EsD3: Macchine a stati finiti: semafori e riconoscitore di fronti

Gruppo 1.AC
Matteo Rossi, Bernardo Tomelleri

6 maggio 2022

1 Misura componenti dei circuiti

Riportiamo per completezza il valore della tensione continua di alimentazione per i circuiti integrati misurata con il multimetro

$$V_{CC} = 4.99 \pm 0.03 \text{ V}$$

e il valore di capacità del condensatore di disaccoppiamento che collega le linee di alimentazione a massa (sempre misurato con il multimetro)

$$C_d = 97 \pm 4 \text{ nF}$$

2 Implementazione di un semaforo con circuiti integrati

Si vuole costruire una macchina a stati finiti in grado di emulare il funzionamento delle luci di un semaforo stradale, nelle modalità

ABILITATO si ripetono in ciclo gli stati: LED verde acceso → LED giallo e verde accesi → LED rosso acceso.

DISABILITATO si alternano gli stati LED giallo acceso e spento (giallo lampeggiante).

in cui tutti gli stati hanno durata pari ad un impulso del clock inviato all'ingresso dei circuiti integrati a disposizione.

2.a Diagramma a stati del semaforo

Quando il semaforo è in modalità "abilitato" ($E = 1$) si susseguono ciclicamente 3 possibili output, per cui sono stati implementarli con 3 stati interni della macchina. Mentre nel caso di semaforo "disabilitato" ($E = 1$) i possibili output si riducono a 2, e si possono esprimere utilizzando due degli stati precedentemente codificati a seconda del valore logico dell'input ENABLE tramite un circuito FSM di Mealy.

Scegliamo di codificare con due bit di memoria (Q_1Q_0) i tre stati: 00, 01 e 10 che corrispondono, quando $E = 1$, ai tre output "verde" (V), "verde-giallo" (VG), "rosso" (R). Quando invece $E = 0$ vengono usati solamente i primi due 00 e 01, che corrispondono agli output del semaforo "spento" (Y^0) e "giallo" (Y^1) come si può vedere già nel diagramma a stati riportato in fig. 1.

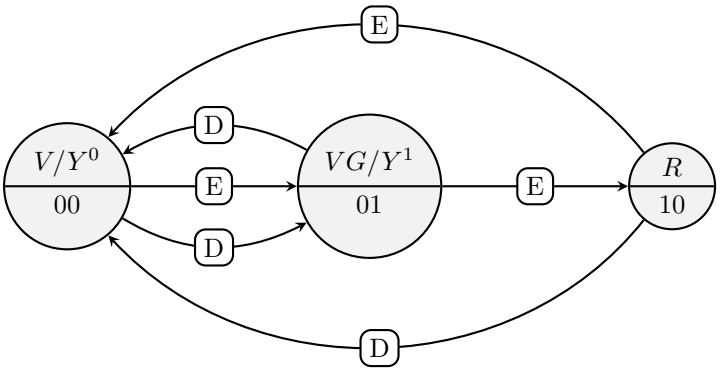

Figura 1: Diagramma degli stati del semaforo FSM di Mealy con Enable

2.b Codifica degli stati della macchina

La codifica in termini di bit degli stati della macchina è riassunta nella tabella 1, dove Q_0 e Q_1 corrispondono rispettivamente al valore logico delle uscite del primo e del secondo flip-flop, mentre S_E e S_D rappresentano lo stato associato al semaforo a seconda che questo sia ENABLED o DISABLED.

S_D	S_E	Q_1	Q_0
Y^0	V	0	0
Y^1	VG	0	1
	R	1	0

Tabella 1: Codifica binaria scelta per gli stati del semaforo.

2.c Tabelle di verità

Come visto nel diagramma degli stati il segnale di input ENABLE modifica l'uscita del circuito in maniera asincrona, mentre dalla tabella di verità (tabella 2) si intuisce come lo stato futuro dipenda da quello precedente incrementando come un contatore in maniera sincrona rispetto al segnale di clock.

Dal diagramma ricaviamo la tabella di verità delle transizioni di stato tra il corrente Q_1Q_0 (che indichiamo con lo stato in uscita della memoria implementata con i Flip-Flop) e il successivo D_1D_0 (che indichiamo con il valore logico in ingresso agli stessi FF). Riportiamo per completezza anche lo stato inutilizzato e non normalmente raggiungibile dalla macchina 11 con $\ominus$, dal momento che la FSM attraversa gli stati come un contatore binario ci aspettiamo che torni spontaneamente negli stati permessi.

		current state		next state		
ENABLE	LED	Q_1	Q_0	D_1	D_0	LED
1	V	0	0	0	1	VG
	VG	0	1	1	0	R
	R	1	0	0	0	V
	$\ominus$	1	1	X	X	$\ominus$
0	Y^0	0	0	0	1	Y^1
	Y^1	0	1	0	0	Y^0
	Y^0	1	0	0	0	Y^0
	$\ominus$	1	1	X	X	$\ominus$

Tabella 2: Tabella di verità per le transizioni di stato del semaforo Mealy

Riportiamo in forma esplicita la tabella di verità per le uscite che identificano i LED Verde, Giallo e Rosso del semaforo nella tabella 3

E	Q_0	Q_1	G	Y	R
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	X	X	X
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	X	X	X
0	1	1	X	X	X

Tabella 3: Corrispondenza tra gli stati della FSM e i valori di tensione digitali in ingresso ai LED del semaforo

.

in cui come prima gli stati contrassegnati da una X sono quelli che non ci interessano (don't care). Questi sono stati che non ci aspettiamo che la macchina attraversi, e la loro codifica in termini di bit ci interessa solo per a minimizzare la logica combinatoria che definisce le transizioni e le espressioni per le uscite; come faremo per costruire le mappe di Karnaugh, in cui i singoli bit X verranno considerati 1 o 0 nella maniera più conveniente.

Da queste vediamo come possiamo collegare gli output della macchina ai led del semaforo in modo tale che nella realizzazione pratica del circuito, in corrispondenza degli opportuni stati della FSM in uscita dai Flip-Flop si accendano le corrette combinazioni di LED (ad esempio: $\mathrm{Y} = Q_0 \implies$ LED giallo collegato in serie a Q_0).

LED verde acceso $\mathrm{G} = Q_1 = 0$ ($\iff \overline{Q_1} = 1$)

LED giallo acceso $\mathrm{Y} = Q_0 = 1$

LED rosso acceso $\mathrm{R} = Q_1 = 1$

Occorre specificare che, nel caso in cui $E = 0$, $Q_0 = 1$ e $Q_1 = 0$ (riportato come X nella tabella 3, produce l'output "rosso". Questo vale a dire che se il semaforo è "rosso" con ENABLE alto e questo viene messo a zero, rimarrà rosso fino al fronte di salita successivo del clock, dopo di cui si spegne come previsto dalla modalità disabilitato.

2.d Mappe di Karnaugh e logica combinatoria

Si riportano in tabella 4 e tabella 5 seguito le mappe di Karnaugh impiegate per derivare le espressioni logiche degli stati futuri D_1D_0 in funzione di quelli correnti Q_1Q_0 e del segnale di ingresso E:

E \ Q_1Q_0	00	01	11	10
0	1	0	X	0
1	1	0	X	0

Tabella 4: Tabella di Karnaugh per $D_0 = \overline{Q_0} \cdot \overline{Q_1}$

E \ Q_1Q_0	00	01	11	10
0	0	0	X	0
1	0	1	X	0

Tabella 5: Tabella di Karnaugh per $D_1 = E \cdot Q_0$

Da queste si vede come per la corretta accensione del led verde, è necessario che questa sia consentita solo quando ENABLE = 1, mentre per il LED rosso questo non è necessario, infatti possiamo riassumere i valori delle uscite del semaforo come:

- $\mathrm{G} = \overline{Q_1} \cdot E \implies$
- $\mathrm{Y} = Q_0$
- $\mathrm{R} = Q_1$

AND-gate per LED rosso Facendo uso di una quarta porta AND è possibile consentire l'accensione del LED rosso solamente in presenza del segnale di ENABLE (ovvero $\mathrm{R} = Q_1 \cdot E$) in maniera analoga a quanto fatto per il LED verde. Dalla tabella 2 infatti si nota che l'unica differenza di funzionamento si avrebbe durante la transizione di $E : 1 \to 0$. Per cui in questa variante del circuito, lo spegnimento diventerebbe asincrono rispetto al segnale di clock anziché sincrono.

Scegliamo quindi di costruire la prima versione del circuito che minimizza il numero di porte logiche necessarie.

2.e Costruzione del circuito

Si è montato il circuito riportato in fig. 2 con due positive-edge-triggered D Flip-Flop (da chip integrato 74LS74) e 3 porte AND (da chip integrato 74LS08) e si sono collegati i pin PRESET e CLEAR dei D-FF alla tensione di alimentazione V_{CC} onde evitare reset o clear spuri.

2.f Analisi e verifica del funzionamento del circuito

Per studiarne il comportamento generiamo nei due pin DIO 0 (CLOCK) e DIO 1 (ENABLE) dell'AD2 rispettivamente due segnali di clock di frequenza $f_{\mathrm{clk}} = 10$ Hz e $f = f_{\mathrm{clk}}/10 = 1$ Hz agli ingressi CLK ed E del circuito.

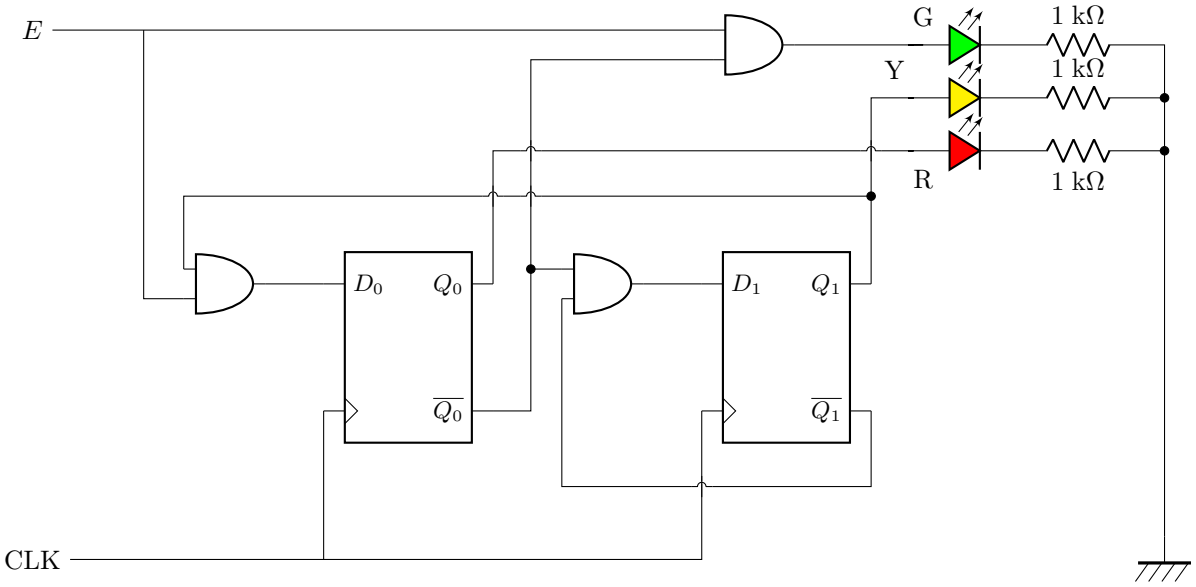

Figura 2: schema del semaforo Mealy con enable.

Così facendo si riesce ad apprezzare il comportamento del circuito non solo con $E = 1$ ed $E = 0$, ma anche durante le transizioni tra le due modalità di funzionamento del semaforo, come visualizzato dall'acquisizione con Logic Analyzer dei segnali in ingresso e in uscita della FSM in fig. 3 (su DIO 2 il LED verde (G), in DIO 3 il giallo (Y) e in DIO 4 il rosso (R)).

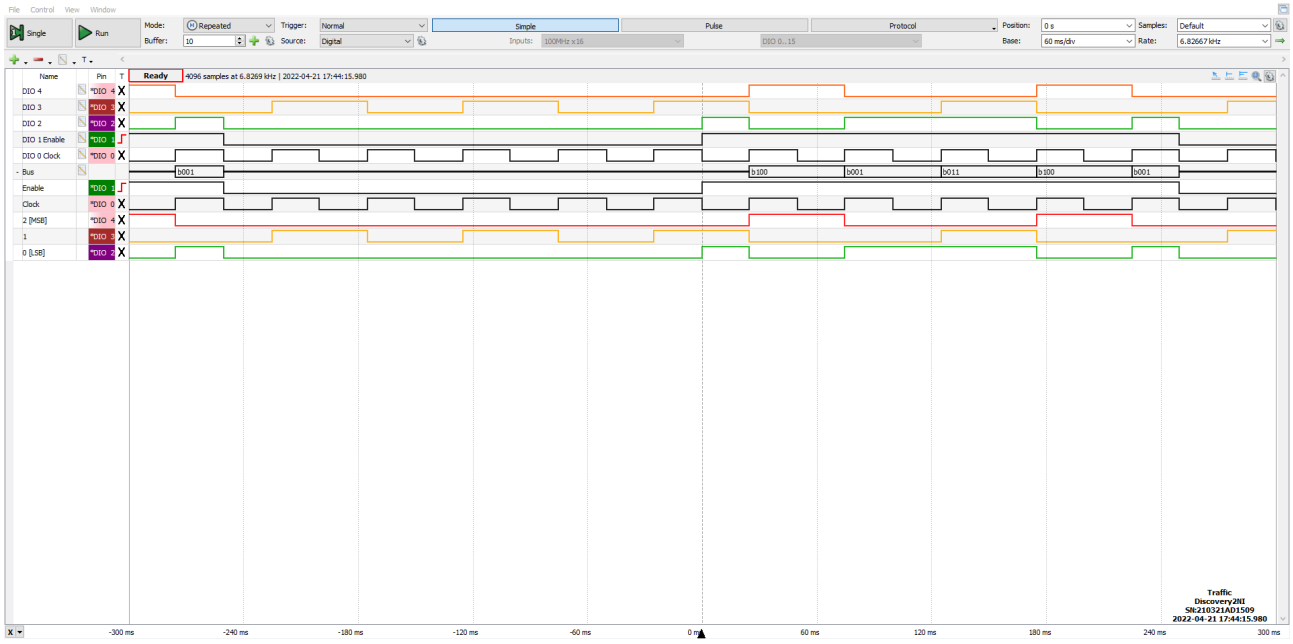

Figura 3: Acquisizione di un ciclo completo (frequenza 1 Hz) con Logic Analyzer dei segnali in ingresso e in uscita dal semaforo.

Osserviamo che nel caso particolare del cambiamento di modalità di funzione del semaforo $E = 0$, $Q_0 = 1$ e $Q_1 = 0$ il LED rosso rimane acceso fino al fronte di salita successivo del clock prima di spegnersi, coerentemente con quanto previsto dalle tabelle di verità.

3 Implementazione software della logica combinatoria con AD2/ROM

3.a Costruzione del circuito

Si vuole ricostruire il semaforo sostituendo alla parte di logica combinatoria con circuiti integrati la funzione ROM dello strumento Patterns dell'AD2, mantenendo quindi gli stessi 2 Flip-Flop di prima per memorizzare i

3 stati interni del semaforo.

Dopo aver costruito il circuito in fig. 4 si è programmata la ROM: innanzitutto si sono impostati ENABLE, Q_0 e Q_1 come Input, per controllare lo stato corrente e se il semaforo risulta essere abilitato, dunque D_0, D_1 e le uscite corrispondenti ai LED come Output, in modo da poter sovrascrivere lo stato corrente e accendere o spengere i led.

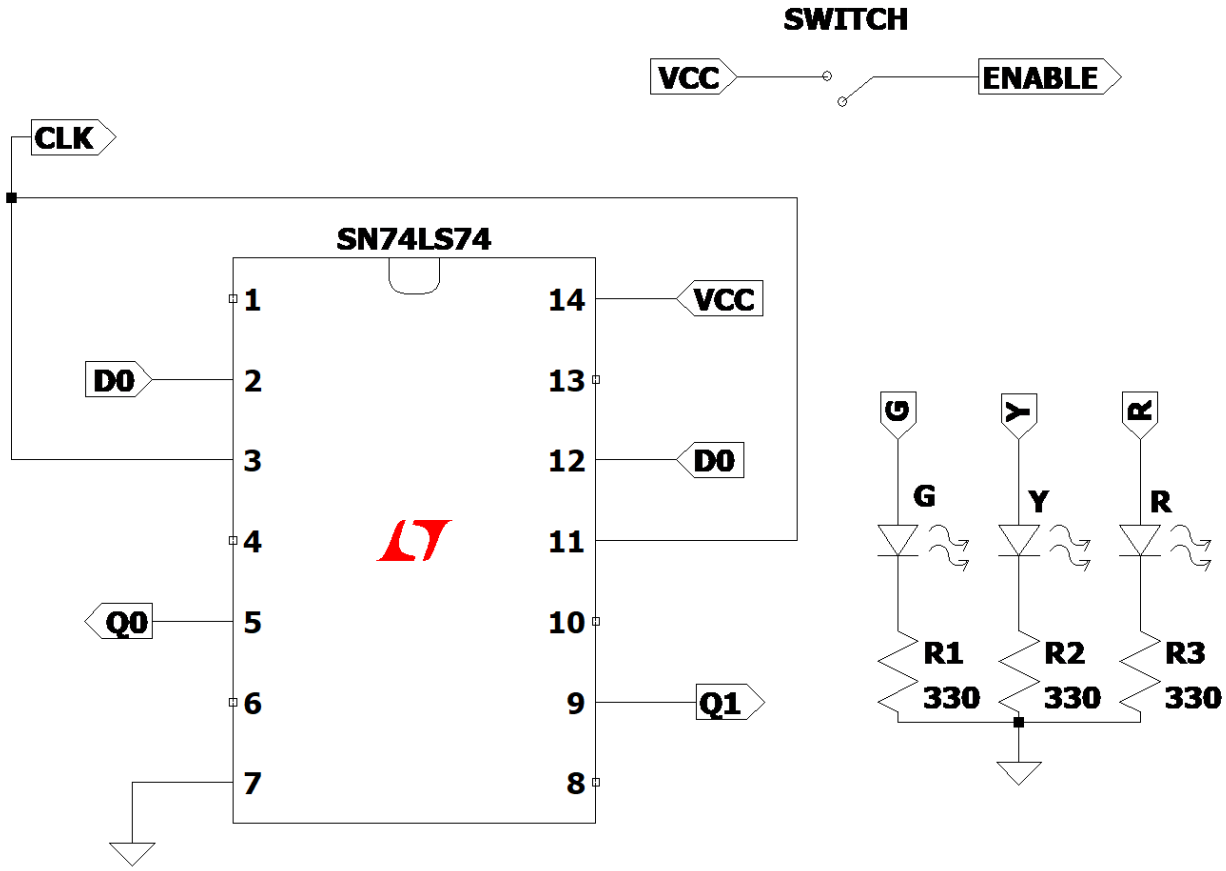

Figura 4: Schematica del circuito utilizzato per il semaforo gestito da ROM

3.b Implementazione delle tabelle di verità in ROM

Utilizzando la tabella di verità presente in tabella 2 abbiamo programmato la seguente ROM in cui è presente un unico stato illegale per entrami i funzionamenti (abilitato e disabilitato):

	Q1	Q0	ENABLE	GREEN	YELLOW	RED	D1	D0
1	0	0	1	1	0	0	0	1
2	0	1	1	1	1	0	1	0
3	1	0	1	0	0	1	0	0
4	1	1	X	0	0	0	X	X
5	0	0	0	0	0	0	0	1
6	0	1	0	0	1	0	0	0
7	1	0	0	0	0	0	0	0
8								

Figura 5: Tabella di verità programmata all'interno della ROM per il controllo del semaforo

3.c Verifica del funzionamento del circuito

Si procede quindi con la verifica del funzionamento del circuito. Si invia al pin CLK un segnale di clock con frequenza pari ad 1 Hz, mantenendo collegati CLEAR e PRESET a Vcc come prima, e impostando la frequenza della ROM a 1 Mhz. Dalle immagini 6 e 7 possiamo concludere che il circuito funzioni come da aspettativa.

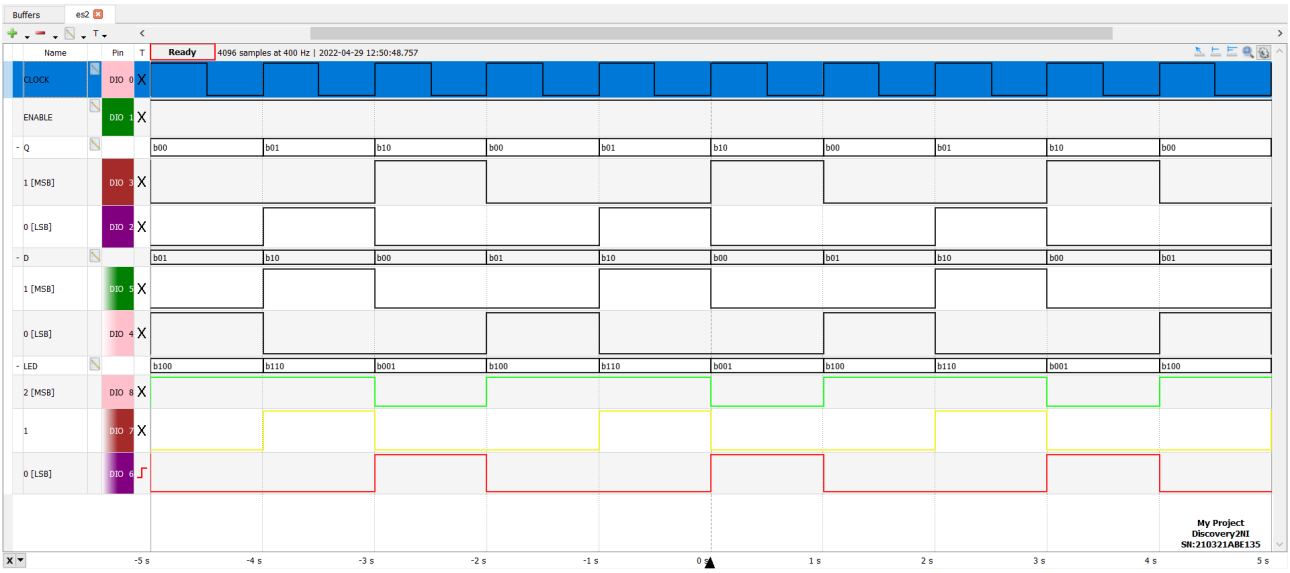

Figura 6: Acquisizione Logic del semaforo gestito da ROM, Abilitato

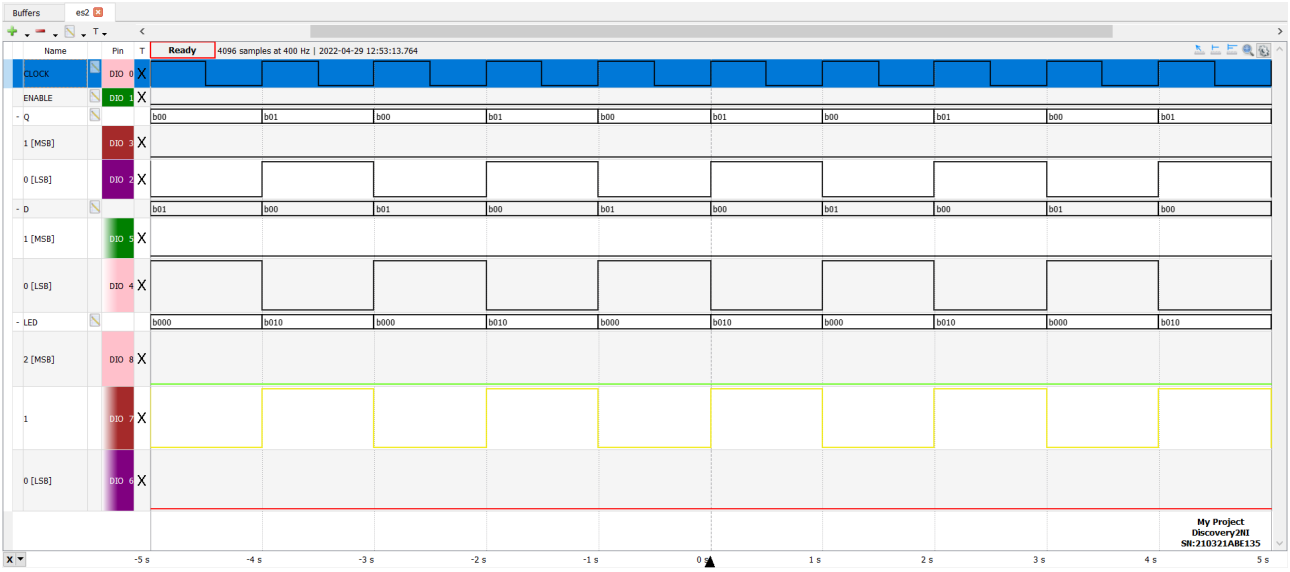

Figura 7: Acquisizione Logic del semaforo gestito da ROM, Disabilitato

3.d Variante svizzera del semaforo ROM

Si va quindi a modificare la fig. 5 in modo da utilizzare anche lo stato 1 1 (successivo al rosso) per visualizzare lo stato rosso e giallo. Si procede quindi a verificare il funzionamento del semaforo svizzero. Come prima inviamo

	Q1	Q0	ENABLE	GREEN	YELLOW	RED	D1	D0
1	0	0	1	1	0	0	0	1
2	0	1	1	1	1	0	1	0
3	1	0	1	0	0	1	1	1
4	1	1	1	0	1	1	0	0
5	0	0	0	0	0	0	0	1
6	0	1	0	0	1	0	0	0
7	1	0	0	0	0	0	0	0
8	1	1	0	0	0	0	0	0
9								

Figura 8: Tabella di verità programmata all'interno della ROM per il controllo del semaforo svizzero

un segnale di clock a 1 Hz e lasciamo la frequenza della ROM a 1 MHz. Osservando la fig. 9 possiamo concludere

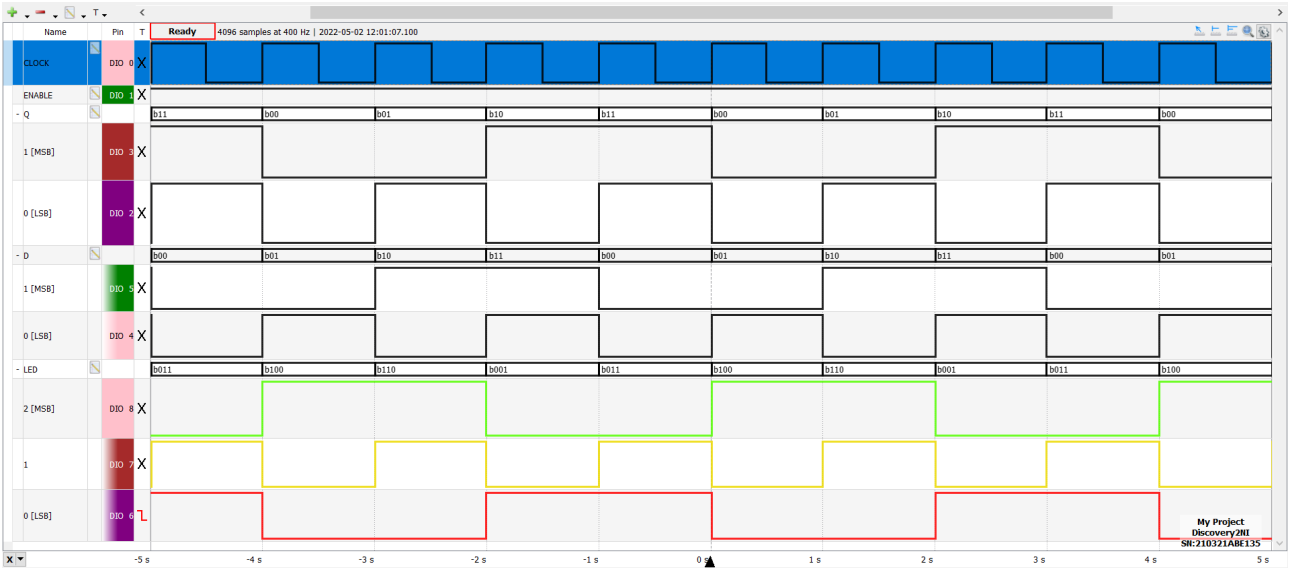

Figura 9: Acquisizione Logic del semaforo svizzero gestito da ROM, Abilitato

che il circuito funziona come da aspettativa

4 Implementazione in software dei semafori con MCU (Arduino)

Si vuole ricostruire il circuito precedente per il controllo di un semaforo tramite un microcontrollore (nel nostro caso utilizzeremo Arduino UNO). Dopo aver montato il circuito descritto in fig. 10, si procede con l'implementazione software del semaforo.

4.a Implementazione del codice per la FSM

Ci basiamo sullo schema per il semaforo FSM di Mealy presente in fig. 1 per controllare il funzionamento di un semaforo tramite enable.

```
/*state
 * 0= ALL OFF
 * 1= RED ON
 * 2= RED YELLOW ON
 * 3= GREEN ON
 * 4= GREEN YELLOW ON
 * 5= YELLOW ON
 *
 */
void setup() {
  pinMode(5, OUTPUT); //Led Rosso
```

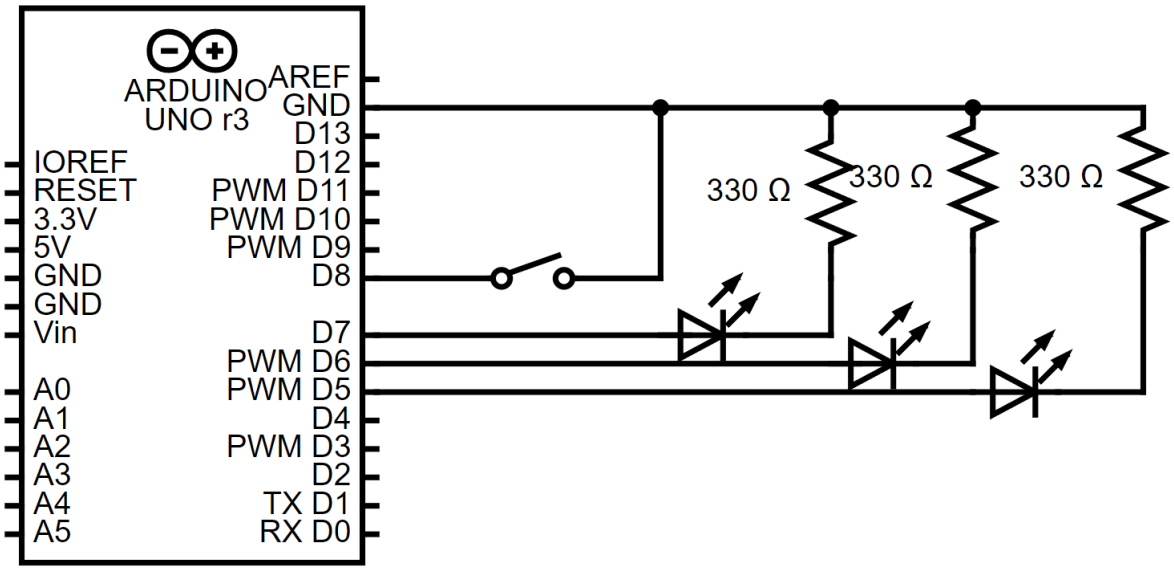

Figura 10: Schematica utilizzata nella gestione del semaforo con software tramite arduino.

```
  pinMode(6, OUTPUT); //Led Giallo
  pinMode(7, OUTPUT); //Led Verde
  pinMode(8, INPUT_PULLUP); //Pin di Enable
}
int tickGREEN=1000; //Temporizzazione dello stato VERDE
int tickRED=1000; //Temporizzazione dello stato ROSSO
int tickYELLOW=1000; //Temporizzazione dello stato GIALLO
int tickREDYELLOW=1000; //Temporizzazione dello stato ROSSOGIALLO
int tickGREENYELLOW=1000; //Temporizzazione dello stato VERDEGIALLO
int tickOFF=1000; //Temporizzazione dello stato spento
int CurrentState=0; //stato iniziale impostato sul tutto spento


void loop() {
  //viene controllato se il semaforo \‘e abilitato
  if(CheckEnable()){

    // nel caso in cui il semaforo sia abilitato
    switch(CurrentState){
    case 1: //se lo stato precedente risulta essere rosso e giallo
      CurrentState=3; //lo stato corrente viene spostato a verde
      digitalWrite(5,LOW);
      digitalWrite(6,LOW);
      digitalWrite(7,HIGH);
      delay(tickGREEN);
      break;
    case 3: //se lo stato precedente risulta essere solo verde
      CurrentState=4; //lo stato corrente viene portato a giallo e verde
      digitalWrite(6,HIGH);
      delay(tickGREENYELLOW);
      break;

    default: // in qualunque altro caso di stato precedente, se il semaforo \‘e abilitato, lo
    stato inizializzato \‘e quello rosso
      CurrentState=1;
      digitalWrite(5,HIGH);
      digitalWrite(6,LOW);
      digitalWrite(7,LOW);
      delay(tickRED);
    }

  }else {
    //nel caso in cui il semaforo sia disabilitato
    switch(CurrentState){
      case 5: // se lo stato precedente risulta essere giallo
        CurrentState=0; //imposta lo stato corrente a tutto spento
        digitalWrite(6,LOW);
```

```
        delay(tickOFF);
59      break;
      default: // in qualunque altro caso di stato precedente, lo stato inizializzato \'e
    quello giallo
61      CurrentState=5;
        digitalWrite(6,HIGH);
63      digitalWrite(5,LOW);
        digitalWrite(7,LOW);
65      delay(tickYELLOW);
      }
67  }
}
69
bool CheckEnable(){
71  //Algoritmo di controllo per Enable

73  int check =digitalRead(8);
  return (check== HIGH);
75}
```

Listing 1: Semaforo-mealy.ino

4.b Versione svizzera del semaforo con Enable via Arduino

Si introduce un quarto stato nel caso in cui il semaforo sia abilitato, successivo al rosso; per fare ciò è necessaria una semplice modifica all'interno dello switch nel caso in cui il semaforo risulti abilitato

```
1 void loop() {
  //viene controllato se il semaforo \'e abilitato
3 if(CheckEnable()){

5   // nel caso in cui il semaforo sia abilitato
    switch(CurrentState){
7   case 1://se lo stato precedente risulta essere solo rosso
      CurrentState=2;
9     digitalWrite(6,HIGH); //imposta lo stato corrente a rosso e giallo
      delay(tickREDYELLOW); //viene fatto passare il tempo specificato all'inizio del file
11    break;
    case 2: //se lo stato precedente risulta essere rosso e giallo
13    CurrentState=3; //lo stato corrente viene spostato a verde
      digitalWrite(5,LOW);
15    digitalWrite(6,LOW);
      digitalWrite(7,HIGH);
17    delay(tickGREEN);
      break;
19  .
    .
21  .
```

Listing 2: semf-svizzero.ino

5 Falling-edge detector

Si vuole realizzare una FSM che riceve uno stream di bit su una linea di ingresso e che accende un LED tutte le volte che su questo si presenta un fronte di discesa secondo il modello di Moore e un'altra secondo quello di Mealy.

5.a Progettazione FSM e costruzione dei circuiti

Seguendo lo stesso procedimento usato nella sezione 2 per realizzare il semaforo disegniamo i diagrammi degli stati delle macchine, ne codifichiamo in bit gli stati, ricaviamo le tabella di verità per le transizioni tra questi e i valori in uscita, così da ottenere le equazioni logiche che le governano da tradurre nella realizzazione pratica delle connessioni tra memoria e circuiti di logica combinatoria.

5.a.1 Edge-detector FSM di Moore

Si è scelto di costruire la FSM di Moore per riconoscere fronti di discesa con 3 stati, dunque 2 bit di memoria Q_1Q_0 secondo il diagramma degli stati in fig. 11. Per cui se il segnale di rumore in ingresso rimane a livello logico basso o alto la macchina resta in stato LOW (00) o HIGH (01) in cui l'uscita è OUT = 0; se invece il

segnale IN scende a 0 durante lo stato HIGH la macchina entra nell'unico stato (EDGE := 10) in cui l'uscita è OUT = 1 prima di tornare ad uno degli stati precedenti a seconda del valore logico dell'ingresso al successivo fronte di salita del clock.

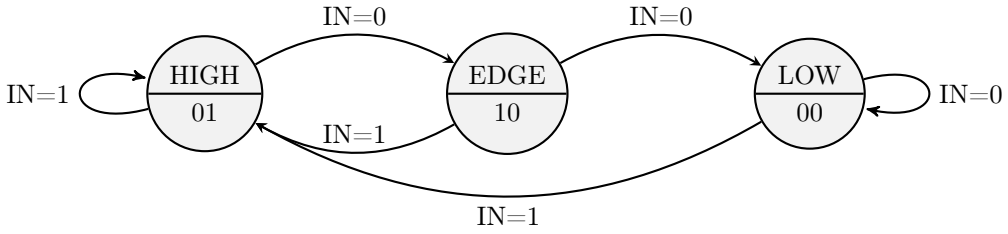

Figura 11: Edge-detector FSM di Moore

Dal diagramma ricaviamo la tabella di verità delle transizioni di stato con la stessa convenzione usata prima tra lo stato corrente Q_i e il successivo D_i come riportata in tabella 6.

State	IN	Q_1	Q_0	D_1	D_0	OUT
LOW	0	0	0	0	0	0
	1	0	0	0	1	0
EDGE	0	1	0	0	0	1
	1	1	0	0	1	1
HIGH	0	0	1	1	0	0
	1	0	1	0	1	0

Tabella 6: Tabella di verità per le transizioni tra gli stati del detector di Moore.

Dunque per semplificare le espressioni logiche delle transizioni e delle uscite si sono costruite le mappe di Karnaugh riportate in tabella 7, tabella 8 e tabella 9.

IN \ Q_1Q_0	00	01	11	10
0	0	0	X	0
1	1	1	X	1

Tabella 7: Mappa di Karnaugh per $D_0 = \text{IN}$

IN \ Q_1Q_0	00	01	11	10
0	0	1	X	0
1	0	0	X	0

Tabella 8: Mappa di Karnaugh per $D_1 = \overline{\text{IN}} \cdot Q_0$

IN \ Q_1Q_0	00	01	11	10
0	0	0	X	1
1	0	0	X	1

Tabella 9: Mappe di Karnaugh per $\text{OUT} = Q_1$

Finalmente dalle equazioni per gli stati futuri e delle uscite della FSM si è costruito il circuito riportato in fig. 12 a partire da due positive-edge-triggered D Flip-Flop (IC 74LS74), 1 porta AND (74LS08) e una porta NOT (74LS04).

5.a.2 Edge-detector FSM di Mealy

Per la realizzazione della FSM di Mealy per riconoscere fronti di discesa invece sono sufficienti 2 stati, quindi 1 bit di memoria, come si vede anche dal diagramma degli stati riportato in fig. 13. Anche in questo caso

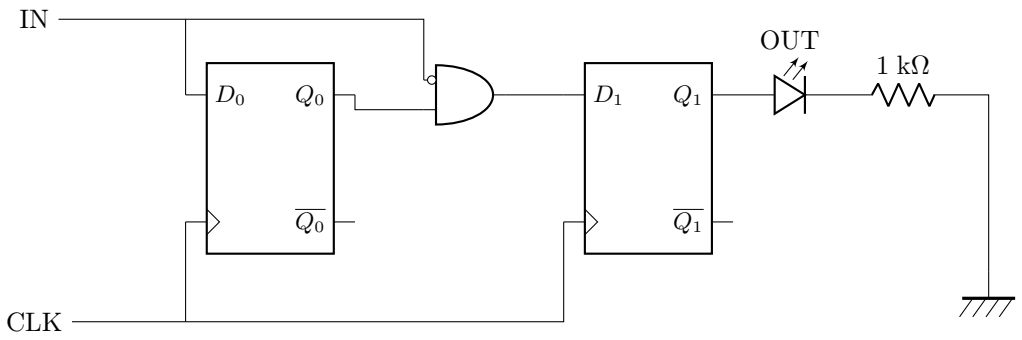

Figura 12: schema del detector Moore.

il segnale di rumore viene inviato all'ingresso D del FF, ma ora viene comparato continuamente con lo stato precedentemente registrato in memoria Q. Dunque il valore in uscita dal circuito è sempre OUT = 0 apparte quando la macchina registra la discesa del segnale IN a 0 durante lo stato HIGH ($Q = 1$), per cui OUT: $0 \to 1$ in maniera asincrona rispetto clock.

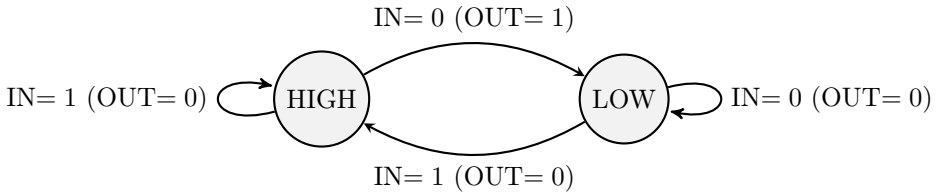

Figura 13: Edge detector FSM di Mealy

Come prima ricaviamo dal diagramma la tabella di verità delle transizioni di stato riportata in tabella 10 con le solite convenzioni; stavolta non è necessario ricorrere alle mappe di Karnaugh per semplificare le espressioni dello stato futuro $D = \text{IN}$ e dell'uscita $\text{OUT} = \overline{\text{IN}} \cdot Q$

$D = \text{IN}$	Q	OUT
0	0	0
0	1	1
1	0	0
1	1	0

Tabella 10: codifica binaria degli stati del detector di Mealy. $D = \text{IN}$; $\text{OUT} = \overline{\text{IN}} \cdot Q$

A partire da queste equazioni si è costruito il circuito riportato in fig. 14 con un positive-edge-triggered D Flip-Flop (IC 74LS74), 1 porta AND (74LS08) e una porta NOT (74LS04).

5.b Definizione dello stream di bit casuali in ingresso

Con la funzione Patterns di Waveform si invia un segnale (in DIO 5) di clock di frequenza $f_{\text{clk}} = 1$ kHz al pin (CLK) dei Flip-Flop e si genera con il canale DIO 6 uno stream di dati random alla stessa frequenza $f = f_{\text{clk}}$, già dalla schermata di Patterns è possibile notare come le commutazioni di stato del segnale pseudocasuale avvengano in corrispondenza dei fronti di discesa del segnale di clock.

Come prima si sono mantenuti i pin PRESET e CLEAR dei D-FF collegati a V_{CC} per evitare reset o clear spuri.

5.c Verifica del funzionamento e analisi della temporizzazione

Si sono acquisiti i segnali in ingresso (CLK = DIO 5, IN = DIO 6) e in uscita (OUT = DIO 7) dai circuiti edge-detector con la funzione Logic Analyzer dell'AD2, di cui riportiamo i risultati per il circuito di Moore in fig. 15 e per la FSM di Mealy in fig. 16.

Notiamo nel primo caso come l'uscita assume valore alto in maniera sincrona rispetto al segnale di clock, mentre nell'implementazione di Mealy l'uscita non aspetta il successivo fronte d'onda del clock per salire al livello logico alto e accendere il LED.

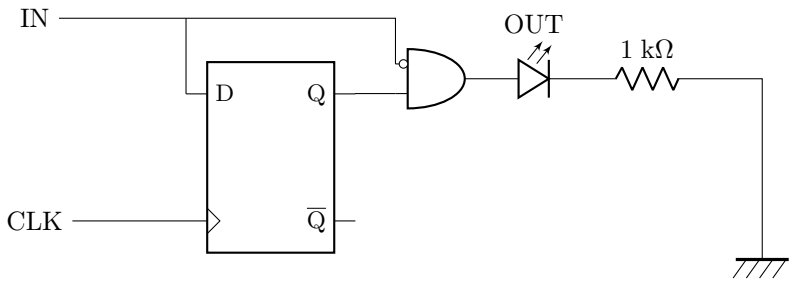

Figura 14: schema del detector Mealy.

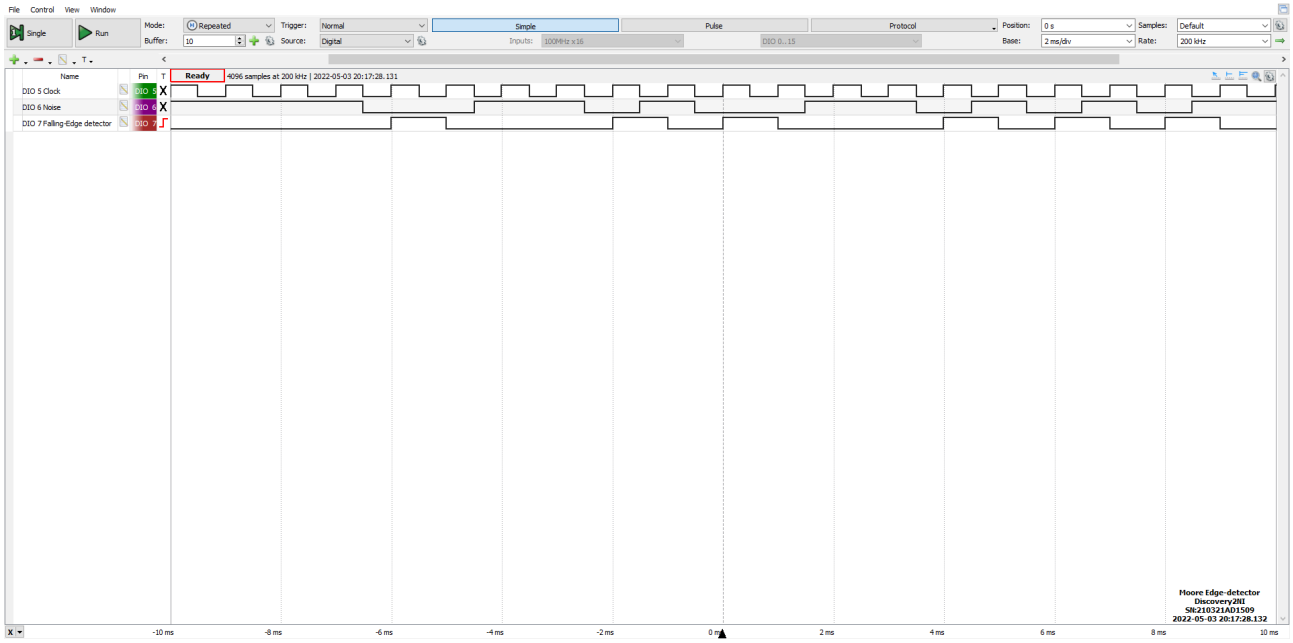

Figura 15: Acquisizione di un ciclo completo (frequenza 1 kHz) con Logic Analyzer dei segnali in ingresso e in uscita dall'edge detector di Moore.

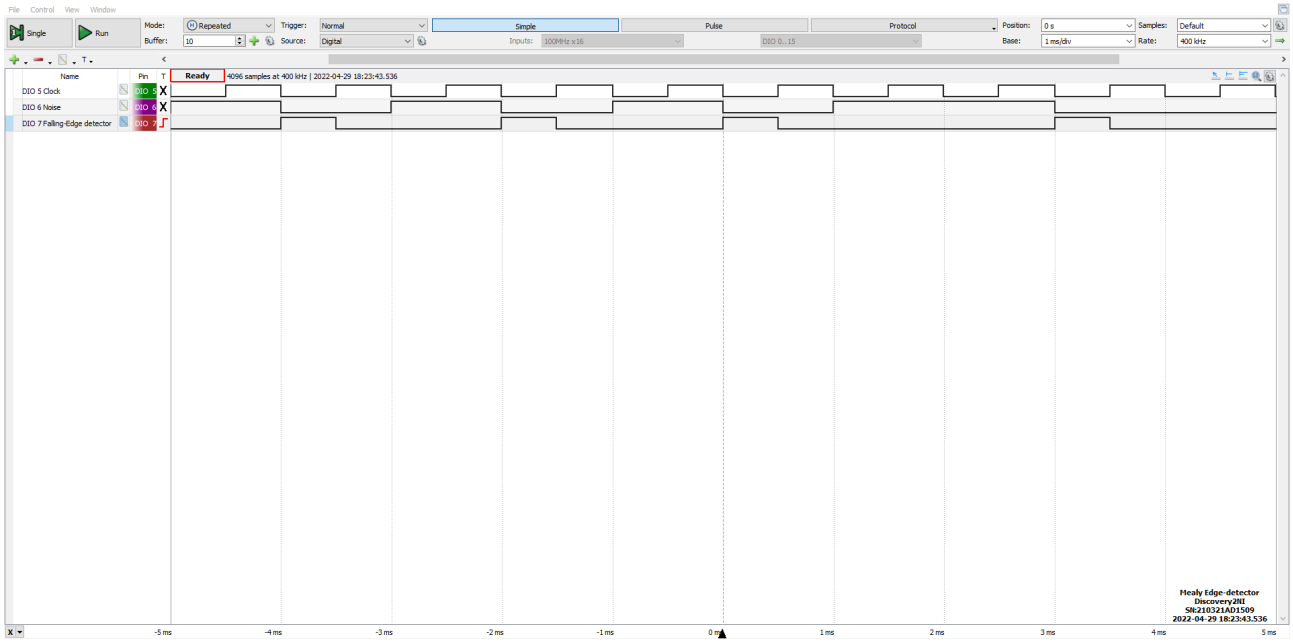

Figura 16: Acquisizione di un ciclo completo (frequenza 1 kHz) con Logic Analyzer dei segnali in ingresso e in uscita dal detector di Mealy.

Questo risulta compatibile con quanto ci si aspetta per le diverse temporizzazioni delle macchine a stati finiti realizzate secondo i modelli di Moore e Mealy

Conclusioni e commenti finali

Si è riusciti a progettare, costruire e verificare il corretto funzionamento di circuiti logici combinatori di diversa complessità e svariate applicazioni (e.g., sistemi di controllo e misura) costruiti con porte NOT, NAND, OR e D-FF. Inoltre si è riusciti ad apprezzare le diverse modalità di funzionamento delle macchine a stati finiti implementate secondo i modelli Moore e Mealy, ponendo particolare attenzione alle loro diverse temporizzazioni nei cambiamenti di stato, nonostante la bassa risoluzione temporale dell'AD2.

Dichiarazione

I firmatari di questa relazione dichiarano che il contenuto della relazione è originale, con misure effettuate dai membri del gruppo, e che tutti i firmatari hanno contribuito alla elaborazione della relazione stessa.