# Immagine che contiene segnale Descrizione generata automaticamente

Progettazione e simulazione in VHDL di un controllore per una Washing Machine

Anno Accademico 2019/2020

Progetto di Digital Programmable Systems

**Prof. Ing. Francesco De Leonardis**

Membri del gruppo:

PACE NICOLA

PACIOLLA FRANCESCO

DAMONE MICHELE

Corso di Laurea Magistrale in Ingegneria dell’Automazione

Sommario

[**Cenni sul linguaggio VHDL** 3](#_Toc40864980)

[ENTITY DECLARATION 4](#_Toc40864981)

[ARCHITECTURE BODY 4](#_Toc40864982)

[PROCESSO COMBINATORIO 5](#_Toc40864983)

[**Progetto del controllore per una Washing Machine** 6](#_Toc40864984)

[SCHEMATICO DEL PROGETTO 6](#_Toc40864985)

[LIBRERIE INCLUSE 7](#_Toc40864986)

[SEGNALI DI I/O UTILIZZATI 7](#_Toc40864987)

[MACCHINA MEALY 8](#_Toc40864988)

[MACCHINA A STATI FINITI (FSM) 9](#_Toc40864989)

[FREQUENCY DIVIDER 12](#_Toc40864990)

[TIMER 13](#_Toc40864991)

[PWM 14](#_Toc40864992)

[LED DI STATO E BCD TO 7SEG 15](#_Toc40864993)

[TOP 16](#_Toc40864994)

[PIN PLANNER E TIMING ANALYZER 17](#_Toc40864995)

[**Simulazioni** 20](#_Toc40864996)

[SIMULAZIONI - FREQUENCY DIVIDER 20](#_Toc40864997)

[SIMULAZIONI - TIMER 20](#_Toc40864998)

[SIMULAZIONI-PWM 20](#_Toc40864999)

[SIMULAZIONI- PROCESSO COMPLETO 21](#_Toc40865000)

[**Conclusioni** 22](#_Toc40865001)

# **Cenni sul linguaggio VHDL**

VHDL (acronimo di VHSIC Hardware Description Language, dove "VHSIC" è la sigla di Very High Speed Integrated Circuits), è un linguaggio di descrizione dell'hardware, usato per la progettazione di sistemi elettronici digitali. È lo strumento fondamentale per la progettazione dei moderni circuiti integrati digitali e le sue applicazioni spaziano dai microprocessori, alle comunicazioni, al campo automobilistico.

Il VHDL si presenta per molti versi simile a un vero e proprio linguaggio di programmazione: in particolare ne usa i tipici costrutti quali *if...then...else, while, for*, tuttavia, essendo un linguaggio che descrive il funzionamento e la struttura di componenti hardware, ha alcune caratteristiche distintive rispetto ai linguaggi software. La principale è la concorrenzialità, ovvero diverse parti di un codice scritto in VHDL, una volta tradotte in un circuito elettronico, funzionano contemporaneamente, in quanto dispongono di hardware dedicato.

Una specifica VHDL può essere simulatamediante opportuni strumenti come Modelsim. Simulare una specifica VHDL significa simulare il comportamento del circuito che la specifica descrive.

Il linguaggio VHDL è estremamente ricco e flessibile e permette di fornire specifiche di circuiti

digitali a diversi livelli di astrazione**:**

* Structural: Al livello più basso di astrazione possiamo vedere un circuito come un grafo in cui i nodi rappresentano elementi logici (semplici o complessi) quali porte logiche, multiplexer, flip-flop oppure interi sotto circuiti, mentre gli archi rappresentano le connessioni tra tali elementi. Tale rappresentazione è comunemente detta netlisted è il modello che più si avvicina alla realizzazione finale del circuito in esame.
* Data flow: Aumentando il livello di astrazione, si passa dal VHDL gate-level al VHDL RTL (Register Transfer Level) o data-flow. Secondo questo paradigma, la specifica descrive esplicitamente le trasformazioni che i dati subiscono durante la propagazione all’interno del circuito. In particolare, il circuito è visto come un insieme di due tipologie di elementi:

1. Reti combinatorie che esprimono in forma esplicita le trasformazioni dei dati mediante

espressioni algebriche, espressioni aritmetiche e condizioni.

2. Registri che sono deputati a memorizzare i risultati intermedi di elaborazioni complesse.

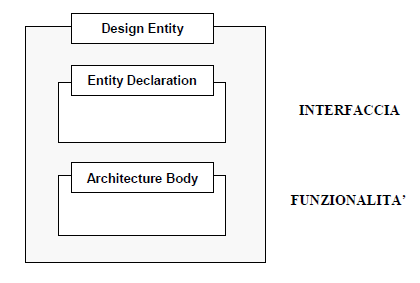
Questa suddivisione spiega il nome Register Transfer: la specifica infatti esprime come avviene il trasferimento e l’elaborazione dei dati tra i registri della rete.

* Behavioural: È il massimo livello di astrazione che il VHDL consente. Esso è in grado di descrivere la funzionalità di un circuito mediante uno o più algoritmi. In questo caso, né la struttura, né le singole trasformazioni che i dati subiscono sono esplicite. In particolare, non è esplicito come le varie operazioni ed elaborazioni siano assegnate ai diversi cicli di clock. Non è quindi evidente la struttura mostrata per le descrizioni a livello RTL: sarà lo strumento di sintesi ad effettuare lo scheduling delle operazioni sui vari cicli di clock sulla base di vincoli imposti dal progettista.

Ogni sistema, dal più semplice al più complesso, si compone di unità funzionali ben identificabili dette moduli o blocchi. Queste hanno lo scopo di isolare una funzione ben precisa del sistema complessivo con il duplice scopo di fornire una visione strutturata del progetto e di scomporre un problema molto complesso in un insieme di sotto problemi di complessità minore.

Una design entity è un modulo e la sua specifica, mediante il linguaggio VHDL, sottolinea e separa i due concetti complementari di interfaccia e comportamento. L’interfaccia è quella parte di blocco che specifica i segnali di ingresso e di uscita e che consente di connettere il blocco stesso ad altri blocchi. Il comportamento, invece, descrive come i segnali di ingresso devono essere elaborati per

produrre i segnali di uscita. L’interfaccia di un modulo prende il nome di entity, mentre il comportamento prende il nome di architecture. Risulta evidente che ogni architecture è associata ad una ed una sola entity. non è invece vero il contrario infatti ad una entity possono essere associate più architecture.



## ENTITY DECLARATION

Definisce una entity e la sua interfaccia con l’ambiente esterno, non definisce la funzionalità del modello. L’entity name definisce il nome della design entity e deve essere unico.



L’istruzione port identifica l’insieme dei segnali di interfaccia della entity per comunicare con l’ambiente esterno.

## ARCHITECTURE BODY

Descrive la funzionalità del modello attraverso la definizione delle relazioni funzionali tra gli ingressi e le uscite di un modello. L’associazione di una architecture specifica ad una entity prende il nome di configuration declaration.

La prima sezione di una architecture, opzionale, è costituita da un elenco di dichiarazioni. Le dichiarazioni possono essere di quattro tipi:

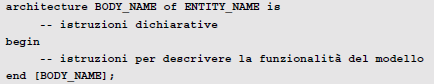
1. Dichiarazioni di costanti: Nomi, tipi e valori delle costanti simboliche utilizzate nella specifica.

2. Dichiarazioni di segnali: Nomi e tipi dei segnali che saranno usati nella specifica della funzionalità del modulo.

3. Dichiarazioni di tipi: Nomi e definizioni di tipi definiti dall’utente ed utilizzati nella specifica.

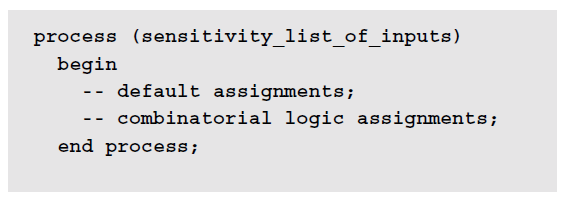
4. Dichiarazioni di componenti: Nomi ed interfacce dei moduli utilizzati nella specifica dell’architecture in esame.

Tra le parole chiave begin ed end compare la sezione destinata a raccogliere la descrizione della funzionalità che il modulo deve svolgere. La funzionalità di un intero circuito, quindi, si trova quindi distribuita nelle architecture declaration delle diverse entity che costituiscono i moduli del sistema.



## PROCESSO COMBINATORIO

Un processo è uno statement concorrente, al pari di un assegnamento, un assegnamento condizionato o una istanziazione. Un primo importantissimo aspetto è che gli statements all’interno di un processo non sono concorrenti bensì sequenziali. Un possibile utilizzo di un processo potrebbe quindi essere quello di forzare una interpretazione sequenziale del codice VHDL del suo corpo.  
Un processo ha un nome opzionale process\_name, utile prevalentemente per motivi di leggibilità, una parte dichiarativa opzionale declaration, simile a quella dell’architecture declaration ed un corpo body. Un processo ha una sensitivity list ovvero una lista di segnali in grado di attivarlo: tutte le volte che si verifica un evento su uno dei segnali della sensitivity list il processo è attivato.



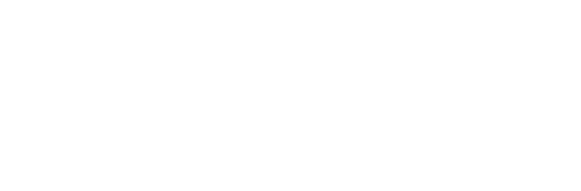
# **Progetto del controllore per una Washing Machine**

L’obiettivo del progetto sviluppato è progettare e simulare un controllore completamente automatico e funzionante di una lavatrice avente:

* 4 diverse modalità di funzionamento (full power, eco ...)
* La possibilità di attivare l’asciugatura o meno
* Un'interfaccia utente (led, 7 seg)
* Diversa durata per ogni stato di funzionamento

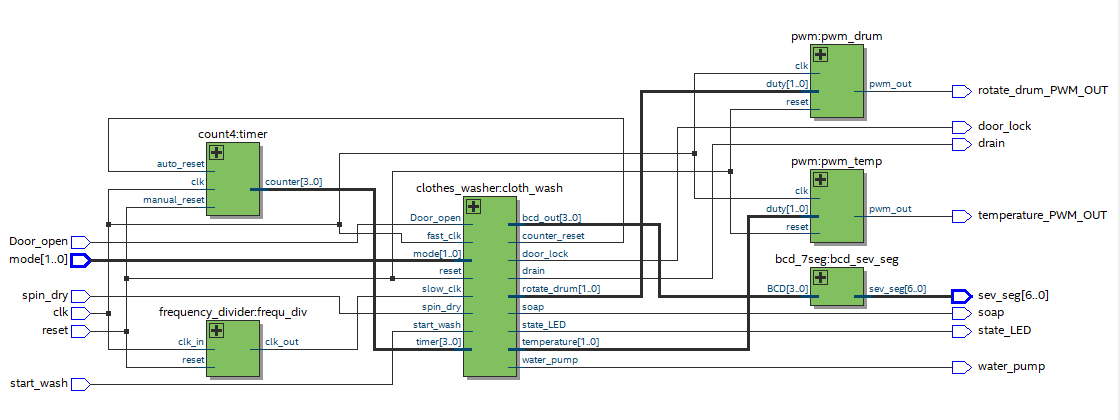
Per la realizzazione del progetto sono stati implementati:

* Un divisore di frequenza per rallentare il clock di sistema
* Una macchina a stati finiti per implementare i diversi stati del processo di lavaggio
* Un timer per attuare le diverse durate degli stati
* Un controllore PWM per tradurre un duty cycle in un'uscita PWM per pilotare la velocità del cestello e la temperatura di lavaggio
* Un display a 7 segmenti per mostrare lo stato attuale della macchina a stati finiti



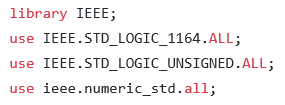
## SCHEMATICO DEL PROGETTO

All’interno del controllore sono stati implementati i seguenti blocchi:



## LIBRERIE INCLUSE

Nella progettazione, prima di definire le entità sono state incluse le librerie IEEE che, come ogni libreria VHDL, è suddivisa in package ognuno dei quali definisce alcuni oggetti che possono essere utilizzati nella progettazione. In particolare, il package std\_logic\_1164 definisce i due tipi risolti std\_logic e std\_ulogic ed i corrispondenti tipi vettoriali std\_logic\_vector e std\_ulogic\_vector, sono stati anche inclusi i package std\_logic\_unsiged per i numeri senza segno e quello numeric\_std.



## SEGNALI DI I/O UTILIZZATI

Per l’implementazione del controllore della Washing Machine è stata implementata una macchina a stati finiti (FSM). Per far ciò è stata creata una entity con tutti i segnali di I/O utilizzati, questi sono riassunti nella seguente tabella.

|  |  |  |  |
| --- | --- | --- | --- |
| SIGNAL | MODE | TYPE | FUNCTION |
| timer | In | std\_logic\_vector (3 downto 0) | Timer a 4 bit |
| slow\_clk | In | std\_logic | System clock |
| fast\_clk | In | std\_logic | Clock FSM |
| spin\_dry | In | std\_logic | Dry |
| start\_wash | In | std\_logic | Start |
| Door\_open | In | std\_logic | Door sensor |
| reset | In | std\_logic | Reset |
| mode | In | std\_logic\_vector (1 downto 0) | Washing mode |
| door\_lock | Out | std\_logic | Door lock button |
| water\_pump | Out | std\_logic | Water pump |
| soap | Out | std\_logic | Soap sensor |
| temperature | Out | std\_logic\_vector (1 downto 0) | Temperature sensor |
| rotate\_drum | Out | std\_logic\_vector (1 downto 0) | Drum velocity |
| drain | Out | std\_logic | Drain water |
| state\_LED | Out | std\_logic | Output LED |
| counter\_reset | Out | std\_logic | Counter reset |
| bcd\_out | Out | std\_logic\_vector (3 downto 0) | BCD to 7 segment output |

I segnali di tipo std\_logic sono bit che possono assumere solo i valori logici 0 ed 1, invece i segnali di tipo std\_logic\_vector sono vettori la cui dimensione è indicata dagli indici presenti.

## MACCHINA MEALY

Una macchina di Mealy è un automa a stati finiti i cui valori dello stato d’uscita sono determinati dallo stato attuale in cui si trova e dagli input che riceve, a differenza della macchina di Moore, che invece lavora solo in funzione dello stato corrente. Tuttavia, non per tutte le macchine di Mealy si può definire una macchina di Moore equivalente.

Una macchina di Mealy è una sestupla, {S, S0, Σ, Λ, T, G}, costituita da:

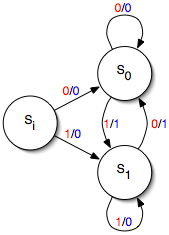
* un insieme finito di stati (S)
* uno stato iniziale S0, che è un elemento di (S)
* un insieme finito chiamato alfabeto degli ingressi (Σ)
* un insieme finito chiamato alfabeto delle uscite (Λ)
* una funzione di transizione (T: S × Σ → S) che tiene conto delle relazioni per passare da una coppia di stato-input al corrispondente stato successivo
* una funzione uscita (G: S × Σ → Λ)

Il diagramma di stato per una macchina Mealy associa un valore di uscita a ciascun fronte di transizione, in contrasto con il diagramma di stato per una macchina Moore, che associa un valore di uscita a ciascuno stato. Una macchina Mealy è un trasduttore deterministico a stati finiti, ovvero per ogni stato e input è possibile al massimo una transizione.

Le macchine Mealy:

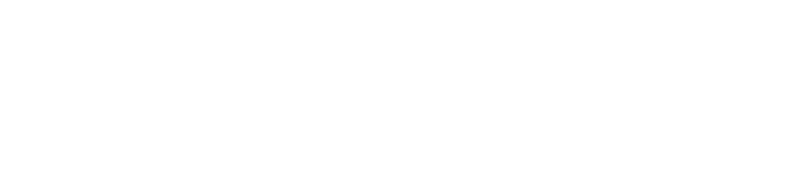
* Tendono ad avere un numero inferiore di stati
* Hanno diverse uscite sugli archi (n^2) ed n stati
* La modifica dell'ingresso può causare la modifica dell'uscita non appena viene eseguita la logica, ciò rappresenta un problema quando due macchine sono interconnesse, in quanto può verificarsi la generazione di un output asincrono.
* Reagiscono più velocemente agli input, infatti queste reagiscono nello stesso ciclo, senza aspettare il clock.

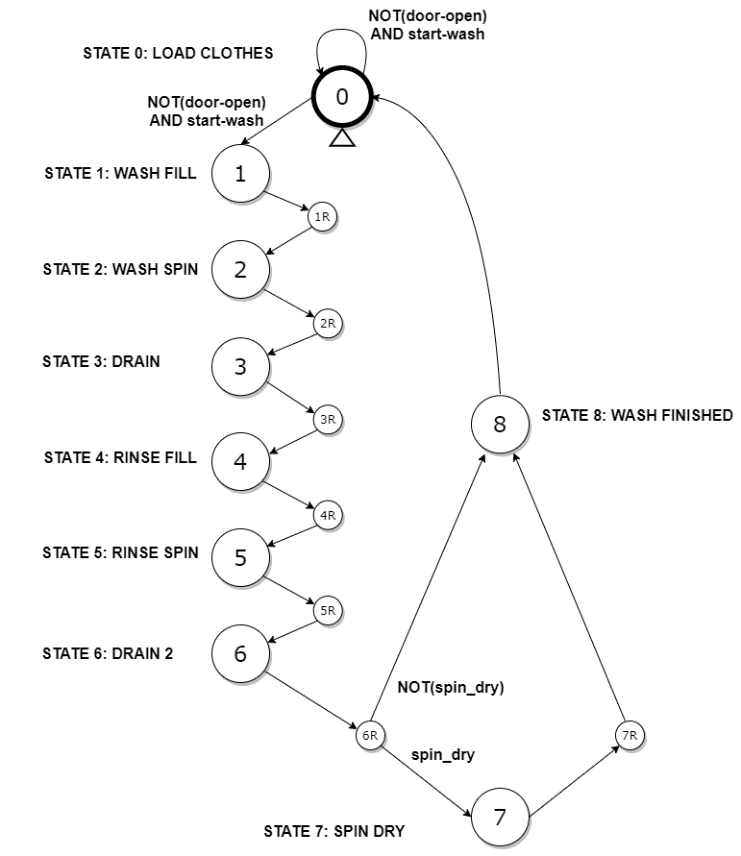
Nella figura è rappresentata una semplice macchina Mealy con un input ed un output. Ogni transizione è etichettata con il valore dell'ingresso (mostrato in rosso) e il valore dell'uscita (mostrato in blu). In questo esempio, La macchina parte dallo stato Si e l'output è l’implementazione della XOR dei due valori di input più recenti; pertanto, la macchina implementa un edge detector, creandone uno ogni volta che si inverte l'ingresso.



## MACCHINA A STATI FINITI (FSM)

È una macchina Mealy e rappresenta il cuore del progetto sviluppato, è costituita da 9 stati principali e da 7 "stati di reset" grazie ai quali viene ripristinato il contatore.

È necessario implementare una reset logic per resettare il contatore su "0" quando c'è un cambiamento di stato.  
Per l’implementazione della FSM viene definita una architecture utilizzando un type definito custom con tutti gli stati della FSM. Il diagramma degli stati della FSM è mostrato nella seguente figura.



All’interno dell’architecture sono stati creati quattro processi denominati reset\_auto, state\_register, next\_state\_logic e output\_logic.

I processi creati sono:

* Reset\_auto: È il segnale di reset del counter
* State\_register: Consente di salvare lo stato corrente della FSM per un periodo di tempo prestabilito diverso per ciascuno stato. La durata degli stati WASH SPIN (2) e RINSE SPIN (5) dipende dalla modalità scelta:

Modalità normale (00): il conteggio del timer arriva fino a “0111”

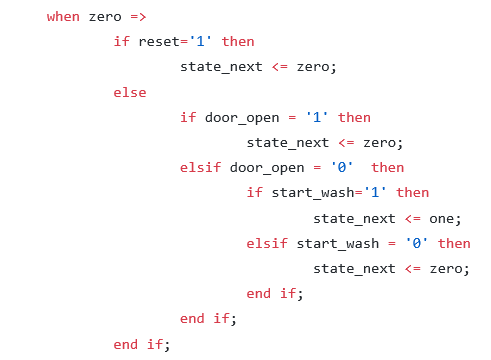
Modalità Eco (01): il conteggio del timer arriva fino a “0011”

Full Power (10): il conteggio del timer arriva fino a “1111”

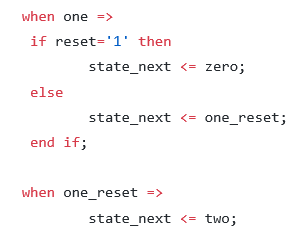
Modalità lana (11): il conteggio del timer arriva fino a “0111”

* Next\_state\_logic: Descrive la sequenza logica della FSM

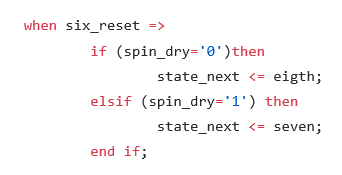
La FSM rimane nello stato LOAD CLOTHES (0) se valgono le seguenti relazioni mostrate nel seguente script. In ogni stato vi è la condizione che se è verificato reset=1 allora la FSM va nello stato 0.

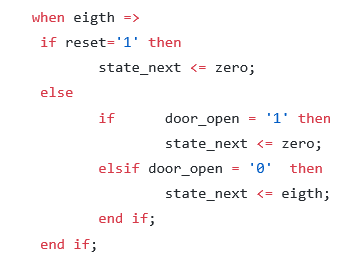


La FSM passa allo stato WASH SPIN (1) solo se start\_wash=1 e door open=0. Per gli altri stati la FSM passa dallo stato corrente a quello successivo senza condizioni, passando per gli stati di reset.



Nello stato di six\_reset (6R) viene verificato lo stato dell’ingresso spin\_dry, se questo è pari a 1 allora è stata selezionata l’asciugatura, quindi la FSM passa nello stato SPIN DRY (7), altrimenti passa direttamente nello stato finale WASH FINISHED (8).



Nello stato finale, se l’ingresso door\_open=0 allora la FSM rimane nello stato 8, altrimenti va nello stato 0 poiché la porta della washing machine viene aperta. 

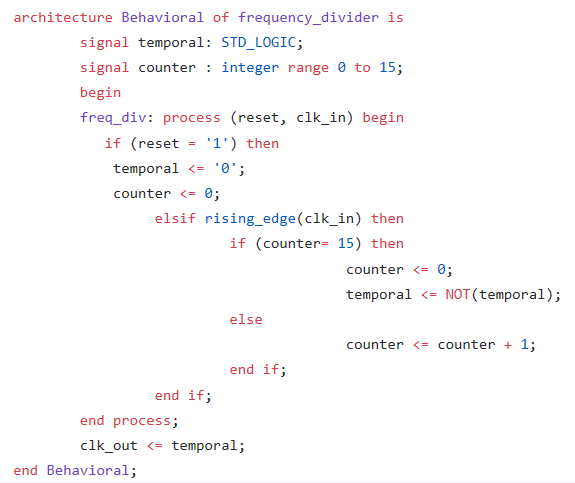
* Output\_logic: Descrive gli outputs del processo in funzione degli input in ingresso alla FSM. In arancione vengono evidenziati gli ingressi, in verde gli output della FSM ed in rosso gli stati correnti e quelli futuri a seconda degli input Nella tabella della verità mostrata vengono indicati gli output per la modalità Normal

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **STATE** | **STATE CURRENT** | **DOOR OPEN** | **START WASH** | **SPIN DRY** | **DOOR LOCK** | **WATER PUMP** | **SOAP** | **ROTATE DRUM** | **TEMPERATURE** | **DRAIN** | **STATE NEXT** |
| RESET | 1R/2R/3R/  4R/5R/6R/7R | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 2/3/  4/5/6/7/8 |
| LOAD CLOTHES | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 0 |
|  | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 0 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 0 |
|  | 0 | X | 1 | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 1 |
| WASH FILL | 1 | X | X | 0 | 1 | 1 | 1 | 00 | 11 | 0 | 1R |
| WASH SPIN | 2 | X | X | 0 | 1 | 0 | 0 | 11 | 10 | 0 | 2R |
| DRAIN | 3 | X | X | 0 | 1 | 0 | 0 | 00 | 00 | 1 | 3R |
| RINSE FILL | 4 | X | X | 0 | 1 | 1 | 0 | 00 | 00 | 0 | 4R |
| RINSE SPIN | 5 | X | X | 0 | 1 | 0 | 0 | 11 | 00 | 0 | 5R |
| DRAIN 2 | 6 | X | X | 0 | 1 | 0 | 0 | 00 | 00 | 1 | 6 R |
|  | 6 | X | X | 1 | 1 | 0 | 0 | 00 | 00 | 1 | 7 R |
| SPIN DRY | 7 | X | X | 1 | 1 | 0 | 0 | 11 | 00 | 1 | 7R |
| WASH FINISHED | 8 | 1 | X | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 8 |
|  | 8 | 0 | X | 0 | 0 | 0 | 0 | 00 | 00 | 0 | 0 |

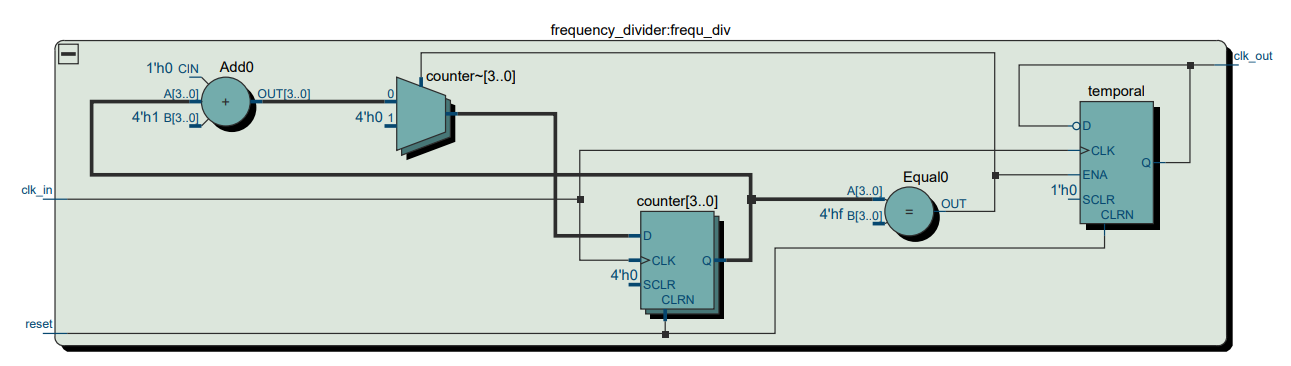
## FREQUENCY DIVIDER

Il clock del core dell’FPGA lavora ad una frequenza troppo elevata, quindi è necessario rallentarlo per l’applicazione in questione.

L’architecture è definita mediante il seguente script, in particolare viene implementato un contatore che conta ad una counting frequency che è una frazione di quella del clock dell’FPGA che lo alimenta. Il counter si attiva sui fronti di salita del clk\_in e quando il conteggio arriva a 15, questo viene resettato, altrimenti aumenta di 1 il conteggio. Il clk\_out in uscita è pari al valore della variabile temporal.



Di seguito viene mostrato lo schema a blocchi del frequency divider: Un contatore conta da 0 a 15 e ogni volta che raggiunge 15, lo stato dell'uscita viene modificato. Il MUX reimposta a 0 il contatore quando viene raggiunto 15.



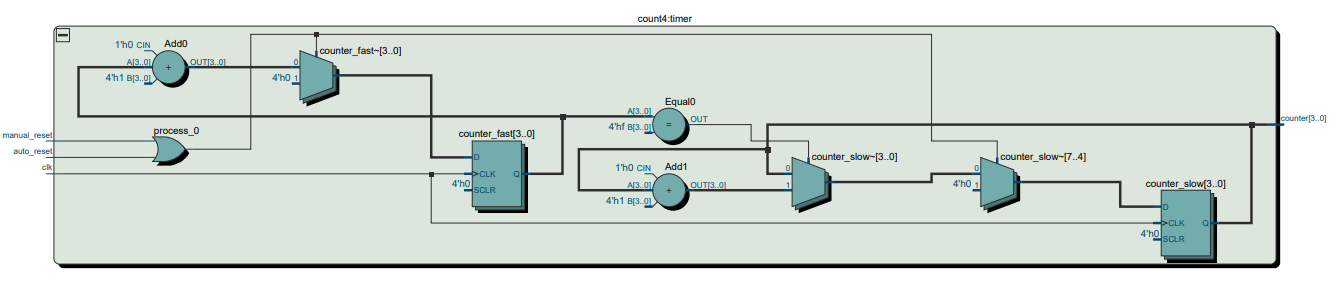
## TIMER

Viene utilizzato un timer per implementare la diversa durata degli stati di funzionamento della FSM.

In particolare, nell’architecture vengono definiti due segnali counter\_fast e counter\_slow, dove il primo rappresenta il conteggio veloce effettuato con il clock di sistema e quello lento invece viene aggiornato quando il counter veloce raggiunge il conteggio “1111”. Il timer utilizzato per implementare la diversa durata degli stati è counter\_slow.

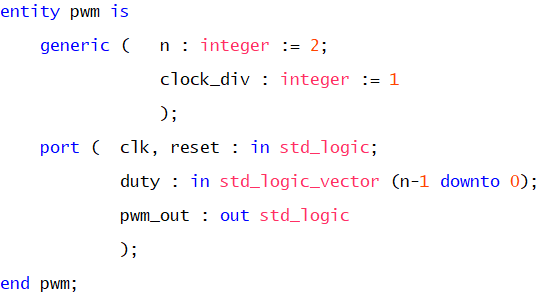


Il timer viene implementato mediante 2 counter: counter\_fast e counter\_slow di 4 bit in cascata con 2 MUX usati per resettare a "0000" i contatori ed un terzo MUX usato per aggiornare il counter\_slow solo quando counter\_fast raggiunge "1111".



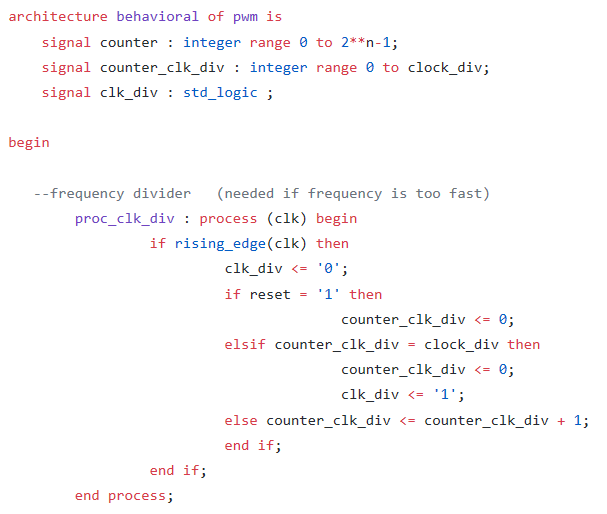
## PWM

Nell’entity vengono definiti due segnali generic, n che rappresenta la risoluzione e il clock\_div che è il rapporto tra la frequenza in ingresso e uscita desiderato.

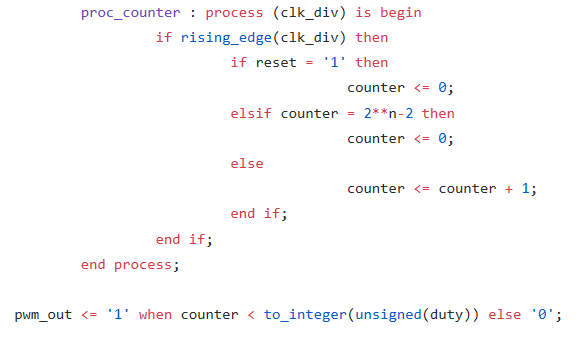


Il driver PWM che è stato implementato è costituito da 2 processi:

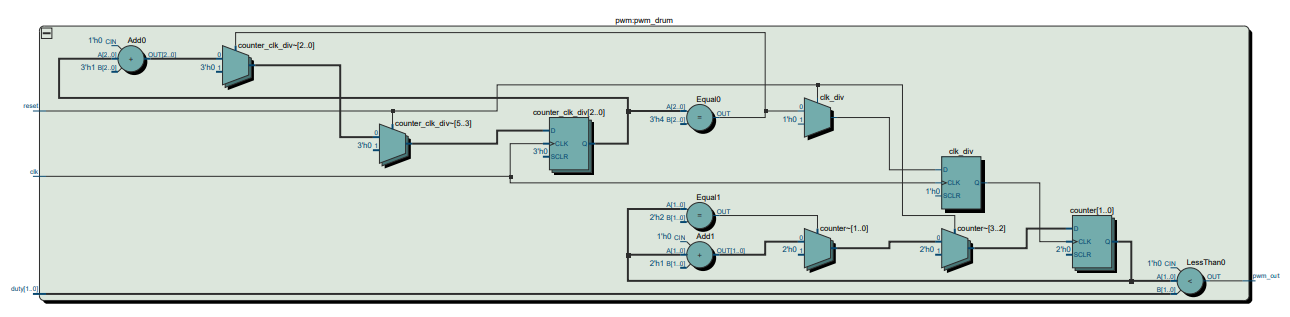
* Un divisore di frequenza per impostare la frequenza PWM corretta, necessario se la frequenza considerata è molto elevata



* Un contatore a n-bit, se counter < duty allora in uscita il PWM=1, altrimenti PWM=0.

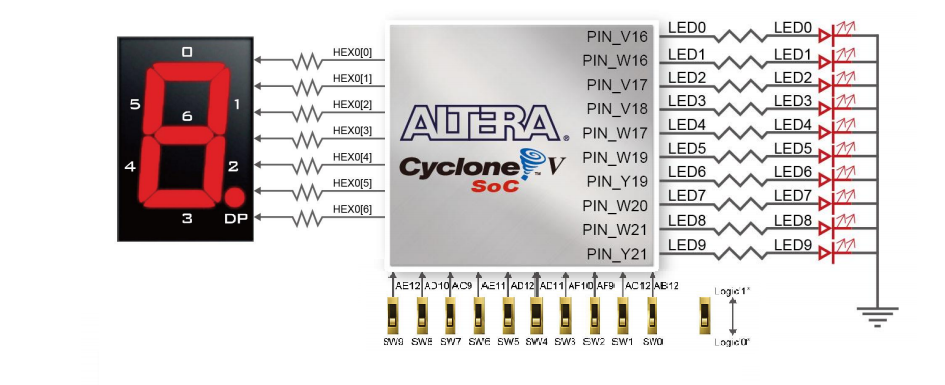


Lo schematico del PWM mostra come la parte sinistra rappresenti un contatore che viene usato per impostare la corretta frequenza del PMW e la parte destra è un contatore che ferma il conteggio a seconda del duty cycle in ingresso.

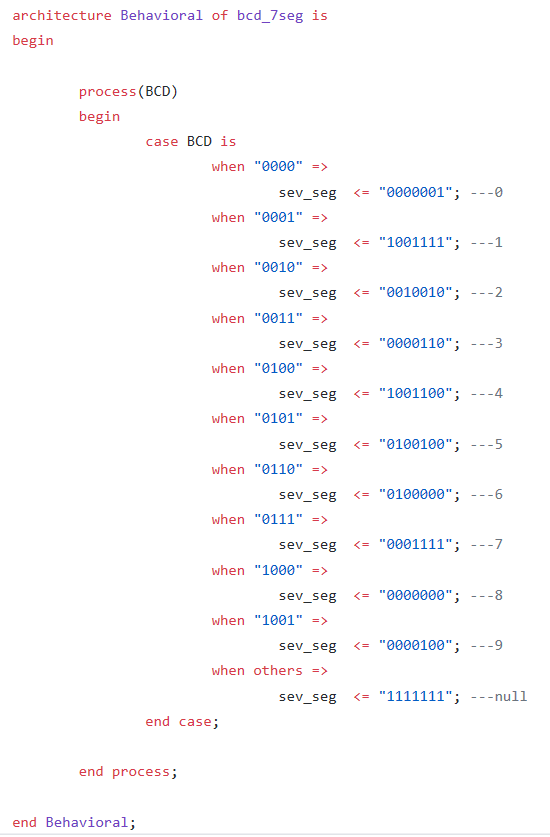


## LED DI STATO E BCD TO 7SEG

Per la realizzazione dell’interfaccia utente è stato utilizzato un LED che si accende per verificare il corretto funzionamento del processo di lavaggio. Inoltre, è stato implementato un decodificatore da BCD a 7 segmenti converte lo stato corrente della FSM da 4 bit a 7 bit, questo può essere visualizzato mediante un display a 7 segmenti.



Per l’implementazione del 7 segmenti è stato implementato il seguente script contenente una case structure con i nove casi legati ai 9 stati principali che la FSM può assumere.

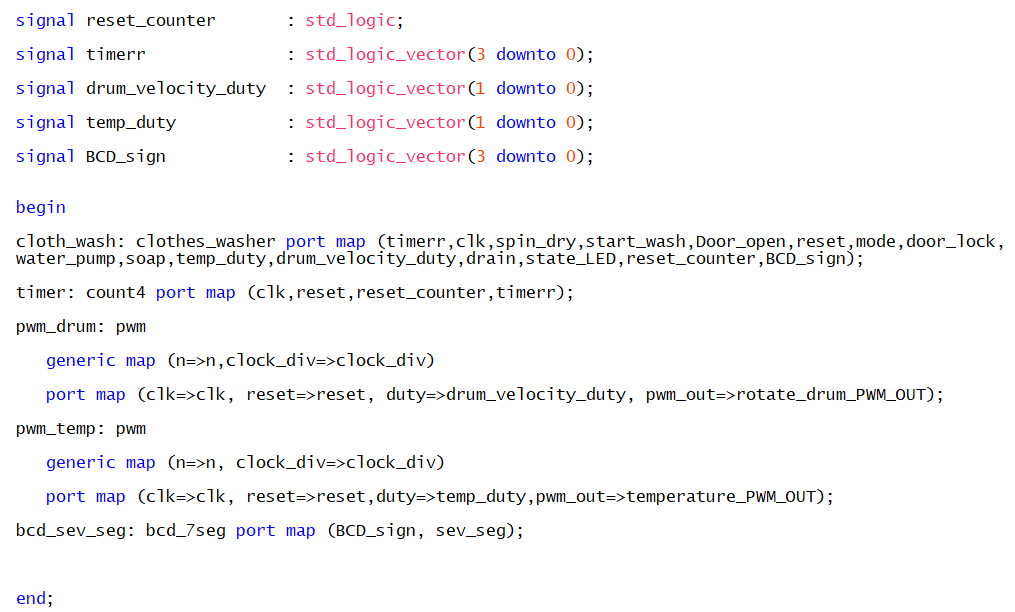


## TOP

Viene definito come top level entity e viene utilizzato per unire tutti i diversi componenti sviluppati nel progetto.



Mediante le diverse port map vengono collegati i segnali interni ed esterni ai diversi componenti.



## PIN PLANNER E TIMING ANALYZER

Per il progetto è stata utilizzata una FPGA ALTERA Ciclone V - 5CSEMA6F31C6 sulla board teraSIC DE1-SoC.

Il Pin Planner:

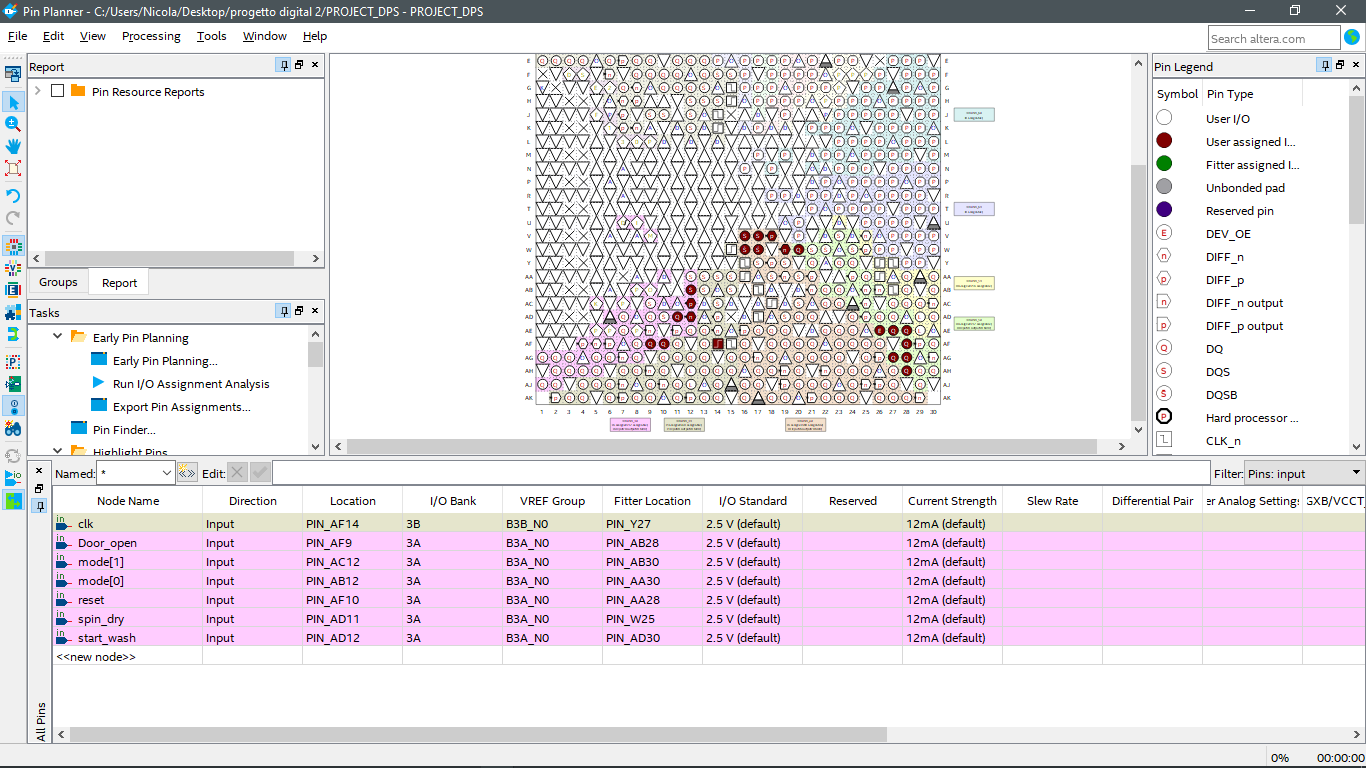
* Aiuta a visualizzare, pianificare ed assegnare i pin del dispositivo in una vista grafica del package del dispositivo utilizzato
* Utilizzando lo user manual della scheda è possibile assegnare il nodo definito a tipi di pin specifici

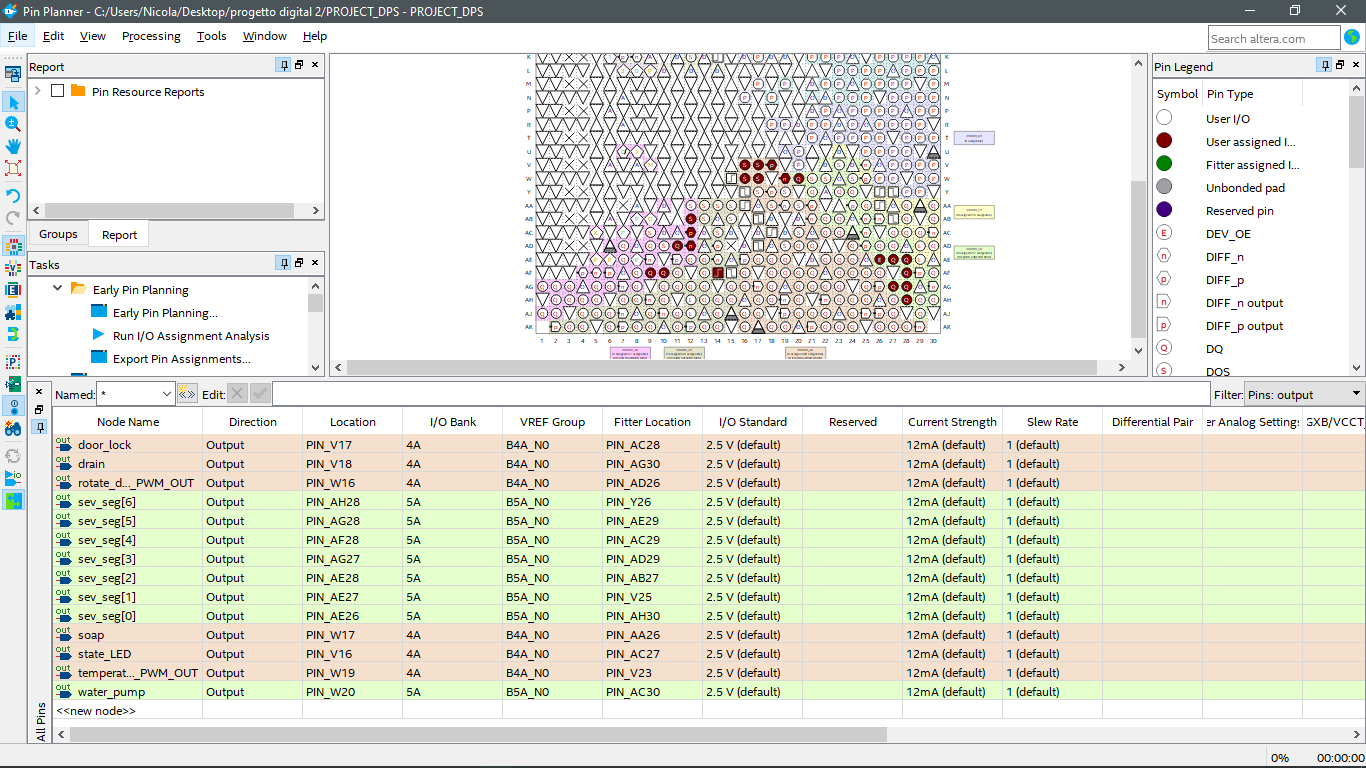
(I / O, VREF, clock)

Il Timing Analyzer:

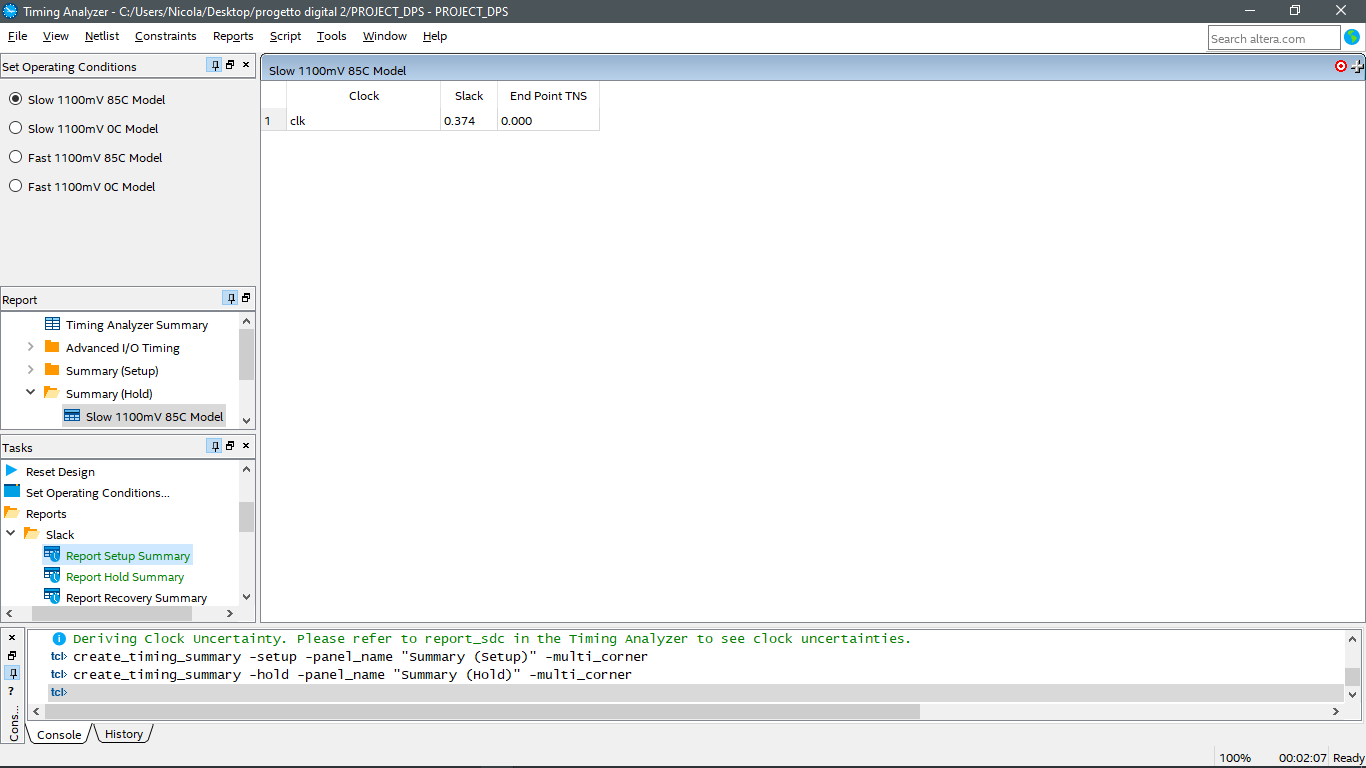
* Determina le relazioni temporali che devono essere soddisfatte per progettare correttamente il funzionamento
* Controlla i tempi di arrivo dei segnali rispetto ai tempi richiesti per verificare il corretto timing
* Il Timing Analyzer Report riporta i risultati dei controlli del setup del clock e verifica il valore dello slack
* Un requisito di temporizzazione è soddisfatto se il valore di slack è positivo

Per il Pin Planning abbiamo effettuato l’assegnazione di tutti i segnali di input e output utilizzati ai pin fisici della scheda a seconda della tipologia riportata sullo user manual della scheda. Nelle seguenti figure sono mostrate le assegnazioni effettuate.





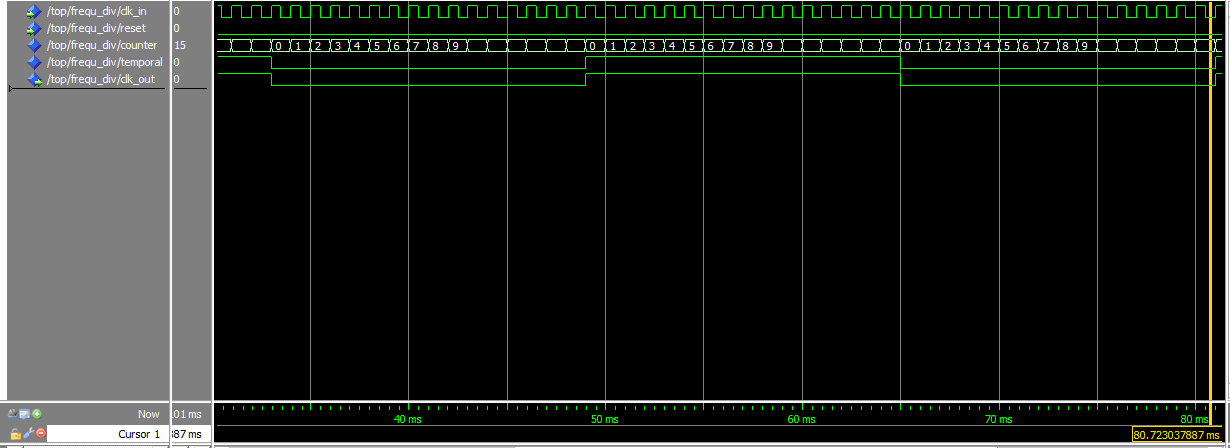
Nella seguente figura è mostrato il valore dello slack ottenuto dopo il Timing Analyzer, questo è positivo quindi la frequenza di funzionamento del progetto in questione è inferiore alla frequenza massima della scheda utilizzata.



# **Simulazioni**

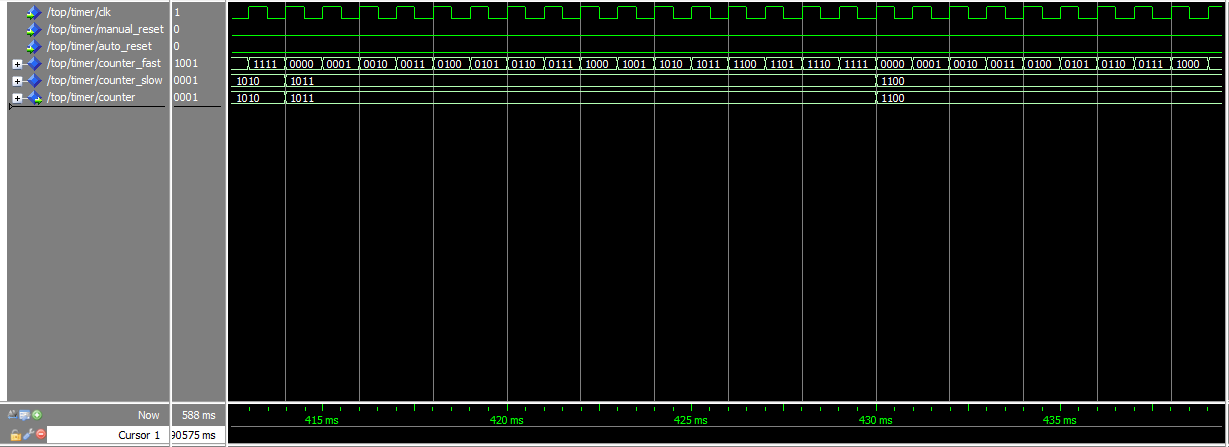
## SIMULAZIONI - FREQUENCY DIVIDER

Dalla simulazione è possibile notare la funzione del divisore di frequenza il quale, avendo un clk\_in in ingresso ad una data frequenza, ha in uscita un clock con una frequenza 15 volte inferiore poiché il counter era stato definito con modulo 15. Una volta terminato il conteggio, lo stato dell'uscita viene modificato.



## SIMULAZIONI - TIMER

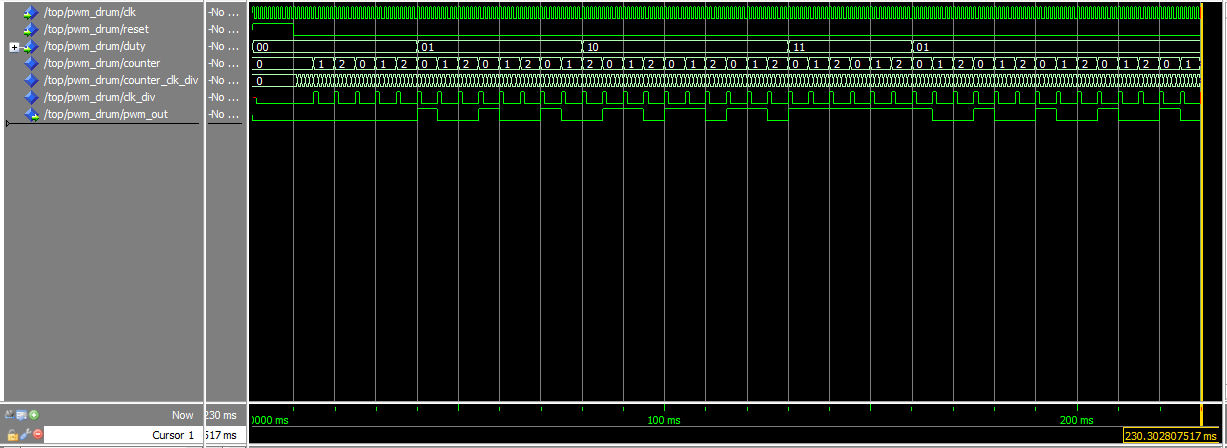
Quando counter\_fast raggiunge "1111", il conteggio counter\_slow viene incrementato di 1. Il timer in uscita viene aggiornato quando si aggiorna counter\_slow.



## SIMULAZIONI-PWM

È possibile notare come:

* Quando il duty cycle in ingresso è “00”, corrispondente al 0% in uscita non vi è modulazione PWM
* Quando il duty cycle è pari a “01” (33%) il PWM in uscita dura un periodo del clock\_div, cioè il clock in uscita dal divisore interno al PWM, ovvero il PWM è alto per un terzo del ciclo di conteggio
* Quando il duty cycle è pari a “10” (66%) il PWM in uscita dura due periodi del clock interno, ovvero il PWM è alto per due terzi del periodo di conteggio
* Quando il duty cycle è pari a “11” (100%) il PWM in uscita dura un ciclo di conteggio del counter interno, ovvero il PWM è sempre alto



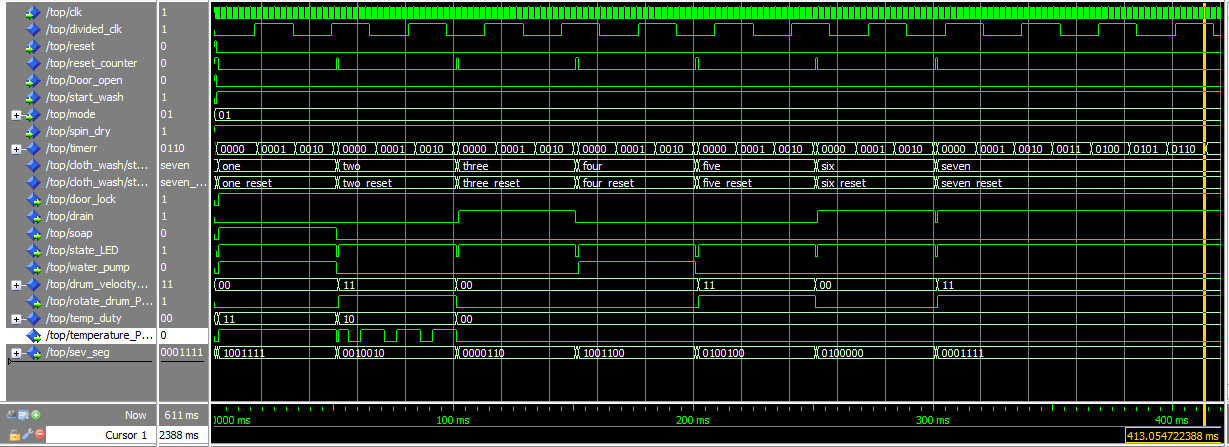
## SIMULAZIONI- PROCESSO COMPLETO

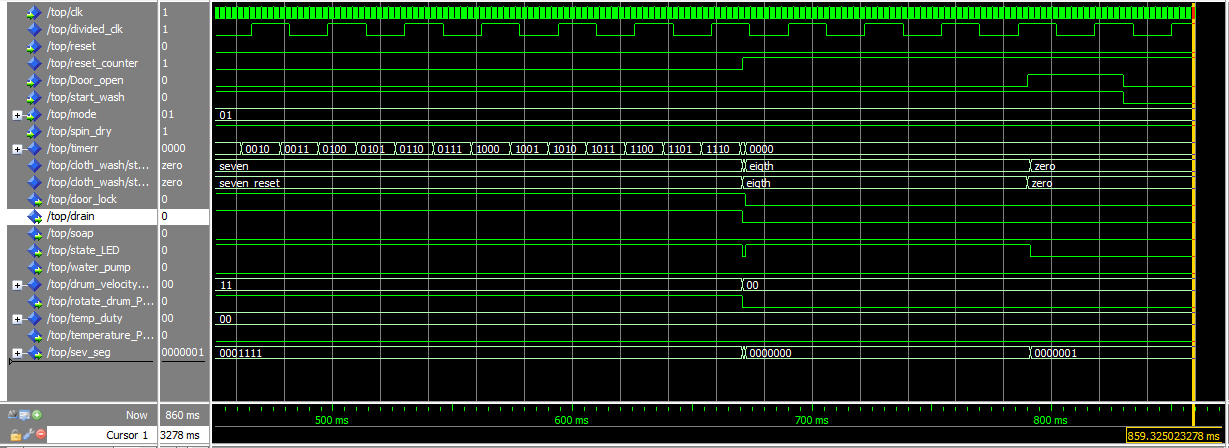
La simulazione inserita rappresenta il funzionamento della Washing Machine in modalità ‘01’, questa può essere modificata real-time.

È possibile notare:

* Il corretto funzionamento del frequency divider in quanto il divided\_clk ha in uscita una frequenza inferiore del clock di sistema
* Come la FSM rimanga negli stati di reset per un tempo molto piccolo, pari alla durata di reset\_counter.
* L’implementazione della diversa durata degli stati, infatti il timer nello sette conta fino a “1110”, invece negli altri stati fino a “0010”.
* L’implementazione del PWM sulla drum velocity e sulla temperatura, infatti nello stato due, quando il duty della temperatura è “10”, viene creata un’onda quadra con duty pari a 66%

Una volta posto a “0” Door open e Start Wash a “1” inizia il ciclo di lavaggio e door lock va a “1” per tutta la durata del lavaggio, solo in questo caso la FSM passa dallo stato zero allo stato uno.   
Il drain viene attivato nello stato tre e negli stati sei e sette poiché è stata impostata come input spin dry a “1”, ovvero è richiesta l’asciugatura. Il sapone viene inserito nello stato uno e il water pump viene attivato negli stati uno e quattro dove questa è necessaria.





# **Conclusioni**

In conclusione, è stato progettato un controllore per una Washing Machine composto da:

* Un divisore di frequenza
* Un contatore
* Una macchina a stati finiti
* Un convertitore da BCD a 7SEG
* 2 convertitori PWM

Sono state eseguite diverse simulazioni in tutti i casi di studio analizzati e gli output ottenuti sono in linea con il corretto funzionamento del controllore.