**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.

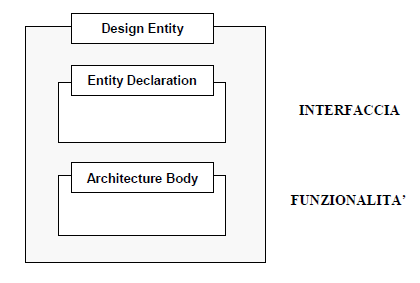
* Behavioural: E’ il massimo livello di astrazione che il VHDL consente. Esso è in grado di descrivere la funzionalità di un circuito mediante uno o più algoritmi. In questo caso, né la struttura, né le singole trasformazioni che i dati subiscono sono esplicite. In particolare, non è esplicito come le 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

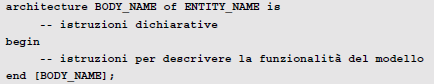
funzionalità del modulo.

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.



**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:

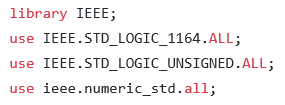
* 4 diverse modalità di funzionamento (full power, eco ...)
* La possibilità di attivare l’asciugatura o meno
* Un'interfaccia utente (led, 7 seg)
* Diversa durata per ogni stato di funzionamento

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 cyle 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

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 ed std\_ulogic\_vector, sono stati anche inclusi i package std\_logic\_unsiged e quello numeric\_std.



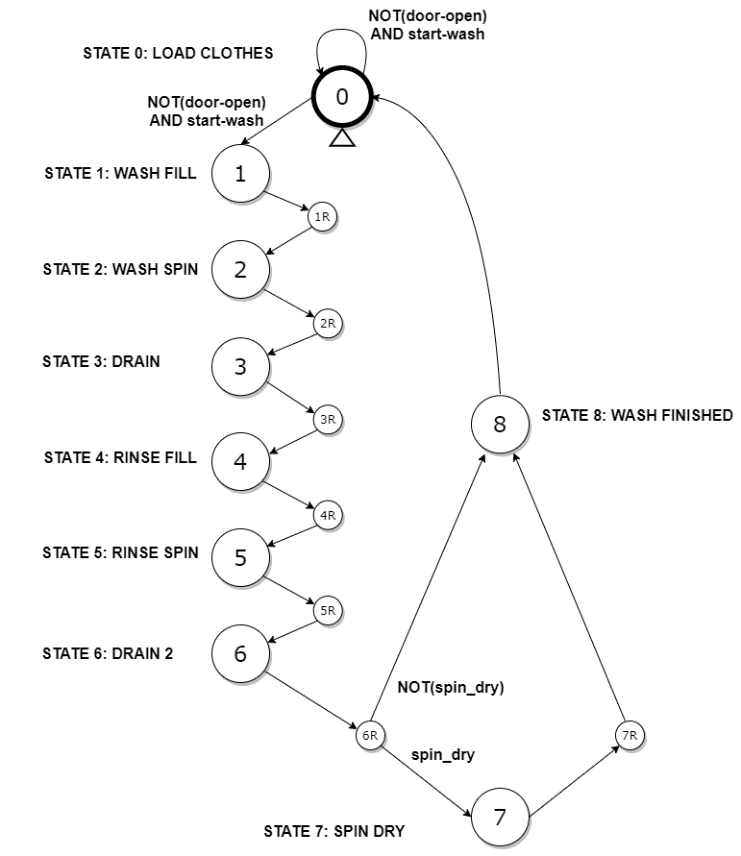
FINITE STATE MACHINE

Per l’implementazione della macchina a stati finiti (FSM) è stata creata una entity con tutti i segnali 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 |
| clk | In | std\_logic | Clock |
| 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.

Successivamente viene definita una architecture utilizzando un type definito custom con tutti gli stati della FSM, in particolare sono presenti 9 stati principali e 7 “stati di reset” grazie a cui viene resettato il counter quando vi è un cambio di stato. 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. 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 primo utlizzo di un processo potrebbe quindi essere quello di forzare una interpretazione sequenziale del codice VHDL del suo corpo.

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. I processi creati sono:

* Reset\_auto: Utile per il 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”

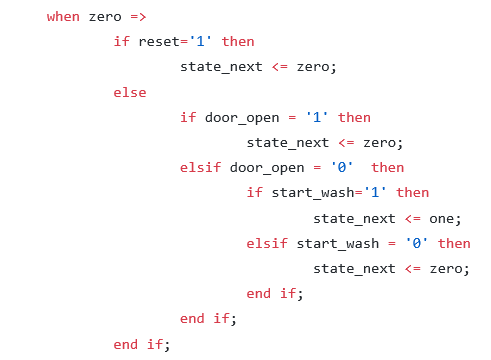
Modalità Eco (01): il conteggio del timer arriva fino a “0011”

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

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

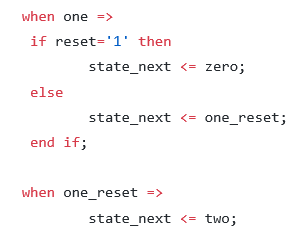
* Next\_state\_logic: Descrive la sequenza logica della FSM

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.

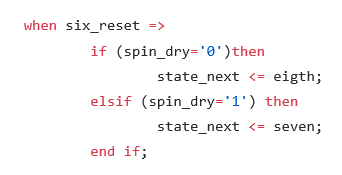


La FSM passa allo stato WASH SPIN (1) solo se start\_wash=1 e door open=0.

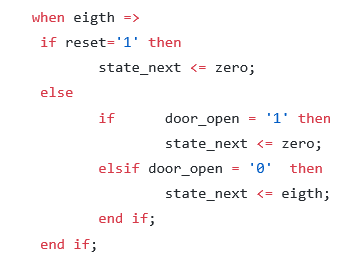
Per gli altri stati la FSM passa dallo stato corrente a quello successivo senza condizioni, passando per gli stati di reset.



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.



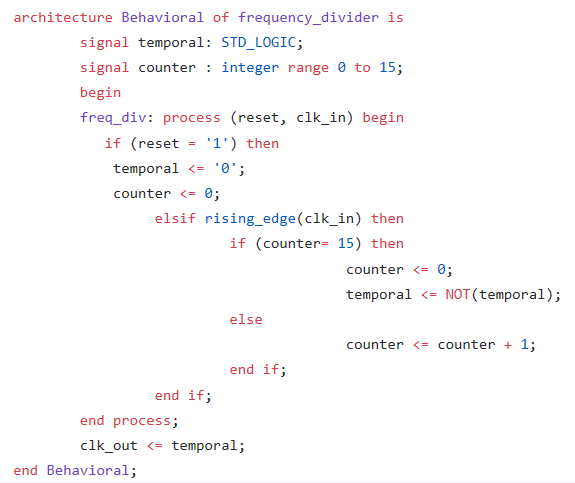
* Output\_logic: Descrive gli outputs del processo. Nella tabella della verità mostrata vengono indicati gli output per la modalità Normal

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FUNCTION | STATE CURRET | STATE NEXT | DOOR OPEN | START WASH | SPIN DRY | DOOR LOCK | WATER PUMP | SOAP | ROTATE DRUM | TEMPERATURE | DRAIN |
| RESET | 1/2/3/  4/5/6/7 | 1R/2R/3R/  4R/5R/6R/7R | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 0 |
| LOAD CLOTHES | 0 | 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 | 1 | X | 1 | 0 | 0 | 0 | 0 | 00 | 00 | 0 |
| WASH FILL | 1R | 2 | X | X | 0 | 1 | 1 | 1 | 00 | 11 | 0 |
| WASH SPIN | 2R | 3 | X | X | 0 | 1 | 0 | 0 | 11 | 10 | 0 |
| DRAIN | 3R | 4 | X | X | 0 | 1 | 0 | 0 | 00 | 00 | 1 |
| RINSE FILL | 4R | 5 | X | X | 0 | 1 | 1 | 0 | 00 | 00 | 0 |
| RINSE SPIN | 5R | 6 | X | X | 0 | 1 | 0 | 0 | 11 | 00 | 0 |
| DRAIN 2 | 6 R | 8 | X | X | 0 | 1 | 0 | 0 | 00 | 00 | 1 |
|  | 6 R | 7 | X | X | 1 | 1 | 0 | 0 | 00 | 00 | 1 |
| SPIN DRY | 7R | 8 | X | X | 1 | 1 | 0 | 0 | 11 | 00 | 1 |
| WASH FINISHED | 8 | 0 | 1 | X | 0 | 0 | 0 | 0 | 00 | 00 | 0 |
|  | 8 | 8 | 0 | X | 0 | 0 | 0 | 0 | 00 | 00 | 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 si resetta, altrimenti aumenta di 1 il conteggio. Il clk\_out in uscita è pari al valore della variabile temporal (??).



TIMER

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

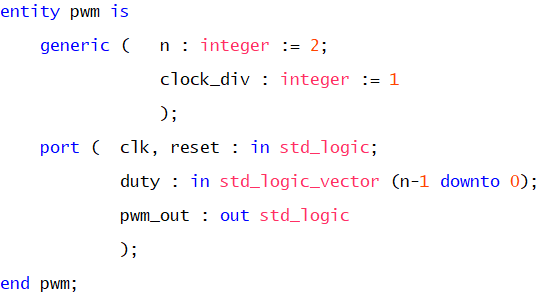
In particolare, nell’architecture vengono definiti due segnali counter\_fast e counter\_slow AGGIUNGI



Il timer viene aggiornato quando anche counter\_slow viene incrementato, ovvero quando counter\_fast arriva a “1111”.

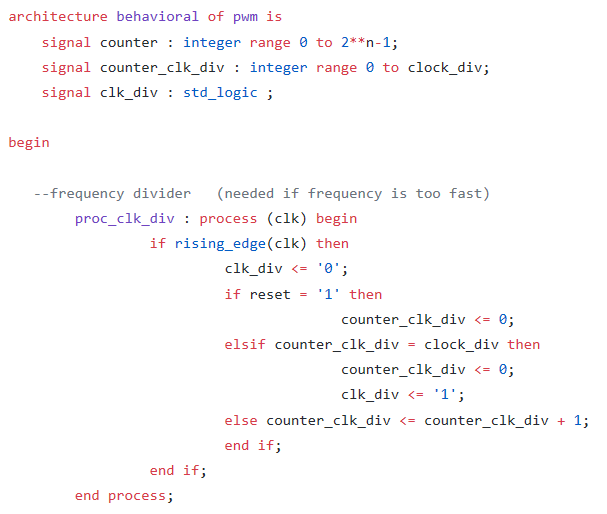
PWM

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.

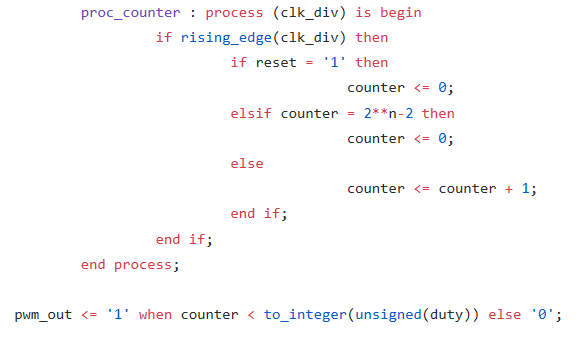


Il driver PWM che è stato implementato è costituito da 2 processi:

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

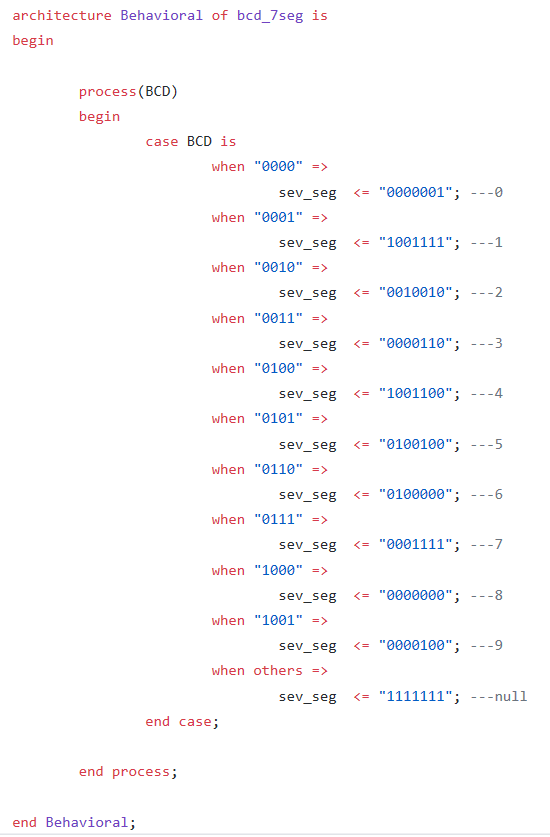


* Un contatore a n-bit, se counter < duty allora in uscita il PWM=1, altrimenti PWM=0.



BCD TO 7SEG

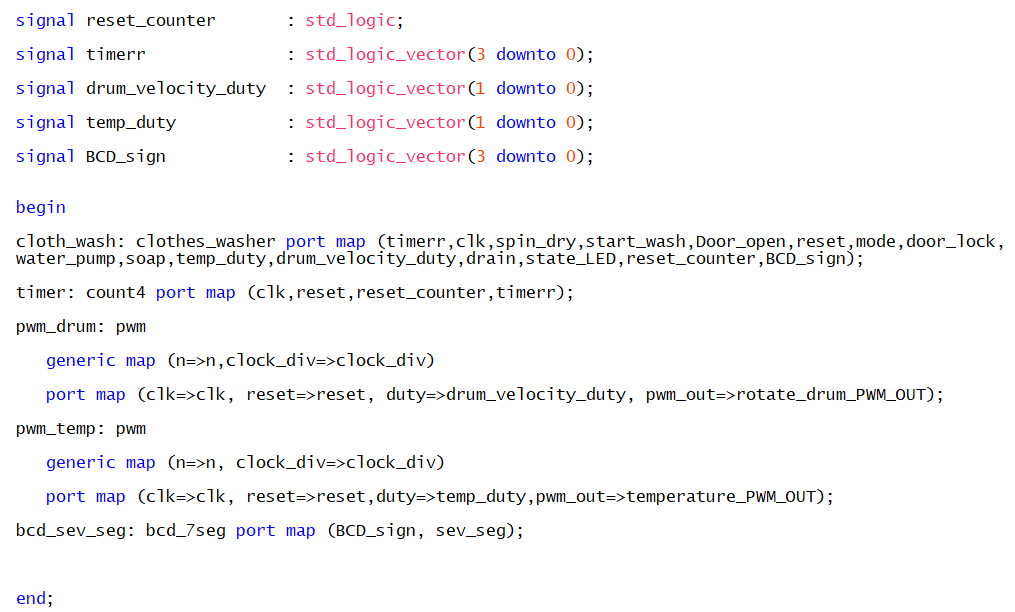
Il decodificatore da BCD a 7 segmenti converte lo stato corrente della FSM da 4 bit a 7 bit, questo può essere visualizzato mediante un display a 7 segmenti.



TOP

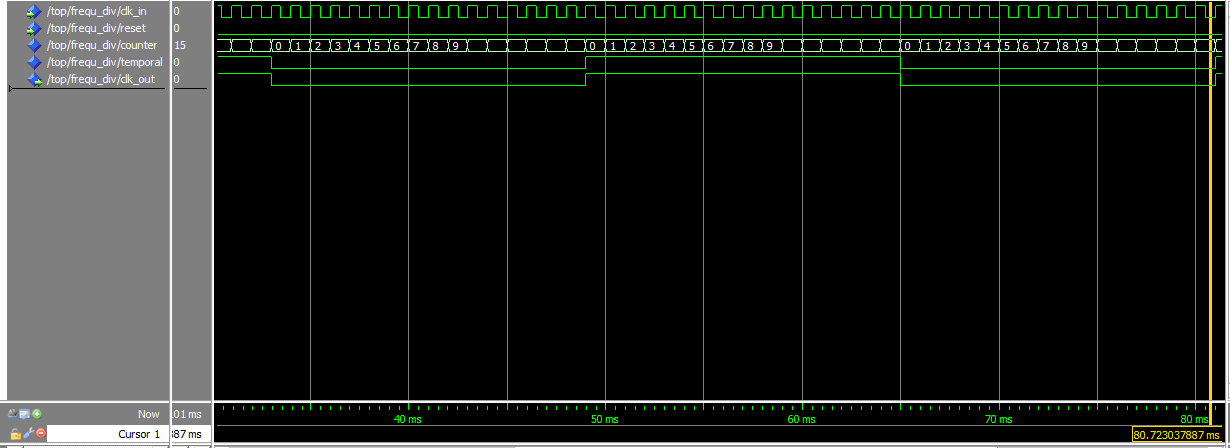
Viene definito come top hierarchy, viene utilizzato per unire tutti i diversi componenti sviluppati nel progetto.





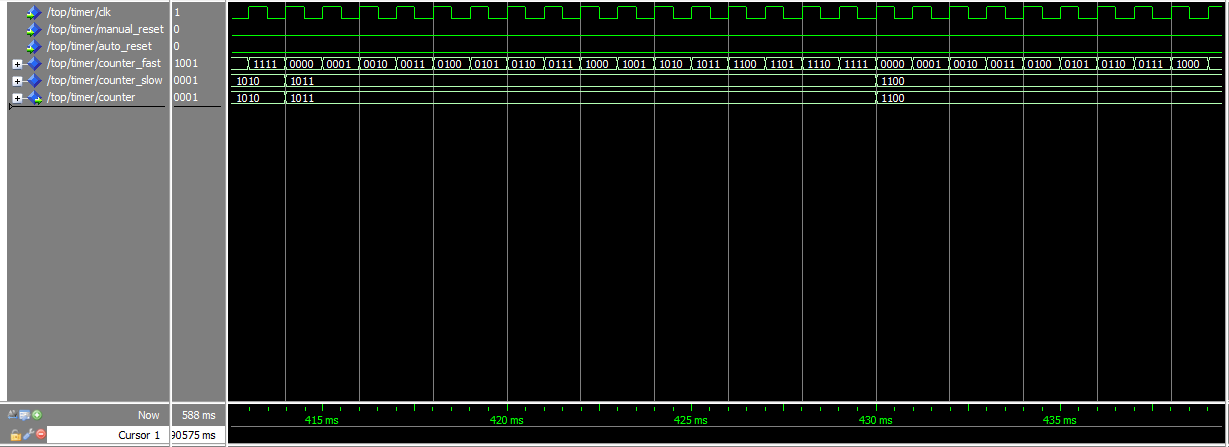
SIMULAZIONI- FREQUENCY DIVIDER

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

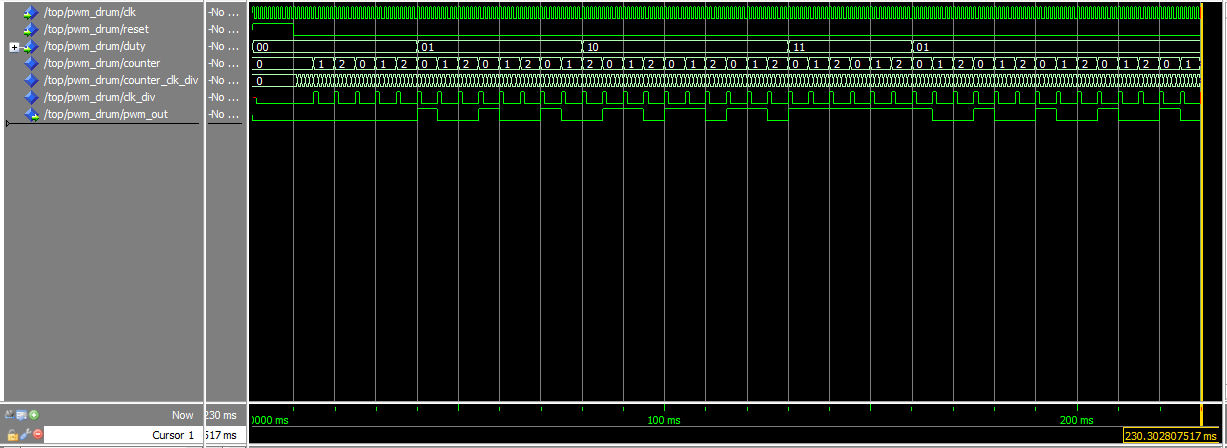
Quando counter\_fast raggiunge "1111", il conteggio counter\_slow viene incrementato di 1. Il timer in uscita viene aggiornato quando si aggiorna counter\_slow.



SIMULAZIONI-PWM

E’ 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 conteggio del counter, 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 conteggi del counter, 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, ovvero il PWM è sempre alto



SIMULAZIONI- PROCESSO COMPLETO

DA COMMENTARE

