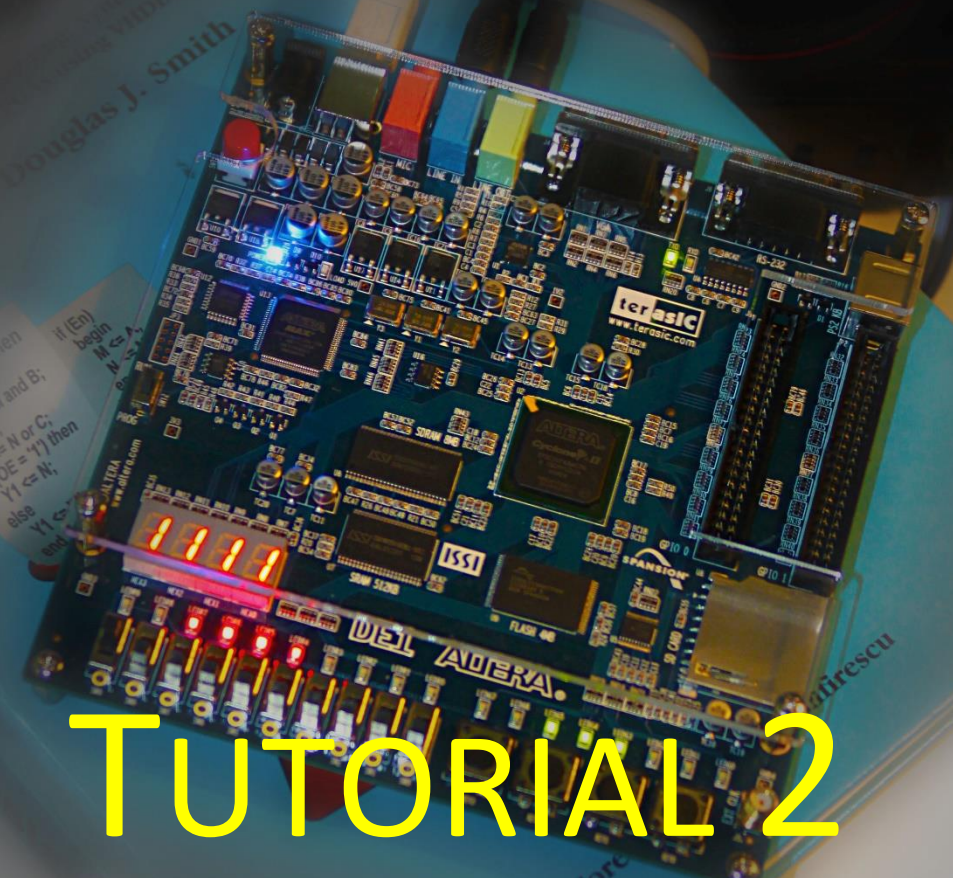
FPGA Tutorial on DE1 Board



TUTORIAL 2

Using Quartus II V13.0

Marsi 2014 - University of TRIESTE

Tutorial FPGA n.2

Realizzazione di un circuito logico gerarchico su DE1

Descrizione: Si realizzi un semplice cronometro digitale su DE1 con possibilità di start, stop, reset tramite pulsanti e visualizzazione del tempo sul dislay LED a 7 segmenti.

Scopo: Familiarizzare con sistemi digitali più complessi ed apprendere l'uso di strumenti di progettazione e di debugging più evoluti.

Apprendimento previsto:

- Sviluppo di un sistema gerarchico
- Utilizzo dei BUS
- Progetto di macchine a stati finiti
- Analisi delle criticità temporali di un circuito
- Impostazione di vincoli temporali

Procedimento:

Si inizi un nuovo progetto per `Ciclone II - EP2C20F484C7N`

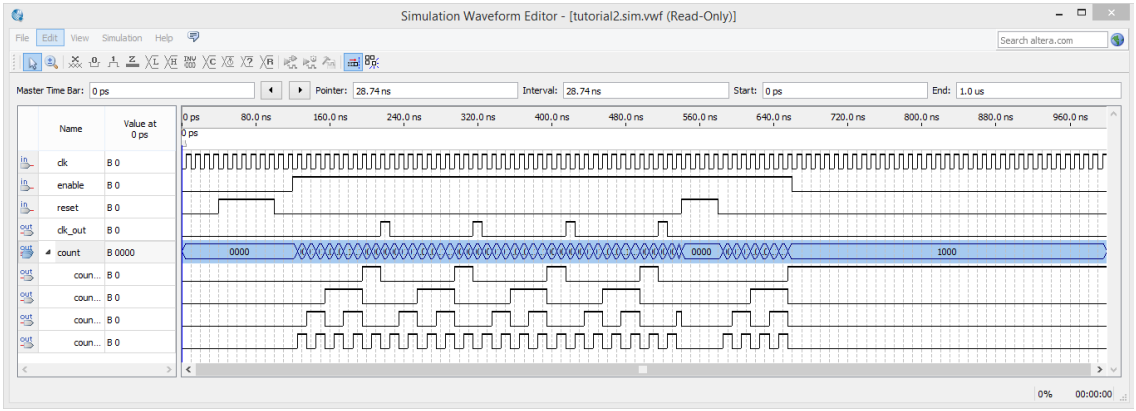
Realizzazione gerarchica Verilog- Schematico

Si predisponga un modulo verilog per la realizzazione di un contatore sincrono mod10 (0…9) comprensivo di segnali di `enable`, `reset` e `riporto`.

```
module count10
(
        input clk, enable, reset,
        output reg clk_out,
        output reg [3:0] count
);

        always @ (posedge clk or posedge reset)
        begin
                if (reset)
                        count <= 0;
                else if (enable == 1'b1)
                begin
                        if (count < 9)
                            begin
                                count <= count + 1;
                                clk_out <= 1'b0;
                            end
                        else
                            begin
                              count <= 0;
                              clk_out <= 1'b1;
                            end
                end
        end

endmodule
```

- Si salvi il file con lo stesso nome del modulo (es. count10.v)
- Lo si predisponga come "top level entity"

- Si definisca un opportuno file di stimoli
- Si simuli a livello funzionale
- Si verifichi il corretto funzionamento



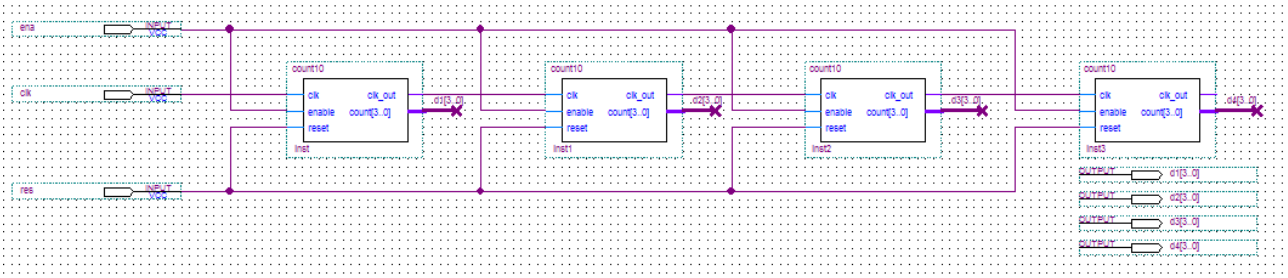
Si noti inoltre che il modulo appena realizzato oltre a generare i quattro bit di conteggio genera un ulteriore segnale (clk_out) che avrà una periodo 10 volte superiore al clock in ingresso ed un duty cycle di 0.1. Questo segnale, utilizzato in seguito per alimentare contatori per le cifre più significativa (decine, centinaia, migliaia) sarà in seguito oggetto di un'approfondita analisi temporale.

Si predisponga il modulo appena realizzato per essere impiegato a livello gerarchico superiore in uno schematic editor:

Nella finestra del "project Navigator"

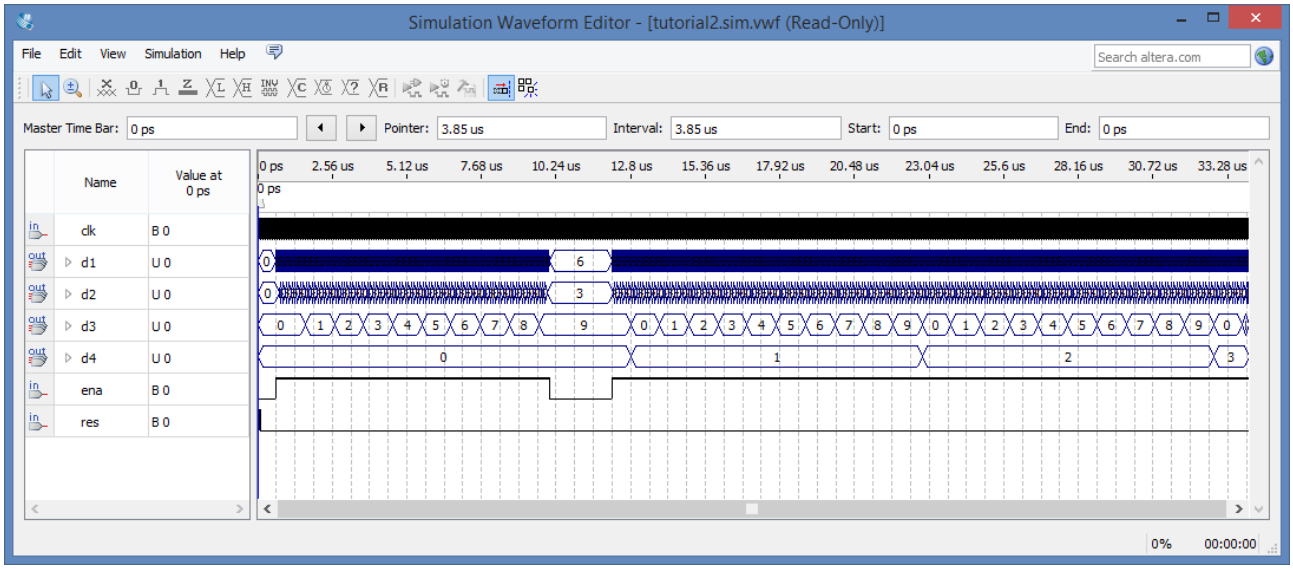
```
Right click sul file > Create symbol file for current file
```

- Si crei ora un nuovo schematico
- Tra le porte logiche disponibili vi sarà anche un blocco rappresentante il modulo verilog appena realizzato
- Realizzare uno schematico per un contatore in base 10 a 4 cifre (0 … 9999)
- Si utilizzino dei bus per rappresentare le 4 cifre.
- Per trasferire i valori sui pin d'uscita non è necessario tracciare un collegamento esplicito, ma basta che il pin d'uscita e la "net" ad esso associata abbiano lo stesso nome
- Si noti che nello schematico il nome convenzionale per i bus è: `nome_bus[n1 ..n0]` ove `nome_bus[n1]` rappresenta il bit più significativo, mentre `nome_bus[n0]` è il meno significativo (es: `D[3..0], a[9..2]`) notare che `D[3..0]` e `D[0..3]` sono entrambe rappresentazioni lecite, ma mentre nel primo caso il bit più significativo è `D[3]`, nel secondo è `D[0]`



- Si salvi lo schematico

- Lo si predisponga come “top level entity”
- Si definisca un nuovo opportuno file di stimoli
- Si simuli a livello funzionale
- Si verifichi il corretto funzionamento



Realizzazione gerarchica Verilog- Verilog

Predisporre ora un modulo Verilog per la conversione da binario a display a sette segmenti (realizzabile tramite una semplice LUT) con uscita sincronizzata su un opportuno segnale di clock.

```
module SEG7_LUT        (clk,oSEG,iDIG);

input clk;
input   [3:0]   iDIG;
output  [6:0]   oSEG;
reg             [6:0]   oSEG;

always @(negedge clk)
        case(iDIG)
                4'h1: oSEG = 7'b1111001;      // ---t----
                4'h2: oSEG = 7'b0100100;      // |      |
                4'h3: oSEG = 7'b0110000;      // lt    rt
                4'h4: oSEG = 7'b0011001;      // |      |
                4'h5: oSEG = 7'b0010010;      // ---m----
                4'h6: oSEG = 7'b0000010;      // |      |
                4'h7: oSEG = 7'b1111000;      // lb    rb
                4'h8: oSEG = 7'b0000000;      // |      |
                4'h9: oSEG = 7'b0011000;      // ---b----
                4'ha: oSEG = 7'b0001000;
                4'hb: oSEG = 7'b0000011;
                4'hc: oSEG = 7'b1000110;
                4'hd: oSEG = 7'b0100001;
                4'he: oSEG = 7'b0000110;
                4'hf: oSEG = 7'b0001110;
                4'h0: oSEG = 7'b1000000;
                endcase
end
endmodule
```

Si noti che in Verilog i BUS prendono il nome di `nome[n1:n0]` ove `nome[n1]` è il bit più significativo mentre `nome[n0]` è il meno significativo. Si noti ancora che il modulo viene attivato sul fronte negativo del clock. La rappresentazione appena adottata è una rappresentazione "comportamentale" del modulo in oggetto, essa infatti ne descrive il funzionamento ma NON fornisce dettagli sull'architettura (in fase di sintesi si potrebbe utilizzare una LUT, ma volendo anche una realizzazione attraverso porte logiche potrebbe servire allo scopo).

Predisporre ora un modulo Verilog per l'integrazione di 4 dei moduli precedenti in una sola unità. Questa a differenza della precedente è una rappresentazione "strutturale" [1]. Essa infatti evidenzia l'architettura del modulo. Vi sono infatti istanziati 4 blocchi identici (denominati rispettivamente u0, u1, u2,u3) ai quali sono collegati opportunamente i vari segnali di ingresso e di uscita.

```
module SEG7_LUT_4 (oSEG0,oSEG1,oSEG2,oSEG3,iDIG,clk );

input clk;
input  [15:0] iDIG;
output [6:0]  oSEG0,oSEG1,oSEG2,oSEG3;

SEG7_LUT      u0     (clk,oSEG0,iDIG[3:0]);
SEG7_LUT      u1     (clk,oSEG1,iDIG[7:4]);
SEG7_LUT      u2     (clk,oSEG2,iDIG[11:8]);
SEG7_LUT      u3     (clk,oSEG3,iDIG[15:12]);

endmodule
```

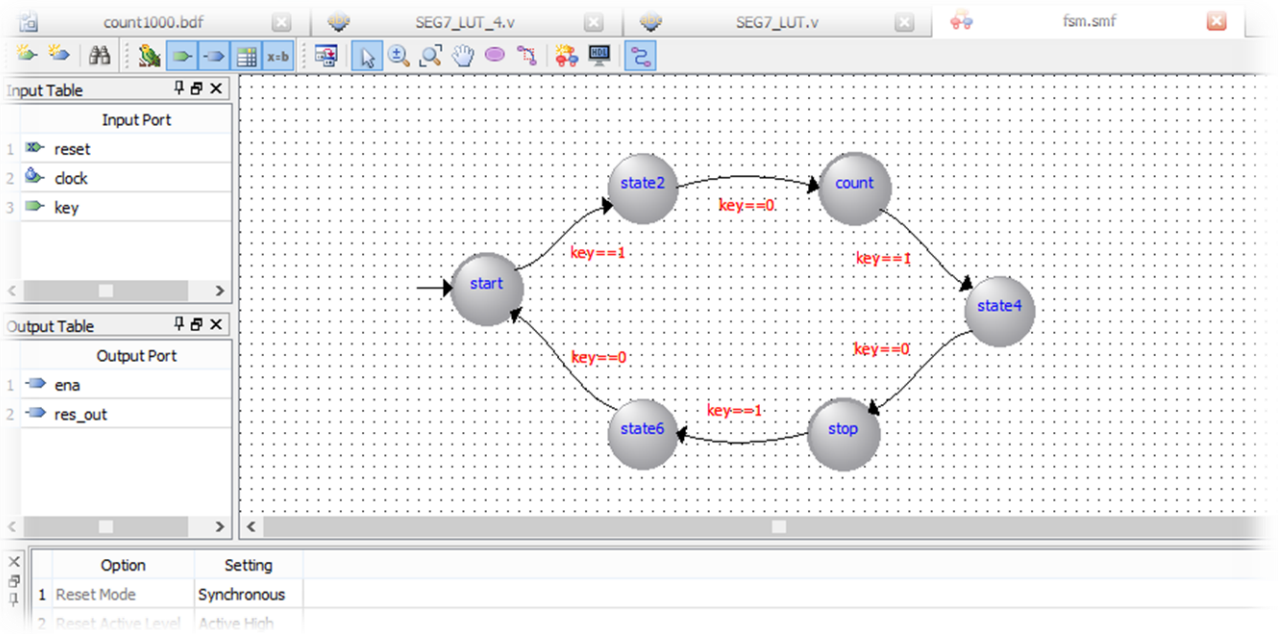
Generare il simbolo corrispondente a questo secondo modulo.

Definizione di una FSM (finite state machine)[2-5]

Si voglia realizzare ora una macchina a stati finiti con un solo pulsante in ingresso che generi gli opportuni segnali per il contatore di `start – stop – reset` secondo questa specifica sequenza.

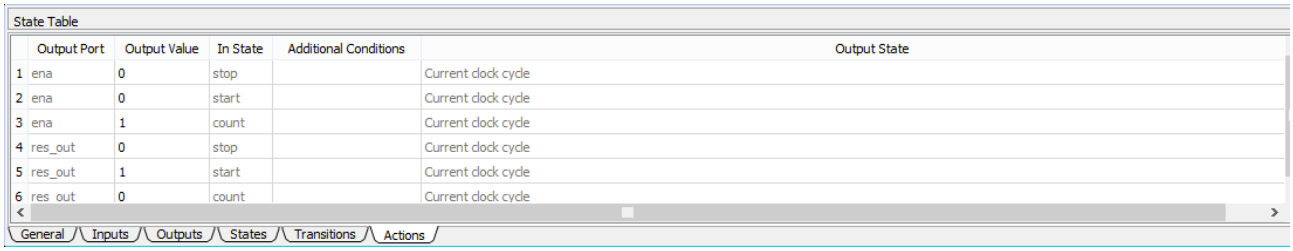
```
File > New
State machine file
```

- Aggiungere alle porte di default (clk, reset) le porte "key" come ingresso e le porte "ena" e "res_out" come uscita (Cliccare col tasto destro all'interno delle schede "Input Table" ed "Output Table"
- Tracciare quindi un grafo degli stati simile a quello riportato in figura (da esso si evince che vi sono tre stati principali (start, count, stop), ma la transizione dall'uno all'altro avviene attraverso una duplice azione: key passa prima al valore alto e successivamente ritorna al valore basso, pertanto per rendere stabile ogni stato si sono dovuti aggiungere gli stati intermedi state2, state4 e state6. Inoltre negli stati principali verranno definiti i valori da assegnare ai segnali d'uscita.



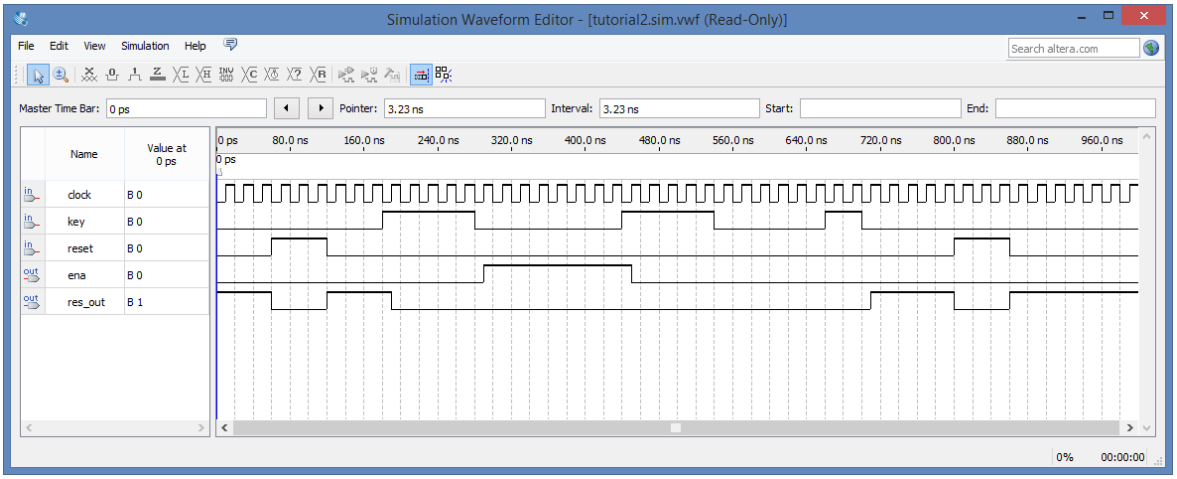
- Definire le transizioni tra i vari stati (doppio click sul ramo)
- Definire le uscite dei vari stati (doppio click sullo stato)
    - Attivare l'uscita "ena" nello stato "count" e disattivarla nello stato "stop"
    - Attivare la linea "res_out" allo stato start e disattivarla negli altri

Nota: le varie schede nella finestra "State Table" riassumono il funzionamento completo della macchina.



- Generare il file Verilog HDL che descriva il grafo usando l'opportuno pulsante
- Verificare la correttezza del file sia tramite analisi sintattica che mediante simulazione funzionale Prestare molta attenzione al risultato perché non sempre coincide col risultato sperato (ad esempio si può notare che a seguito di un reset esterno entrambe le uscite "ena" e "res_out" assumono il valore basso … se ciò non dovesse rispondere alle nostre specifiche si può eventualmente modificare a mano il file verilog HDL appena generato.

    - Importare il file Verilog rappresentante la macchina a stati finiti nel proprio progetto
    - Settare tale file come Top-level entity
    - Generare un opportuno file di stimoli
    - Verificare il corretto funzionamento della macchina a stati finiti appena realizzata



Notare in particolare che sebbene il funzionamento risulti corretto negli stati principali, non avendo definito le uscite negli stati intermedi, in esse queste ultime assumono funzionamenti diversi (ad esempio l'attivazione del conteggio avviene solo quando il tasto viene rilasciato, mentre lo stop del conteggio avviene nella fase di pressione del tasto e l'azzeramento del conteggio nuovamente quando il tasto sarà rilasciato). Ove il funzionamento non risultasse quello desiderato si può tornare a modificare la macchina a stati finiti o nella sua versione grafica oppure agendo direttamente sul file HDL agendo sugli stati dove le uscite non sono state definite.

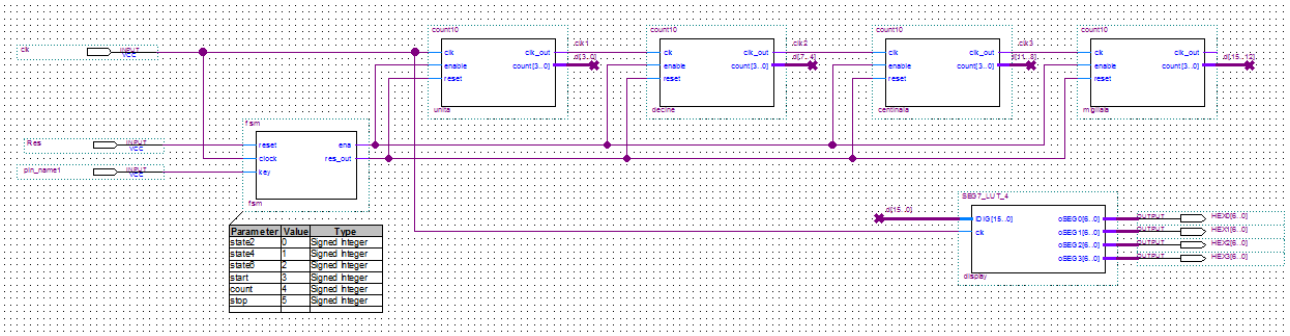
- Generare il simbolo grafico associato alla macchina a stati finiti (FSM) appena realizzata

Definizione del sistema completo

A questo punto abbiamo definito tutti i blocchi che realizzano il nostro sistema:

- Una macchina a stati finiti che con un solo ingresso (key) genera ciclicamente i segnali di attivazione del conteggio, stop del conteggio e azzeramento del medesimo.
- Un contatore in codice BCD a 4 cifre che incrementa il suo valore ad ogni fronte positivo del clock
- Un visualizzatore delle cifre per display a sette segmenti che "congela" la cifra da visualizzare ad ogni fronte negativo del clock.

Si realizzi pertanto uno schematico che riunisca tutti i blocchi finora sviluppati con gli opportuni collegamenti.



Si consiglia, cliccando col tasto destro sui vari blocchi e scegliendo “properties” di assegnare a ciascun blocco un nome mnemonico che aiuti in futuro a distinguere i segnali che fanno riferimento al medesimo. Nell’immagine riportata si son scelti ad esempio i nomi (fsm, display, unita,decine,centinaia,migliaia)

Utilizzo dell’ analizzatore “Time Quest Timing Analyzer” [6,7,8]

Sebbene evidentemente il funzionamento di questo dispositivo, utilizzerà successivamente una frequenza di lavoro di qualche decina di Hertz (onde rendere visibile il conteggio sui display), per studiarne il funzionamento temporale in condizioni più critiche supponiamo di volerlo far funzionare a 100 MHz. Si voglia pertanto verificare se esso possa funzionare correttamente a questa frequenza [9-10].

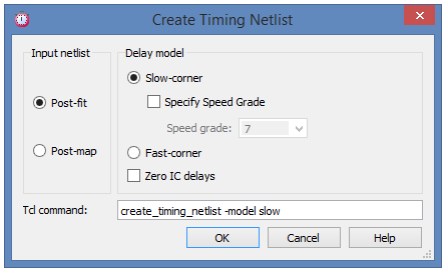
- Si definisca lo schematico completo appena realizzato come “Top Level Entity” `(ctr-Shift-J)`
- Si esegua una compilazione completa `(ctr-L)`
- Si attivi il tool “Time Quest Timing Analyzer”

```
Tools > Time Quest Timing Analyzer
```

Si crei ora una nuova Netlist

```
Netlist > Create Timing Netlist
```

Si faccia riferimento al modello post-fit in versione pessimistica (Slow-corner)




```
> OK
```

Si definiscano ora le specifiche richieste dal clock del sistema

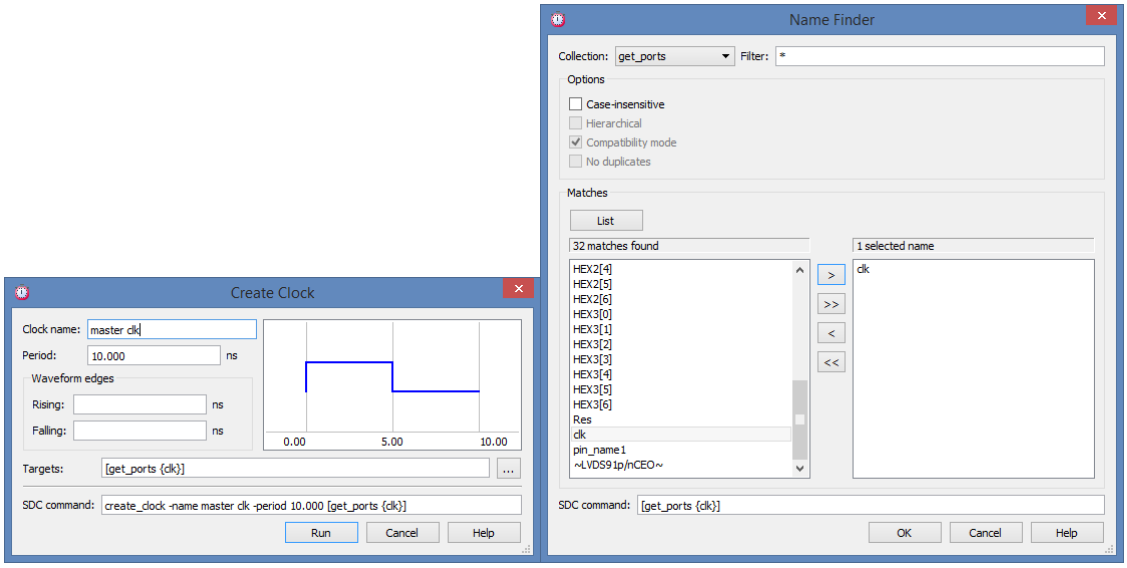
```
Constraints > Create Clock
```

Da notare che la finestra interattiva che si apre serve sostanzialmente solo a definire le specifiche del comando “tcl” e che in futuro, con un po’ di pratica, esso potrà essere scritto direttamente senza ricorrere all’interfaccia grafica.

Si fissino quindi le specifiche. In particolare:

- Si dia un nome mnemonico al clock (può essere il nome dello stesso segnale di clock)
- Si definisca il periodo in 10 ns
- Si associ questo vincolo al segnale presente nel piedino d’ingresso denominato “clk”
    - Per quest’ultimo passo si faccia click sul tasto “…”
    - Nella finestra che appare, sfruttando eventalmente opportuni filtri
        - Si scelga “Get_ports”
        - List

- Si aggiunga nella lista di selezione il segnale clk



> OK
> RUN

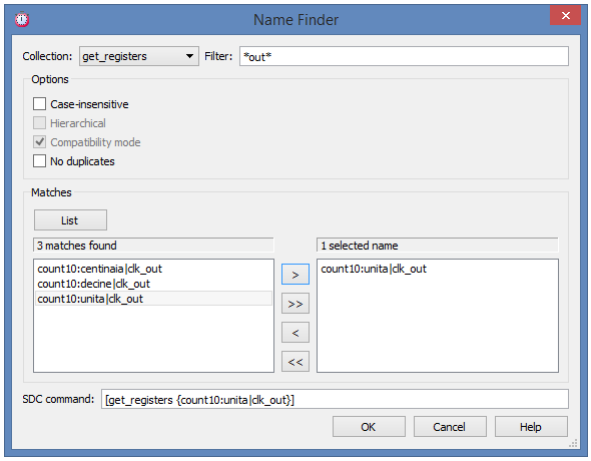
Tutta la procedura all'atto pratico si traduce nella generazione del comando tcl:

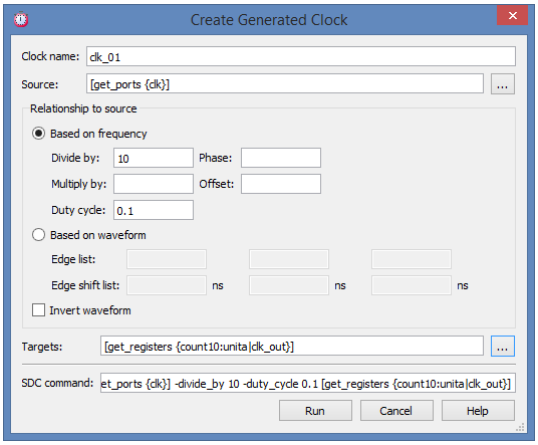
```
tcl> create_clock -name clk -period 10.000 [get_ports {clk}]
```

A questo punto siccome i diversi clock che pilotano i contatori dedicati alle cifre più significative sono generati in cascata ciascuno dal precedente (con un periodo 10 volte superiore ed un duty cycle pari a 0,1), bisogna passare questa informazione al nostro analizzatore.

```
Constraints > Create Generated Clock
```

Nella finestra che segue si va pertanto a specificare che il clock che entra nel contatore delle decine è generato attraverso il clock del contatore delle unità, che esso è 10x più lento e con un duty cycle di 0,1. Per fare questo bisogna trovare nella liste (…), attraverso eventuali filtri, i segnali coinvolti (sia come **source** che come **target**). Per questo scopo è stato utile dare un nome riconoscibile alle varie istanziazioni dei blocchi nello schematico





Corrispondente al comando tcl:

```
tcl> create_generated_clock -name clk_01 -source [get_ports {clk}] -
divide_by 10 -duty_cycle 0.1 [get_registers {count10:unita|clk_out}]
```

e proseguire analogamente anche per gli altri 2 clock generati rispettivamente dal blocco che effettua il conteggio delle decine e delle centinaia tenendo conto che tali clock vengono generati dai clock precedenti.

```
tcl> create_generated_clock -name clk_001 -source [get_registers
{count10:unita|clk_out}] -divide_by 10 -duty_cycle 0.1 [get_registers
{count10:decine|clk_out}]
```

```
tcl> create_generated_clock -name clk_0001 -source [get_registers
{count10:decine|clk_out}] -divide_by 10 -duty_cycle 0.1 [get_registers
{count10:centinaia|clk_out}]
```

Aggiornare la Netlist

```
Netlist > Update Timing Netlist
```

E generare il report relative alle temporizzazioni

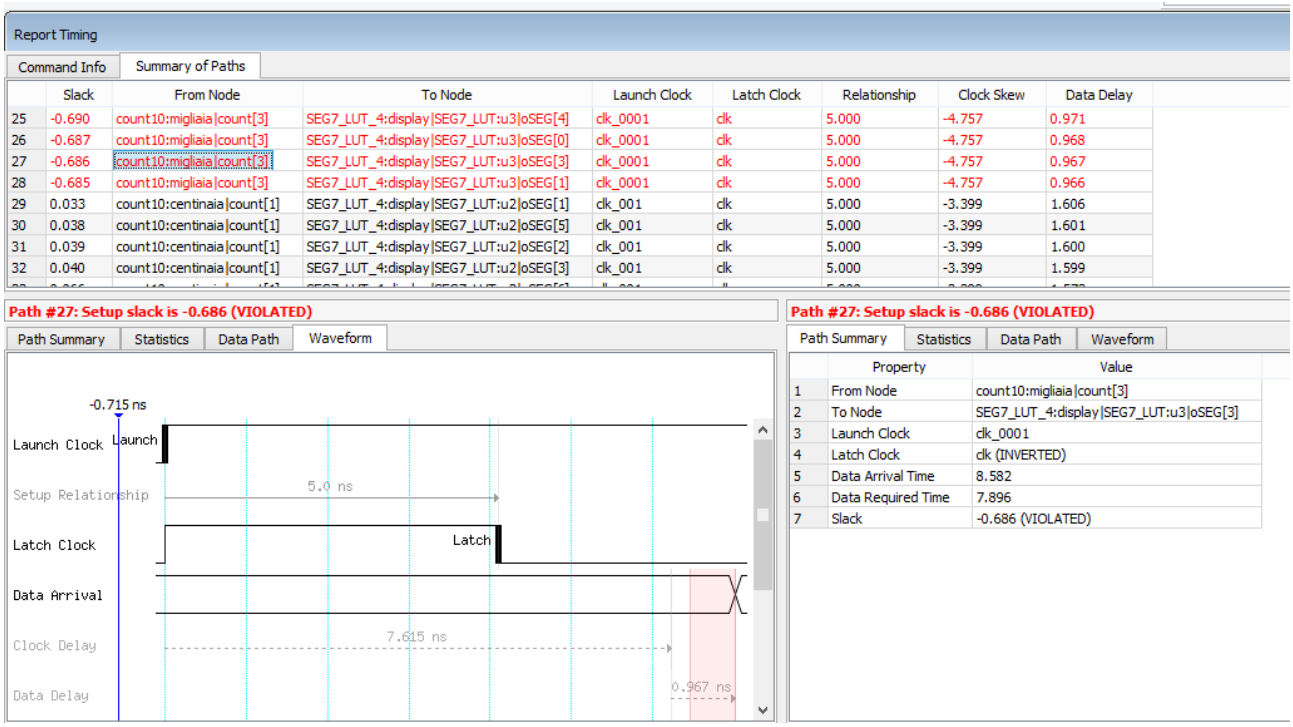
```
Reports > Custom Report > Report Timing
```

Scegliere

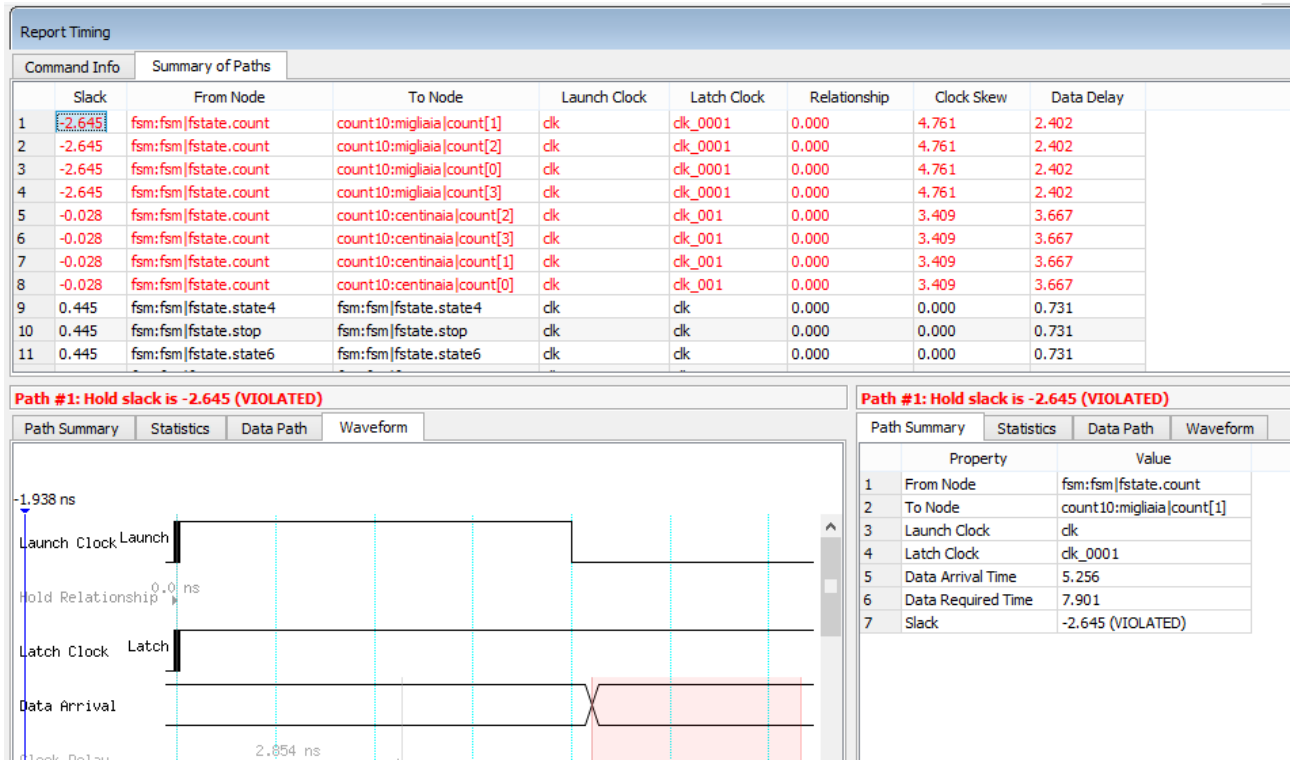
- Analysis Type: Setup
- Number of Path: 100
- Click on "**Report Timing**"

La finestra che appare illustra i dettagli dei vari percorsi ordinati da più lento verso il più veloce e se questi rispettano i vincoli (seppur minimi che abbiamo fornito). Si nota in particolare che alcuni di questi risultano scritti in rosso, mentre altri sono in nero: i primi sono quelli che NON rispettano i vincoli, infatti si può notare nella scheda waveform, relativa a qualcuno di questi percorsi come il segnale arrivi in ritardo rispetto il fronte di clock (di discesa) che dovrebbe campionarlo). In verità il sistema tiene conto di tutti i ritardi coinvolti nel percorso, sia quelli dei dati che quelli relativi ad entrambi i segnali di clock (quello che

“lancia” il conteggio e quello che “congela” il risultato sul display). I risultati possono essere numericamente diversi e si differenziano in base a come è stato realizzato il piazzamento della logica all’interno dell’FPGA.



Lanciando altre tipologie di analisi (ad esempio Hold) si può notare come vi siano anche altre criticità inerenti la sincronizzazione tra la macchina a stati finiti ed i contatori di decine e centinaia



In questo caso è il segnale che esce dalla macchina a stati finiti (ena) a rischiare di commutare prima di essere stato acquisito correttamente da contatori di centinaia e migliaia.

Una comoda utilità può essere quella di localizzare il path o gli elementi coinvolti in tali malfunzionamenti nelle varie viste del circuito. Per fare ciò basta cliccare col tasto destro sull'elemento coinvolto e scegliere

- **Locate** oppure **Locate Path**
- E scegliere successivamente la vista nella quale localizzarlo

Soffermarsi in particolare sulle varie viste del medesimo circuito:

- Design File – vista relativa ai file sorgenti
- RTL Viewer – vista a livello RTL (il punto di partenza per la sintesi)
- Tecnology Map Viewer – Vista in base ad uno schematico basato sulla mappatura su FPGA
- Chip Planner – Vista in base all'occupazione di risorse su FPGA
- Pin Planner – Utile per modificare i pin di I/O per migliorare le prestazioni
- Assignment Editor – Utile per assegnare, modificare I vincoli realizzativi

Bisogna notare inoltre che le informazioni sulle diverse frequenze di clock che sono state appena definite hanno una validità duplice: da un lato servono a verificare se il sistema rispetta le specifiche, ma da un altro canto, esse possono essere passate Quartus per cercare di "forzare" il sistema in fase di sintesi, mapping e routing a rispettare tali vincoli.
Per fare ciò

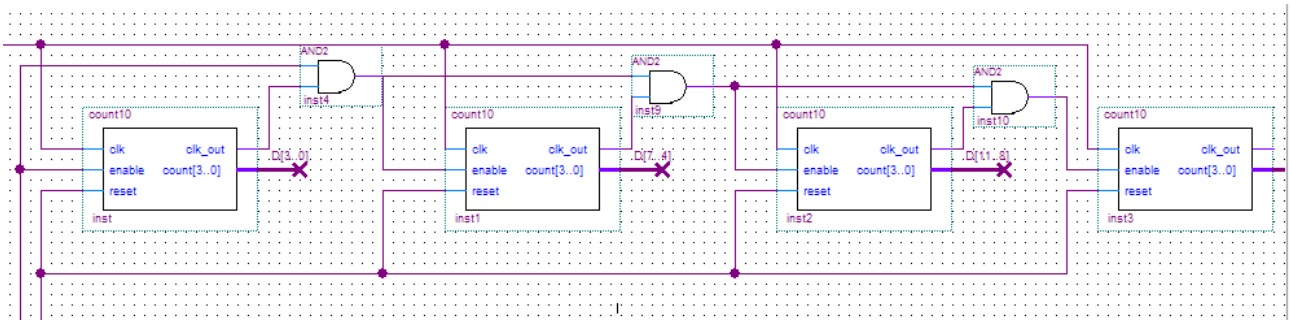
- Si salvi il file di vincoli (Doppio click nella finestra Tasks su: "Write SDC File".)
- Importare il fale appena salvato all'interno dei file del progetto di Quartus

Quartus utilizzerà i vincoli imposti per cercare una realizzazione che li soddisfi ed eventualmente indicherà nei reports se tali vincoli non dovessero essere rispettati.

Considerazioni

I malfunzionamenti evidenziati sono sostanzialmente legati all'architettura stessa del circuito, infatti non è mai una buona idea utilizzare circuiti che rigenerino un segnale di sincronismo attraverso funzioni logiche, perché così facendo si creano disallineamenti nel clock (clock skew) [9,10] e sebbene attraverso vincoli costruttivi si potrebbe limitare almeno in parte il malfunzionamento, la strategia migliore dovrebbe essere quella di ridisegnare il sistema facendo in modo che ogni contatore sia alimentato dal medesimo clock e che il conteggio venga eventualmente fermato o riattivato attraverso il segnale di enable.

Una soluzione al problema esposto potrebbe essere la seguente:



Questa architettura non dovrebbe più creare problemi dal punto di vista del disallineamento del clock in quanto tutti i blocchi risultano sincronizzati sullo stesso segnale ed inoltre non vi è alcun elemento di ritardo nella propagazione del clock.

Per contro il conteggio in questo caso risulta affetto da un altro problema, ovvero la commutazione del contatore a livello più altro avviene in ritardo di un ciclo di clock rispetto quello più basso, pertanto il passaggio da 09 a 10 in realtà avviene erroneamente in due cicli 09 – 00 – 10. Oltre al fatto che il segnale di clk_out dovrebbe venir rivisto in modo che la sua durata sia esattamente uguale ad un singolo periodo del clock di ingresso. Ovviare a questi inconvenienti richiede di modificare più profondamente il progetto.

Modifica dei Vincoli realizzativi.

Si deve però ora, all'atto pratico, tener presente che il più basso valore del clock della scheda DE1 è a 24 MHz. Qualora si utilizzasse direttamente questo clock per effettuare il conteggio esso sarebbe talmente rapido che i display a 7 segmenti apparirebbero sempre illuminati. Si dovrà pertanto predisporre un ulteriore blocco utile per ridurre a valori più ragionevoli la frequenza di conteggio.

Un ulteriore vantaggio nella riduzione della frequenza di conteggio sta nel fatto che, nonostante le problematiche finora viste (relative ad una frequenza a 100MHz), Il circuito dovrebbe funzionare correttamente se la sua frequenza di lavoro venisse abbassata sufficientemente. Si può infatti verificare con lo strumento "Time Quest Timing Analyzer" che, già abbassando la frequenza di lavoro già di un fattore 10 le problematiche legate ai tempi di setup non sono più presenti.

Per fare ciò basta aprire il file di vincoli e modificare la riga

```
create_clock -name {clk} -period 10.000 -waveform {0.000 5.000} [get_ports {clk}]
```

in

```
create_clock -name {clk} -period 100.000 -waveform {0.000 50.000} [get_ports {clk}]
```

si salvi, si ricompili quindi il progetto e quindi si rilanci il "Time quest Timing analyzer"

eseguire poi di seguito

```
> Create Timing Netlist
> Read SDC File
> Upadte Timing netlist
> Report Timing
```

Si Noterà dai reports che permangono solo le problematiche legate ai tempi di Hold, ma per come si susseguono i segnali in uscita dalla macchina a stati finiti, questi effetti solo in casi rarissimi potrebbero produrre un malfunzionamento rilevabile.

Comunque, un'altra strada percorribile per migliorare anche questo aspetto è quella di imporre opportune metodologie in fase di Sintesi per far rientrare il circuito nei vincoli richiesti:

```
Assignments > Settings (ctrl-Shift-E)
Fitter Settings
(set - Optimize hold timing)
Choose item – All Path
```

Si può ora verificare (attraverso il TQTA oppure tramite i reports) che tutti i vincoli risultano rispettati.

Realizzazione fisica su scheda DE1.

Si voglia adottare una frequenza di conteggio pari a 100 Hz, così da poter impiegare il sistema per conteggiare fino i centesimi di secondo, si dovrà pertanto ridurre la frequenza del clock della scheda fino a raggiungere la frequenza di conteggio desiderata

Il riduttore di frequenza può essere così descritto in Verilog HDL:

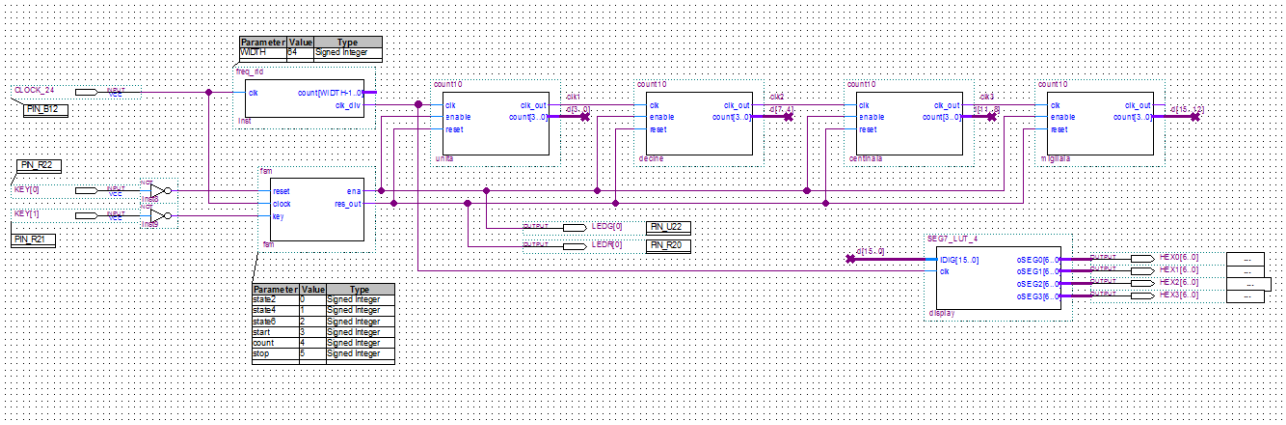
```
module freq_rid
#(parameter WIDTH=64)
(
        input clk,
        output reg [WIDTH-1:0] count,
        output reg clk_div
);

        always @ (posedge clk)
        begin
             if (count<240000)
                begin
                       count <= count + 1;
                       clk_div <=1'b0;
                       end
                 else
                    begin
                       count <=0;
                       clk_div <=1'b1;
                       end
        end
endmodule
```

Una simulazione di questo blocco sarebbe quanto mai problematica, si dovrebbe infatti estendere per 240.000 cicli di clock prima di vedere l'uscita "clk_div" commutare. Eventualmente si può simulare modificando il limite di conteggio.

- Appena realizzato anche questo blocco, generare il "symbol" corrispondente.

Giunti a questo punto si può definire il sistema completo via schematico

- Si apra un nuovo schematico
- Utilizzando i blocchi costituiti in precedenza si realizzi uno schema simile a quello riportato in figura



- Si assegni un nome opportuno ai vari bus onde garantire la correttezza dei collegamenti tra le varie unità
- Si assegnino dei nomi opportuni ai piedini di ingresso e di uscita in modo che essi siano rispecchiati da un eventuale file di vincoli
- Si importi un file di vincoli ove siano specificati nel dettaglio le posizioni degli swithes, del clock, dei display a sette segmenti e dei led (per comodità le posizioni di alcuni dispositivi di I/O sono qui di seguito elencate) [11-13] - un file con questi vincoli è peraltro disponibile sulla pagina web del corso.

```
SW[0],PIN_L22
SW[1],PIN_L21
SW[2],PIN_M22
SW[3],PIN_V12
SW[4],PIN_W12
SW[5],PIN_U12
SW[6],PIN_U11
SW[7],PIN_M2
SW[8],PIN_M1
SW[9],PIN_L2

KEY[0],PIN_R22
KEY[1],PIN_R21
KEY[2],PIN_T22
KEY[3],PIN_T21

LEDR[0],PIN_R20
LEDR[1],PIN_R19
LEDR[2],PIN_U19
LEDR[3],PIN_Y19
LEDR[4],PIN_T18
LEDR[5],PIN_V19
LEDR[6],PIN_Y18
LEDR[7],PIN_U18
LEDR[8],PIN_R18
LEDR[9],PIN_R17

LEDG[0],PIN_U22
LEDG[1],PIN_U21
LEDG[2],PIN_V22
LEDG[3],PIN_V21
LEDG[4],PIN_W22
LEDG[5],PIN_W21
LEDG[6],PIN_Y22
LEDG[7],PIN_Y21

HEX0[0],PIN_J2
HEX0[1],PIN_J1
HEX0[2],PIN_H2
HEX0[3],PIN_H1
HEX0[4],PIN_F2
HEX0[5],PIN_F1
HEX0[6],PIN_E2
HEX1[0],PIN_E1
HEX1[1],PIN_H6
HEX1[2],PIN_H5
HEX1[3],PIN_H4
HEX1[4],PIN_G3
HEX1[5],PIN_D2
HEX1[6],PIN_D1
HEX2[0],PIN_G5
HEX2[1],PIN_G6
HEX2[2],PIN_C2
HEX2[3],PIN_C1
HEX2[4],PIN_E3
HEX2[5],PIN_E4
HEX2[6],PIN_D3
HEX3[0],PIN_F4
HEX3[1],PIN_D5
HEX3[2],PIN_D6
HEX3[3],PIN_J4
HEX3[4],PIN_L8
HEX3[5],PIN_F3
HEX3[6],PIN_D4

CLOCK_27,PIN_D12
CLOCK_24,PIN_A12
CLOCK_50,PIN_L1
EXT_CLOCK,PIN_M21
```

Si noti che entrambi i pulsanti KEY[0] e KEY[1] sono di default nello stato alto, mentre vanno nello stato basso quando premuti. Di congruenza con la macchina a stati finiti realizzata, si dovranno introdurre due invertitori in serie ai rispettivi piedini.

- Si salvi lo schematico
- Si definisca quest'ultimo come Top-Level-Entity
- Si compili l'intero sistema
- Si verifichino il layout del sistema ed i reports
- Si operi il download della bitstream così generata su scheda DE1.

- Si verifichi il corretto funzionamento su scheda.

Per una corretta analisi temporale del nuovo sistema bisogna ovviamente generare un nuovo file di vincoli che rispecchi il funzionamento del circuito.

Ulteriore approfondimento sui vincoli realizzativi

Il progetto finora seguito non ha visto ancora l'imposizione di vincoli sui percorsi, fatta eccezione per la definizione delle frequenze di clock, e per la posizione dei piedini di I/O e pertanto il risultato ottenuto non ha seguito particolari strategie di ottimizzazione.

Si può notare ad esempio attraverso il TQTA – Report Paths che il ritardo di propagazione tra KEY[0] e LEDR[0] di circa 16 ns

Report Path

Command Info | Summary of Paths

	Delay	From Node	To Node
1	16.083	KEY[0]	LEDG[0]
2	13.958	KEY[0]	LEDR[0]
3	10.009	KEY[0]	count10:unita|clk_out
4	10.009	KEY[0]	count10:unita|count[2]
5	10.009	KEY[0]	count10:unita|count[3]
6	10.009	KEY[0]	count10:unita|count[0]

Path #1: Delay is 16.083

Path Summary | Statistics | Data Path

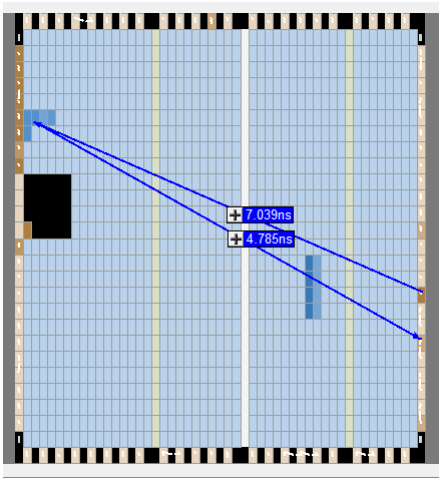
	Total	Incr	RF	Type	Fanout	Location	Element
1	16.083	16.083					data path
1	0.000	0.000			1	PIN_R22	KEY[0]
2	0.864	0.864	RR	CELL	8	IOC_X50_Y10_N1	KEY[0]|combout
3	7.903	7.039	RR	IC	1	LCCOMB_X2_Y21_N26	fsm|ena~0|dataa
4	8.448	0.545	RR	CELL	20	LCCOMB_X2_Y21_N26	fsm|ena~0|combout
5	13.233	4.785	RR	IC	1	IOC_X50_Y7_N0	LEDG[0]|datain
6	16.083	2.850	RR	CELL	0	PIN_U22	LEDG[0]

Path #1: Delay is 16.083

Path Summary | Statistics | Data Path

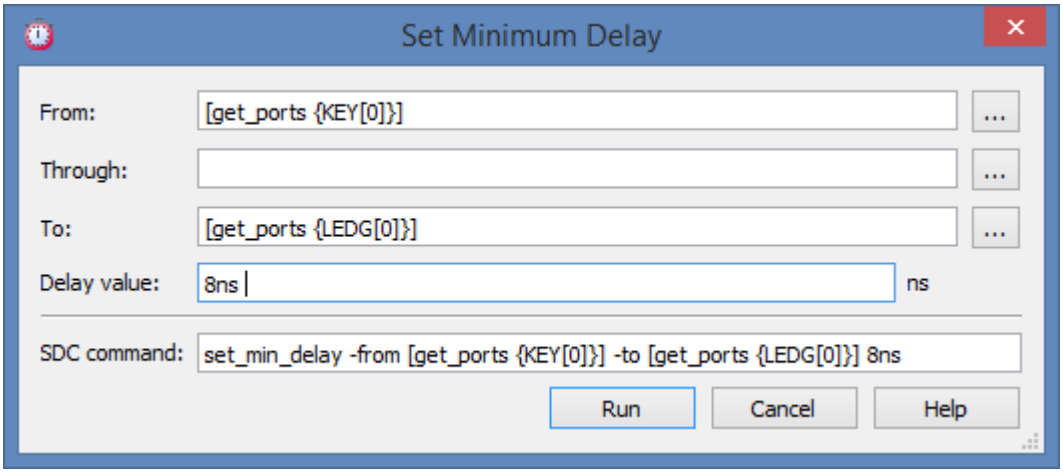
	Property	Value
1	From Node	KEY[0]
2	To Node	LEDG[0]
3	Delay	16.083

Si può peraltro evidenziare nelle varie viste dove si localizza tale ritardo e da cosa è generato (chip Planner)



Si può ora, sempre tramite lo strumento TQTA, provare ad imporre ad esempio il vincolo che questo ritardo non sia superiore a 8ns.

```
Constraints > Set Maximum Delay
```

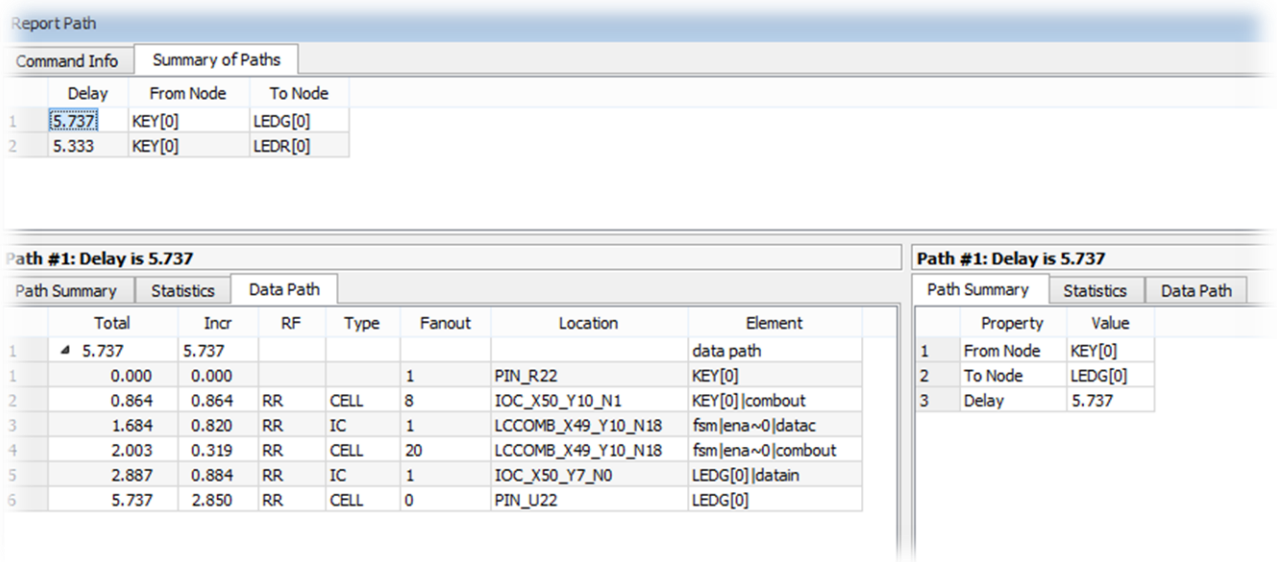


Nota i vincoli vanno messi con oculatezza, mettere dei vincoli irraggiungibili rischia di non consentire al sistema di trovare una soluzione accettabile.

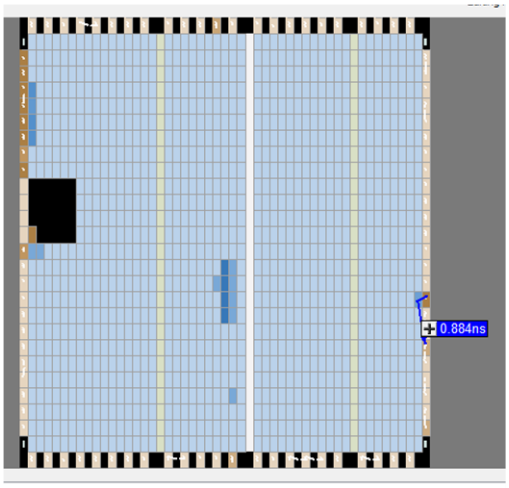
```
Update Timing Netlist
Write SDC File
```

Il file di vincoli così salvato viene a far parte dei files di progetto di Quartus (se così non fosse importare il file appena salvato tra I files di progetto e ricompilare con Quartus.

Localizzando ora attraverso il TQTA il medesimo path



si nota che il sistema ha scelto una diversa dislocazione di blocchi logici pur di garantire le specifiche richieste.



In verità l'uso corretto dello strumento TQTA dovrebbe essere iterativo:

- Si compila il progetto
- Si esegue una prima analisi post-map (ovvero senza informazioni sui ritardi di interconnessione
- Si fissano i vincoli
- Si ricompila il progetto
- Si esegue un'analisi post-fitting
- Si analizzano i veri report
- Si modificano le parti critiche e si ricomincia

Bibliografia

[1] David Harris: "**Structural Design with Verilog**"
http://classes.soe.ucsc.edu/cmpe100/Winter03/Resources/verilog.pdf

[2] **"Tutorial: Modeling and Testing Finite State Machines (FSM)**"
http://courses.cs.tamu.edu/rabi/cpsc617/Assignments/Assignments%202013/Tutorial%20Three%20Cycle%20High%20Timer.pdf

[3] CSE 20221: Logic Design **"Verilog FSM Design Example"**
http://www.eece.hw.ac.uk/teaching/ee3_project_11_12/Verilog_resources/FSM%20Design%20Example%20with%20Verilog.pdf

[4] "**Finite State Machine Simulation**"
http://lyle.smu.edu/~jgd/ee2381/007/labs/fsm_007.pdf

[5] Clifford E. Cummings "**The Fundamentals of Efficient Synthesizable Finite State Machine"**
http://www.sunburst-design.com/papers/CummingsICU2002_FSMFundamentals.pdf

[6] Altera Corporation: "**TimeQuest Timing Analyzer - Quick Start Tutorial**"
http://www.altera.com/literature/ug/ug_tq_tutorial.pdf

[7] Altera Corporation: "**Quartus II TimeQuest Timing Analyzer Cookbook**"
http://www.altera.com/literature/manual/mnl_timequest_cookbook.pdf

[8] Altera Corporation: "**The Quartus II TimeQuest Timing Analyzer**"
http://www.altera.com/literature/hb/qts/qts_qii53018.pdf

[9] Kahng, A.B., Lienig, J., Markov, I.L., Hu, J: "**VLSI Physical Design: From Graph Partitioning to Timing Closure**"- Springer - ISBN 978-90-481-9591-6 (cap.8 available on-line)

[10] Altera Corporation: "**AN 584: Timing Closure Methodology for Advanced FPGA Designs**"
http://www.altera.com/literature/an/an584.pdf

[11] Altera Corporation: "**DE1 Development and Education Board - User Manual**"
https://www.terasic.com.tw/attachment/archive/83/DE1_UserManual_v1018.pdf

[12] Altera Corporation: **"Cyclone II FPGA Starter Development Board -Reference Manual**"
http://www.altera.com/literature/manual/mnl_cii_starter_board_rm.pdf

[13] Richard Lokken "**DE1 Board**"
http://ecampus.matc.edu/lokkenr/elctec-131/labs/de1%20board%20pin%20list%20rev%20aa.pdf