Tutorial 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.

Apprendimenti previsto:

- Sviluppo di un sistema gerarchico
- Utilizzo dei BUS
- Progetto di macchine a stati finiti
- Impostazione di vincoli temporali
- Evidenziazione di problematiche temporali nei "reports"

Procedimento:

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

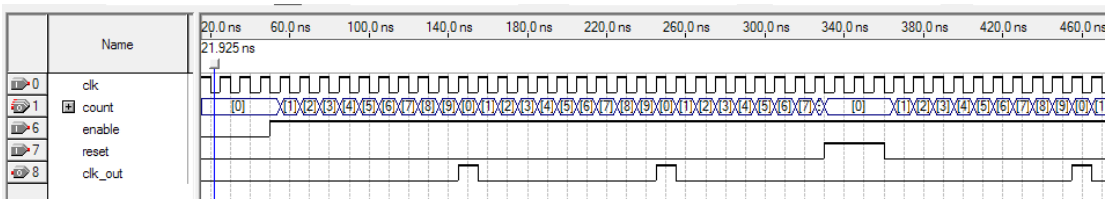
Realizzazione gerarchica Verilog- Schematico

Si predisponga un modulo verilog per la realizzazione di un contatore sincrono mod10 (0…9) con 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 modulo
- Lo si predisponga come "top level entity"

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

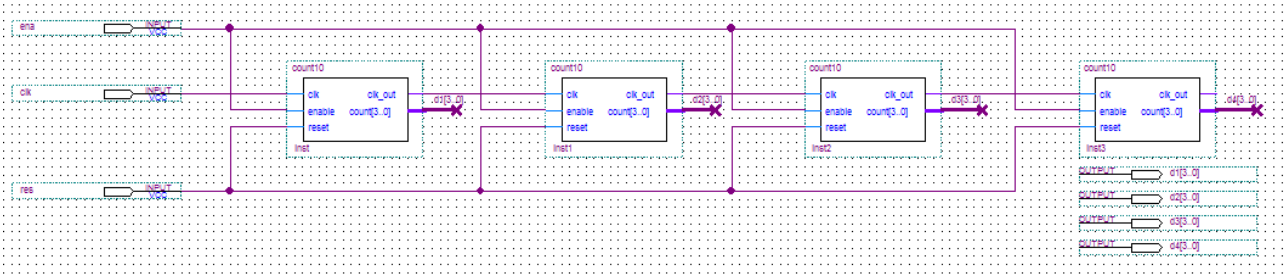


Si predisponga il modulo appena realizzato per essere impiegato a livello gerarchico superiore in uno schematic editor:

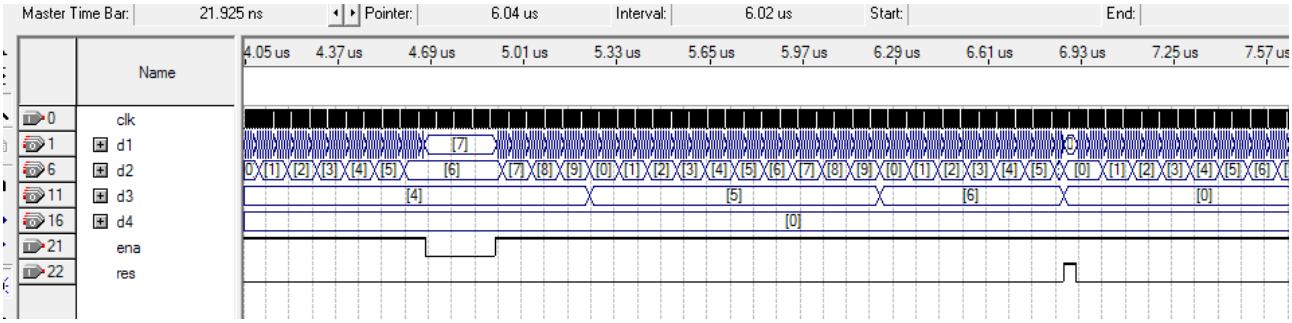
Nella finestra del “project Navigator”

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

- Si crei 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” 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[n1]` è il meno significativo (es: `D[3..0], a[9..2]`) notare che `D[3..0]` e `D[0..3]` sono entrambi leciti, ma mentre nel primo 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 modifichi il settaggio (ctrl – shift – E) onde garantire di usare il corretto file di stimoli
- Si simuli a livello funzionale
- Si verifichi il corretto funzionamento



Realizzazione gerarchica Verilog- Verilog

Predisporre un modulo Verilog per la conversione da binario a display a sette segmenti (una semplice LUT)

```
module SEG7_LUT        (oSEG,iDIG);
input   [3:0]   iDIG;
output  [6:0]   oSEG;
reg             [6:0]   oSEG;

always @(iDIG)
begin
                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

Predisporre un modulo Verilog per l'integrazione di 4 dei moduli precedenti in una sola unità

```
module SEG7_LUT_4 (oSEG0,oSEG1,oSEG2,oSEG3,iDIG );
input   [15:0]  iDIG;
output  [6:0]   oSEG0,oSEG1,oSEG2,oSEG3;

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

endmodule
```

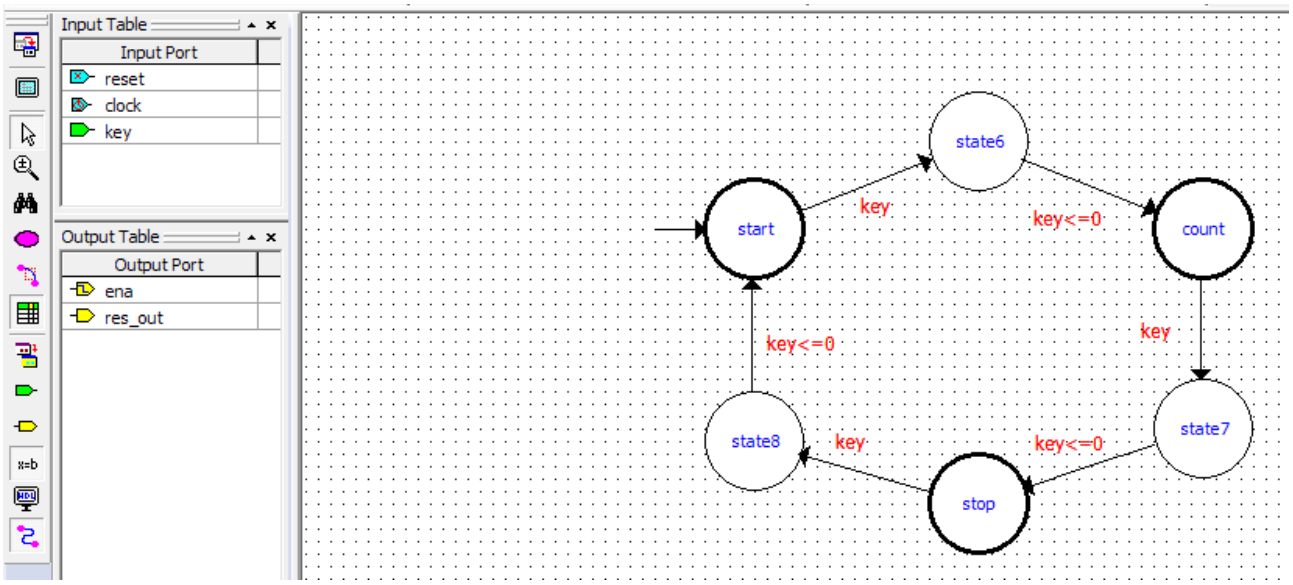
Generare il simbolo corrispondente a questo secondo modulo.

Definizione di una FSM (finite state machine)

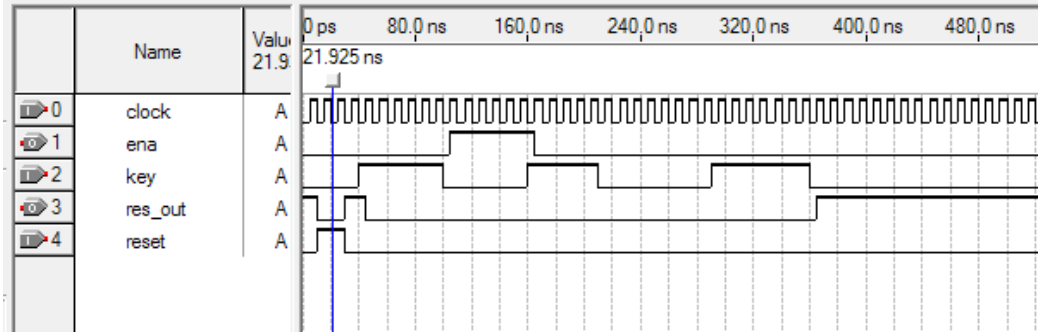
Si voglia realizzare una macchina a stati finiti con un solo pulsante in ingresso che generi gli opportuni segnali per il contatore di `start – stop – reset` secondo questa specifica sequenza.

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

- Aggiungere alle porte di default (clk, reset) le porte “key” come ingresso e “res_out” come uscita
- Tracciare un grafo degli stati simile a quello riportato in figura



- 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 “freeze”
  - Attivare la linea “res_out” allo stato start e disattivarla negli altri
- 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
  - Settare il file verilog appena generato come Top-level entity
  - Generare un opportuno file di stimoli
  - Si modifichi il settaggio (ctrl – shift – E) onde garantire l’uso del corretto file di stimoli



- Generare il simbolo grafico associato all fsm appena realizzata

Definizione del sistema completo

Sebbene possa sembrare che vi siano ormai a disposizione tutti i blocchi necessari per sviluppare il sistema definitivo, si deve tener conto che il più basso valore del clock della scheda DE1 è a 24 MHz. Se si utilizzasse questo clock per effettuare il conteggio esso sarebbe talmente rapido che i display a 7 segmenti apparirebbero sempre illuminati. Si deve pertanto predisporre un ulteriore blocco utile per ridurre a valori più ragionevoli la frequenza di conteggio.

Si adotti una frequenza di conteggio pari a 100 Hz, cosicchè da poter impiegare il sistema per conteggiare fino i centesimi di secondo.

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

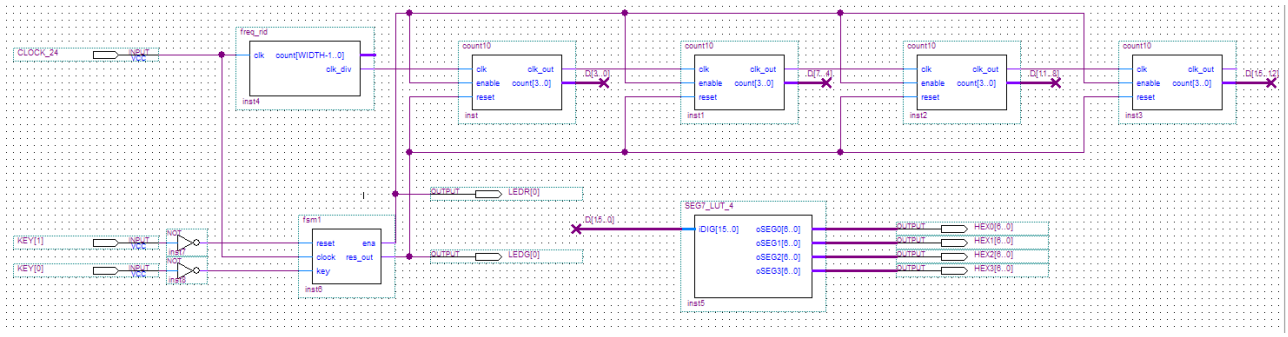
```
module freq_rid
#(parameter WIDTH=64)
(
        input clk,
        output reg [WIDTH-1:0] count,
        output reg clk_div
);

        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 che 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

```
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. Per congruenza con la macchina a stati finiti realizzata, si dovranno pertanto 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.

Analisi dei Reports

Premessa: Il circuito appena realizzato, per quanto semplice da un punto di vista intuitivo, presenterebbe grosse problematiche nel momento in cui la sua frequenza di lavoro dovesse essere superiore a poche centinaia di Hz, infatti non è una strategia efficace quella di ritardare il clock con della logica interposta, essa infatti può comportare problematici disallineamenti del clock con conseguente errato funzionamento del circuito stesso.

Se si analizzano i timing report del circuito si notano infatti, evidenziati in rosso collegamenti che non sono coerenti con un regolare funzionamento. In particolare si nota che

Clock Hold: 'CLOCK_24'

	Minimum Slack	From	To	From Clock	To Clock	Required Hold Relationship	Required Shortest P2P Time	Actual Shorte P2P Time
1	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst3|count[0]	CLOCK_24	CLOCK_24	None	None	4.531 ns
2	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst3|count[2]	CLOCK_24	CLOCK_24	None	None	4.531 ns
3	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst3|count[1]	CLOCK_24	CLOCK_24	None	None	4.531 ns
4	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst3|count[3]	CLOCK_24	CLOCK_24	None	None	4.531 ns
5	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst2|count[0]	CLOCK_24	CLOCK_24	None	None	4.551 ns
6	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst2|count[2]	CLOCK_24	CLOCK_24	None	None	4.551 ns
7	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst2|count[3]	CLOCK_24	CLOCK_24	None	None	4.551 ns
8	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst2|count[1]	CLOCK_24	CLOCK_24	None	None	4.551 ns
9	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst2|clk_out	CLOCK_24	CLOCK_24	None	None	4.548 ns
10	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst1|count[0]	CLOCK_24	CLOCK_24	None	None	4.540 ns
11	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst1|count[1]	CLOCK_24	CLOCK_24	None	None	4.540 ns
12	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst1|count[3]	CLOCK_24	CLOCK_24	None	None	4.540 ns
13	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst1|count[2]	CLOCK_24	CLOCK_24	None	None	4.540 ns
14	Not operational: Clock Skew > Data Delay	fsm1:inst6|fstate.count	count10:inst1|clk_out	CLOCK_24	CLOCK_24	None	None	4.539 ns

diversi path che vanno dalla FSM ai vari contatori presentano un ritardo sul disallineamento del clock superiore al ritardo di propagazione dei dati (in particolare si fa riferimento al segnale "ena" generato dalla FSM). All'atto pratico il fronte di clock sul quale si è generato il segnale "ena" arriva ai contatori solo molto tempo dopo il segnale "ena" stesso. L'effetto è che il conteggio può partire sfasato rispetto il suo segnale di abilitazione. In questo caso la cosa non crea evidenti problemi, ma in altre situazioni potrebbe comportare malfunzionamenti. Si noti altresì che il sistema non segnala malfunzionamenti analoghi relativi al segnale "res_out" perché questo agisce in modo asincrono nei confronti dei contatori.

Per ovviare a questo inconveniente si dovrebbe ripensare al contatore evitando di realizzarlo come cascata di 4 contatori in cui ciascuno genera il clock per il successivo, bensì tutti i contatori dovrebbero essere pilotati dallo stesso clock e lo stadio precedente al più abilita o disabilita il conteggio dello stadio successivo.

Si possono evidenziare i path interessati dal fenomeno:

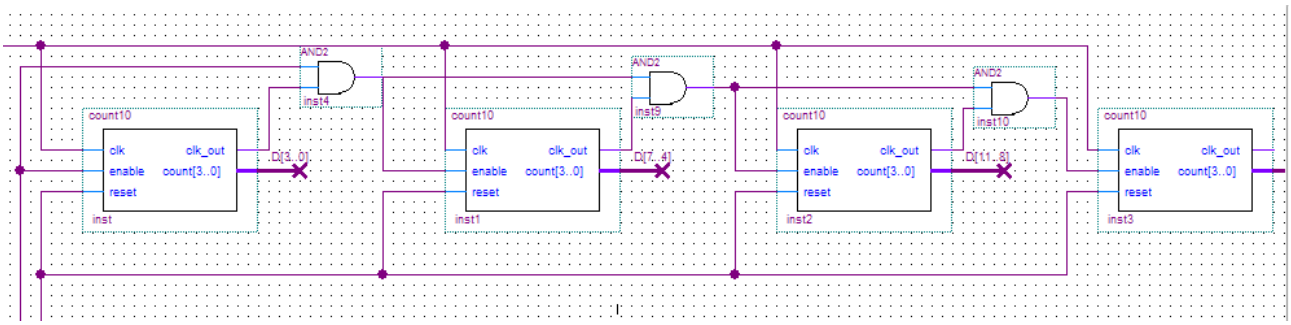
```
Aprire il timing report (ctrl-R)
Timing Analizer  > Clock Hold: Nomeclock
Right click on report
```

```
Locate > Locate in ...
```

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 opportunamente I vincoli realizzativi

Una soluzione al problema esposto potrebbe essere la seguente:



Questa architettura non crea più 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 problema, ovvero la commutazione del contatore a livello più altro avviene in ritardo di un clock rispetto quello più basso, pertanto il passaggio da 09 a 10 in realtà avviene erroneamente in due cicli 09 – 00 – 10. Ovviare a questo ulteriore problema richiede di modificare più profondamente il progetto.

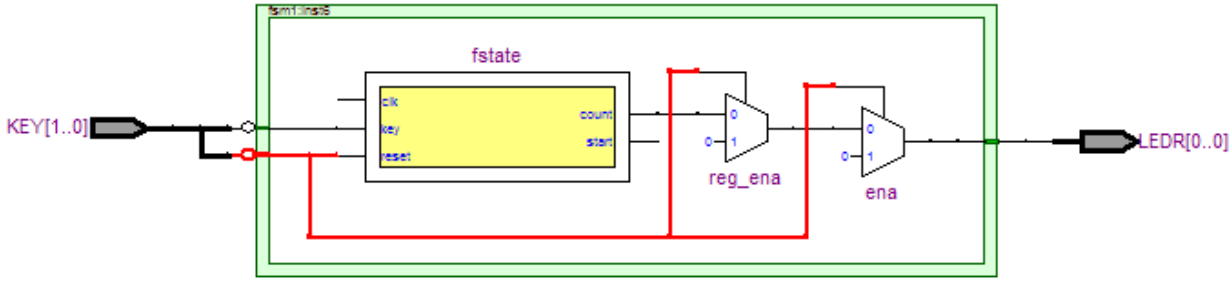
Vincoli realizzativi

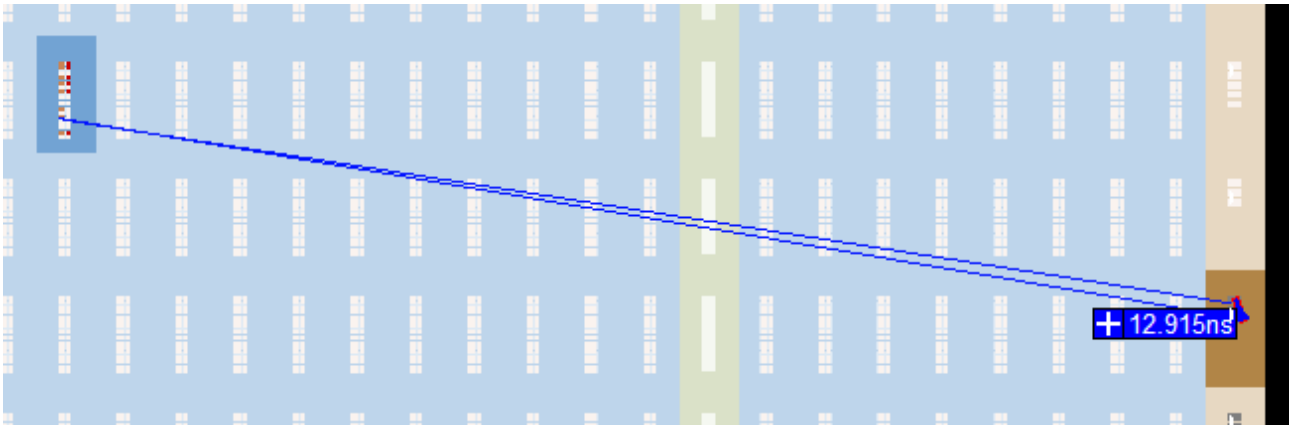
Il progetto finora seguito non ha visto ancora l'imposizione di vincoli, fatta eccezione per la posizione dei piedini di I/O e pertanto il risultato ottenuto non ha seguito nessuna particolare strategia di ottimizzazione.

Si noti ad esempio il ritardo di propagazione asincrona tra KEY[1] e LEDR[0] di 12.95 ns

```
Reports
Timing Analizer  > tpd
```

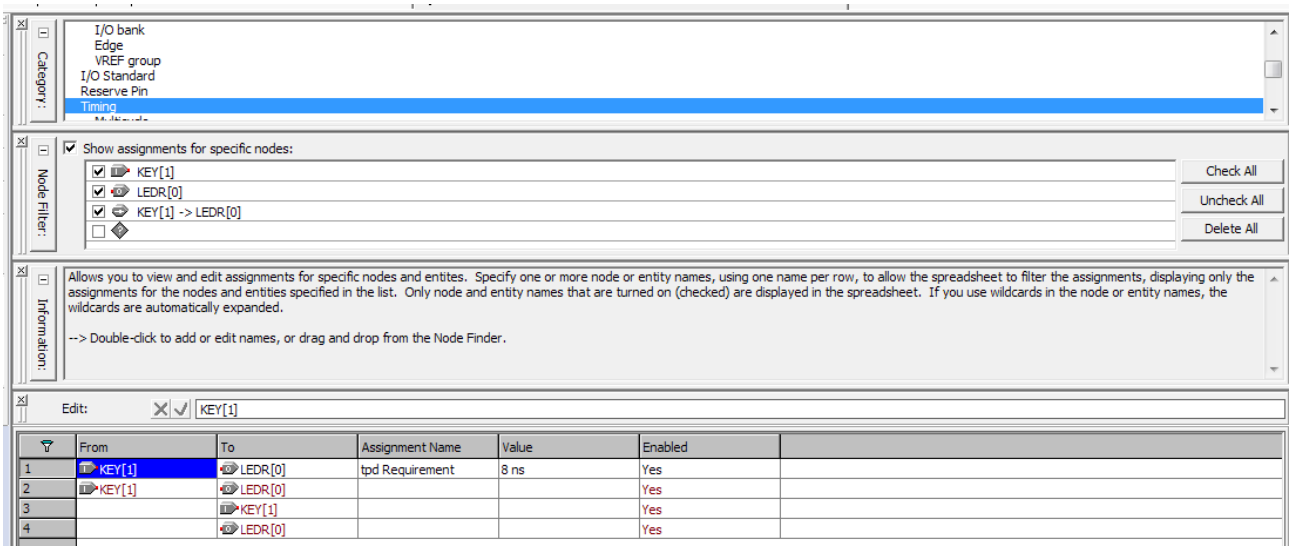
E si evidenzi nelle varie viste **dove** si localizza il ritardo (RTL viewer) e **da cosa è generato** (chip Planner)





Si provi ora ad imporre il vincolo che questo ritardo non sia superiore a 8ns.
Nota i vincoli vanno messi con oculatezza, mettere dei vincoli irraggiungibili rischia di non consentire al sistema di trovare una soluzione accettabile.

```
Right click on report
Locate > Locate > Locate in Assignments Editor
Editare la tabella
```



- Salvare il file e ricompilare
- Analizzare i reports e soprattutto il floorplanning con particolare attenzione al path in oggetto
- Provare a modificare ulteriormente il vincolo a 4ns e analizzare i risultati