

Testing di un decoder con Arduino

USO DEI REGISTRI INTERNI DI ARDUINO

Sommario

1. DESCRIZIONE GENERALE.....	2
2. SCOPO.....	2
3. ACRONIMI E DEFINIZIONI	2
4. LA MANUTENZIONE PREVENTIVA E CORRETTIVA	2
a. Polling.....	2
b. Interrupt.....	2
5. IL DECODER	3
6. I REGISTRI DI ARDUINO.....	3
7. MICROCONTROLLORE ATmega328	4
8. I REGISTRI DI ARDUINO.....	4
a. DDRx (dove x sta per B, C e D).....	4
b. PORTx (dove x sta per B, C e D)	4
c. PINx (dove x sta per B, C e D)	4
9. TRASFERIMENTO DEI DATI: I BUS.....	5
a. BUS a trasmissione parallela	5
b. BUS a trasmissione seriale.....	5
a. BUS dati	5
b. BUS indirizzi.....	5
c. BUS controlli.....	6
10. ANALISI	6
a. Funzionamento Decoder 2:4 CD74HC139	6
b. Tabella della verità del decoder	6
11. PROGETTAZIONE	7
12. TESTING	12
a. Obbiettivi	12
b. Montaggio su breadboard	12
13. DUBBI	13
14. CONCLUSIONI	13
15. RIFERIMENTI BIBLIOGRAFICI E SITOGRAFICI	13

1. DESCRIZIONE GENERALE

L'enunciato del problema è il seguente:

“Effettuare la diagnosi di un decoder 2:4 simulando le 4 combinazioni e riportando il risultato, nei registri di Arduino”

2. SCOPO

Lo scopo di questa relazione è mettere in pratica le nozioni teoriche sui registri di Arduino apprese in classe, per effettuare la risoluzione del problema.

3. ACRONIMI E DEFINIZIONI

- **Task:** per task, o processo, si intende un programma in esecuzione.
- **Pin:** è un contatto presente nei circuiti elettronici di tutti i tipi.
- **Logica negativa:** viene utilizzato per indicare una codifica delle informazioni binarie nella quale la rappresentazione dei significati vero e falso è invertita rispetto a quella usuale.
- **Core:** il core di un microcontrollore è il suo nucleo elaborativo.
- **MCU:** MicroController Unit o Unità del MicroControllore.
- **Datasheet:** sono la documentazione che riassume le caratteristiche di un componente elettronico, sono reperibili online sui siti dei costruttori del componente.
- **Jumper Wire:** cavi elettrici, con i due capi sguainati, oppure cavi con dei connettori o pin alle proprie estremità.
- **Breadboard:** tavoletta forellata, utilizzata in elettronica per creare circuiti di cui non si vogliono saldare i componenti; possiede 4 file di fori collegati in orizzontale (alimentazione, + e -) e 10 righe di fori collegati in verticale (per i componenti).
- **IDE:** l'Integrated Development Environment o Ambiente di Sviluppo Integrato è un software che in fase di programmazione aiuta i programmatori nello sviluppo di un codice o programma.
- **CPU:** la Central Processing Unit o Unità di Elaborazione Centrale è un tipo di microprocessore che, nei computer, si occupa della maggior parte delle operazioni.

4. LA MANUTENZIONE PREVENTIVA E CORRETTIVA

La manutenzione consiste nell'agire su un dato componente prima che esso si guasti, agendo, quindi, d'anticipo oppure dopo aver diagnosticato un problema. L'azione di manutenzione consiste nell'avviare una diagnostica della memoria, che può essere eseguita in 2 casi:

a. Polling

È una manutenzione di tipo preventivo: viene eseguita una scansione o verifica delle periferiche o **task** per accertarsi che siano tutte perfettamente funzionanti.

b. Interrupt

È una manutenzione di tipo correttivo: da una periferica o task qualsiasi viene richiesta un controllo; l'interrupt può essere di due tipi:

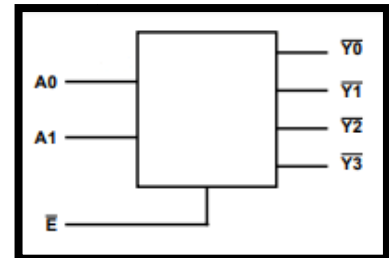
- **Maskable:** un interrupt che “viene messo in atto” solo quando la *Task* giunge al termine.
- **Non Maskable:** un interrupt di priorità maggiore che agisce immediatamente sulla *Task*, uno dei casi in cui il *Not Maskable Interrupt (NMI)* entra in funzionamento, è un guasto fisico all’hardware.

5. IL DECODER

Il decoder in ambito informatico è un dispositivo composto da una rete combinatoria che ci permette di “Decodificare” dei valori; nel nostro caso, il decoder, decodifica i valori binari in input e li converte nei corrispettivi valori decimali in output.

Il decoder è dotato di un **Pin** denominato Enable (Abilitazione) il quale, tramite l’immissione di un valore High o Low (1-0), determina lo stato di funzionamento del decoder. Nei decoder che utilizzano ingressi di tipo attivo alto, esso infatti sarà abilitato a svolgere il suo normale funzionamento, se il pin di Enable riceverà 1 logico (High), disabilitato invece, se riceverà lo 0 logico (Low). (Figura 1)

Figura 1 - Decoder 2:4



Oltre al pin di Enable il decoder dispone di vari ingressi e uscite, nel nostro caso il decoder è di tipo 2:4, perciò avrà due ingressi e 4 uscite. (Secondo il calcolo delle combinazioni):

$$\text{Combinazioni} = \text{Base del sistema di numerazione}^{\text{numero degli input}}$$

$$\rightarrow C = 2^2 = 4$$

Un fatto importante di cui tener conto è il livello logico degli ingressi e delle uscite infatti, nel decoder che andremo ad utilizzare, gli input e gli output sono di tipo Attivo Basso.

Cosa significa?

Vuol dire semplicemente che tali ingressi e tali uscite sono negate, si parla perciò di logica negativa.

Per riconoscere un dispositivo che funziona in **logica negativa** basta osservare il suo Datasheet: di solito gli input e output Attivi Bassi sono contrassegnati da un simbolo di negazione sopra al nome, o ad un pallino prima del pin. (Figura 2)

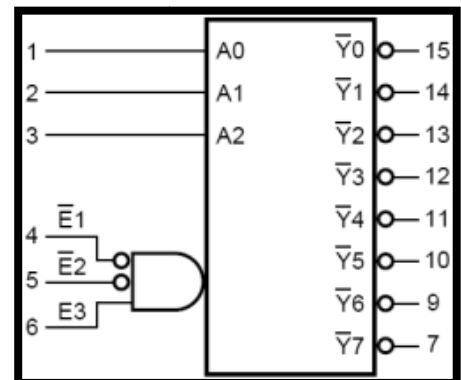


Figura 2 - Decoder 2:4 con Enable e uscite Attive Basse

6. I REGISTRI DI ARDUINO

Normalmente, se vogliamo accendere un LED collegato ad un pin dell’Arduino sappiamo che dobbiamo, nell’ordine:

- 1) impostare quel pin come OUTPUT usando il comando pinMode;
- 2) “scrivere” su quel pin il valore HIGH usando il comando digitalWrite.

Sembra semplice ma se andiamo ad analizzare il codice che viene utilizzato nel “cervello” di Arduino ci accorgiamo che non lo è. La semplicità dell’IDE di Arduino permette sì di fare quest’operazione con 2 istruzioni ma nel “retroscena” le operazioni che vengono eseguite sono diverse, perché il **core** di Arduino deve effettuare una serie di calcoli prima di poter eseguire l’operazione.

Se non si ricerca una grande velocità, tutti questi passaggi possono anche non disturbare, ma se vogliamo fare le cose in modo più rapido e diretto la soluzione è una sola: bisogna manipolare direttamente le porte logiche del microcontrollore.

Un microcontrollore è un dispositivo elettronico e, per questo tutte le operazioni sono svolte a livello digitale. Esistono dei particolari registri salvati nella memoria del microcontrollore che contengono lo stato di ogni singola linea di input/output (I/O) del microcontrollore. Ogni registro rappresenta lo stato di una porta logica: una porta logica è l'insieme dei bit collegati ad un gruppo di piedini fisici della **MCU**. Leggendo o scrivendo in questi registri si manipola in maniera diretta lo stato dei pin.

7. MICROCONTROLLORE ATmega328

Le porte logiche dell'ATmega328 sono ad 8 bit, per cui ogni porta logica contiene lo stato di 8 pin del microcontrollore. Nella *Figura 3* vedete la piedinatura dell'ATmega328 montato sull'Arduino UNO: ogni pin è identificato da una sigla come PDx, PCx o PBx (riquadri giallini).

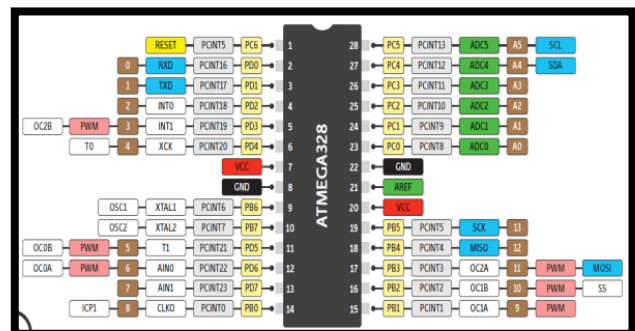


Figura 3 - Pinout ATmega328

Ad esempio, PD0 identifica il pin 0 della porta logica “D”, il quale fa riferimento al pin 0 dell'Arduino (riquadri marroni).

L'ATmega328 ha 23 linee di I/O raggruppati in 3 porte logiche da 8 bit l'una (tranne la C che ne ha 7) denominate B, C e D.

La porta D controlla i pin digitali dallo 0 al 7, la porta B quelli dall' 8 al 13 e la porta C i pin analogici.

8. I REGISTRI DI ARDUINO

Ogni porta logica del microcontrollore è gestita da 3 registri:

a. DDRx (dove x sta per B, C e D)

- Data Direction Register;
- Questo registro è sia di lettura che di scrittura e determina la direzione (INPUT-OUTPUT) dei pin ad esso collegati. Un bit a 0 rappresenta un pin impostato come INPUT; un bit a 1 rappresenta un pin impostato come OUTPUT;
- Equivale alla funzione standard: `pinMode(pin, DIREZIONE)`.

b. PORTx (dove x sta per B, C e D)

- Port Driver Register;
- Questo registro è sia di lettura che di scrittura e determina lo stato (HIGH-LOW) dei pin ad esso collegati. Un bit a 1 indica uno stato HIGH sul relativo pin, mentre un bit a 0 indica la presenza dello stato LOW sullo stesso pin;
- Equivale alla funzione standard: `digitalWrite(pin, STATO)`.

c. PINx (dove x sta per B, C e D)

- Port Pins Register;

-Questo registro è solo di lettura e legge contemporaneamente lo stato (HIGH-LOW) di tutti gli 8 pin di una porta logica;

-Equivale alla funzione standard: `digitalRead(pin)`.

Da ciò si evince che per impostare un pin come OUTPUT con un livello HIGH basta impostare la direzione con il registro DDRx ed il suo stato con il registro PORTx; successivamente parleremo della sintassi da utilizzare per operare con i registri.

9. TRASFERIMENTO DEI DATI: I BUS

Per il trasferimento dei dati entranti ed uscenti da una periferica o un componente, nel nostro caso il decoder, vengono utilizzati i BUS: è l'acronimo di Binary Unit System e indica un canale di comunicazione che permette a periferiche e componenti di un sistema elettronico di interfacciarsi tra loro scambiandosi dati, attraverso la trasmissione di segnali.

In generale, esistono due grandi famiglie di BUS, a seconda della loro modalità di trasmissione dei dati:

a. BUS a trasmissione parallela

Un BUS a **trasmissione parallela**, indica la trasmissione di dati in cui tutti i bit sono trasferiti contemporaneamente lungo canali separati. È utilizzato prevalentemente per trasferire informazioni digitali all'interno di sistemi elettronici (ad esempio, un computer).

La caratteristica peculiare dei BUS a trasmissione parallela è che vengono utilizzati più conduttori per trasmettere simultaneamente le informazioni: un cavo che effettua una trasmissione parallela a n bit è formato da almeno n fili conduttori separati.

Rispetto alla trasmissione seriale la trasmissione parallela risulta avere prestazioni più elevate in termini di velocità di trasmissione, ma ovviamente sarà anche più ingombrante e costosa. Normalmente per comunicazioni a lunga distanza si preferisce infatti utilizzare delle trasmissioni seriali dato che i cavi necessari alla trasmissione parallela renderebbero non economicamente conveniente il progetto.

b. BUS a trasmissione seriale

Un BUS a **trasmissione seriale** indica la modalità di comunicazione tra dispositivi digitali nella quale i bit sono trasferiti lungo un solo canale di comunicazione uno di seguito all'altro e giungono di seguito al ricevente nello stesso ordine in cui li ha trasmessi il mittente.

Un tipo di BUS a trasmissione seriale è il cavo dell'Ethernet.

Un BUS di sistema, utilizzato per il collegamento di due dispositivi, solitamente nei computer, può essere diviso in tre bus minori:

a. BUS dati

È il BUS utilizzato per il trasferimento delle informazioni. È utilizzabile da tutti i componenti del sistema, sia in scrittura sia in lettura; inoltre è un tipo di BUS bidirezionale, cioè permette il passaggio dati in più direzioni contemporaneamente.

b. BUS indirizzi

È un BUS unidirezionale attraverso il quale la **CPU** decide in quale indirizzo andare a scrivere o a leggere informazioni; sia le celle di memoria sia le periferiche di I/O sono infatti divise in zone, ognuna delle quali ha un dato indirizzo. Dopo aver comunicato l'indirizzo tramite questo bus, la scrittura o lettura avviene normalmente tramite il bus dati. Naturalmente questo bus è fruibile in scrittura solo dalla CPU e in lettura da tutti gli altri componenti, in quanto tramite questo bus viene dato solo l'indirizzo della cella, che è deciso dalla CPU.

c. BUS controlli

Il bus controlli è un insieme di collegamenti il cui scopo è coordinare le attività del sistema; tramite esso, la CPU può decidere quale componente deve scrivere sul bus dati in un determinato momento, quale indirizzo leggere sul bus indirizzi, quali celle di memoria devono scrivere e quali invece leggere, etc. Infatti la memoria e tutti gli altri componenti comunicano con la CPU attraverso un unico bus condiviso; questo significa che senza un controllo da parte della CPU si verrebbero a creare dei conflitti e delle collisioni.

10. ANALISI

Prima di cominciare con l'analisi, bisogna procedere con la creazione del circuito ideale dal quale, poi, si procederà a realizzarlo fisicamente.

Attenendosi sempre all'enunciato del problema:

“Effettuare la diagnosi di un decoder 2:4 simulando le 4 combinazioni e riportando il risultato, nei registri di Arduino”

Come primo punto bisogna analizzare il funzionamento del nostro decoder (CD74HC139).

a. Funzionamento Decoder 2:4 | CD74HC139

Stando al **Datasheet**, il decoder CD74HC139 contiene due ingressi binari indipendenti su ognuno dei due decodificatori presenti nell'integrato, ciascuno con un singolo ingresso di abilitazione attivo basso (\bar{E}). Con \bar{E} a livello logico 0 (massa), le combinazioni sugli ingressi di selezione (A0 e A1) portano a 0 il livello logico di una delle quattro uscite normalmente alte (1) (a seconda della combinazione inserita).

Se l'ingresso di abilitazione \bar{E} è alto (1), tutte e quattro le uscite rimangono alte.

b. Tabella della verità del decoder

Per creare la tabella di verità, però, bisogna attenersi a determinate regole, che sono:

- Calcolare le combinazioni (l'esatto numero di righe da inserire) elevando alla Base del sistema di numerazione (in questo caso binario), il numero degli ingressi (in questo caso 2):

$$\text{Combinazioni} = \text{Base del sistema di numerazione}^{\text{numero degli input}}$$

$$\rightarrow C = 2^2 = 4$$

- Ordinare i bit da meno significativo (**LSB, bit che cambia più velocemente**) al più significativo (**MSB, bit che cambia più lentamente**);
- Compilare la tabella attenendosi al punto precedente (partendo dal LSB al MSB), compilando poi la colonna degli output per ultima.

Possiamo notare, come esposto nel paragrafo precedente, che per \bar{E} con valore logico alto, non importa che valori ci siano sui pin di selezione A0 e A1 (la X ha questo significato), poiché le uscite saranno sempre alte.

Inoltre, l'input \bar{E} risulta negato, così come gli output; questo perché il decoder in questione “lavora” sulla base di un segnale attivo basso, rendendo così negate le uscite.

Il risultato che si otterrà sarà esattamente inverso ad un decoder con segnale attivo alto: con il decoder in questione è possibile prevedere che l'uscita selezionata tramite le varie combinazioni, sarà rappresentata con uno 0 logico, mentre le restanti tre saranno a 1 logico.

Nel decoder con segnale attivo alto accade invece l'operazione opposta, infatti l'uscita selezionata sarà a 1 logico mentre le restanti tre saranno a 0 logico.

INPUTS ENABLE SELECT			OUTPUTS			
\bar{E}	A1	A0	\bar{Y}_3	\bar{Y}_2	\bar{Y}_1	\bar{Y}_0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	X	X	1	1	1	1

Figura 4 - Tabella di verità del decoder CD74HC139

11. PROGETTAZIONE

Nei seguenti paragrafi, viene illustrata la progettazione di tutto il circuito ed il programma utilizzati per verificare il corretto funzionamento del decoder.

a. Elenco strumenti, apparecchiature, dispositivi e software

▪ 1 Breadboard

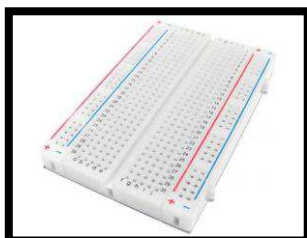


Figura 5 - Breadboard con 400 fori

▪ Dei Jumper Wires o Monofilamento

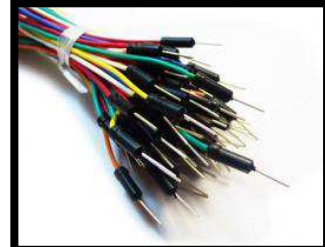


Figura 6 - Jumper wires

▪ 1 Integrato CD74HC139

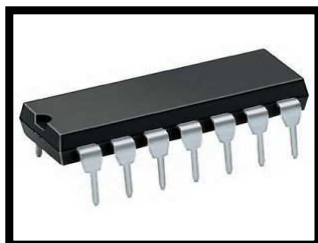


Figura 7 - Integrato CD74HC139

▪ 4 LED gialli



Figura 8 - LED giallo

▪ 1 Scheda Arduino UNO



Figura 9 - Scheda Arduino UNO

▪ IDE di Arduino



Figura 10 - IDE di Arduino

b. Introduzione al Programma

Per redigere un programma con Arduino, andando a lavorare con i comandi che agiscono direttamente sui registri, bisogna seguire una sintassi specifica.

Come è stato precedentemente esposto nella *sezione 8*, ogni registro ha un comando specifico da utilizzare; poniamo per esempio di voler accendere un LED montato sulla breadboard, tramite il pin 5 dell'Arduino (appartenente alla porta logica D).

Per prima cosa dobbiamo dichiarare la direzione che vogliamo assegnare al pin 5, utilizzando il comando DDRD; dato che ogni porta logica controlla lo stato e la direzione di 8 pin, dobbiamo assegnare al comando, 8 bit che rappresentano gli 8 pin:

DDRD = B00100000; (l'assegnazione dei pin ai bit va fatta dalla destra alla sinistra del byte, partendo dal pin con il valore più basso: il bit più a destra sarà lo 0 e quello più a sinistra il 7)

La B davanti agli 8 bit specifica che vogliamo utilizzare la scrittura del byte in Binario (è possibile usare anche la scrittura in esadecimale: 0x20), mentre l'1 posizionato in terza posizione a partire da sinistra, significa che vogliamo che il pin 5 sia posto come OUTPUT.

Successivamente, per far accendere il LED, dobbiamo determinare lo stato del pin 5 come HIGH tramite il comando PORTD; utilizzando la sintassi del comando precedente scriviamo: PORTD = B00100000;

Assegnando il valore logico di 1 al bit corrispondente al pin 5, abbiamo portato quest'ultimo allo stato HIGH (+5V) e, se abbiamo collegato l'anodo del LED al pin 5, e il catodo a massa, il nostro LED sarà acceso. Se volessimo anche spegnere il led dopo averlo acceso, basterebbe utilizzare di nuovo il registro PORTD, riportando il pin 5 a LOW (0V):

PORTD = B00000000;

Fino ad ora, il comando PINx non è stato usato; facciamo allora in modo di comprenderne il suo utilizzo!

Per prima cosa scolleghiamo il LED dalla breadboard, non ci servirà più.

Ora colleghiamo il pin 5, dal quale abbiamo erogato prima 5V e poi 0V, ad un altro pin, per esempio l'8 e impostiamo la direzione di quest'ultimo in OUTPUT. Come è stato esposto nella *sezione 8*, il registro PINx è solo di lettura, e legge appunto tutti gli 8 pin della porta;

DDRB = B00000000;

Per verificare quindi che il pin 5 sia nello stato esatto, leggiamo il pin 8 a cui è collegato: assegniamo per prima cosa una variabile di tipo byte (può contenere solo 8 bit) alla lettura della porta B (a cui appartiene il pin 8) in modo da immagazzinare in essa la lettura dei pin:

verifica = PINB;

successivamente inseriamo questo comando:

verifica&=B00000001;

Esso serve per la **mascheratura** dei pin non utilizzati: il registro tiene conto solo della lettura effettuata sul pin 8, mentre porta tutte le altre letture automaticamente a 0 logico.

Per verificare che la lettura sul pin 8, dello stato del pin 5 sia corretta utilizziamo il seguente comando:

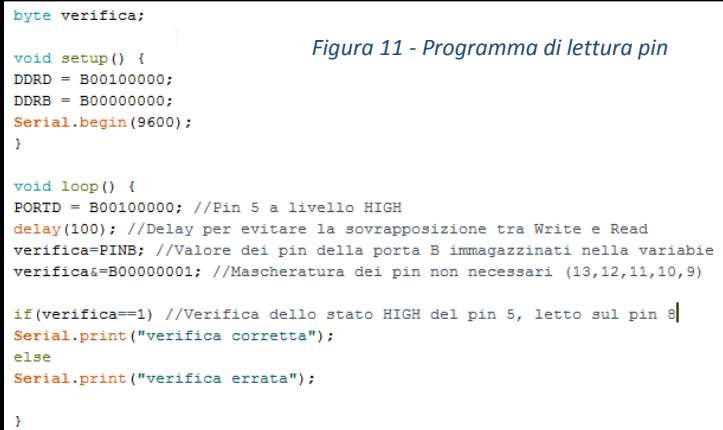
```
if(verifica==1)
```

```
Serial.print("lettura corretta");
```

```
Else
```

```
Serial.print("lettura errata");
```


Verifichiamo, tramite la condizione if, che la variabile `verifica` sia uguale a 1 poiché, convertendo il byte di lettura della porta B in decimale troviamo che esso equivale appunto ad 1, ma solo se il pin 5 è nello stato HIGH. Il codice completo per la verifica di una lettura è il seguente:



```
byte verifica;

void setup() {
  DDRD = B00100000;
  DDRB = B00000000;
  Serial.begin(9600);
}

void loop() {
  PORTD = B00100000; //Pin 5 a livello HIGH
  delay(100); //Delay per evitare la sovrapposizione tra Write e Read
  verifica=PINB; //Valore dei pin della porta B immagazzinati nella variabile
  verifica<=B00000001; //Mascheratura dei pin non necessari (13,12,11,10,9)

  if(verifica==1) //Verifica dello stato HIGH del pin 5, letto sul pin 0
    Serial.print("verifica corretta");
  else
    Serial.print("verifica errata");
}
```

Figura 11 - Programma di lettura pin

c. Programma

Nel seguente paragrafo viene esposto lo sviluppo del programma che è stato creato per la verifica del funzionamento del decoder:

prima di tutto vengono inizializzate le 4 variabili `combo` di tipo byte che andranno ad immagazzinare la lettura della porta logica B per ogni combinazione e la variabile intera `i` che servirà per il conteggio delle verifiche eseguite; successivamente, nel Setup, vengono poste le direzioni sia per i due pin che generano le combinazioni (D2 e D3), i quali sono in OUTPUT, e per i pin di lettura delle uscite del decoder (D8, D9, D10, e D11), i quali sono in INPUT. Viene poi inizializzata la comunicazione seriale con il monitor e viene stampata la richiesta del comando di start.

Il Loop, ha come primo comando un if, che serve ad eseguire la verifica solamente quando il comando di start: 'p' viene scritto nel monitor seriale; successivamente viene eseguita la scrittura delle combinazioni entranti nel decoder, e la lettura delle sue uscite, per ognuna delle 4 combinazioni possibili. Dopo che è stata terminata l'ultima lettura, il programma stampa per ogni variabile assegnata alle combinazioni, la sua lettura effettiva.

Infine verifica che tutte e quattro le combinazioni siano corrette e in tal caso stampa a schermo: "Verifica delle combinazioni corretta" e aspetta il comando di start per eseguire nuovamente la verifica; in caso contrario stampa a schermo: "Verifica delle combinazioni errata" e aspetta il comando di start per eseguire nuovamente la verifica.

```
byte combo1, combo2, combo3, combo4;
int i = 1;

void setup()
{
    DDRB = B00000000; //PORTB tutta in Input (Pin da 8 a 13)
    DDRD = B00001100; //PORTD tutta in Input (Tranne pin 2 e 3 in Output)

    Serial.begin(9600);
    Serial.println();
    Serial.println(" ► Inserisci il comando d'avvio: p ");
    Serial.println();
}

void loop()
{
    if(Serial.read()=='p') //Attesa del comando d'avvio: p
    {
        Serial.print("    — Verifica N° ");
        Serial.print(i,DEC);
        Serial.println(" —");
        Serial.println();
        Serial.print(" ► Attendere: acquisizione dati in corso");
        i++;

        /***** Prima lettura *****/
        PORTD=B00000000; //Combinazione 1: pin 2 e 3 LOW
        delay(100); //Questo delay è inserito per evitare problemi di lettura, dati dalla rapidità delle operazioni
        combo1=PINB; //Lettura completa di tutti i pin della porta B, e assegnamento del byte alla variabile combo1
        combo1&=B00001111; //Questa stringa serve a mascherare i primi quattro pin della porta B, in modo che non diano false
        letture
        delay(500); //Delay posto tra una lettura, e quella successiva

        Serial.print(".");

        /***** Seconda lettura *****/
        PORTD=B00000100; //Combinazione 2: pin 2 LOW e pin 3 HIGH
        delay(100);
        combo2=PINB; //Lettura completa di tutti i pin della porta B, e assegnamento del byte alla variabile combo2
        combo2&=B00001111;
        delay(500);

        Serial.print(".");

        /***** Terza lettura *****/
        PORTD=B00001000; //Combinazione 3: pin 2 HIGH e pin 3 LOW
        delay(100);
        combo3=PINB; //Lettura completa di tutti i pin della porta B, e assegnamento del byte alla variabile combo3
        combo3&=B00001111;
        delay(500);

        Serial.print(".");
```

```

/***** Quarta lettura *****/
PORTD=B00001100; //Combinazione 4: pin 2 HIGH e pin 3 HIGH
delay(100);
combo4=PINB; //Lettura completa di tutti i pin della porta B, e assegnamento del byte alla variabile combo4
combo4&=B00001111;
delay(500);

Serial.println(".");
Serial.println();

/***** Stampa di ogni combinazione *****/
Serial.println(" ► Risultato della verifica:");
Serial.println();

Serial.print("    • combo1 = ");
Serial.print(combo1,BIN);
Serial.println();
Serial.println();
Serial.print("    • combo2 = ");
Serial.print(combo2,BIN);
Serial.println();
Serial.println();
Serial.print("    • combo3 = ");
Serial.print(combo3,BIN);
Serial.println();
Serial.println();
Serial.print("    • combo4 = ");
Serial.print(combo4,BIN);
Serial.println();
Serial.println();

if(combo1==14&&combo2==13&&combo3==11&&combo4==7)
{
    Serial.println(" ► Verifica delle combinazioni corretta");
    Serial.println();
    Serial.println(" ► Per eseguire nuovamente la verifica, inserire il comando d'avvio: p ");
    Serial.println();
    Serial.println("~~~~~");
    Serial.println();
}
else
{
    Serial.println(" ► Errore nella verifica delle combinazioni");
    Serial.println();
    Serial.println(" ► Per eseguire nuovamente la verifica, inserire il comando d'avvio: p ");
    Serial.println();
    Serial.println("~~~~~");
    Serial.println();
}
}
}
}

```

12. TESTING

Nei seguenti paragrafi viene esposto il Testing di ciò che è stato progettato nel precedente paragrafo.

a. Obiettivi

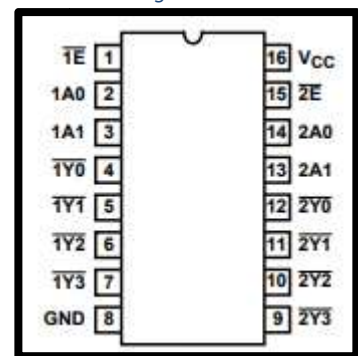
L'obiettivo del Testing è quello di verificare la validità del progetto precedentemente esposto, quindi di assemblare il circuito ed eseguire il programma di verifica del funzionamento del decoder 2:4.

b. Montaggio su breadboard

Prima di procedere con il montaggio del circuito, è necessario conoscere la “piedinatura” dell'integrato:

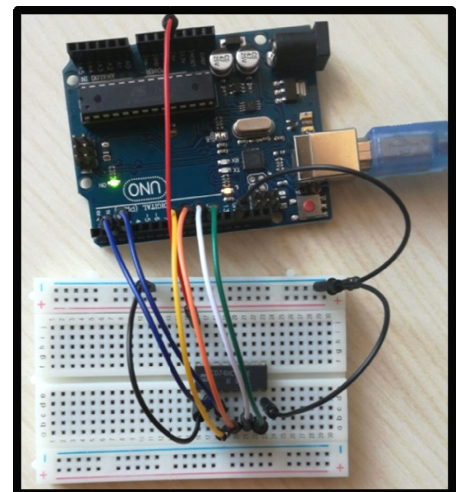
- Pin 1 → Enable decoder 1
- Pin 2, Pin 3 → Pin di selezione decoder 1
- Pin 4, Pin 5, Pin 6, Pin 7 → Uscite decoder 1
- Pin 8 → **GND**
- Pin 9, Pin 10, Pin 11, Pin 12 → Uscite decoder 2
- Pin 13, Pin 14 → Pin di selezione decoder 2
- Pin 15 → Enable decoder 2
- Pin 16 → **Vcc**

Figura 15 - Piedinatura dell'integrato CD74HC139



Procediamo ora con il montaggio: per prima cosa ho inserito l'integrato tra le due metà della breadboard, successivamente ho collegato il pin della 16 dell'integrato al pin dei 5 Volt dell'Arduino; ho collegato poi i pin digitali 2 e 3 dell'Arduino, rispettivamente ai pin 2 e 3 dell'integrato. Ho collegato sia il pin 1 che il pin 8 dell'integrato, con il pin GND dell'Arduino; a questo punto ho collegato i pin dell'Arduino, impostati per la lettura (8, 9, 10 e 11) ai pin 4, 5, 6 e 7 dell'integrato. (Figura 16)

Figura 16 - Montaggio del circuito



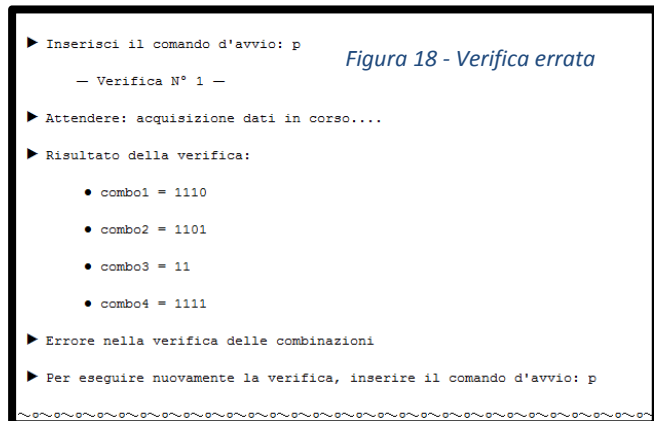
Dopo aver Montato la breadboard, ho caricato il programma sulla scheda Arduino. Aprendo il monitor seriale dall'IDE di Arduino, immediatamente è comparso il messaggio “Inserisci il comando d'avvio: p”, e dopo averlo digitato nella barra superiore del monitor, è apparso il caricamento della verifica e poco dopo anche il risultato. (Figura 17)

```

▶ Inserisci il comando d'avvio: p
    — Verifica N° 1 —
▶ Attendere: acquisizione dati in corso...
▶ Risultato della verifica:
    • combo1 = 1110
    • combo2 = 1101
    • combo3 = 1011
    • combo4 = 111
▶ Verifica delle combinazioni corretta
▶ Per eseguire nuovamente la verifica, inserire il comando d'avvio: p
  
```

Figura 17 - Verifica corretta

Successivamente, per controllare che tutto funzionasse per il meglio, anche con le combinazioni errate, ho scollegato un cavo che collegava l'uscita del decoder, al pin di lettura e ho fatto partire la verifica. (Figura 18)



```
► Inserisci il comando d'avvio: p
— Verifica N° 1 —
► Attendere: acquisizione dati in corso...
► Risultato della verifica:
• combo1 = 1110
• combo2 = 1101
• combo3 = 11
• combo4 = 1111
► Errore nella verifica delle combinazioni
► Per eseguire nuovamente la verifica, inserire il comando d'avvio: p
```

13. DUBBI

È stato abbastanza difficile comprendere la teoria dei registri e completare il programma poiché l'argomento non è molto facile da assimilare, ma una volta compresa la metodologia, sembra tutto molto più facile. Ho riscontrato molta difficoltà da parte dei compagni, a capire l'argomento e la programmazione.

14. CONCLUSIONI

Dopo aver testato con successo il funzionamento del decoder, tramite il programma di Arduino, possiamo dire di aver risolto il problema solo in parte: nella relazione non è stata inserita la parte comprendente il collegamento tra il decoder e l'Arduino tramite il latch.

15. RIFERIMENTI BIBLIOGRAFICI E SITOGRAFICI

- Datasheet Decoder CD74HC139: <https://www.ti.com/lit/ds/symlink/cd74hc139.pdf?HQS=TI-null-null-alldatasheets-df-pf-SEP-ww>
- Teoria sul decoder: <http://www.dacrema.com/Informatica/Decoder.htm>
- Teoria sui registri: <https://www.leonardomiliani.com/2013/manipoliamo-direttamente-le-porte-logiche-di-una-mcu/>
- Slides sui registri: <https://www.slideshare.net/SardegnaRicerche/il-cuore-di-arduino-un-sistema-di-sviluppo-basato-su-microcontrollore-atmel-avr>