Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell’Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Immagine che contiene cerchio, simbolo, schizzo

Descrizione generata automaticamente

Elaborato di Architettura dei Sistemi Digitali

*Prof.ssa Alessandra De Benedictis*

a.a. 2023-24

Studenti:

<nome cognome matricola >

Sommario

[Capitolo 1: reti combinatorie elementari 4](#_Toc158999250)

[Esercizio 1.1 4](#_Toc158999251)

[Progetto e architettura 4](#_Toc158999252)

[Implementazione 4](#_Toc158999253)

[Simulazione 7](#_Toc158999254)

[Sintesi su board di sviluppo 7](#_Toc158999255)

[Timing analysis 7](#_Toc158999256)

[Appendice 8](#_Toc158999257)

[Componente 1: multiplexer 4:1 8](#_Toc158999258)

[Progetto e architettura 8](#_Toc158999259)

[Implementazione 8](#_Toc158999260)

[Componente 2: multiplexer 2:1 9](#_Toc158999261)

[Progetto e architettura 9](#_Toc158999262)

[Implementazione 9](#_Toc158999263)

[Esercizio 1.2 10](#_Toc158999264)

[Progetto e architettura 10](#_Toc158999265)

[Implementazione 10](#_Toc158999266)

[Simulazione 12](#_Toc158999267)

[Appendice 12](#_Toc158999268)

[Componente 1: demultiplexer 1:4 12](#_Toc158999269)

[Progetto e architettura 12](#_Toc158999270)

[Implementazione 13](#_Toc158999271)

[Componente 2: demultiplexer 1:2 14](#_Toc158999272)

[Progetto e architettura 14](#_Toc158999273)

[Implementazione 14](#_Toc158999274)

[Esercizio 1.3 14](#_Toc158999275)

[Esercizio 2.1 14](#_Toc158999276)

[Progetto e architettura 15](#_Toc158999277)

[Implementazione 15](#_Toc158999278)

[Simulazione 16](#_Toc158999279)

[Appendice 17](#_Toc158999280)

[Componente 1: ROM combinatoria 17](#_Toc158999281)

[Progetto e architettura 17](#_Toc158999282)

[Implementazione 17](#_Toc158999283)

[Componente 2: Macchina M 17](#_Toc158999284)

[Progetto e architettura 17](#_Toc158999285)

[Implementazione 18](#_Toc158999286)

[Capitolo 2: reti sequenziali elementari 18](#_Toc158999287)

[Esercizio 3.1 18](#_Toc158999288)

[Progetto e architettura 18](#_Toc158999289)

[Implementazione 19](#_Toc158999290)

[Simulazione 21](#_Toc158999291)

[Esercizio 3.2 22](#_Toc158999292)

[Esercizio 4.1 23](#_Toc158999293)

[Progetto e architettura 23](#_Toc158999294)

[Implementazione 24](#_Toc158999295)

[Simulazione 27](#_Toc158999296)

[Appendice 27](#_Toc158999297)

[Componente 1 : FF\_D 27](#_Toc158999298)

[Progetto e architettura 27](#_Toc158999299)

[Implementazione 27](#_Toc158999300)

[Componente 2 : SR\_Component 28](#_Toc158999301)

[Progetto e architettura 28](#_Toc158999302)

[Implementazione 28](#_Toc158999303)

[Esercizio 5.1 29](#_Toc158999304)

[Progetto e architettura 29](#_Toc158999305)

[Implementazione 29](#_Toc158999306)

[Simulazione 29](#_Toc158999307)

[Appendice 29](#_Toc158999308)

[Componente 1 : Counter 29](#_Toc158999309)

[Progetto e architettura 29](#_Toc158999310)

[Implementazione 29](#_Toc158999311)

# Capitolo 1: reti combinatorie elementari

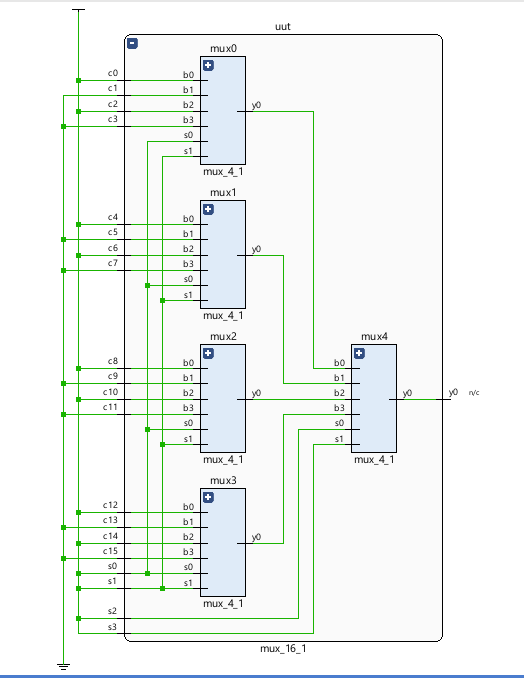
## Esercizio 1.1

Progettare, implementare in VHDL e testare mediante simulazione un **multiplexer indirizzabile 16:1**, utilizzando un approccio di progettazione per composizione a partire da **multiplexer 4:1.**

#### Progetto e architettura

<descrizione dell’approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

Abbiamo seguito un approccio strutturale, basandoci sull’utilizzo di 5 multiplexer 4:1, i primi 4 prendono gli ingressi del sistema, il quinto multiplexer prende le uscite dei primi 4 multiplexer e ci fornisce l’uscita finale del sistema. I primi 4 multiplexer prendono in ingresso i bit di selezione meno significativi, mentre l’ultimo multiplexer prende in ingresso i 2 bit di selezione più significativi. Ognuno dei multiplexer al primo livello prende in ingresso 4 bit e seleziona uno di questi in base al valore dei bit di selezione, questi multiplexer ci forniranno delle uscite intermedie che vanno in ingresso al multiplexer all’ultimo livello, che in base ai bit di selezione ci fornirà l’uscita.

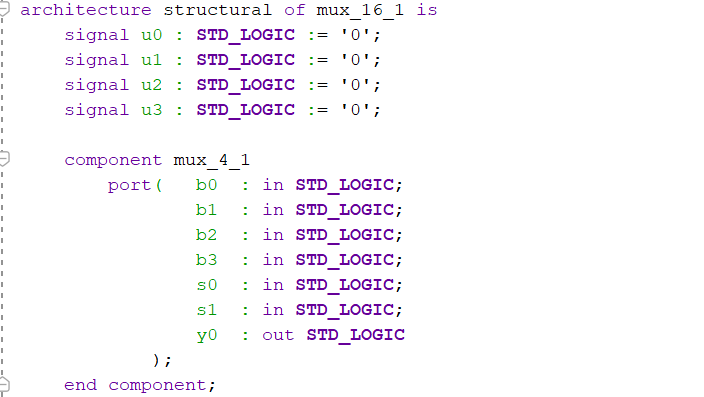


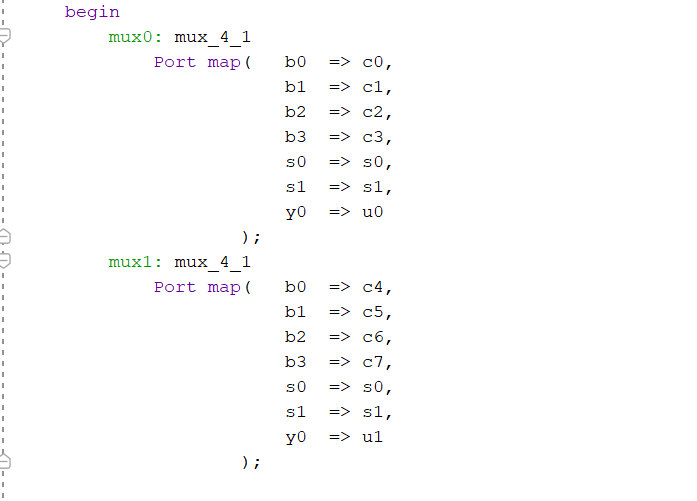
#### Implementazione

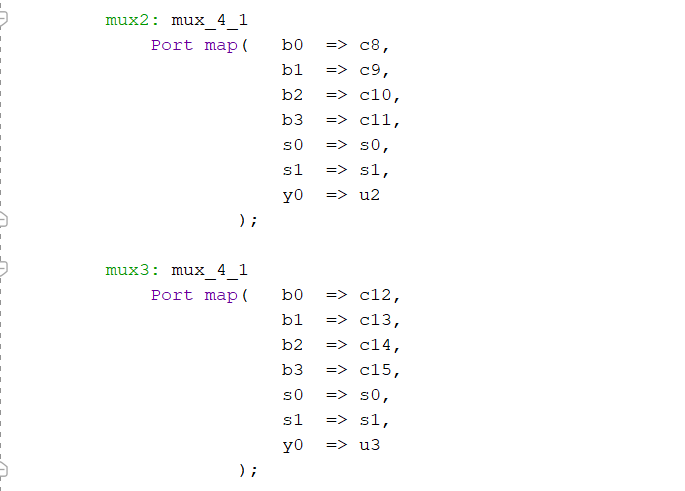
<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un’appendice>

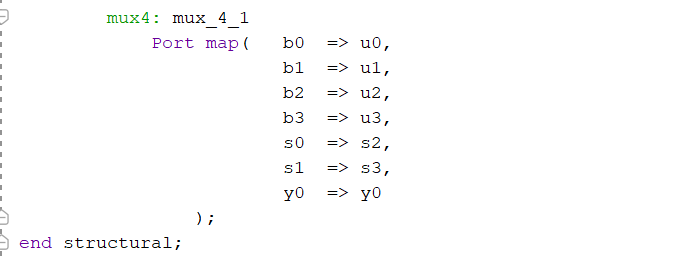
Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente









#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

Abbiamo dato in input una stringa di bit con 0 e 1 alternati, tramite i bit di selezione abbiamo abilitato tutte le varie uscite. Infatti dalla simulazione otteniamo che l’uscita finale è composta da un onda quadra, ovvero un alternarsi di 0 e 1.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene schermata

Descrizione generata automaticamente

#### Sintesi su board di sviluppo

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

#### Timing analysis

<se richiesto: discussione dei risultati della timing analysis sui circuiti realizzati>

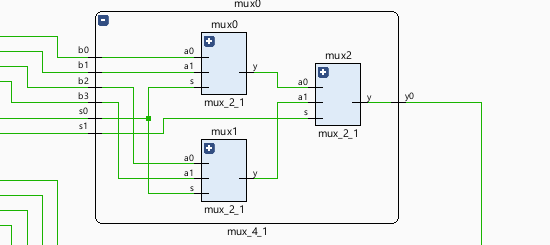
## Appendice

### Componente 1: multiplexer 4:1

### Progetto e architettura

<descrizione dell’approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

Abbiamo utilizzato un approccio strutturale, con 3 multiplexer 2:1, i primi due prendono gli ingressi che attraverso il bit di selezione ci fornirà un’uscita intermedia. Il terzo multiplexer 2:1 prende le uscite di questi 2 e in base al bit di selezione ci fornisce l’uscita finale.



### Implementazione

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un’appendice>

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, numero

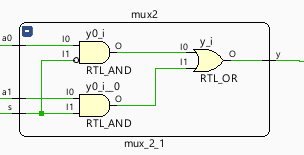
Descrizione generata automaticamente

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

### Componente 2: multiplexer 2:1

### Progetto e architettura

Abbiamo utilizzato un approccio seguendo l’espressione logica del multiplexer.

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## Esercizio 1.2

Utilizzando il componente sviluppato al punto precedente, progettare, implementare in VHDL e testare mediante simulazione una **rete di interconnessione a 16 sorgenti e 4 destinazioni**.

#### Progetto e architettura

Abbiamo utilizzato il multiplexer 16:1 implementato nell’esercizio precedente, e abbiamo aggiunto un demultiplexer 1:4 seguendo un approccio strutturale. Il multiplexer 16:1 ha la stessa funzionalità dell’esercizio precedente. Il Demux è formato da 3 Demux 1:2, il primo prende in ingresso l’uscita del multiplexer 16:1 e un bit di selezione, questo ci fornirà due uscite. Le due uscite andranno in ingresso agli altri due Demux 1:2, con un ulteriore bit di selezione. I due Demux 1:2 quindi ci forniranno l’uscita totale del sistema.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

#### Implementazione

Di seguito l’implementazione del nostro sistema.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

#### Simulazione

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene schermata

Descrizione generata automaticamente

## Appendice

### Componente 1: demultiplexer 1:4

### Progetto e architettura

Abbiamo utilizzato un approccio strutturale, con 3 demultiplexer 1:2, il primo prende in ingresso l’uscita del mux 16:1 e un bit di selezione. Questo ci fornirà due uscite, le quali andranno in ingresso ai due demux 1:2 dell’ultimo livello che ci forniranno l’uscita totale del sistema.

Immagine che contiene testo, diagramma, Piano, numero

Descrizione generata automaticamente

### Implementazione

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

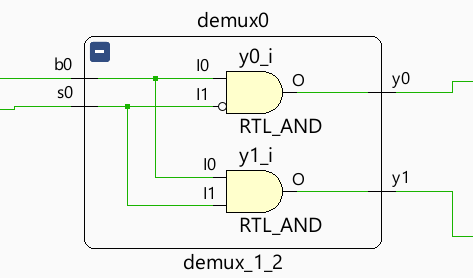
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

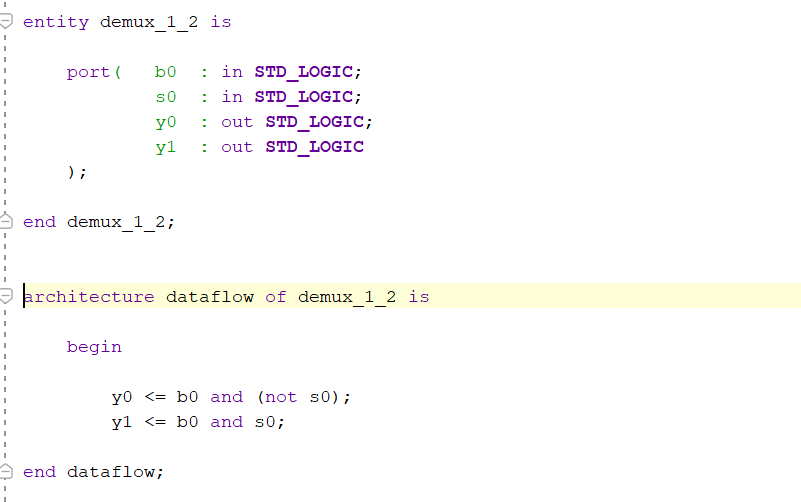
### Componente 2: demultiplexer 1:2

### Progetto e architettura

Abbiamo utilizzato un approccio seguendo l’espressione logica del demultiplexer.



### Implementazione



## Esercizio 1.3

Sintetizzare ed implementare su board il progetto della rete di interconnessione sviluppato al punto 1.2, utilizzando gli switch per fornire gli input di selezione e i led per visualizzare i 4 bit di uscita. Per quanto riguarda i 16 bit dato in input, essi devono essere immessi mediante switch, 8 bit alla volta, sviluppando un’apposita “rete di controllo” per l’acquisizione che utilizzi due bottoni della board per caricare rispettivamente la prima e la seconda metà del dato in ingresso.

#### Progetto e architettura

Abbiamo progettato questo esercizio partendo dai componenti dell’esercizio precedente, in aggiunta abbiamo creato un componente denominato Control e abbiamo unito tutto nel sistema rete\_16\_4\_board. Abbiamo quindi utilizzato un approccio strutturale, il sistema prevede due segnali di abilitazione denominati load\_data1 e load\_data2, collegati rispettivamente al bottone P17 e M1. Abbiamo un vettore denominato switches che si occupa di ricevere l’input 8 bit alla volta, questi collegati agli switch J15 fino a R13. Poi utilizziamo un ulteriore vettore denominato sel\_switches che ci permette di ricevere in input e quindi di scegliere quale selezione utilizzare, in particolare sel\_switches da 0 a 3 vengono utilizzati per il mux\_16\_1 e sel\_switches da 4 a 5 vengono utilizzati come selezione del demux, questi sono collegati agli switch R16 fino a V10. Infine abbiamo l’uscita denominata led che è un vettore di 4 bit collegato ai led H17 fino ad N14.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

#### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

#### Sintesi su board di sviluppo

Immagine che contiene testo, schermata, numero, software

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

## Appendice

#### Componente 1: Control

#### Progetto e architettura

Per questo componente abbiamo utilizzato un approccio comportamentale, esso prevede un segnale denominato enable\_data1 che abilita la ricezione dei primi 8 bit di input, un segnale denominato enable\_data2 che abilita la ricezione dei secondi 8 bit di input, un vettore denominato input\_data che identifica gli 8 bit di input, un vettore input\_sel che identifica le selezioni da utilizzare nella rete\_16\_4, un vettore di value\_data sul quale salviamo i 16 bit di input totale, e un vettore di value\_sel dove salviamo i 6 bit di selezione della rete. In particolare in questo componente implementiamo la logica con la quale inseriamo l’input, quando abbiamo enable\_data1 pari ad 1 abilitiamo a ricevere i primi 8 bit dell’input salvandoli sulle prime 8 locazioni di un vettore di appoggio, in maniera identica quando enable\_data2 è pari ad 1 abilitiamo a ricevere i secondi 8 bit dell’input salvandoli sulle seconde 8 locazioni della stessa variabile d’appoggio, infine questa variabile viene copiata nel vettore value\_data per darla alla rete.

Immagine che contiene testo, diagramma, linea, schermata

Descrizione generata automaticamente

#### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## Esercizio 2.1

Progettare, implementare in VHDL e testare mediante simulazione un **sistema S** composto da una **ROM** puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria **M** che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto nella ROM all’indirizzo A opportunamente “trasformato” attraverso la macchina M. Il comportamento della macchina M è totalmente a scelta dello studente, l’unico vincolo è che essa prenda in ingresso 8 bit e ne fornisca in uscita 4.

#### Progetto e architettura

Abbiamo utilizzato un approccio strutturale, includendo una ROM ed una Macchina denominata M. Il funzionamento del sistema S prevede in ingresso un address di 4 bit, e attraverso questo, prendiamo la locazione di memoria dalla ROM, che attraverso la Macchina andremo ad elaborare secondo una logica scelta da noi .

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

#### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene schermata, testo, diagramma, linea

Descrizione generata automaticamente

#### Simulazione

Abbiamo simulato un testbench che fornisce in input i vari indirizzi, e in base al valore delle locazioni, seguendo la logica implementata nella macchina M, ci fornisce l’output finale. Troviamo corretta la simulazione in base ai valori delle locazioni di memoria e alla logica della macchina M, che mostreremo in seguito nell’appendice.

Immagine che contiene testo, schermata, documento, numero

Descrizione generata automaticamente

Immagine che contiene schermata, testo, Carattere, linea

Descrizione generata automaticamente

## Appendice

### Componente 1: ROM combinatoria

### Progetto e architettura

La ROM implementata ha 16 locazioni da 8 bit, e dato un address di 4 bit in ingresso ci fornisce in output il contenuto di una delle 16 locazioni.

### Implementazione

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

### Componente 2: Macchina M

### Progetto e architettura

La macchina M è un’entità che prende in ingresso una stringa di 8 bit e ci restituisce in uscita una stringa di 4 bit. La stringa di uscita viene elaborata concatenando i bit in posizione 1, 3, 5 e 7 della stringa in ingresso.

### Implementazione



## Esercizio 2.2

Sintetizzare ed implementare su board il progetto del sistema ROM+M sviluppato al punto 2.1, utilizzando gli switch per fornire l’indirizzo della ROM da cui leggere i valori da trasformare e i led per visualizzare i 4 bit di uscita.

### Progetto e architettura

Per implementare questo esercizio abbiamo modificato l’esercizio precedente, modificando semplicemente il nome della variabile di ingresso al sistema con switches e quello della uscita con leds, entrambi sono sempre dei vettori di 4 bit. In particolare switches li abbiamo affidati agli switch da J15 fino a R15, leds da led H17 fino a led N14.

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente

#### Sintesi su board di sviluppo

Immagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente

# Capitolo 2: reti sequenziali elementari

## Esercizio 3.1

Progettare, implementare in VHDL e testare mediante simulazione una macchina in grado di riconoscere la sequenza 101. La macchina prende in ingresso un segnale binario i che rappresenta il dato, un segnale A di tempificazione e un segnale M di modo, che ne disciplina il funzionamento, e fornisce un’uscita Y alta quando la sequenza viene riconosciuta. In particolare,

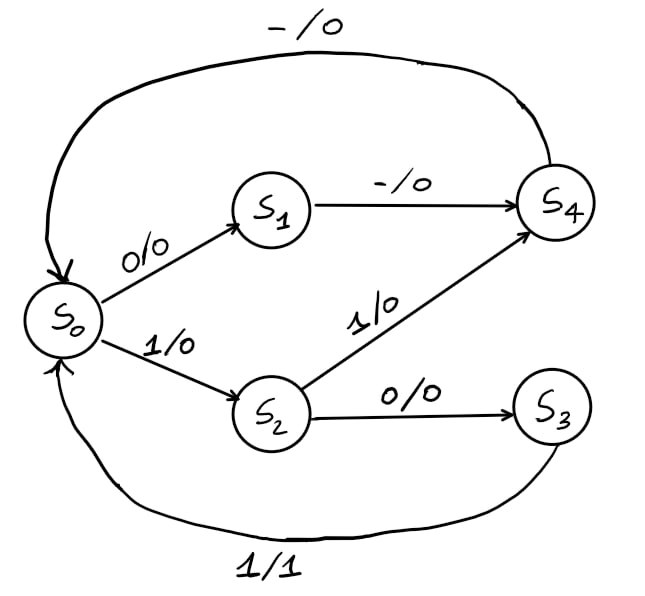
- se M=0, la macchina valuta i bit seriali in ingresso a gruppi di 3 (sequenze non sovrapposte),

- se M=1, la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

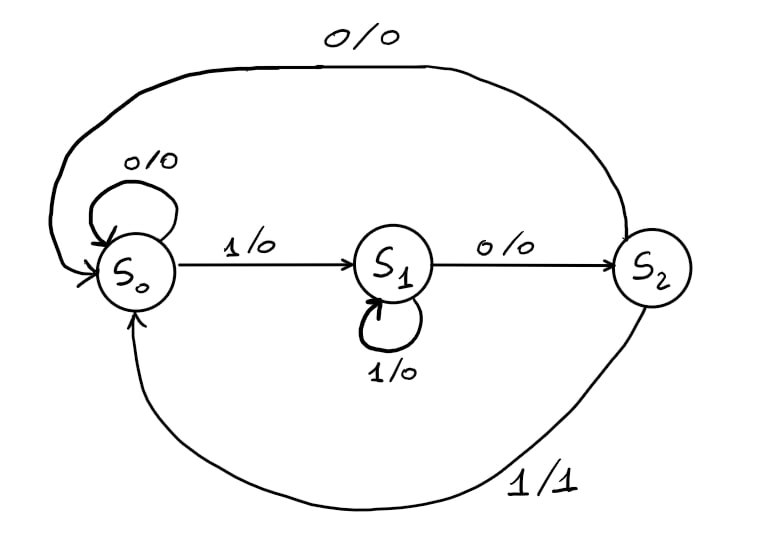
### Progetto e architettura

Abbiamo utilizzato un approccio behavioral con 2 process. All’ interno del primo process abbiamo implementato la funzionalità del riconoscitore, separando con due costrutti if le due modalità, ovvero quando M = 0 abbiamo implementato la modalità senza sovrapposizione degli eventi, quando invece M = 1 abbiamo implementato la modalità con sovrapposizione parziale. All’interno del secondo process, invece sono rappresentati gli elementi di memoria e gli eventi per i quali ripristinare la macchina.

Automa a stati finiti del riconoscitore senza sovrapposizione :



Automa a stati finiti del riconoscitore con sovrapposizione parziale:



### Implementazione

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente



Immagine che contiene testo, schermata, algebra

Descrizione generata automaticamente

#### Simulazione

Abbiamo simulato un testbench dove il riconoscitore ha M = 0 inizialmente, dunque è in modalità senza sovrapposizione, e dopo la terza variazione dell’ingresso i notiamo che l’uscita si alza. In contemporanea alla quarta variazione diamo anche il cambiamento di M = 1, infatti notiamo dalla simulazione che alla settima variazione abbiamo di nuovo la Y che si alza perché riconosce una sequenza 101 essendo in modalità sovrapposizione parziale, come era previsto dalla progettazione e dalla implementazione.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Inoltre abbiamo effettuato un'ulteriore simulazione, senza inserire la variazione della M, dunuqe lasciando il riconoscitore senza sovrapposizione, e notiamo che in effetti non viene riconosciuta la seconda sequenza riconosciuta prima.

Immagine che contiene schermata, linea, testo, software

Descrizione generata automaticamente

## Esercizio 3.2

Sintetizzare e implementare su board la rete sviluppata al punto precedente, utilizzando uno switch S1 per codificare l’input i e uno switch S2 per codificare il modo M, in combinazione con due bottoni B1 e B2 utilizzati rispettivamente per acquisire l’input da S1 e S2 in sincronismo con il segnale di tempificazione A, che deve essere ottenuto a partire dal clock della board. Infine, l’uscita Y può essere codificata utilizzando un led.

### Progetto e architettura

Per progettare questo esercizio abbiamo utilizzato il componente ButtonDebouncer per gestire la logica dei bottoni, il Riconoscitore sviluppato al punto precedente leggermente modificato, abbiamo utilizzato degli enable per abilitare la ricezione del riconoscitore, e come al punto precedente quando il segnale di modo cambia tra due colpi di clock riportiamo l’automa allo stato s0. In particolare questi segnali sono rappresentati nel nostro sistema da B1 ovvero il bottone utilizzato per abilitare la ricezione degli input, associato al bottone M17 e B2 utilizzato per abilitare la ricezione dei segnali di modo, associato al bottone P17. Il segnale di modo è denominato S2 ed associato allo switch V10, il segnale di input è denominato S1 ed associato allo switch J15. Per quanto riguarda il clock abbiamo utilizzato quello della board. Il reset è abbinato al bottone M18. Infine abbiamo l’uscita led associata al led H17, che si accende quando la sequenza è riconosciuta.

Immagine che contiene testo, diagramma, linea, schermata

Descrizione generata automaticamente

### Implementazione

Immagine che contiene testo, schermata, Carattere, viola

Descrizione generata automaticamente Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Sintesi su board e sviluppo

Per poter effettuare la sintesi su board era necessario utilizzare il componente BottonDebouncer, perché ci serviva ripulire il segnale proveniente dal bottone per far sì che durasse quanto un colpo di clock.

Immagine che contiene testo, schermata, Carattere, lettera

Descrizione generata automaticamente

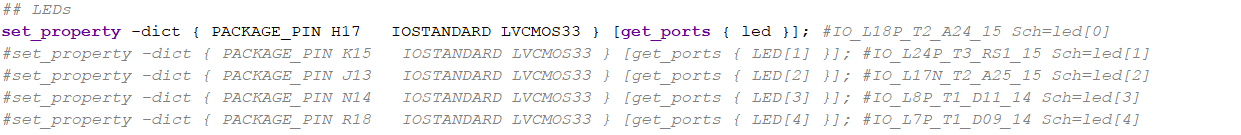


Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

## Appendice

### Componente 1: BottonDebouncer

### Progetto e architettura

Questo componente è un automa a stati finiti che rappresenta lo stato in cui si trova un bottone, dando un bit di uscita CLEARED\_BTN. In particolare ad ogni fronte di salita se il reset è uguale ad 1 lo stato è NOT\_PRESSED, altrimenti, se il bottone viene premuto lo stato diventa CHK\_PRESSED, in questo stato controlla se la variabile count è uguale al max\_count – 1, se lo è ed il bottone risulta uguale ad 1 lo stato diventa PRESSED e il bit di uscita CLEARED\_BTN è uguale ad 1, altrimenti lo stato diventa NOT\_PRESSED, il conteggio viene azzerato in entrambi i casi. Nel caso in cui count non è uguale a max\_count – 1, allora il count viene incrementato di 1 e lo stato rimane in CHK\_PRESSED. La variabile max\_count è data dal rapporto della durata stimata dell’oscillazione del bottone sul periodo del clock. Ritornando all’automa, quando siamo nello stato PRESSED se il bottone è uguale a 0 lo stato diventa CHK\_NOT\_PRESSED, altrimenti rimane PRESSED. Nello stato CHK\_NOT\_PRESSED, si ripete il controllo tra la variabile count e max\_count – 1, se questa è vera ed il bottone è pari a 0, lo stato diventa NOT\_PRESSED, se il bottone è alto allora lo stato è PRESSED. Se la condizione count uguale max\_count-1 allora si incrementa il conteggio è lo stato rimane CHK\_NOT\_PRESSED. In tutti gli altri casi lo stato del bottone è NOT\_PRESSED.

### Implementazione

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

## Esercizio 4.1

Progettare, implementare in VHDL e testare mediante simulazione un registro a scorrimento di N bit in grado di shiftare a destra o a sinistra di un numero Y variabile di posizioni a seconda di una opportuna selezione. In particolare, i valori possibili di Y sono 1 e 2. L’utente tramite selezione deve scegliere di quante posizioni shiftare. Il componente deve essere realizzato utilizzando sia un a) approccio comportamentale sia un b) approccio strutturale.

Nota: il numero di bit del registro deve essere implementato come un generic, e dall’esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

### Progetto e architettura

Per lo shift register con approccio comportamentale, abbiamo pensato di utilizzare 2 ingressi seriali uno da destra e uno da sinistra (EI, EO), per l’uscita abbiamo usato un solo segnale SO che si aggiorna in base alla modalità di shifting. Per gestire la modalità di shift abbiamo utilizzato un ingresso di selezione (s) che quando è posto uguale a 0 abbiamo shift verso sinistra, mentre quando s è uguale a 1 abbiamo lo shift verso destra. Abbiamo utilizzato un ingresso Y che quando è 0 sposta i bit di registro in registro, mentre quando Y è posto a 1 abbiamo lo shift di due registri.

Immagine che contiene diagramma, linea, Piano, Disegno tecnico

Descrizione generata automaticamente

Per lo shift register con approccio strutturale, abbiamo pensato di creare un componente che gestisca le varie modalità di funzionamento. Ogni componente shifta verso destra o sinistra di 1 o 2 posizioni in base ai segnali di selezione s ed Y. In particolare quando s è uguale a 0 abbiamo lo shift verso destra, quando s è uguale a 1 abbiamo lo shift verso sinistra, mentre per la Y quando essa è uguale a 0 abbiamo lo shift di una posizione, quando è uguale a 1 abbiamo lo shift di due posizioni. Per gli ingressi dello shift register abbiamo utilizzato EI per l’ingresso da sinistra ed EO per l’ingresso da destra. Il segnale di uscita è SO, questo si aggiorna in base a se shiftiamo da destra o da sinistra con il valore del registro opportuno.

Immagine che contiene diagramma, Piano, Disegno tecnico, schematico

Descrizione generata automaticamente

### Implementazione

La nostra implementazione comportamentale prevede in base alla modalità di shifting ( sinistra/destra, 1 o 2 posizioni) di aggiornare l’uscita al colpo di clock successivo a quello che popola l’ultimo registro, per fare in modo che sia fedele all’implementazione strutturale.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

La nostra implementazione strutturale prevede l’utilizzo di n componenti (SR\_Component) , ognuno di questi componenti è formato da 3 MUX\_2:1 e un Flip-Flop di tipo D. Per implementare lo shift register abbiamo generato n componenti assegnando gli ingressi e le uscite in base alla logica interna dell’SR\_Component.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

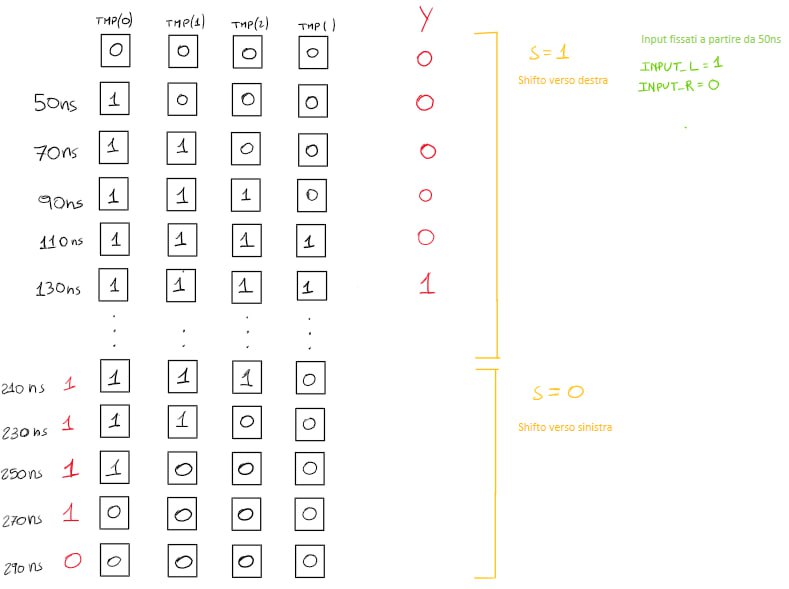
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Simulazione

Immagine che contiene schermata

Descrizione generata automaticamente



Abbiamo svolto un ulteriore simulazione, dove andiamo ad abilitare lo shifting di due bit alla volta. Inoltre abbiamo alzato anche l’input da destra per notare se le tempistiche di propagazione erano corrette, in entrambe le direzioni. Infatti, abbiamo ottenuto i seguenti risultati:

- Inizio ricezione e shift a destra = 50ns , notiamo l’output a 90ns.

- Inizio ricezione e shift a sinistra = 190ns, notiamo che l’output si abbassa per 40ns perché riporta i registri che non erano riempiti, e si alza nuovamente a 230ns.

Immagine che contiene schermata

Descrizione generata automaticamente

## Appendice

### Componente 1 : FF\_D

### Progetto e architettura

Abbiamo utilizzato un Flip-Flop di tipo D. CLK rappresenta il segnale di clock e RST rappresenta il segnale di reset, il segnale d è il dato che va in ingresso al flip flop, q è il segnale di uscita dal registro.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Componente 2 : SR\_Component

### Progetto e architettura

Per implementare questo componente abbiamo utilizzato un approccio strutturale, infatti esso è formato da 3 MUX\_2:1 e un Flip-Flop di tipo D. Esso prevede il segnale di clock (CLK), il segnale di reset (RST), 4 ingressi a1,a2,b1,b2, due ingressi di selezione s e Y e l’uscita O. Il primo MUX prende in ingresso a1, a2 e la selezione s, s in questa logica ci selezione il segnale per effettuare uno shift verso destra o verso sinistra, l’uscita sarà un segnale di output. Il secondo MUX funziona esattamente come il primo con la differenza che gli ingressi sono b1 e b2. Le uscite di questi due MUX vanno in ingresso ad un ulteriore MUX, questo ha come segnale di selezione la Y che, selezione quello corretto in base a se shiftiamo di una o di due posizioni. L’ultimo livello dell’SR\_Component è formato da un Flip-Flop di tipo D, questo prende in ingresso l’uscita del terzo MUX citato.

Immagine che contiene testo, diagramma, Piano, schermata

Descrizione generata automaticamente

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## Esercizio 5.1

Progettare, implementare in VHDL e testare mediante simulazione un cronometro, in grado di scandire secondi, minuti e ore a partire da una base dei tempi prefissata (es. si consideri il clock a disposizione sulla board). Il progetto deve prevedere la possibilità di inizializzare il cronometro con un valore iniziale, sempre espresso in termini di ore, minuti e secondi, mediante un opportuno ingresso di set, e deve prevedere un ingresso di reset per azzerare il tempo.

Il componente deve essere realizzato utilizzando un approccio strutturale, collegando opportunamente dei contatori secondo uno schema a scelta.

### Progetto e architettura

Abbiamo progettato il cronometro con un approccio strutturale partendo da un counter generico comportamentale. Abbiamo quindi utilizzato un counter modulo 60 per i secondi, un ulteriore counter modulo 60 per i minuti e infine un counter modulo 24 per le ore. In particolare il primo counter viene abilitato ad ogni colpo di clock, per simulare i secondi, il counter dei minuti viene abilitato quando il counter dei secondi arriva a 59, il counter delle ore invece viene abilitato quando il primo e il secondo counter arrivano entrambi contemporaneamente a 59. Il cronometro, oltre al reset che ci permette di azzerarlo, presenta un segnale di set che ci permette di impostare un orario di inizio conteggio.

Immagine che contiene testo, diagramma, linea, schermata

Descrizione generata automaticamente

### Implementazione

Per implementare la funzionalità di abilitazione dei counter, abbiamo utilizzato dei segnali di enable, in particolare enable\_s posto uguale al clock, per fare in modo che il counter dei secondi contasse ad ogni colpo di clock, enable\_m che si alza quando il counter\_s arriva a 59 (con la variabile count) e arriva il prossimo colpo di clock. Infine enable\_h viene alzato quando il counter\_s e il counter\_m arrivano a 59 e arriva il prossimo colpo di clock.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

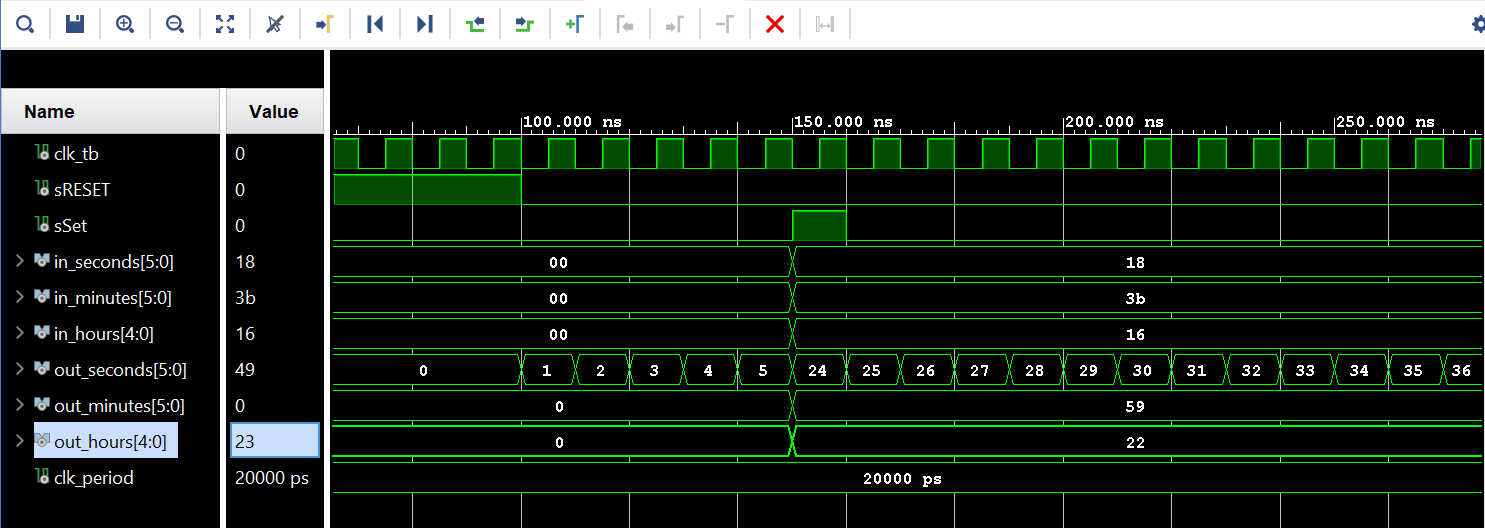
Descrizione generata automaticamente

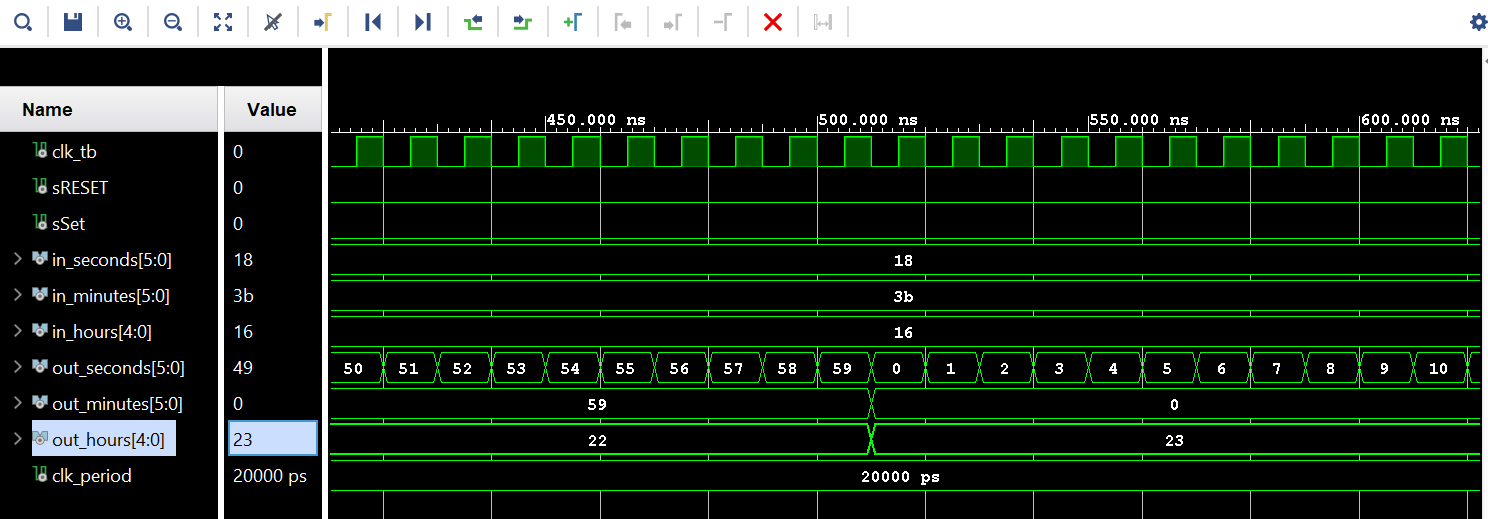
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Simulazione

Abbiamo creato un testbench attraverso il quale settiamo i secondi a 18 , i minuti a 59 e le ore a 22. In questo modo riusciamo a vedere tutte le transizioni di abilitazione dei vari contatori. In particolare vediamo come a 520 ns abbiamo che il counter dei secondi finisce il conteggio di 59, e di conseguenza il counter dei minuti passa da 59 a 00 e il counter delle ore passa da 22 a 23.





## Appendice

### Componente 1 : Counter

### Progetto e architettura

Per progettare il counter abbiamo utilizzato un approccio comportamentale. Abbiamo pensato di progettare un unico counter generico per poi adattarlo ad ore, minuti e secondi. Il counter prevede un segnale di reset che ci permette di azzerarlo, un segnale di enable che ci abilita il conteggio, un segnale di load che ci permette di caricarlo, un vettore data che prendiamo in ingresso per caricarlo, la y che è il vettore di uscita e infine il count un segnale che noi utilizziamo quando il contatore arriva al massimo conteggio per cui è stato progettato.

Immagine che contiene diagramma, linea, testo, Piano

Descrizione generata automaticamente

### Implementazione

Per implementare il counter generico abbiamo utilizzato una libreria che ci permettesse di contare e di fermare il conteggio a nostro piacimento. Abbiamo utilizzato 2 process:

-il primo process per il conteggio, che si aziona ad ogni fronte di discesa del clock, se il reset è alto allora azzeriamo il conteggio, se il load è alto carichiamo il contatore dall’esterno, se abbiamo l’enable alto e il conteggio raggiunge il massimo allora azzeriamo il contatore, altrimenti aggiungiamo 1 al conteggio.

- il secondo process, per l’abilitazione dove, alziamo il segnale di count quando abbiamo raggiunto il massimo conteggio per fare in modo di azionare il contatore successivo. In particolare questo process si azione al fronte di salita, per fare in modo che al fronte di discesa successivo si ha il contatore prossimo già abilitato.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Immagine che contiene testo, schermata

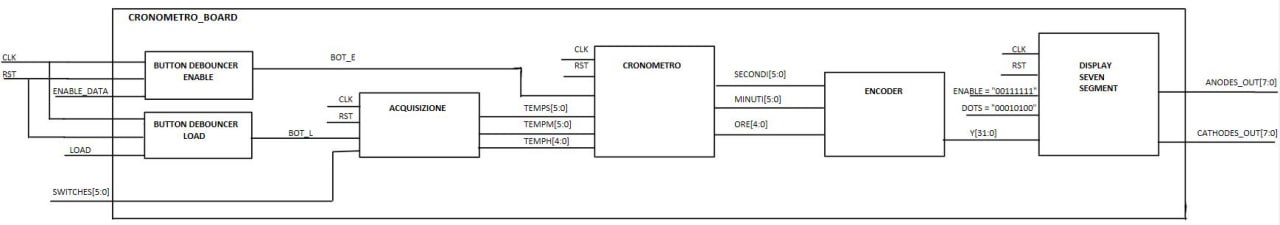
Descrizione generata automaticamente

## Esercizio 5.2

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando i display a 7 segmenti per la visualizzazione dell’orario (o una combinazione di display e led nel caso in cui i display a disposizione siano in numero inferiore a quello necessario), gli switch per l’immissione dell’orario iniziale e due bottoni, uno per il set dell’orario e uno per il reset. Si utilizzi una codifica a scelta dello studente per la visualizzazione dell’orario sui display (esadecimale o decimale).

### Progetto e architettura

Per questo esercizio abbiamo utilizzato un approccio strutturale, con top module il Cronometro\_Board, esso è formato da due ButtonDebouncer, un’acquisizione, il cronometro dell’esercizio precedente con la sola tempificazione modificata, attraverso un clock filter per fare in modo di avere una frequenza di 1 Hertz, Un Encoder per codificare l’uscita dei contatori del cronometro, e infine il display seven segment per gestire la visualizzazione su board. Il sistema prevede il clock della board, il reset che abbiamo affidato ad un bottone(M18), un enable\_data che ci permette di immettere un orario attraverso un bottone(P18), gli switches che utilizziamo per immettere l’orario (J15 fino a T18), il load che ci abilita a fornire i secondi, i minuti e le ore singolarmente, e infine abbiamo anodes\_out e cathodes\_out per la visualizzazione su board.



### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

#### Sintesi su board di sviluppo

Per effettuare la sintesi su board, abbiamo dovuto aggiungere alcuni componenti già citati prima, in particolare l’encoder per visualizzare l’orario con la codifica da noi scelta, il componente display seven segment per la visualizzazione sulla board, il ButtonDebouncer per permettere ai bottoni di reset e set l’acquisizione corretta trasformando la pressione del bottone in un segnale dalla durata di un colpo di clock, e un file di acquisizione dove andiamo a implementare proprio come acquisiamo i secondi, i minuti e le ore. Inoltre abbiamo modificato il Cronometro dell’esercizio precedente utilizzando il clock filter che ci permette di rallentare la durata di un colpo di clock ad un secondo.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

La modifica effettuata al cronometro è semplicemente l’istanziazione di un componente clock filter che ci rallenta il clock dandogli una durata di un secondo, enable\_s sarà come prima il segnale di clock del counter dei secondi. Il resto del cronometro è rimasto invariato.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

## Appendice

### Componente 1: Acquisizione

### Progetto e architettura

Per progettare questo componente abbiamo utilizzato un approccio comportamentale, dove ad ogni fronte di salita del clock, se il bottone di load viene premuto, il componente acquisisce il valore fornito dagli switch dei secondi, dei minuti o delle ore in base al valore della variabile x.

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

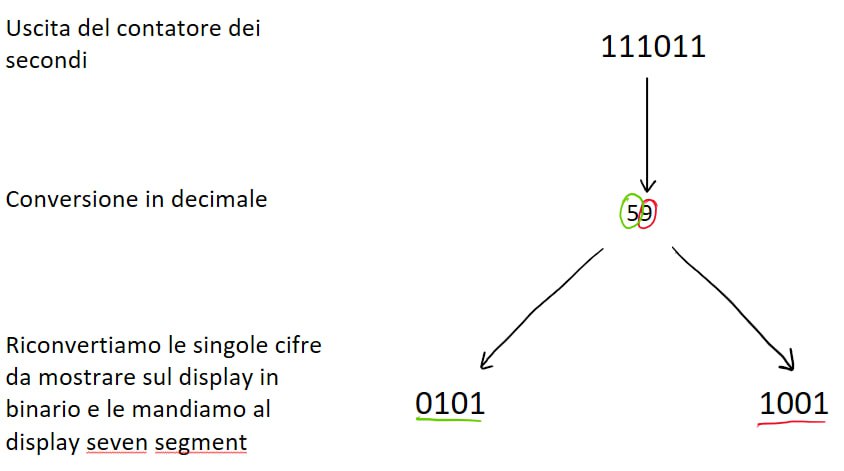
Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

### Componente 2: Encoder

### Progetto e architettura

Per questo componente abbiamo modificato quello che ci è stato fornito durante il corso, in particolare esso prende in ingresso dei vettori che rappresentano minuti, secondi ed ore. Esso ci fornisce in uscita un vettore che otteniamo concatenando quelli presi in ingresso. La funzionalità di questo componente è quella di convertire i vettori che prende in ingresso per fare in modo di visualizzare sulla board le cifre corrette. Per la codifica esso prende in ingresso il vettore in binario, lo converte in decimale e riconverte in binario le singole cifre su 4 bit. Ad esempio



### Implementazione

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

## Esercizio 5.3

Estendere il componente sviluppato ai punti precedenti in modo che sia in grado di acquisire e memorizzare internamente fino ad un numero N di intertempi in corrispondenza di un ingresso di stop. Opzionalmente, il componente può prevedere una modalità di visualizzazione in cui, alla pressione di un bottone, vengano visualizzati sui display gli intertempi memorizzati (uno per ogni pressione).

### Progetto e architettura

Per progettare questo esercizio, abbiamo esteso quello precedente creando un componente denominato Intertempi che ci permette di acquisire un certo numero di intertempi e di visualizzarli. Il componente Intertempi segue un approccio strutturale, infatti è formato da una memoria ed un counter. Tutti questi componenti sono generici, ovvero ci permettono di acquisire e visualizzare N intertempi. Intertempi prevede un clock, il segnale di reset, un enable che è rappresentato da un bottone denominato bot\_int(P17), un value ovvero un vettore di 32 bit che ci rappresenta il valore codificato da salvare in memoria, un bot\_v(M17) che scorre la memoria ed infine un vettore di uscita sempre di 32 bit che ci permette di visualizzare il contenuto salvato nella memoria.

Attraverso la pressione del bot\_int acquisiamo gli intertempi che vengono salvati nella memoria ad N locazioni di 32 bit, una volta acquisiti un certo numero di intertempi, all’interno del Cronometro\_Board abbiamo inserito un process, che si attiva sul fronte di salita del clock e in base allo stato dello switch (V10) ci permette di entrare in modalità visualizzazione, ovvero di mostrare gli intertempi acquisiti, premendo ripetutamente il bottone bot\_v.

### Implementazione

Di seguito inseriamo il process del Cronometro\_Board, che ci permette di entrare in modalità visualizzazione.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Implementazione del componente Intertempi:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, numero

Descrizione generata automaticamente

Di seguito il process aggiunto alla memoria che ci permette di visualizzare il contenuto di essa:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Sintesi su board di sviluppo

Oltre ai componenti citati già abbiamo modificato il file di constraint per effettuare la sintesi su board.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, schermata, documento, Carattere

Descrizione generata automaticamente

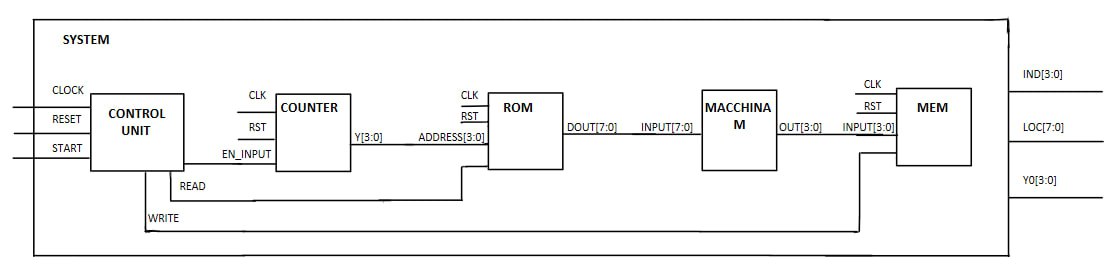
## Esercizio 6.1

Progettare, implementare in VHDL e verificare mediante simulazione un sistema dotato di una memoria ROM di N locazioni da 8 bit ciascuna, una macchina combinatoria M in grado di trasformare (secondo una funzione a scelta dello studente) la stringa di 8 bit letta dalla ROM in una stringa di 4 bit, e una memoria MEM di N locazioni che memorizza la stringa in output da M.

Il sistema si avvia in corrispondenza di un segnale di START che viene fornito esternamente. Una volta avviato, tramite un’apposita unità di controllo che gestisce la tempificazione del sistema, viene scandita una locazione alla volta della ROM e viene scritta la corrispondente locazione di MEM. Gli indirizzi di memoria sono forniti da un contatore. Le memorie ROM e MEM hanno rispettivamente un read e un write sincrono.

### Progetto e architettura

Per progettare questo sistema abbiamo utilizzato un approccio strutturale, è dunque composto da una Control Unit, un counter, una rom, una macchina\_M e una memoria. Abbiamo supposto una rom e una memoria entrambe di 16 locazioni e quindi il counter è modulo 16. Il sistema prevede un segnale di start che va in ingresso alla Control Unit che gestisce la tempificazione e abilita il counter, il counter manda il conteggio alla rom e questo sarà l’address della locazione che cerchiamo. La locazione della rom viene fornita in ingresso alla macchina\_M che la elabora (abbiamo utilizzato la stessa dell’esercizio 2.1) e la manda alla memoria che la salva.



### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

### Simulazione

Abbiamo simulato un testbench che dopo 20ns fornisce il segnale di start, mostrando i vari segnali di abilitazione forniti al counter, alla rom e alla mem. Infatti notiamo che i segnali sono alti in sequenza, tinput sarebbe il segnale di abilitazione al counter, tread identifica l’abilitazione alla rom di leggere la locazione, e infine twrite identifica l’abilitazione a scrivere sulla locazione di memoria il dato elaborato. A 170ns abbiamo alzato in reset infatti vediamo tutti i segnali bassi.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene schermata, Software multimediale

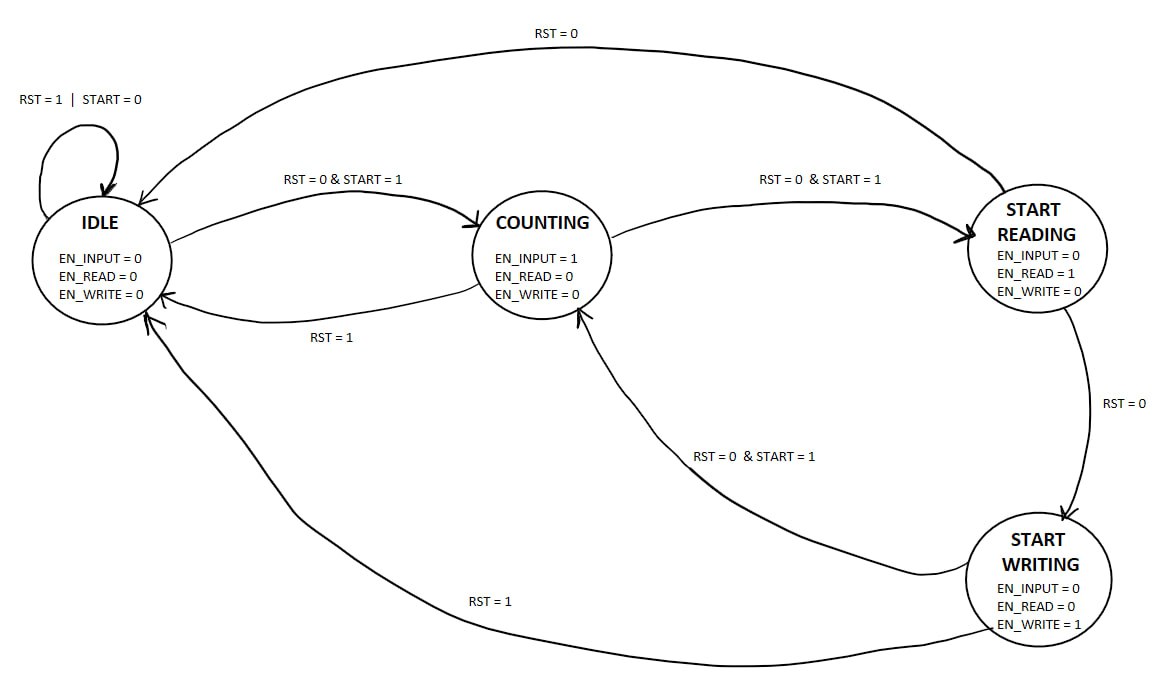
Descrizione generata automaticamente

## Appendice

### Componente 1 : ControlUnit

### Progetto e architettura

Abbiamo progettato questo componente come un automa a stati finiti, che presenta in ingresso il clock, il reset e il segnale di start, mentre in uscite ci fornisce le abilitazioni per i vari componenti del sistema. Il progetto prevede l’utilizzo di quattro stati idle, counting, start\_reading, start\_writing. Alla ricezione del segnale di start l’automa lascia lo stato di idle e passa a quello di counting, dove fornisce l’abilitazione attraverso la variabile en\_input al counter per genere gli address. Lo stato prossimo è quello di start\_reading, dove fornisce l’abilitazione alla rom di poter leggere la locazione attraverso la varibile en\_read. Infine abbiamo lo stato start\_writing, dove fornisce l’abilitazione alla mem di poter salvare il dato elaborato attraverso la variabie en\_write. L’automa quando vede il segnale di reset alto, porta tutti i segnali a 0 e torna nello stato di idle.



### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Componente 2 : Counter

Abbiamo utilizzato lo stesso dell’esercizio precedente. Abbiamo semplicemente fatto in modo che sia modulo 16.

### Componente 3: ROM

### Progetto e architettura

Per progettare la rom abbiamo utilizzato un approccio comportamentale. I segnali utilizzati per progettare la rom sono il clock per la tempificazione, abbiamo pensato di leggere il dato sul fronte di discesa, il reset che ci permette di azzerare il contenuto, un address per identificare la locazione di memoria da selezione, un’uscita denominata dout che ci permette di fornire il contenuto della locazione ai prossimi componenti del sistema e infine un read sincrona che ci permette di leggere il contenuto della locazione solo se essa è alta.

Immagine che contiene testo, diagramma, schermata, Carattere

Descrizione generata automaticamente

### Implementazione

Abbiamo modificato la rom utilizzata già in precedenza, per renderla sequenziale con i segnali sopra citati.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

### Componente 4: Macchina\_M

Abbiamo riutilizzato la macchina\_M progettata nell’esercizio 2.1.

### Componente 5: MEM

### Progetto e architettura

Abbiamo utilizzato un approccio comportamentale, i segnali utilizzati sono il clock che ci permette di scrivere il dato elaborato dalla macchina\_M sul fronte di discesa, il reset che se alto ci azzera il contenuto delle celle di memoria, l’input che è il dato proveniente dalla macchina\_M, l’address utilizzato per scrivere il dato nella corretta locazione di memoria, e infine la write che quando è alta ci permette di scrivere.

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## Esercizio 6.2

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando due bottoni per i segnali di read e reset rispettivamente e i led per la visualizzazione delle uscite della macchina istante per istante.

### Progetto e architettura

Per progettare questo esercizio abbiamo riutilizzato i componenti dell’esercizio precedente, abbiamo aggiunto il componente System\_Board, come top module. In particolare esso prevede un clock C, un reset tramite bottone(P17), uno start tramite bottone(M17), un read (P18)che ad ogni pressione del bottone ci premette di leggere il contenuto della MEM, e infine i led per visualizzare il contenuto.

Il funzionamento di System\_Board, prevede che attraverso la pressione del bottone di start la CU inizia ad elaborare. Abbiamo effettuato una modifica alla CU, per prevedere che ad ogni pressione del bottone di read venga letto il contenuto della ROM, e quindi solo quando questo risulta premuto abbiamo il cambiamento di stato da start\_reading a start\_writing, che darà poi in output sui led il contenuto della MEM.

### Implementazione

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, diagramma

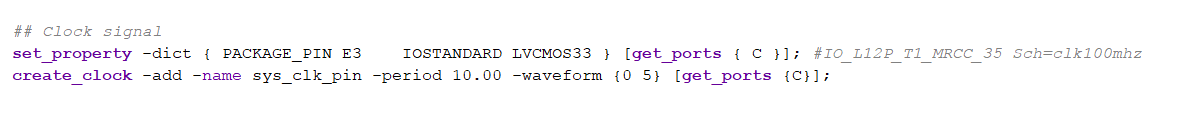
Descrizione generata automaticamente

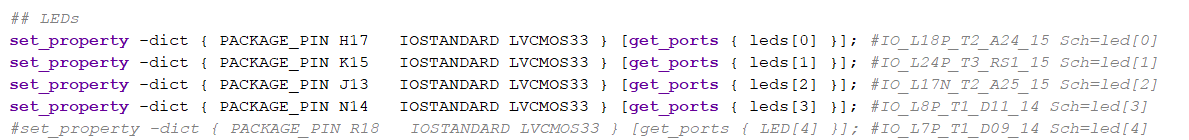
Immagine che contiene testo, schermata, Carattere

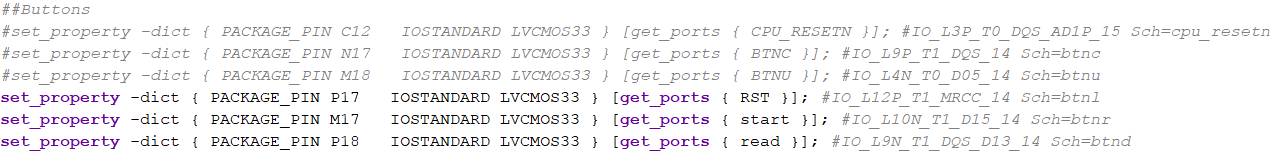
Descrizione generata automaticamente



### Sintesi su board di sviluppo







## Esercizio 7.1