Domenico
Gagliardo
0612704485
Gruppo N°32

RELAZIONE PROGETTO TECNOLOGIE DIGITALI 2019/2020

TRACCIA 5

- Descrizione Del Sistema

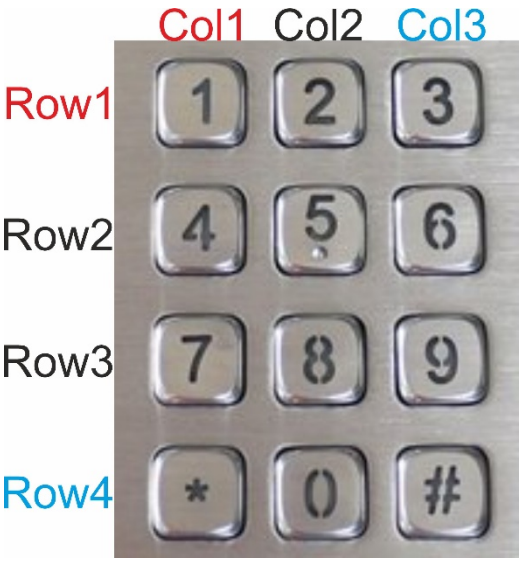
Il Sistema da realizzare consiste in un controllore che gestisce la corretta apertura e chiusura di un cancello. La fase di apertura è comandata da una sequenza di 3 cifre decimali fornita da un tastierino avente 12 pulsanti (cifre da 0 a 9, # e*). Ricevuta la sequenza, il cancello si porta in fase di apertura. Una volta aperto, vi rimane per 10 cicli di clock, dopodichè si porta in fase di chiusura. Se però qualcosa transita davanti la fotocellula il cancello si porta in fase di blocco. In questa fase attende 5 cicli di clock, dopodichè se l'ostacolo non è presente, continua la fase di chiusura, altrimenti si porta in fase di apertura.

- Scelte Progettuali Principali

Per Realizzare Il Progetto è stata scelta un' implementazione con 2 automi, combinati opportunamente con una vista strutturale:

1. Automa Principale (Automa del Cancello): Automa che gestisce il corretto funzionamento del cancello (Automa di Moore). Per questo automa abbiamo ritenuto fosse necessario l'aggiunta di due ulteriori segnali: uno in ingresso che controlla se il cancello è in fine corsa o meno (**Sensore**) e uno come uscita che gestisce il movimento del cancello, vale a dire il **Motore**.
2. Automa Secondario (Automa del Riconoscitore Sequenza): Automa che gestisce la corretta acquisizione ed elaborazione della sequenza attraverso il tastierino (Automa di Mealy) che viene poi collegato all'automa principale attraverso un port map.

Per quanto riguarda il tastierino, è stata scelta una codifica a matrice, considerando Un vettore colonna STD_LOGIC_VECTOR da 3 bit (COL, formato da Col1, Col2, Col3) per codificare le colonne e un vettore righe STD_LOGIC_VECTOR da 4 bit (ROW, formato da Row1, Row2, Row3, Row4) per codificare le righe. Abbiamo inoltre scelto come sequenza che comporta l'apertura del cancello la **sequenza 0-0-0**. Abbiamo inoltre considerato il tastierino di tipo non impulsivo.



Rosso=Bit più significativo.

Blu=Bit meno significativo.

Lo 0 in questo caso quindi è codificato come COL=010 e ROW=0001, perché premendo 0 attivo la seconda colonna e la quarta riga.

Quando non premo nulla quindi avrò COL=000 e ROW=0000.

Domenico
Gagliardo
0612704485
Gruppo N°32

- Realizzazione Dell'Automa Principale

Per la Realizzazione dell'Automa Principale Abbiamo Implementato Un Automa a 5 stati. Essendo le uscite in questo caso strettamente correlate agli ingressi abbiamo preferito utilizzare il modello di Moore. In questo caso in ingresso all'Automa non diamo COL e ROW ma un solo segnale (T), che varrà 1 se la sequenza inserita è corretta, 0 altrimenti. Oltre alla fotocellula e al segnale del tastierino abbiamo utilizzato anche un altro segnale (sensore), che monitora il movimento del cancello. Se il sensore è alto (S=1) indica che il cancello è in fine corsa (quindi completamente aperto o completamente chiuso). Se il sensore è basso (S=0) indica che il cancello è in movimento (non in fine corsa, in quanto non completamente aperto oppure non completamente chiuso).

- Descrizione Dell' Automa Principale

A prescindere dalla presenza o meno di un ostacolo davanti la fotocellula, a partire da **S0** (cancello chiuso), se la combinazione inserita è esatta (T=1) e il cancello è in fine corsa (S=1) si passa allo stato **S1** (cancello in apertura) accendendo sia il lampeggiante che il motore (L=1 e M=1).

Se invece non viene inserita la combinazione corretta (T=0) oppure si inserisce la combinazione corretta e il sensore ci dice che il cancello non è in fine corsa (T=1 e S=0, combinazione impossibile per lo stato inziale, perché abbiamo ipotizzato che il cancello è chiuso e quindi in fine corsa), a prescindere dal valore della fotocellula, permaniamo in **S0** con entrambe le uscite basse (M=0, L=0).

Il valore della combinazione è significativo solo se si vuole aprire il cancello (solo per passare da **S0** a **S1**), mentre per le altre fasi risulta ininfluente.

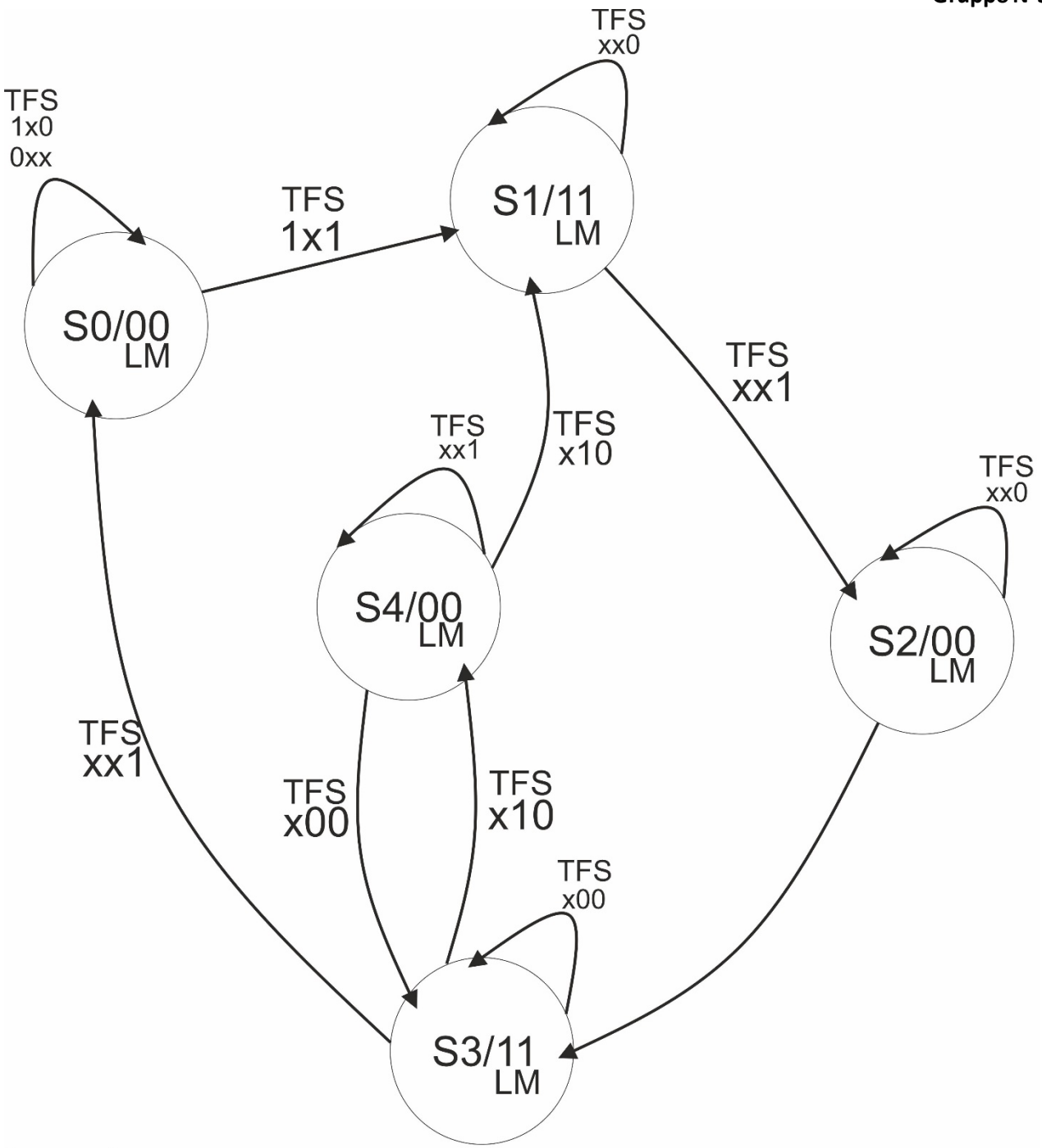
Ci troviamo in **S1** (cancello in apertura): in questo stato ci sono solo 2 possibilità. A prescindere dalla presenza o meno di un ostacolo, o permaniamo in **S1** con entrambe le uscite alte (M=1 L=1) se il cancello non è arrivato a fine corsa (S=0), oppure, passiamo allo stato **S2** (cancello aperto completamente) con entrambe le uscite basse (M=0 L=0) se il sensore ci "comunica" di essere arrivato a fine corsa (S=1).

Arrivati in **S2** si deve aspettare 10 cicli di clock. Alla fine di questi cicli si passa automaticamente allo stato **S3** (cancello in chiusura) con entrambe le uscite alte (L=1 M=1) tenendo soltanto conto del valore del sensore (S=1), perché S=0 è una combinazione impossibile dato che il cancello è aperto e quindi fermo (in questo caso si resta in **S2**), e non degli altri due segnali.

Lo stato **S3** (cancello in chiusura) ha entrambe le uscite alte (L=1 M=1). Da questo momento, sempre trascurando il valore del segnale T, se non è presente nessun ostacolo e, inoltre, se il cancello non è arrivato a fine corsa (F=0 S=0) il cancello continua a chiudersi (rimanendo in **S3**). Appena viene raggiunto il fine corsa (S=1), a prescindere dall'ostacolo (perché se il cancello è chiuso non ha senso tenere conto della fotocellula), il cancello è chiuso completamente (passa allo stato **S0** con entrambe le uscite basse). Se, altrimenti, il cancello non è chiuso completamente ed è presente un ostacolo (S=0 F=1) il cancello si blocca, restando semi-aperto, e andiamo nello stato **S4** con entrambe le uscite basse (si è scelto, dato che non era specificato, che nello stato di blocco il lampeggiante rimanga spento, quindi M=0 L=0).

Entrati in **S4** si permane per un periodo pari a 5 colpi di clock. Una volta terminata questa attesa ci sono 3 possibilità: a prescindere dai valori di T ed F, se il sensore di fine corsa è alto (S=1) si rimane in **S4** perché è una combinazione impossibile (il cancello non è chiuso completamente); se l'ostacolo non è più presente e il cancello non è in fine corsa (F=0 S=0) si ritorna nello stato di chiusura **S3**; mentre se l'ostacolo è ancora presente e il cancello non è in fine corsa (F=1 S=0) si va allo stato di apertura **S1**.

Domenico
Gagliardo
0612704485
Gruppo N°32



Ingressi:
T=Tastierino (1=Combinazione Inserita Esatta, 0=Combinazione Inserita Errata)
F=Fotocellula (1=Rileva Un Ostacolo, 0=Nessun Ostacolo Rilevato)
S=Sensore (1=Fine Corsa, 0=Non in Fine Corsa)
Uscite:
L=Lampeggiante (1=Acceso, 0=Spento)
M=Motore (1=Acceso, 0=Spento)
Stati:
S0=Cancello Chiuso (Stato Iniziale)
S1=Cancello In Fase Di Apertura
S2=Cancello Aperto (qui attendo 10 cicli di clock prima di transitare)
S3=Cancello In Fase Di Chiusura
S4=Cancello In Blocco (qui attendo 5 cicli di clock prima di transitare)

Domenico
Gagliardo
0612704485
Gruppo N°32

- Realizzazione Dell'Automa Secondario

Per la Realizzazione Dell'Automa Secondario abbiamo invece implementato un automa a 11 stati. In questo caso invece abbiamo utilizzato il modello di Mealy, in quanto utilizzare Moore avrebbe comportato uno stato in più per gestire la commutazione del segnale di uscita. Questo automa ha quindi in ingresso COL e ROW e ha una sola uscita (Tastierino_Out) che vale 1 se la sequenza riconosciuta è quella corretta, 0 altrimenti.

- Descrizione Dell'Automa Secondario

L'Automa è caratterizzato da 2 percorsi, uno percorso nel caso di inserimento di una sequenza corretta, e un altro percorso in caso di pressioni non conformi oppure anomale (premo più pulsanti contemporaneamente).

Inizialmente mi trovo nello stato iniziale, ovvero lo stato **S0**, con COL=000, ROW=0000 e T=0. Se Premo Il Tasto 0 (quindi ho COL=010 e ROW=0001) vado in **S0AttGiusto** con T=0, in cui dopo aver premuto per una volta lo 0 mi posiziono in attesa. Se invece premo qualsiasi altra cosa diversa dallo 0 vado in **S0AttSbagliato** con T=0, in cui dopo aver premuto per una volta qualcosa diverso da 0 mi posiziono in attesa.

Se sono nello stato **S0AttGiusto**, vi permango finchè mantengo premuto 0 (quindi con COL=010, ROW=0001 e T=0). Non appena smetto di premere 0, mi porto in **S1Giusto** (quindi con COL=000, ROW=0000 e T=0), dove ho riconosciuto il 1° elemento della sequenza correttamente. Se invece durante la pressione di 0 premo contemporaneamente altri pulsanti del tastierino, mi porto in **S0AttSbagliato** con T=0, stato di attesa dove ho immesso il 1° elemento di una sequenza errata.

Se sono nello stato **S1Giusto**, vi permango finchè non premo nulla (quindi con COL=000, ROW=0000 e T=0). Non appena premo un altro 0, mi porto in **S1AttGiusto** (quindi con COL=010, ROW=0001 e T=0), dove ho immesso uno 0. Se invece premo una sequenza errata, mi porto in **S1AttSbagliato** con T=0, stato di attesa dove ho immesso il 2° elemento di una sequenza che diventerà quindi errata.

Se sono nello stato **S1AttGiusto**, vi permango finchè mantengo premuto 0 (quindi con COL=010, ROW=0001 e T=0). Non appena smetto di premere 0, mi porto in **S2Giusto** (quindi con COL=000, ROW=0000 e T=0), dove ho riconosciuto il 2° elemento della sequenza correttamente. Se invece durante la pressione di 0 premo contemporaneamente altri pulsanti del tastierino, mi porto in **S1AttSbagliato** con T=0, stato di attesa dove ho immesso il 2° elemento di una sequenza che diventerà quindi errata.

Se sono nello stato **S2Giusto**, vi permango finchè non premo nulla (quindi con COL=000, ROW=0000 e T=0). Non appena premo un altro 0, mi porto in **S2AttGiusto** (quindi con COL=010, ROW=0001 e T=0), dove ho immesso uno 0. Se invece premo una sequenza errata, mi porto in **S2AttSbagliato** con T=0, stato di attesa dove ho immesso il 3° elemento di una sequenza che diventerà quindi errata.

Se sono nello stato **S2AttGiusto**, vi permango finchè mantengo premuto 0 (quindi con COL=010, ROW=0001 e T=0). Non appena smetto di premere 0, mi porto in **S0** (quindi con COL=000, ROW=0000 e T=1), dove ho riconosciuto la sequenza correttamente. Se invece durante la pressione di 0 premo contemporaneamente altri pulsanti del tastierino, mi porto in **S2AttSbagliato** con T=0, stato di attesa dove ho immesso il 3° elemento di una sequenza che diventerà quindi errata.

Se sono nello stato **S0AttSbagliato**, vi permango finchè mantengo premuto qualcosa sul tastierino con T=0. Non appena smetto di premere mi porto in **S1Sbagliato** (quindi con COL=000, ROW=0000 e T=0), dove ho riconosciuto il 1° elemento di una sequenza errata.

Domenico
Gagliardo
0612704485
Gruppo N°32

Se sono nello stato **S1Sbagliato**, vi permango finchè non premo nulla (quindi con COL=000, ROW=0000 e T=0). Non appena premo qualcosa sul tastierino, che sia 0 o meno, mi porto in **S1AttSbagliato**, dove ho immesso il 2° elemento di una sequenza errata.

Se sono nello stato **S1AttSbagliato**, vi permango finchè mantengo premuto qualcosa sul tastierino con T=0. Non appena smetto di premere mi porto in **S2Sbagliato** (quindi con COL=000, ROW=0000 e T=0), dove ho riconosciuto il 2° elemento di una sequenza errata.

Se sono nello stato **S2Sbagliato**, vi permango finchè non premo nulla (quindi con COL=000, ROW=0000 e T=0). Non appena premo qualcosa sul tastierino, che sia 0 o meno, mi porto in **S2AttSbagliato**, dove ho immesso il 3° elemento di una sequenza errata.

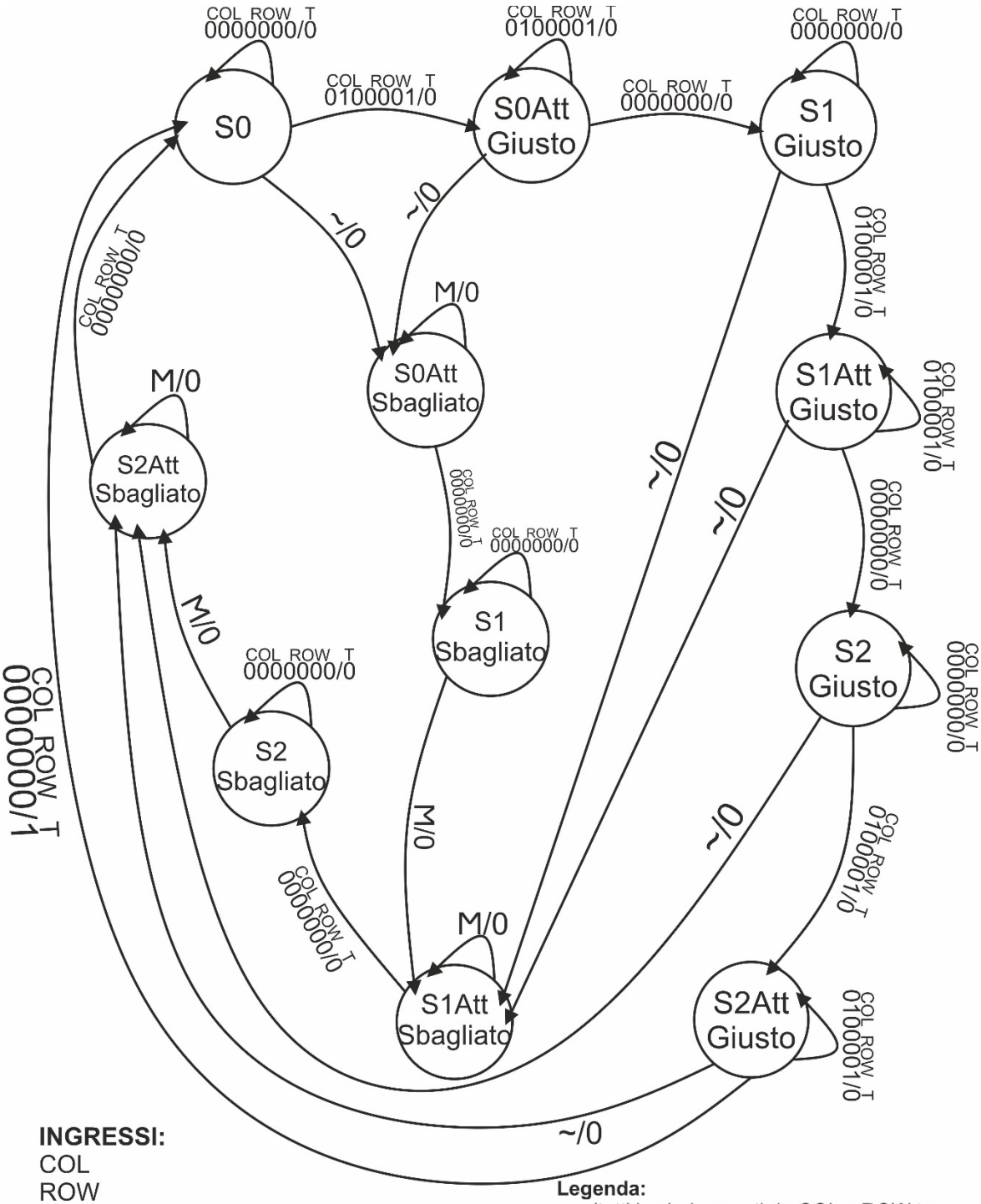
Se sono nello stato **S2AttSbagliato**, vi permango finchè mantengo premuto qualcosa sul tastierino con T=0. Non appena smetto di premere mi porto in **S0** (quindi con COL=000, ROW=0000 e T=0), dove ho riconosciuto il 3° elemento di una sequenza che però essendo errata, non mi fa alzare l'uscita.

Abbiamo scelto di implementare un automa di questo tipo proprio per garantire che l'immissione e il riconoscimento della sequenza sia di 3 cifre decimali per volta. Infatti con un automa molto più semplice, ma che al primo input errato si sarebbe resettato, portandosi allo stato iniziale S0, avremmo potuto avere problemi legati alla corretta elaborazione di sequenze di 3 cifre decimali.

ESEMPIO :

In questo caso infatti, il nostro automa non porterà mai l'uscita T ad 1 in quanto riconoscerà 2 sequenze di 3 cifre decimali, rispettivamente 100 ed 010, che non corrispondono alla sequenza corretta. L'automa più semplice avrebbe invece riconosciuto 000 ed alzando l'uscita proprio perché a fronte del primo elemento della sequenza in input (1) si sarebbe resettato.

Domenico
Gagliardo
0612704485
Gruppo N°32



INGRESSI:
COL
ROW
USCITA:
TASTIERINO_OUT (T)
STATI:

Legenda:
~ (tutti i valori assunti da COL e ROW tranne 0000000 ed 0100001)
M (tutti i valori assunti da COL e ROW tranne 0000000)

S0=Stato di Attesa senza aver premuto nulla (Stato Iniziale)
S0AttGiusto=Stato di Attesa dove premo 0 (uno 0)
S1Giusto=Stato In Cui Ho Riconosciuto uno 0
S1AttGiusto=Stato di Attesa dove premo 0 (due 0)
S2Giusto=Stato In Cui Ho Riconosciuto due 0
S2AttGiusto=Stato di Attesa dove premo 0 (tre 0)
S0AttSbagliato=Stato di Attesa dove premo erroneamente (una pressione)
S1Sbagliato=Stato in Cui Riconosco una pressione errata
S1AttSbagliato=Stato di Attesa dove premo erroneamente (due pressioni)
S2Sbagliato=Stato in Cui Riconosco una o più pressioni errate
S2AttSbagliato=Stato di Attesa dove premo erroneamente (tre pressioni)

Domenico
Gagliardo
0612704485
Gruppo N°32

Codifica Stati Automa Principale e Automa Secondario

State	Encoding
s0	000
s1	001
s2	010
s3	011
s4	100

State	Encoding
s0	0000
s0att_giusto	0001
s0att_sbagliato	0010
s1_giusto	0011
s1_sbagliato	0100
s1att_giusto	0101
s1att_sbagliato	0110
s2_giusto	0111
s2_sbagliato	1000
s2att_giusto	1001
s2att_sbagliato	1010

- Realizzazione Automa Principale VHDL (Modello di Moore)

Coerentemente con quanto stabilito nelle scelte progettuali, l'Automa Principale viene descritto con questa entity dove :

- **Col** è un STD_LOGIC_VECTOR di 3 bit in ingresso per codificare le colonne del tastierino
- **Row** è un STD_LOGIC_VECTOR di 4 bit in ingresso per codificare le righe del tastierino
- **Fotocellula** è un STD_LOGIC in ingresso che controlla la presenza di ostacoli
- **Sensore** è un STD_LOGIC in ingresso che controlla il "fine corsa" del cancello
- **CLK** è un STD_LOGIC in ingresso ed è il segnale di sincronizzazione
- **RESET** è un STD_LOGIC in ingresso che serve per resettare l'automa
- **count_S2_debug** è un STD_LOGIC_VECTOR di 4 bit in uscita. Questo è un segnale inserito per controllare l'evoluzione del comportamento dell'automa che assume il valore corrispondente al conteggio in atto nello Stato S2.
- **count_S4_debug** è un STD_LOGIC_VECTOR di 3 bit in uscita. Questo è un segnale inserito per controllare l'evoluzione del comportamento dell'automa che assume il valore corrispondente al conteggio in atto nello Stato S4.
- **state_debug** è un STD_LOGIC_VECTOR di 3 bit in uscita. Questo è un segnale inserito per controllare l'evoluzione del comportamento dell'automa che assume il valore corrispondente allo stato corrente dell'automa.
- **Lampeggiante** è un STD_LOGIC in uscita e serve per attivare a seconda dello stato il lampeggiante
- **Motore** è un STD_LOGIC in uscita e serve per attivare a seconda dello stato il motore.

Nella prima parte del codice, dopo l'entity, vi è l'istanziazione del componente riconoscitore sequenza, vale a dire l'automa secondario che andiamo a richiamare nell' automa principale attraverso un Port map. Poi vi è la dichiarazione degli stati attraverso la definizione di un **tipo stato,** e anche la dichiarazione di altri due segnali **current state** e **next state** sempre di **tipo stato** che servono per la transizione tra i vari stati. Dichiaro poi due segnali di tipo std logic **time over S2 e time over S4** che mi indicano rispettivamente quando i cicli di clock di attesa nello stato S2 e nello stato S4 sono terminati. Inizializzo poi due variabili di tipo intero **count S2 ed ncountS2** che mi servono nel processo di timer S2 . Queste variabili sono vincolate in un range che va da 0 a 10. Inizializzo allo stesso modo poi due variabili di tipo intero **count S4 ed ncountS4** che mi servono nel processo di timer S4 . Queste variabili sono vincolate in un range che va da 0 a 5. Inizializzo poi un segnale di **enable** di tipo std logic che mi servirà successivamente per assegnare l'uscita al lampeggiante. Inizializzo inoltre anche altri segnali che mi servono per effettuare il Port map del

component riconoscitore sequenza. Questi sono **Tastierino_state_debug** (all'interno del nostro automa principale non serve e non verrà utilizzato, serve però per effettuare un corretto port map)che è un std_logic_vector di 4 bit, **Tastierino_Out** (uscita dell' automa riconoscitore sequenza che andrò ad utilizzare per eseguire le transizioni di stato)di tipo std logic e **Reset_Tastierino** (reset dell'automa riconoscitore sequenza, inizializzato a 0) che è sempre un std logic. Inizializzo infine un ultimo segnale di tipo std logic che mi serve per il lampeggiante **toggle**. A questo punto effettuo il Port map del componente riconoscitore sequenza con i segnali dichiarati precedentemente ed alcuni segnali dell' entity del cancello, vale a dire **COL** e **ROW** e **CLK** (Quest ultimo segnale in particolare mi garantirà la sincronia fra i due automi in quanto è lo stesso clock del cancello). A questo punto iniziano i vari processi che compongono il codice VHDL:

- Processo Sincrono (registro dello stato corrente)

Questo processo innanzitutto contiene nella sensitivity list solamente il segnale di **clock**. Questo vuol dire che questo processo verrà eseguito solo a fronte della variazione del segnale di **clock**. In questo processo se il fronte del clock è alto (**rising edge**) allora andiamo a effettuare due possibili operazioni. Se il **reset** è alto impostiamo **current state** allo stato iniziale vale a dire **S0** e inoltre azzeriamo **count S2** e **count S4**. In caso contrario aggiorniamo **current state** con **next state**, **count S2** con **ncount S2** e **count S4** con **ncount S4**. A questo punto si conclude il processo.

- Processo di Transizione di Stato

Questo processo invece nella sensitivity list contiene **current state**, la **fotocellula**, il **sensore**, **time over S2**, **time over S4** e **tastierino out**. Quindi questo processo verrà eseguito a fronte della variazione di uno di questi segnali. A questo punto all'interno del processo vi è un case when che viene influenzato da **current state**. A seconda quindi del valore di **current state** verranno eseguite le transizioni di stato. In questo processo in particolare catturo anche la transizione di Stato e la memorizzo in **state debug**. Questo mi servirà poi nel test bench per individuare univocamente lo stato in cui mi trovo. Ciò viene eseguito a prescindere del valore di **current state** in quanto in ogni caso vi è questa istruzione. In particolare **stato'POS(current state)** mi restituisce proprio la posizione di **current state**. Questa verrà poi convertita in un std_logic_vector (di 3 bit) e assegnata a **state debug**. In questo case when inoltre impongo che **reset tastierino** sia pari a **0** solo nello stato iniziale, vale a dire **S0**. Questo perché è solo in quello stato che mi interessa il valore in uscita dal tastierino. In tutti gli altri stati escluso **S0** vado a imporre che **reset tastierino** sia pari ad **1**. In questo modo il tastierino non avrà più efficacia all'interno del cancello tranne che nello stato inziale, vale a dire **S0**. Questa scelta progettuale è scaturita dal fatto che se nello stato di chiusura, vale a dire **S3**, avremmo digitato ad esempio 1 o 2 zeri, arrivati allo stato iniziale, vale a dire **S0**, sarebbero bastati rispettivamente 2 zeri oppure anche uno solo per **sbloccare il cancello**. Questo sarebbe stato un comportamento anomalo, e quindi non conforme alle specifiche.

- Processo di Output (Uscita Motore)

Anche questo processo all'interno della sua sensitivity list contiene solo **current state**. In particolare anche il suo corpo consiste in un case when sensibile al segnale **current state**. A seconda del valore che **current state** assume si assegnerà un' uscita corrispondente a quello stato. In particolare in questo processo mi limito ad assegnare solo l'uscita **motore** mentre per quanto riguarda l'uscita **lampeggiante** vado solamente a modificare il segnale di **enable**, vale a dire il segnale di appoggio per assegnare poi l'uscita al **lampeggiante.** L'assegnazione dell'uscita lampeggiante infatti sarà gestita in un processo a parte.

- Processo di Output (Uscita Lampeggiante)

Anche questo processo contiene nella sua sensitivity list solamente il segnale di **clock**. in questo processo in presenza del fronte alto del clock (**rising edge**) vado a complementare il segnale di appoggio **toggle**. Subito

Domenico
Gagliardo
0612704485
Gruppo N°32

dopo la conclusione del processo vado ad assegnare l'uscita **lampeggiante** come una **and** tra questo segnale di appoggio **toggle** e **enable** (istruzione che verrà quindi eseguita in modo concorrente rispetto alle altre). Questo mi consente di ottenere il comportamento desiderato: quando **enable** è uguale a **0** il lampeggiante sarà spento. Quando invece **enable** è uguale a **1** il lampeggiante assumerà il valore del segnale **toggle**, quindi un comportamento oscillante. In particolare sia durante la fase di dichiarazione delle variabili che in questo processo vi è del codice commentato: Questo perché per frequenze inferiori di molto rispetto a quella del clock avrebbe senso utilizzare una costante max count e un contatore count e adoperare quelle istruzioni commentate all'interno del processo. In questo caso per le forme d'onda del test bench si è ritenuto più semplice adoperare una frequenza pari alla metà di quella del clock, avendo quindi un periodo pari al doppio e quindi all'interno delle forme d'onda una visione più immediata dell'oscillazione del segnale **lampeggiante**. In caso si volesse ottenere un'altra frequenza per il lampeggiante basterebbe adoperare quelle istruzioni commentate all'interno del processo e calcolarsi la costante max count appropriata a quella frequenza come:

maxcount=1/2*frequenza_clock/frequenza_desiderata

Dove la frequenza del clock è proprio la frequenza di ingresso. La frequenza desiderata può essere calcolata semplicemente come il reciproco del periodo. f=1/T.

- Processo Timer S2

Questo processo serve per attuare il conteggio dei cicli di clock una volta arrivati all' interno dello Stato **S2**. all'interno della sua sensitivity list vi è quindi **current state** e **count S2**. A questo punto il processo valuta se **current state** è uguale a **S2**. Infatti qualora **current state** non fosse uguale ad **S2** si pone **ncount S2** a **0** e **time over S2** a **0**. Se **current state** è uguale a **S2** si valuta **count S2**: se quest'ultimo è uguale a **10** si pone **time over S2** ad **1** e si azzera **ncount S2**, che andrà poi ad azzerare **count S2** nel processo sincrono al prossimo fronte di salita del clock. Se **count S2** non è uguale a **10** si pone **ncount S2** a **count S2 +1** e si pone **time over S2** a **0**. A questo punto assegno a **count S2 debug** il valore di **count S2** convertito in std_logic_vector su 4 bit (istruzione che verrà quindi eseguita in modo concorrente rispetto alle altre).

- Processo Timer S4

Questo processo serve per attuare il conteggio dei cicli di clock una volta arrivati all' interno dello Stato **S4**. all'interno della sua sensitivity list vi è quindi **current state** e **count S4**. A questo punto il processo valuta se **current state** è uguale a **S4**. Infatti qualora **current state** non fosse uguale ad **S4** si pone **ncount S4** a **0** e **time over S4** a **0**. Se **current state** è uguale a **S4** si valuta **count S4**: se quest'ultimo è uguale a **5** si pone **time over S4** ad **1** e si azzera **ncount S4**, che andrà poi ad azzerare **count S4** nel processo sincrono al prossimo fronte di salita del clock. Se **count S4** non è uguale a **5** si pone **ncount S4** a **count S4 +1** e si pone **time over S4** a **0**. A questo punto assegno a **count S4 debug** il valore di **count S4** convertito in std_logic_vector su 4 bit (istruzione che verrà quindi eseguita in modo concorrente rispetto alle altre).

Domenico
Gagliardo
0612704485
Gruppo N°32

CODICE VHDL AUTOMA PRINCIPALE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Cancello is
   Port ( COL : in  STD_LOGIC_VECTOR (2 downto 0);
          ROW : in  STD_LOGIC_VECTOR (3 downto 0);
          Fotocellula : in  STD_LOGIC;
          Sensore : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          RESET : in  STD_LOGIC;
          count_S2_debug : out  STD_LOGIC_VECTOR (3 downto 0);
          count_S4_debug : out  STD_LOGIC_VECTOR (2 downto 0);
          state_debug : out  STD_LOGIC_VECTOR (2 downto 0);
          Lampeggiante : out  STD_LOGIC;
          Motore : out  STD_LOGIC);
end Cancello;
architecture Behavioral of Cancello is
Component Riconoscitore_Sequenza port (
COL : in STD_LOGIC_VECTOR (2 downto 0);
ROW : in STD_LOGIC_VECTOR (3 downto 0);
CLK: in STD_LOGIC;
RESET : in STD_LOGIC;
state_debug : out  STD_LOGIC_VECTOR (3 downto 0);
Tastierino_Out : out  STD_LOGIC);
end component;
type stato is (S0,S1,S2,S3,S4);
signal current_state,next_state : stato;
signal Timeover_S2,Timeover_S4 : std_logic;
signal count_S2,ncount_S2 : integer range 0 to 10;
signal count_S4,ncount_S4 : integer range 0 to 5;
-- Segnale di appoggio per abilitare  il lampeggiante e uscita del tastierino
signal Enable : STD_LOGIC;
signal Tastierino_state_debug : STD_LOGIC_VECTOR (3 downto 0);
signal Tastierino_Out : STD_LOGIC;
signal RESET_Tastierino : STD_LOGIC :='0';
-- Segnali per processo lampeggiante (Inizializzo toggle ad 1 per comodità nelle forme d'onda)
--constant max_count : natural := 1;
--signal count : natural range 0 to max_count;
signal toggle : STD_LOGIC :='1';
begin
RC : Riconoscitore_Sequenza port map
(COL,ROW,CLK,RESET_Tastierino,Tastierino_state_debug,Tastierino_Out);
--Processo Sincrono (registro dello stato corrente)
process (clk)
begin
if (rising_edge(clk)) then
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
        if(reset='1') then
                current_state<=S0;
                count_S2<=0;
                count_S4<=0;
        else
                current_state<=next_state;
                count_S2<=ncount_S2;
                count_S4<=ncount_S4;
        end if;
end if;
end process;
--Processo di Transizione di Stato
process (current_state,Fotocellula,Sensore,Timeover_S2,Timeover_S4,Tastierino_Out)
begin
        case (current_state) is
                                        when S0=>
                                                RESET_Tastierino<='0';
                                                if (Tastierino_Out='1' and Sensore='1') then --In fase di
apertura non mi interessa della fotocellula,apro lo stesso
                                                        next_state<=S1;
                                                else
                                                        next_state<=S0;
                                                end if;
        state_debug<=conv_std_logic_vector(stato'POS(current_state),3);
                                        when S1=>
                                                RESET_Tastierino<='1';
                                                if (Sensore='1') then
                                                        next_state<=S2;
                                                else
                                                        next_state<=S1;
                                                end if;
        state_debug<=conv_std_logic_vector(stato'POS(current_state),3);
                                        when S2=>
                                                RESET_Tastierino<='1';
                                                if (Timeover_S2='1' and Sensore='1') then
                                                        next_state<=S3;
                                                else
                                                        next_state<=S2;
                                                end if;
        state_debug<=conv_std_logic_vector(stato'POS(current_state),3);
                                        when S3=>
                                                RESET_Tastierino<='1';
                                                if (Sensore='1') then
                                                        next_state<=S0;
                                                elsif (Fotocellula='0' and Sensore='0') then
                                                        next_state<=S3;
                                                else
                                                        next_state<=S4;
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
                                        end if;
        state_debug<=conv_std_logic_vector(stato'POS(current_state),3);
                                    when S4=>
                                        RESET_Tastierino<='1';
                                        if (Timeover_S4='0') then
                                            next_state<=S4;
                                        elsif (Sensore='1') then
                                            next_state<=S4;
                                        elsif (Fotocellula='1' and Sensore='0') then
                                            next_state<=S1;
                                        else
                                            next_state<=S3;
                                        end if;

        state_debug<=conv_std_logic_vector(stato'POS(current_state),3);
                                    when others=>
                                        RESET_Tastierino<='1';
                                        next_state<=S0;
                end case;
end process;
-- Processo di Output (Uscita Motore)
process (current_state)
begin
        case (current_state) is
                when S0 =>  Enable<='0'; Motore<='0';
                when S1 =>  Enable<='1'; Motore<='1';
                when S2 =>  Enable<='0'; Motore<='0';
                when S3 =>  Enable<='1'; Motore<='1';
                when S4 =>  Enable<='0'; Motore<='0';
                when others => Enable<='0'; Motore<='0';
        end case;
end process;
-- Processo di Output (Uscita Lampeggiante)
process(CLK)
begin
if(rising_edge(CLK)) then
                --if (count=max_count-1) then
                toggle<= not toggle;
                --count<=0;
                --else
                --count<=count +1;
                --end if;
        end if;
end process;
Lampeggiante<=toggle and Enable;
-- Processo Timer Stato S2
Timer_S2:process(current_state,count_S2)
begin
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
	if (current_state=S2) then
			if (count_S2=10) then
				ncount_S2<=0;
				timeover_S2<='1';
			else
				ncount_S2<=count_S2+1;
				timeover_S2<='0';
			end if;
	else
		ncount_S2<=0;
		timeover_S2<='0';
	end if;
end process;
count_S2_debug<=conv_std_logic_vector(count_S2,4);
-- Processo Timer Stato S4
Timer_S4:process(current_state,count_S4)
begin
	if (current_state=S4) then
			if (count_S4=5) then
				ncount_S4<=0;
				timeover_S4<='1';
			else
				ncount_S4<=count_S4+1;
				timeover_S4<='0';
			end if;
	else
		ncount_S4<=0;
		timeover_S4<='0';
	end if;
end process;
count_S4_debug<=conv_std_logic_vector(count_S4,3);
end Behavioral;
```

Domenico
Gagliardo
0612704485
Gruppo N°32

- Realizzazione Automa Secondario VHDL (Modello di Mealy)

Coerentemente con quanto stabilito nelle scelte progettuali, l'Automa Principale viene descritto con questa entity dove :

- **Col** è un STD_LOGIC_VECTOR di 3 bit in ingresso per codificare le colonne del tastierino
- **Row** è un STD_LOGIC_VECTOR di 4 bit in ingresso per codificare le righe del tastierino
- **CLK** è un STD_LOGIC in ingresso ed è il segnale di sincronizzazione
- **RESET** è un STD_LOGIC in ingresso che serve per resettare l'automa
- **state_debug** è un STD_LOGIC_VECTOR di 4 bit in uscita. Questo è un segnale inserito per controllare l'evoluzione del comportamento dell'automa che assume il valore corrispondente allo stato corrente dell'automa.
- **Tastierino_Out** è un STD_LOGIC in uscita e serve per controllare se la sequenza è stata riconosciuta o meno.

Nella prima parte del codice, dopo l'entity, vi è la dichiarazione degli stati attraverso la definizione di un **tipo stato,** e anche la dichiarazione di altri due segnali **current state** e **next state** sempre di **tipo stato** che servono per la transizione tra i vari stati. A questo punto iniziano i vari processi che compongono il codice VHDL:

- Processo Sincrono (registro dello stato corrente)

Questo processo innanzi tutto contiene nella sensitivity list solamente il segnale di **clock**. Questo vuol dire che questo processo verrà eseguito solo a fronte della variazione del segnale di **clock**. In questo processo se il fronte del clock è alto (**rising edge**) allora andiamo a effettuare due possibili operazioni. Se il **reset** è alto impostiamo **current state** allo stato iniziale vale a dire **S0**. In caso contrario aggiorniamo **current state** con **next state**. A questo punto si conclude il processo.

- Processo di Transizione di Stato e di Output

Questo processo invece nella sensitivity list contiene **current state**, **COL** e **ROW**. Quindi questo processo verrà eseguito a fronte della variazione di uno di questi segnali. Prima del processo dichiaro due costanti di tipo std_logic_vector (rispettivamente per **COL** di 3 bit e per **ROW** di 4 bit). Queste costanti rappresentano il valore corretto di **COL** e **ROW** (valore da assegnare a **COL** e **ROW** per ottenere la sequenza corretta). A questo punto all'interno del processo vi è un case when che viene influenzato da **current state**. A seconda quindi del valore di **current state** verranno eseguite le transizioni di stato. In questo processo in particolare catturo anche la transizione di Stato e la memorizzo in **state debug**. Questo mi servirà poi nel test bench per individuare univocamente lo stato in cui mi trovo. Ciò viene eseguito a prescindere del valore di **current state** in quanto in ogni caso vi è questa istruzione. In particolare **stato'POS(current state)** mi restituisce proprio la posizione di **current state**. Questa verrà poi convertita in un std_logic_vector (di 4 bit) e assegnata a **state debug**. Sempre all'interno di questo case when, subito dopo aver gestito le transizioni di stato in ognuno dei possibili valori che **current state** può assumere, assegno anche l'uscita **Tastierino Out**.

Domenico
Gagliardo
0612704485
Gruppo N°32

CODICE VHDL AUTOMA SECONDARIO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Riconoscitore_Sequenza is
   Port ( COL : in  STD_LOGIC_VECTOR (2 downto 0);
          ROW : in  STD_LOGIC_VECTOR (3 downto 0);
          CLK : in  STD_LOGIC;
          RESET : in  STD_LOGIC;
          state_debug : out  STD_LOGIC_VECTOR (3 downto 0);
          Tastierino_Out : out  STD_LOGIC);
end Riconoscitore_Sequenza;
architecture Behavioral of Riconoscitore_Sequenza is
type stato is
(S0,S0att_giusto,S0att_sbagliato,S1_giusto,S1_sbagliato,S1att_giusto,S1att_sbagliato,S2_giusto,S2_sbagliat
o,S2att_giusto,S2att_sbagliato);
signal current_state,next_state : stato;
begin
--Processo Sincrono (registro dello stato corrente)
process (clk)
begin
if (rising_edge(clk)) then
          if(reset='1') then
                    current_state<=S0;
          else
                    current_state<=next_state;
          end if;
end if;
end process;
--Processo di Transizione di Stato e di Output(Separo l'assegnazione fra stati e l'assegnazione delle uscite
con 2 if separati per una migliore leggibilità del codice)
process (current_state,COL,ROW)
constant sequenza_col : STD_LOGIC_VECTOR (2 downto 0) := "010";
constant sequenza_row : STD_LOGIC_VECTOR (3 downto 0) := "0001";
begin
case (current_state) is
                    when S0 =>
                              if (COL=sequenza_col and ROW=sequenza_row) then
                                        next_state<=S0att_giusto;
                              elsif (COL="000" and ROW="0000") then
                                        next_state<=S0;
                              else
                                        next_state<=S0att_sbagliato;
                              end if;
                                        Tastierino_Out<='0';
                                        state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
                    when S0att_giusto=>
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
		if (COL=sequenza_col and ROW=sequenza_row) then
			next_state<=S0att_giusto;
		elsif (COL="000" and ROW="0000") then
			next_state<=S1_giusto;
		else
			next_state<=S0att_sbagliato;
		end if;
			Tastierino_Out<='0';
			state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S1_giusto=>
		if (COL=sequenza_col and ROW=sequenza_row) then
			next_state<=S1att_giusto;
		elsif (COL="000" and ROW="0000") then
			next_state<=S1_giusto;
		else
			next_state<=S1att_sbagliato;
		end if;
			Tastierino_Out<='0';
			state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S1att_giusto=>
		if (COL=sequenza_col and ROW=sequenza_row) then
			next_state<=S1att_giusto;
		elsif (COL="000" and ROW="0000") then
			next_state<=S2_giusto;
		else
			next_state<=S1att_sbagliato;
		end if;
			Tastierino_Out<='0';
			state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S2_giusto=>
		if (COL=sequenza_col and ROW=sequenza_row) then
			next_state<=S2att_giusto;
		elsif (COL="000" and ROW="0000") then
			next_state<=S2_giusto;
		else
			next_state<=S2att_sbagliato;
		end if;
			Tastierino_Out<='0';
			state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S2att_giusto=>
		if (COL=sequenza_col and ROW=sequenza_row) then
			next_state<=S2att_giusto;
		elsif (COL="000" and ROW="0000") then
			next_state<=S0;
		else
			next_state<=S2att_sbagliato;
		end if;
		if (COL="000" and ROW="0000") then
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
                Tastierino_Out<='1';
        else
                Tastierino_Out<='0';
        end if;
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S0att_sbagliato=>
        if (COL="000" and ROW="0000") then
                next_state<=S1_sbagliato;
        else
                next_state<=S0att_sbagliato;
        end if;
                Tastierino_Out<='0';
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S1_sbagliato =>
        if (COL="000" and ROW="0000") then
                next_state<=S1_sbagliato;
        else
                next_state<=S1att_sbagliato;
        end if;
                Tastierino_Out<='0';
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S1att_sbagliato =>
        if (COL="000" and ROW="0000") then
                next_state<=S2_sbagliato;
        else
                next_state<=S1att_sbagliato;
        end if;
                Tastierino_Out<='0';
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S2_sbagliato =>
        if (COL="000" and ROW="0000") then
                next_state<=S2_sbagliato;
        else
                next_state<=S2att_sbagliato;
        end if;
                Tastierino_Out<='0';
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when S2att_sbagliato =>
        if (COL="000" and ROW="0000") then
                next_state<=S0;
        else
                next_state<=S2att_sbagliato;
        end if;
                Tastierino_Out<='0';
                state_debug<=conv_std_logic_vector(stato'POS(current_state),4);
when others =>
        next_state<=S0;
        Tastierino_Out<='0';
```

```
end case;
end process;
end Behavioral;
```

- ## Test Bench

Per quanto riguarda le simulazioni, attuate attraverso il simulatore ISIM, si è deciso di utilizzare 2 TestBench per ogni automa implementato:

- TestBench Principale: TestBench dove viene testato il percorso di normale funzionamento dell'automa, dove quindi gli ingressi vengono forniti in maniera corretta e non vi sono situazioni di errore.
- TestBench Secondario: TestBench dove viene testato l'automa nel suo complesso, quindi tutte le possibili combinazioni di ingressi che si possono verificare in ogni singolo stato.

In questi testbench ci siamo poi serviti di eventuali segnali e variabili d'appoggio che sono serviti semplicemente per effettuare cicli (**i**), per assegnare determinate combinazioni agli ingressi (**Sequenza,Sequenza_Corretta**) e per conteggio (**count,number trans**). **Count** serve per contare le pressioni del tasto mentre **number trans** serve per contare il numero di combinazioni testate.

- ## Test Bench Principale Automa Cancello

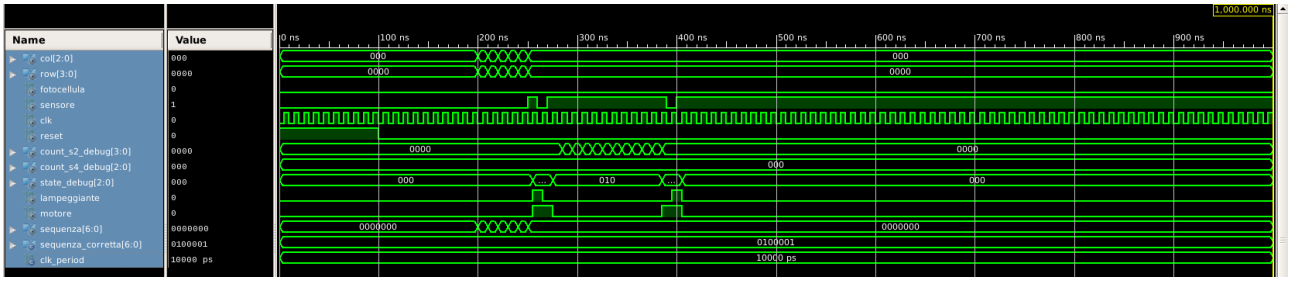
Il TestBench Principale dell'automa Cancello consiste, come detto in precedenza, nel percorrere il percorso di normale funzionamento dell'automa: partendo dallo stato iniziale **S0** (cancello chiuso), passo in fase di apertura (**S1**). Una volta che il cancello si è aperto (**S2**), attendo per 10 cicli di clock. Poi passo in fase di chiusura (**S3**) e infine ritorno ad **S0** (cancello chiuso). Nel codice del TestBench vi è inoltre un processo di debug, sensibile al **RESET** ed alle uscite di debug (**state debug, count S2 debug e count S4 debug**), che effettua delle stampe di alcuni messaggi, per una maggiore comprensione del TestBench. Abbiamo inoltre utilizzato una funzione (**function std_logic_vector to string**) che converte un vettore di tipo std_logic in una stringa pronta per essere letta nel processo di debug. La stampa dei valori del test bench avviene attraverso questo processo di debug che sostanzialmente è sensibile al **reset, state debug , count S2 debug, count S4 debug**. All'interno di questo processo se il **reset** è alto (1) viene stampato fase di reset altrimenti a seconda dello Stato in cui mi trovo che è indicato da **state debug,** e dal valore degli ingressi (**sensore,fotocellula**) e della variabile **count** vi sono diversi messaggi di report che vengono stampati. In particolare negli stati S2 ed S4 viene stampato anche il valore del contatore in binario.

Domenico
Gagliardo
0612704485
Gruppo N°32

CODICE PROCESS DEBUG TB PRINCIPALE

```
Debug: process(RESET,state_debug,count_S2_debug,count_S4_debug)
begin
if (RESET='1') then
report "fase di reset" severity note;
elsif (state_debug="000" and sensore='1') then
report "Vado in S0" severity note;
elsif (state_debug="000" and ((count=0) or (count=3 and Sensore='0'))) then
report "Stato S0" severity note;
elsif ((state_debug="001" and sensore='1' and count=3) or (state_debug="001" and fotocellula='1' and
sensore='0' and count_S4_debug="000")) then
report "Vado in S1" severity note;
elsif (state_debug="001" and sensore='0') then
report "Stato S1" severity note;
elsif (state_debug="010" and sensore='1' and count_S2_debug="0000") then
report "Vado in S2" severity note;
elsif (state_debug="010" and sensore='0' and count_S2_debug="0000") then
report "Stato S2 : Sensore fine corsa non attivo (combinazione impossibile)" severity note;
elsif (state_debug="010") then
report "Stato S2 : Il Conteggio dei colpi di clock  ad : " &to_string(count_S2_debug) severity note;
elsif ((state_debug="011" and count_S2_debug="0000") or (state_debug="011" and fotocellula='0' and
sensore='0' and count_S4_debug="000")) then
report "Vado in S3" severity note;
elsif (state_debug="011" and sensore='0' and fotocellula='0') then
report "Stato S3" severity note;
elsif (state_debug="100" and fotocellula='1' and sensore='0' and count_S4_debug="000") then
report "Vado in S4" severity note;
elsif (state_debug="100" and sensore='1' and count_S4_debug="000") then
report "Stato S4 : Sensore fine corsa attivo (combinazione impossibile)" severity note;
elsif (state_debug="100") then
report "Stato S4 : Il Conteggio dei colpi di clock  ad : " &to_string(count_S4_debug) severity note;
else
report "Transizione Scorretta" severity error;
end if;
end process;
```

FORME D'ONDA (OTTENUTE DAL SIMULATORE ISIM)

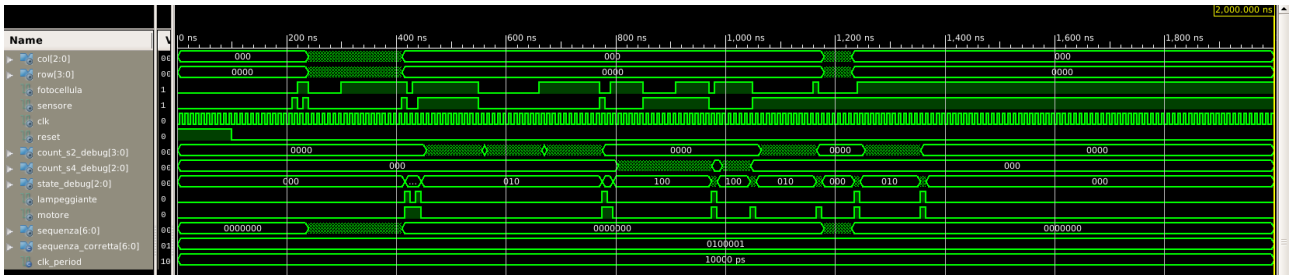


Domenico
Gagliardo
0612704485
Gruppo N°32

- ## TestBench Secondario Automa Cancello

Il TestBench Secondario invece va ad analizzare tutte le possibili combinazioni degli ingressi per ogni stato del Cancello. Qui oltre ad utilizzare la funzione (**function std_logic_vector to string**) che converte un vettore di tipo std_logic in una stringa pronta per essere letta nel processo di debug, abbiamo utlizzato anche un'altra funzione (**function to std logic**) che converte un intero in un std logic. In questo testbench, ad esclusione dello stato **S0**, le combinazioni testate vedono sempre il segnale di uscita del tastierino come un don't care, perché sostanzialmente per come abbiamo strutturato l'automa cancello, l'uscita del tastierino sarà sempre 0 tranne che nello stato iniziale **S0**. Questo perché forziamo il reset del tasterino in tutti gli stati tranne che in **S0**. La stampa dei valori del test bench avviene attraverso lo stesso processo di debug del testbench principale.

FORME D'ONDA (OTTENUTE DAL SIMULATORE ISIM)



- ## Test Bench Principale Automa Riconoscitore Sequenza

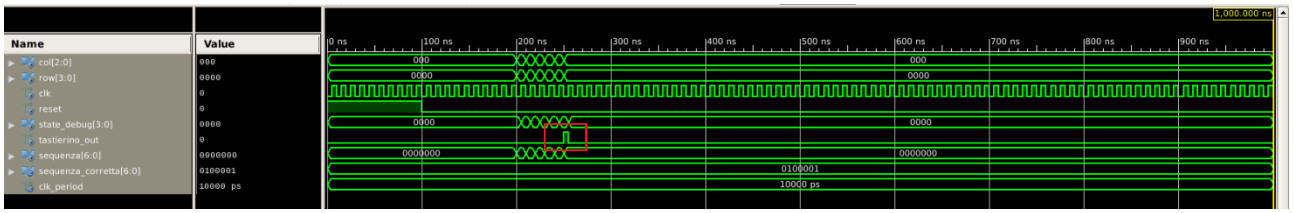
Nel testbench principale dell'automa riconoscitore sequenza abbiamo implementato il percorso di normale funzionamento dell'automa quindi abbiamo premuto per tre volte 0 sul tastierino in conseguenza di ciò l'uscita si è alzata, riconoscendo una combinazione di 3 cifre decimali.

CODICE RICONOSCITORE_SEQUENZA_TB_PRINCIPALE

```
  -- Stimulus process
  stim_proc: process
  begin
    RESET<='1';
    wait for 100 ns;
                    RESET<='0';
    wait for CLK_period*10;
                    Sequenza<=Sequenza_Corretta;
                    wait for CLK_period;
                    Sequenza<="0000000";
                    wait for CLK_period;
                    Sequenza<=Sequenza_Corretta;
                    wait for CLK_period;
                    Sequenza<="0000000";
                    wait for CLK_period;
                    Sequenza<=Sequenza_Corretta;
                    wait for CLK_period;
                    Sequenza<="0000000";
    wait;
  end process;
```

Domenico
Gagliardo
0612704485
Gruppo N°32

FORME D'ONDA (OTTENUTE DAL SIMULATORE ISIM)



- **Test Bench Secondario Automa Riconoscitore Sequenza**

Nel test bench secondario dell'automa riconoscitore sequenza abbiamo invece utilizzato un approccio che ci consentisse di testare tutte le combinazioni di ogni stato nella maniera migliore possibile visto la complessità dell'automa. Questo automa infatti ha 128 combinazioni per ogni stato. Siamo partiti testando il percorso errato vale a dire il percorso che parte da **S0** fino ad arrivare ad **S2AttSbagliato**. Dopodiché ci siamo concentrati ad esaminare ogni singolo stato del percorso corretto transitando quindi nel percorso sbagliato per effettuare il test dello stato stesso. Questo fino ad arrivare ad **S2AttGiusto**, dove, concluso il test anche per quest'ultimo stato, si è transitati in **S0**, riconoscendo così una sequenza di 3 cifre decimali e alzando quindi l'uscita del tastierino. La stampa dei valori del test bench avviene attraverso un altro processo di debug sensibile ai segnali di **Reset,Col,ROW e Tastierino_Out**. In questo processo dichiariamo due costanti che servono a confrontare il valore di **COL e ROW** con la sequenza corretta. In questo processo sei il **reset** è alto (1) viene stampato fase di reset. Altrimenti a seconda dei valori di **state debug**, **Tastierino_Out**, **COL**, **ROW**, **count e number trans** vengono stampati altri messaggi (in un messaggio in particolare viene stampata anche la combinazione corrente appena testata, corrispondente al valore corrente di **number trans**).

CODICE RICONOSCITORE_SEQUENZA_TB_SECONDARIO

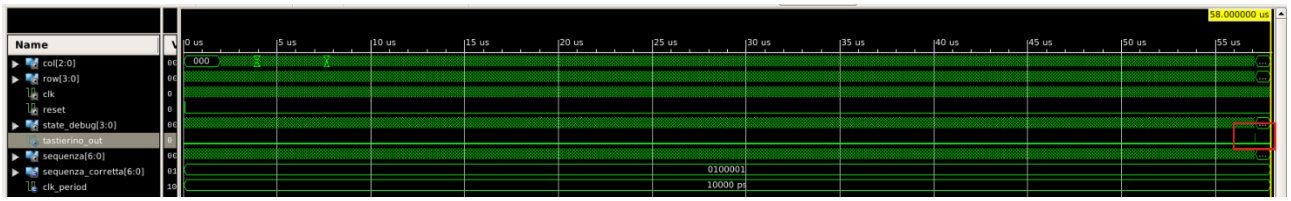
```
Debug: process (RESET,COL,ROW,Tastierino_Out)
constant Sequenza_Col : STD_LOGIC_VECTOR (2 downto 0) := "010";
constant Sequenza_Row : STD_LOGIC_VECTOR (3 downto 0) :="0001";

begin
if (RESET='1')  then report "Fase di Reset" severity note;
else
        if (state_debug="1001" and count=0 and Tastierino_Out='1' and COL="000" and ROW="0000" and
number_trans=128) then
        report "Ho Riconosciuto La Sequenza Corretta" severity note;
        elsif (state_debug="0000" and count=0 and COL="000" and ROW="0000" and (number_trans=128
or number_trans=0)) then
        report "Permango in S0" severity note;
        elsif (state_debug="0001" and count=0 and COL="000" and ROW="0000" and number_trans=128)
then
        report "Passo da S0Att_Giusto ad S1_Giusto e Permango in S1_Giusto" severity note;
        elsif (state_debug="0101" and count=0 and COL="000" and ROW="0000" and number_trans=128)
then
        report "Passo da S1Att_Giusto ad S2_Giusto e Permango in S2_Giusto" severity note;
        elsif (state_debug="0000" and count=0 and COL=Sequenza_Col and ROW=Sequenza_Row and
number_trans=128) then
        report "Passo Da S0 ad S0att_giusto e Permango in S0att_giusto" severity note;
```

Domenico
Gagliardo
0612704485
Gruppo N°32

```
        elsif (state_debug="0011" and count=0 and COL=Sequenza_Col and ROW=Sequenza_Row and
number_trans=128) then
        report "Passo da S1_Giusto ad S1att_giusto e Permango in S1_att_giusto" severity note;
        elsif (state_debug="0111" and count=0 and COL=Sequenza_Col and ROW=Sequenza_Row and
number_trans=128) then
        report "Passo da S2_giusto ad S2_att_giusto e Permango in S2_att_giusto" severity note;
        elsif (count/=3 and (COL/=Sequenza_Col or ROW/=Sequenza_Row)) then
        report "Testo il percorso errato" severity note;
        elsif (count/=3 and COL=Sequenza_Col and ROW=Sequenza_Row) then
        report "Testo il percorso errato (Combinazione 33)" severity note;
        elsif (count=3 and COL="000" and ROW="0000") then
        report "Combinazione testata, transizione numero " & integer'image(number_trans) severity note;
        else report "C' una transizione Scorretta" severity warning;
        end if;
end if;
end process Debug;
```

FORME D'ONDA (OTTENUTE DAL SIMULATORE ISIM)



- ## DESIGN ALTERNATIVO

Una possibile strategia alternativa alla risoluzione dell'elaborato sarebbe stata quella di sintetizzare le uscite e le funzioni di stato prossimo (quindi le transizioni tra gli stati) attraverso le K-Mappe (mappe di Karnaugh) e disegnare il circuito con porte logiche opportune e flip-flop opportuni, avendo così un approccio di tipo RTL (Register Transfer Level).