulla base del MOS.



Il grafico mostra l'andamento della corrente nel motore applicando, nella prima metà, un'onda quadra (quindi un duty cycle del 50%) e, nella seconda metà, un segnale rettangolare con duty cycle del 75%. Il grafico è idealizzato e nell'ipotesi di motore fermo (si noti la corrente iniziale nulla). Si noti anche il caratteristico andamento "seghettato" ma sostanzialmente costante

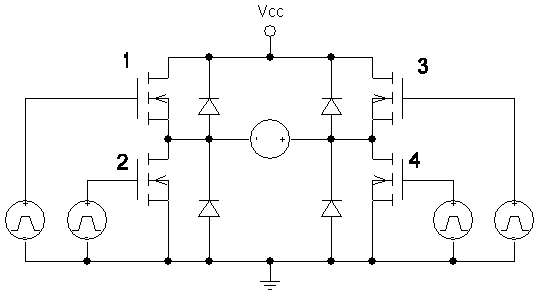
Una osservazione è relativa al fatto che non tutti i motori sono adatti al pilotaggio PWM, soprattutto alla frequenze più elevate: infatti la perdita nei circuiti magnetici è proporzionale alla frequenza e potrebbe divenire inaccettabile, soprattutto se la frequenza di pilotaggio supera la decina di KHz. Anche il transistor, se inadatto all'applicazione o mal pilotato, può presentare surriscaldamenti a frequenza superiori alla decina di KHz

Una nota di colore: un effetto del pilotaggio pwm (se la frequenza di pilotaggio è inferiore a 20KHz) è il caratteristico fischio prodotto. In tutti i film di fantascienza questo suono indica il movimento dei robot...

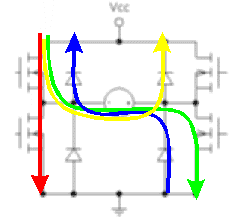
## *Il ponte ad H*

Un ponte ad H è costituito da quattro interruttori e permette il funzionamento bidirezionale del motore in presenza di una alimentazione singola.

Il nome deriva dalla somiglianza del circuito alla lettera maiuscola H, dove il motore costituisce il segmento orizzontale ed i quattro transistor i quattro segmenti verticali. L'idea è già stata [esposta in precedenza](https://www.vincenzov.net/tutorial/motoridc/driver.htm#ponte) ma in questo caso il segnale di comando è di tipo on/off oppure pwm e non lineare.



Lo schema mostra come i quattro transistor sono connessi. In genere i due transistor inferiori (2 e 4 nello schema) sono detti di sink in quanto assorbono la corrente proveniente da motore oppure low side switch; i due transistor connessi direttamente alla Vcc sono detti di source oppure high side switch.



A seconda di quali transistor sono in attivi, abbiamo diversi possibili percorsi per la corrente, illustrati nell'immagine sopra riportata:

* (linea verde) Se è attiva un transistor di sink ed uno di source appartenenti a lati opposti del ponte, abbiano passaggio di corrente nel motore. In questo caso il motore è in rotazione in un verso; per ottenere la rotazione opposta è evidentemente necessario attivare la coppia simmetrica.
* (linea rossa) Se è attiva un transistor di sink ed uno di source appartenenti allo stesso lato del ponte abbiamo un corto circuito. Inutile dire che questa situazione deve essere evitata nel modo più assoluto in quanto porterebbe alla distruzione del ponte o dell'alimentazione in tempi brevissimi.
* (linea blu) Se tutti i transistor sono spenti non abbiamo maglie in cui possa passare la corrente fornita dall'alimentatore. Quella indicata è la via che l'eventuale corrente accumulata dall'induttore percorre: si tratta ovviamente di un fenomeno temporaneo ma che deve necessariamente essere previsto per i [quanto già illustrato](https://www.vincenzov.net/tutorial/motoridc/driver.htm#diodi). Terminata la scarica dell'induttore non si ha più passaggio di corrente e se il motore era precedentemente in moto si arresta lentamente  a causa degli attriti meccanici.
* (linea gialla) Se è attivo almeno uno dei transistor di source e nessuno di quelli di sink non vi sono percorsi in cui passa la corrente fornita dall'alimentatore. La differenza rispetto alla situazione precedente è il sostanziale cortocircuito che si viene a creare ai capi del motore: infatti la tensione ai capi del motore è pari alla tensione diretta del diodo sommata a quella di conduzione del transistor. L'effetto è una vigorosa azione frenante causata dalla presenza del generatore equivalente Eg già citato a proposito delle [equazioni di funzionamento](https://www.vincenzov.net/tutorial/motoridc/driver.htm#equazioni) e dalla conseguente corrente generata dal motore.

Il ponte ad H è utilizzabile in funzionamento on/off semplicemente applicano gli opportuni segnali per ottenere rotazione in un verso o nel verso opposto, frenata rapida o frenata lenta.

In alternativa è possibile utilizzare un segnale pwm per pilotare il ponte, secondo due modalità:

* **Sign/magnitude pwm**: occorre un segnale pwm per l'ampiezza ed uno costante per il verso di rotazione. Il duty cycle del segnale pwm varia tra lo 0% (motore fermo) ed il 100% (motore a piena velocità); esso deve essere applicato direttamente ad una coppia diagonale di transistor mentre l'altra deve essere spenta.  l'ingresso di segno serve per scegliere a quale coppia di transistor applicare il segnale pwm.
* **Locked anti-phase pwm**: occorre un solo segnale pwm tale che quando è altro è in conduzione una coppia di transistor, quando è basso l'altra. L'effetto è quello di forzare la corrente nel motore prima in un verso, poi nell'altra: a causa dell'induttanza si ha una stabilizzazione intorno al valor medio. In particolare quando un duty cycle è del 50% la corrente è nulla, la tensione media nulla e quindi il motore è fermo. Quando il duty cycle tende al 100% il motore ruota a piena velocità in un verso, quando è dello 0%, ruota a piena velocità nell'altro verso.

Va evitata nella maniera più assoluta la conduzione contemporanea dei due transistor dello stesso ramo in quanto ciò causerebbe una corrente elevata tra alimentazione e massa. Per evitare il problema in genere si attende un certo tempo tra lo spegnimento di una coppia di transistor e l'accensione di un'altra (dead time).

Da notare che a volte nella configurazione ad H non si usano diodi discreti in quanto sono già integrati all'interno dei transistor: questa soluzione va però valutata attentamente perché a volte tali diodi non sono sufficientemente veloci.

Un problema di questo circuito è legato alla difficoltà pratica che si ha nel pilotare i due transistor superiori, soprattutto se la tensione di alimentazione è elevata: infatti i loro emettitori (o drain) non sono connessi a massa ma ad una tensione variabile. Come effetto si ha che:

* l'ampiezza degli impulsi di pilotaggio non deve essere riferita a massa ma all'emettitore, con notevoli complicazioni circuitali
* l'ampiezza degli impulsi deve essere superiore alla tensione di alimentazione del motore, soprattutto nel caso dei MOS

La soluzione prevede in genere traslatori di livello associati a circuiti a pompa di carica (per ottenere tensioni superiori a quella di alimentazione) e, per le tensioni più elevate, isolamento ottico. In commerci si trovano appositi circuiti integrati che integrano queste funzioni.