

TUTORIAL 2

Ottimizzazione di un progetto su scheda XSA50

1 Problema proposto

Lo scopo di questo tutorial è quello di acquisire dimestichezza nella realizzazione di un progetto gerarchico e di operare successivamente la simulazione e la sintesi. Si vuole inoltre sottolineare come uno stesso algoritmo, realizzato secondo metodologie differenti, possa condurre a risultati con prestazioni molto diverse.

Si vuole progettare un sommatore ad 8 bit sincrono, un blocco cioè che abbia in ingresso due bus a 8 bit ed in uscita uno a 9 bit che ne contiene la somma; il tutto deve essere sincronizzato in una pipeline in modo che la somma in uscita si presenti il ciclo di clock successivo all'arrivo dei dati in ingresso.

Questo venga fatto con due metodologie diverse:

- utilizzando una gerarchia di fulladder opportunamente collegati tra loro in cascata secondo l'architettura "Ripple Carry"
- adoperando l'elemento già presente nella libreria del software ISE.

Si proceda quindi secondo i seguenti passi:

1. Si crei lo schematico;
2. Si estragga lo schematic symbol;

Si ripetano i passi 1. e 2. ogni qual volta si debba realizzare uno schematico a livello gerarchico più alto, impiegando il simbolo creato al passo precedente.

3. Si crei un segnale di test;
4. Si esegua una simulazione comportamentale e se ne verifichi il corretto funzionamento;
5. Si impostino le opzioni di sintesi volute (Synthesize->Properties);
6. Si sintetizzi il circuito complessivo
7. Si proceda con l'analisi del report risultante e si simuli nuovamente il circuito sintetizzato evidenziando i tempi di ritardo

A questo punto si possono eventualmente modificare il segnale di test (operando sulla frequenza di clock o sugli *Input Setup Time/Output Valid Delay*) o le opzioni di sintesi (cambiando il tipo ed il livello di ottimizzazione) e ripetere gli ultimi due passi.

8. Confronto dei risultati ottenuti

Per un corretto svolgimento sarà pertanto necessario aver installato il pacchetto software Xilinx ISE 5.x, nonché il simulatore ModelSim XE II v5.6e; in particolare, per la corretta cooperazione di entrambi i software, è necessario inizialmente impostare il ModelSim come simulatore predefinito per il pacchetto ISE:

dopo aver lanciato il Project Navigator (Programmi -> Xilinx ISE 5.x -> Project Navigator), si clicchi su Edit -> Preferences -> Partner Tools e si indichi il percorso del file win32xoem.exe (Corrispondente all'eseguibile del ModelSim) all'interno della casella denominata come Model Tech Simulator. Infine si confermi il tutto.

2 Realizzazione proposta

2.1 Sommatore con fulladder

Si avvii l'ISE Webpack andando sul menu di avvio e cliccando su Programmi -> Xilinx ISE 5.x -> Project Navigator

A questo punto bisogna creare un nuovo progetto selezionando File -> New Project

Apparirà la finestra in cui si settano le impostazioni della scheda; inoltre nel campo Project Location si scelga il path della cartella di destinazione del nuovo progetto e nel campo Project Name si inserisca il nome del progetto; verrà creata quindi una directory con quel nome sul percorso indicato.

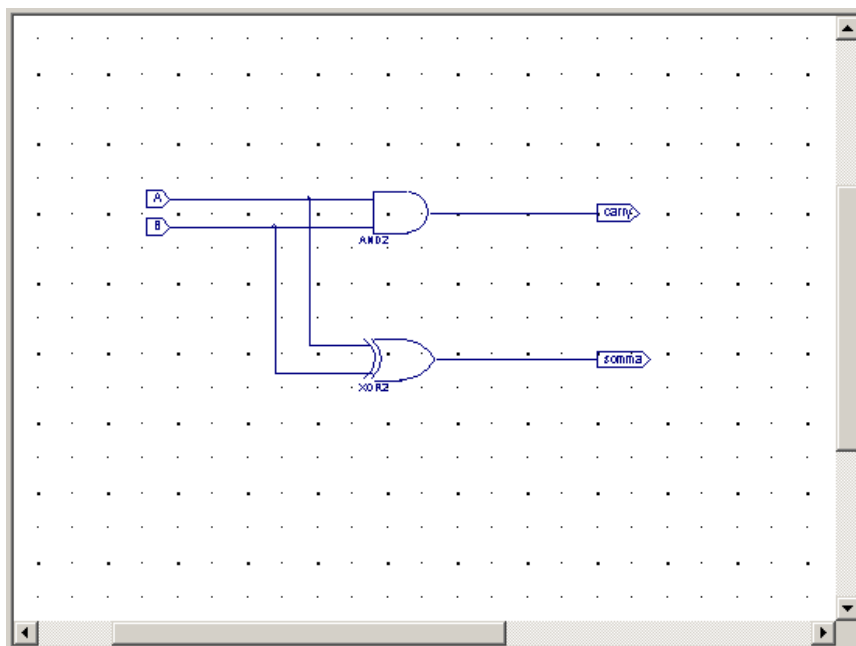
Si preme quindi su OK e si tornerà sulla schermata principale.

Osserviamo innanzitutto che ciascun fulladder è costituito da due halfadder, essi stessi collegati opportunamente in cascata. Prima di tutto, quindi, si andrà a realizzare un halfadder.

Nel Source Pane si evidenzi il nome del chip con le specifiche che si sono date in precedenza e si clicchi col tasto destro del mouse, selezionando New Source e di seguito Schematic, indicando il nome dello schematico nel campo File Name, ad esempio halfadder.

Si confermi cliccando su Avanti> e successivamente su Fine.

Si aprirà quindi il tool Xilinx ECS e si andrà a disegnare lo schematico posizionando la logica e i pins di I/O opportunamente collegati tra loro; dopo averli rinominati si dovrebbe ottenere un risultato del tipo



in cui si è scelto di nominare i due bit in ingresso *A* e *B* e quelli in uscita *carry* (risultante dalla porta and) e *somma* (risultante dalla porta xor).

Al solito, cliccando su , si verifichi la correttezza dello schematico e si salvi.

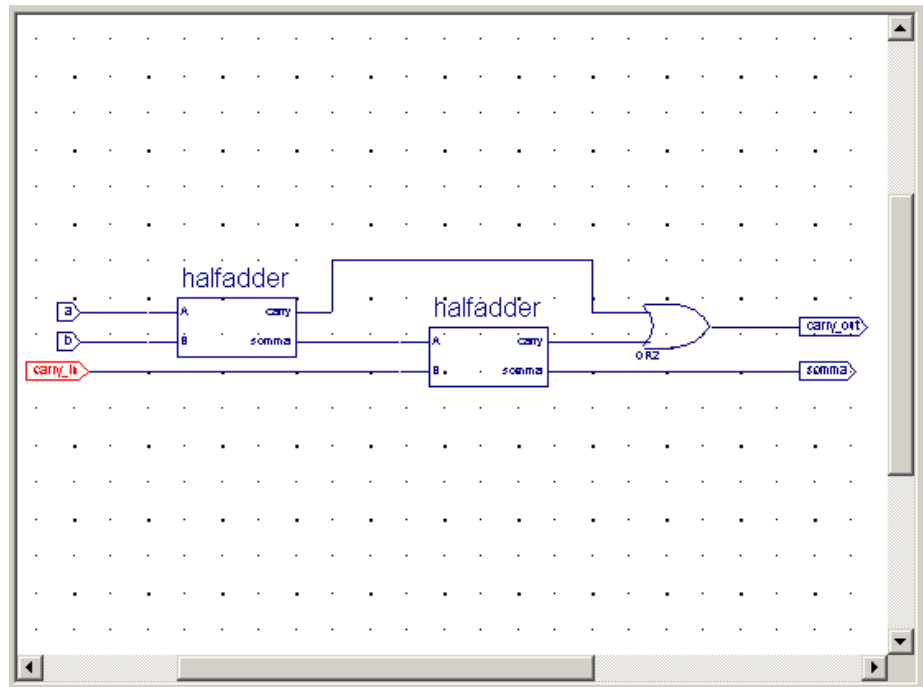
Si ritorni quindi al Project Navigator; se si è eseguito correttamente, si dovrebbe avere in aggiunta al progetto il file halfadder.sch.

A questo punto nuovamente si clicchi sul nome del chip col tasto destro e si aggiunga un nuovo schematico, questa volta denominato fulladder. Ora si aprirà come al solito l'ECS (oppure, se non lo si aveva chiuso in precedenza, si dovrebbe essere aperto un nuovo schematico vuoto accanto al precedente già salvato).

Si vada al Project Navigator ed, evidenziando sul progetto il nome del file halfadder.sch, nel Process pane si vada a cliccare due volte su **Create Schematic Symbol** sotto **Design Entry Utilities**; così facendo viene creato sullo Schematic Editor un nuovo simbolo col nome halfadder.

Si torni quindi all'ECS, si clicchi su **Symbols** e qui sulla voce che riporta il path del progetto; nella finestra sottostante, dove normalmente vi sono i simboli disponibili, dovrebbe trovarsi il nuovo simbolo *halfadder*.

Si colleghino quindi in cascata i due halfadder, si aggiungano i pins, li si rinominino.



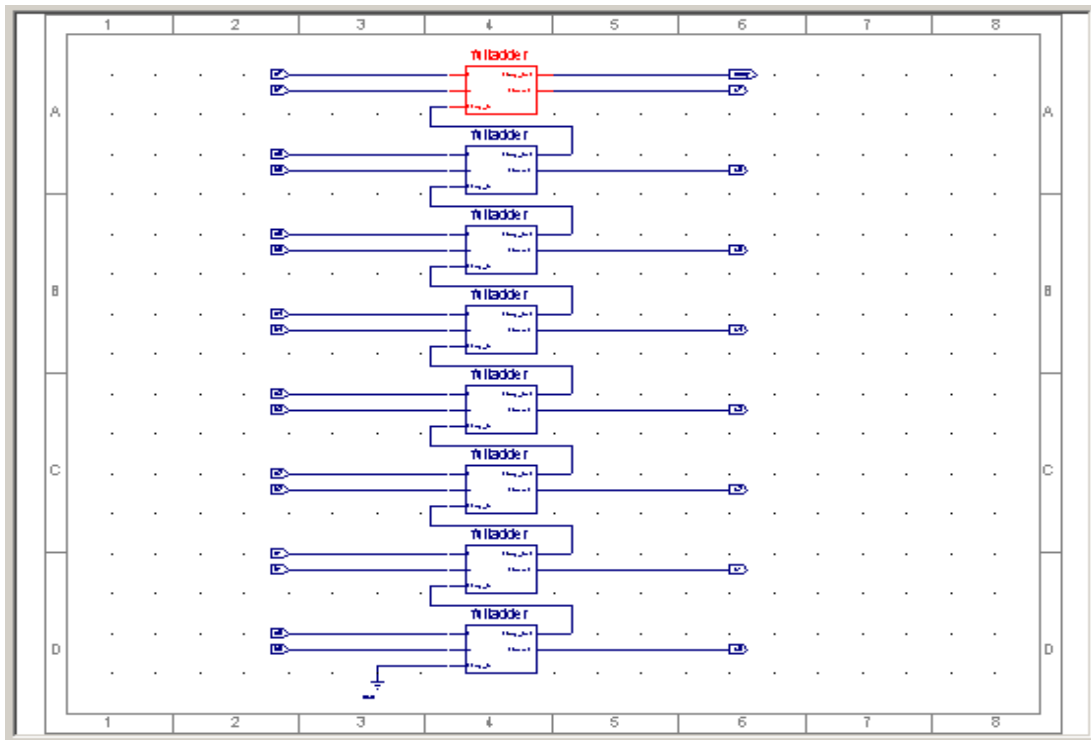
Si operi , al solito, il controllo degli errori e si salvi.

Ora si ritorni al Project Navigator, si clicchi col tasto destro sul nome del chip nel Source pane e si aggiunga un nuovo schematico col nome di, ad esempio, *sommatore8bit*; per questo, ovviamente, si utilizzerà il fulladder appena creato.

Di nuovo nel Project Navigator si evidenzi nel Source pane il nome del file fulladder.sch e nel Process Pane sottostante **Create Schematic Symbol**.

Si torni quindi allo schematic editor e si recuperi il nuovo simbolo *fulladder* sotto il path del progetto nella finestra **Symbols**.

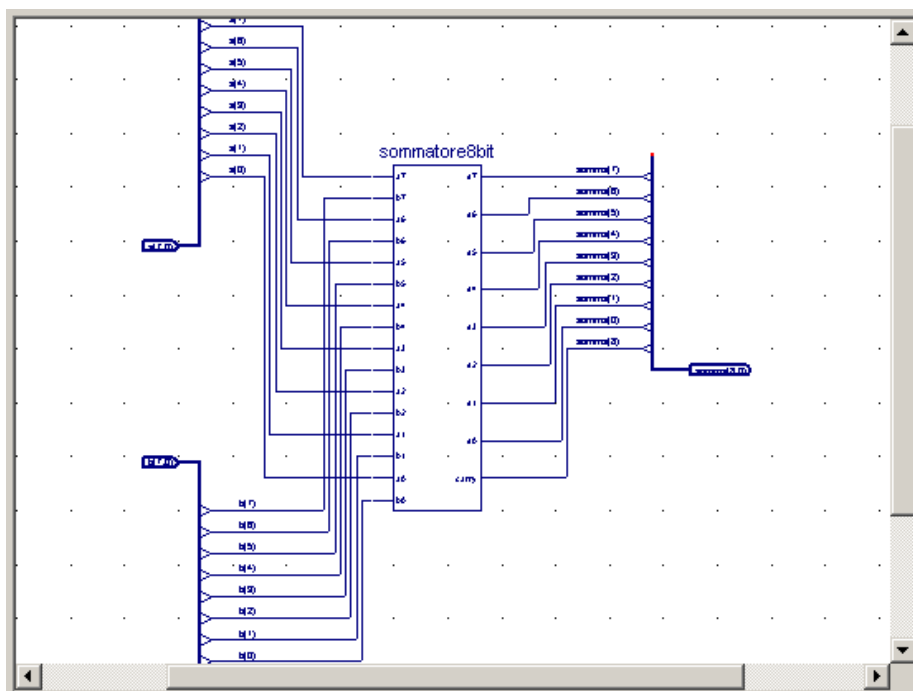
Si colleghino quindi opportunamente 8 fulladder: il carry in del bit meno significativo va messo a massa ed il carry out di ciascun fulladder va collegato col carry in del fulladder successivo, secondo l'architettura RC (Ripple Carry)




ciascun fulladder ha in ingresso una coppia di bit da sommare, il carry della somma della coppia precedente ed in uscita il risultato della somma attuale ed il nuovo carry, eventualmente nullo. Si faccia il check error e si salvi.

Ora si devono ancora riunire i singoli bit che compongono i vari segnali entro opportuni "bus". Si torni, quindi, al Project Navigator e si crei il simbolo per lo schematico *sommatore8bit* come già illustrato in precedenza. Si aggiunga un nuovo schematico con il nome *sommatorefin* e, una volta tornati all'ECSS, si posizioni il nuovo simbolo appena creato e si traccino gli opportuni collegamenti con le pad.

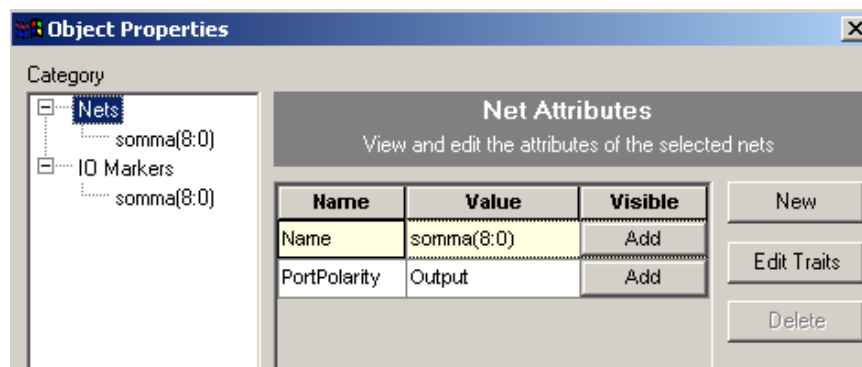
Si dovrà ottenere una cosa del tipo




in cui, per creare ciascun bus, ci si è serviti di “interfacce” di collegamento tra wire e bus appunto, dette bus-tap ed indicate col simbolo .

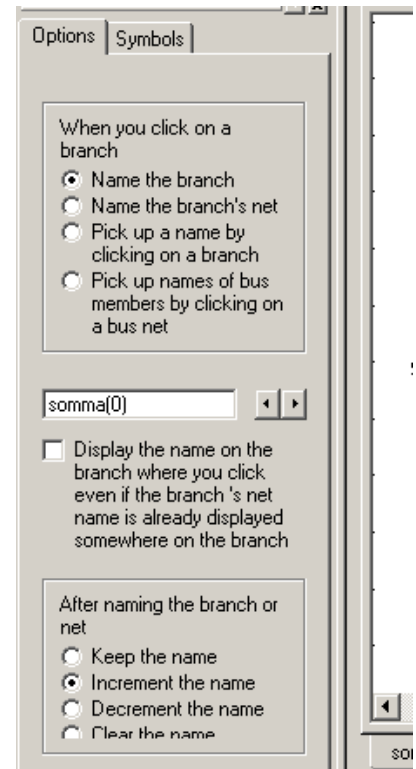
Meglio:

- si selezioni il simbolo del bus-tap e si vadano a posizionare nello schematico quanti ne servono, scegliendone l'opportuno orientamento nella finestra che compare accanto all'area di lavoro;
- si traccino i collegamenti tra l'estremità superiore del tap e l'uscita o l'ingresso del simbolo principale;
- si tracci un filo di collegamento tra le estremità inferiori dei bus-tap (la base della “piramide”) e se ne lasci libera una parte in modo da poterci aggiungere la pad;
- si clicchi due volte sulla pad appena posizionata e la si faccia diventare la pad d'ingresso o d'uscita di un bus dandole un nome e tra parentesi tonde la dimensione (Per esempio, per un bus a 8 linee si scriverà **<nome bus>(7:0)**)




N.B. Questo sta a significare che ciascun elemento del bus è del tipo **<nome bus>(i)** in cui viene considerato il settimo filo come il bit meno significativo; per il viceversa si usa **<nome bus>(0:7)**

- si noti che ora il filo di collegamento con la pad è di maggior spessore ad indicare per l'appunto che si tratta di un bus; bisogna quindi associare opportunamente ciascun filo afferente al bus col corrispondente segnale nel bus stesso. Per far questo si preme il pulsante  e nella finestra che compare a fianco si inserisca il nome del primo filo del bus (ad esempio **<nome bus>(0)**) e si spunti la voce sottostante **Increment the name**. Si noti che nel frattempo il puntatore del mouse è diventato un'etichetta col nome che si è appena scelto. Si vada quindi nell'area di lavoro e si clicchi sul ramo a cui si vuole dare quel nome: esso viene immediatamente aggiunto ed è subito pronta un'altra etichetta con il numero tra parentesi già incrementato. Si ripeta l'operazione per ciascun filo.

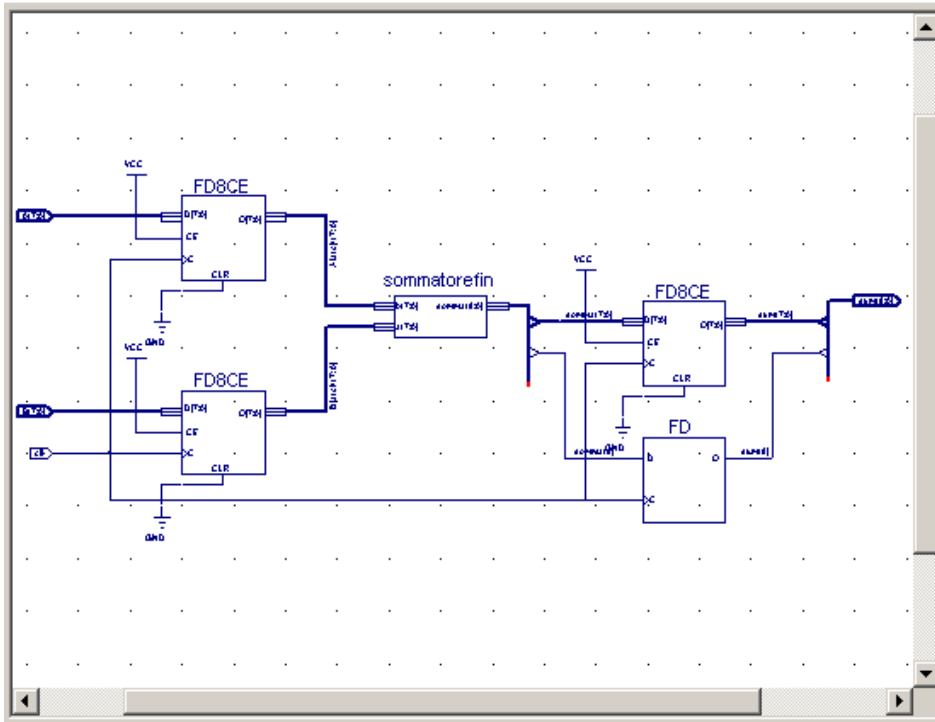


Finito lo schematico, si esegua il check error e si salvi.

Curiosità: se si seleziona il blocco principale e poi si preme il

tasto  , è possibile vedere di quale logica è costituito al suo interno e come è collegata. Questo si può fare, ovviamente, con qualsiasi blocco, anche per quelli già presenti nelle librerie del software.

Ora bisogna aggiungere il sincronismo: si crei allora lo "Schematic Symbol" del sommatore appena realizzato, si aggiunga al progetto un nuovo schematico che chiameremo, ad esempio, *sommasinc*;



si aggiungano quindi un flip-flop del tipo *FD8CE* per ciascuno dei due bus in ingresso.

In uscita si hanno 9 bit, quindi si scomponga il bus in due rami, uno a 8 fili a cui verrà aggiunto un flip-flop uguale ai precedenti, ed uno a filo singolo a cui verrà collegato un flip-flop del tipo *FD*.

Si riuniscano poi i due bus in uscita sotto un unico bus col nome di *sum(8:0)*, ad esempio. Si colleghino

opportunamente all'alimentazione i ChipEnable dei flip-flop ed i Clear a massa. Si estragga inoltre la linea di clock.

Si dovrebbe ottenere uno schematico del tipo qui sopra.

Si presti molta attenzione a come vengono nominati i bus: molto spesso la causa degli errori è dovuta proprio a disattenzioni sul collegamento dei fili nei bus.

Si esegua la verifica degli errori e si salvi.

Ora si deve procedere con la simulazione comportamentale: per far ciò bisogna innanzitutto creare un segnale di test.

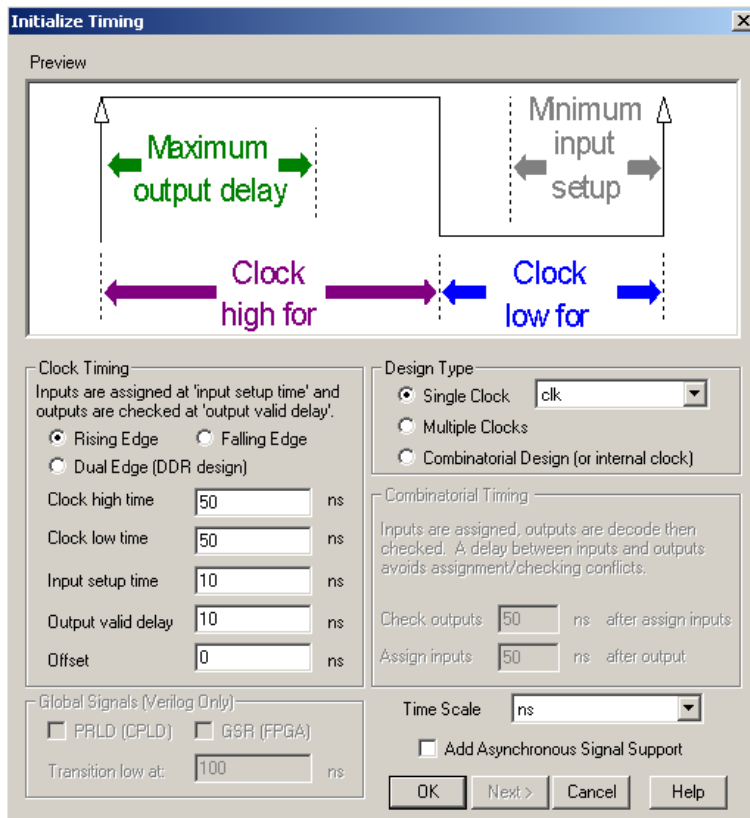
Cliccando col tasto destro sulla parte del progetto a cui si vuole associare detto segnale nel Project Navigator ed aggiungendo una nuova sorgente del tipo **Test Bench Waveform**.

In questo modo si associa alla parte da simulare un nuovo file con estensione *tbw*; cliccandoci sopra due volte si apre il tool HDL Bencher in cui si sceglie il tipo di clock ed i dati in ingresso per ogni ciclo di clock, o anche, se opportuno, il risultato che si vorrebbe ottenere in uscita.

Si noti che se ora si clicca una sola volta in corrispondenza della forma d'onda nel Source pane, compare nel Process pane sottostante un nuovo menù di simulazione in cui è possibile, con la data forma d'onda, verificare il funzionamento del progetto nei suoi vari stadi di implementazione; ovvero si passa da una simulazione comportamentale ad una finale dopo il processo di piazzamento delle piste.

Cliccando due volte su una di queste voci si lancia il Modelsim col tipo di simulazione voluta e la forma d'onda di test selezionata.

Si crei un segnale di test associato al file più in alto nella gerarchia di quelli componenti il progetto con la procedura spiegata precedentemente; si apre automaticamente l'HDL Benchner.



Appare innanzitutto una finestra di inizializzazione in cui è possibile impostare il clock: se il progetto è fornito di una linea di clock esplicita, questa viene di default impostata come clock singolo e nella parte a sinistra si può fissare la forma d'onda; altrimenti si possono scegliere clock multipli o un clock interno se questo non è esplicitato nello schematico.

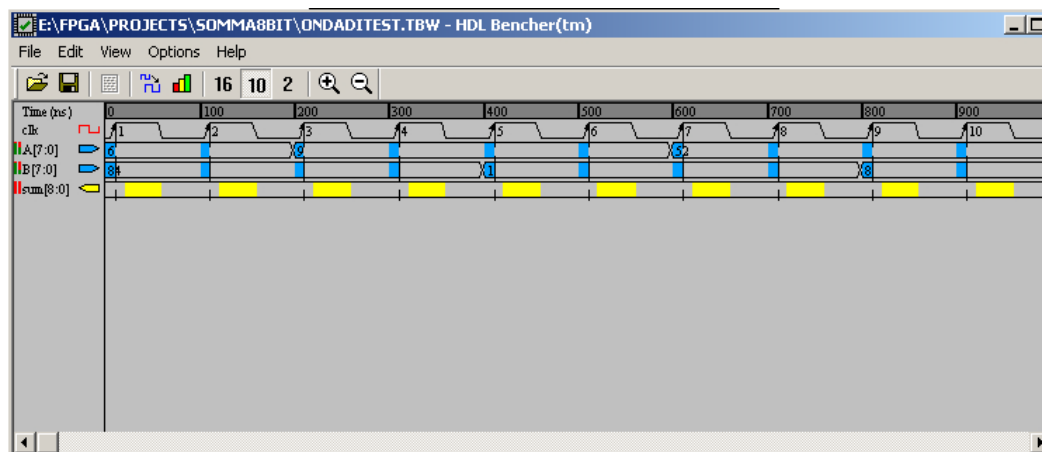
Si sceglierà Single Clock e se ne imposteranno le caratteristiche a sinistra: quelle di default possono essere ritenute soddisfacenti. Si scelga la scala temporale e si confermi.

Eventualmente è possibile modificare le impostazioni appena scelte andando su Options->Timing

Constraints.

A questo punto vengono mostrati gli ingressi (in blu) e le uscite (in giallo); da notare che un ciclo di clock è marcato da un settore parzialmente colorato, questo serve a rendere più evidenti i periodi di possibile commutazione dei segnali, contrapposti ai periodi nei quali i viceversa detti segnali dovrebbero essere stabili.

Ora si può assegnare, per ciascun input, il valore nel corrispondente ciclo di clock, cliccando sul rispettivo tratto blu ed inserendo il numero in forma



esadecimale, decimale o binario, a seconda di quale pulsante risulta selezionato tra 16 10 2 .

La durata del segnale di test è evidenziata da una linea blu verticale, che può essere trascinata col mouse fino al limite voluto.

Si salvi e si chiuda.

Si torni quindi al Project Navigator e si evidenzi il file che contiene l'onda appena creata.




Nel Source pane compaiono le simulazioni possibili:

- Simulate Behavioral VHDL Model (simulazione comportamentale);
- Generated Expected Simulation Results (in cui si possono comparare le uscite effettivamente ottenute con quelle "desiderate");
- Simulate Post-Translate VHDL Model (simulazione post traduzione e quindi post sintesi);
- Simulate Post-Map VHDL Model (simulazione post mappatura);
- Simulate Post-Place&Route VHDL Model (simulazione dopo il piazzamento delle piste);

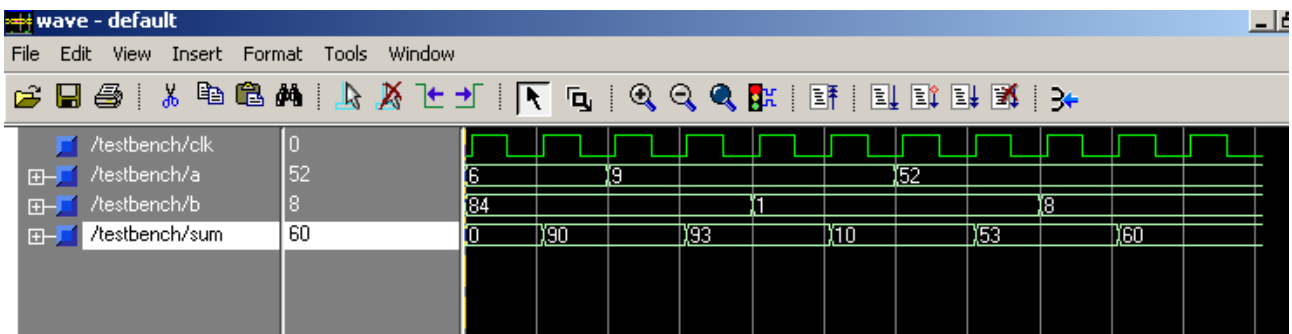
Si esegua un doppio clic sulla prima e verrà lanciato il simulatore.

Il simulatore Modelsim è costituito principalmente da quattro aree:

- una finestra principale in cui si controlla lo svolgimento della simulazione;
- una finestra denominata Structure Window, in cui vi è il progetto da simulare, sotto forma di elenco di componenti costituenti in cui transita il segnale;
- una finestra, detta Signals Window, in cui compare l'elenco degli ingressi e delle uscite da valutare;
- una Wave Window, in cui viene visualizzata la simulazione vera e propria.

Nella Wave window, si possono posizionare dei cursori  magari per fissare l'attenzione sul tempo trascorso tra due posizioni successive, oppure visualizzare più comodamente una determinata porzione di simulazione (con ) o l'intera simulazione su tutta la finestra (con .



Si dovrebbe ottenere un risultato del tipo qui sotto



in cui, per una più comoda visualizzazione, si sono impostate i valori in decimale, cliccando col tasto destro su ciascun segnale e scegliendo Radix->Decimal.

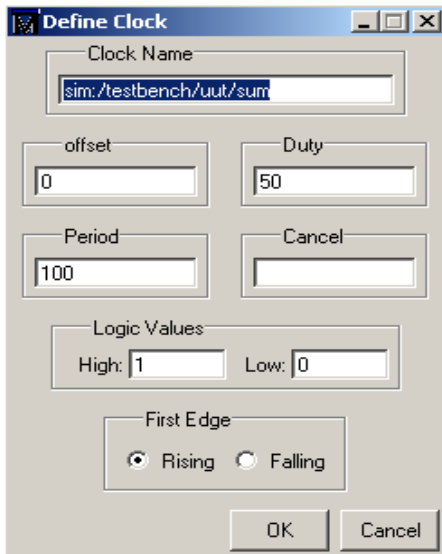
Si noti che il modello è corretto almeno dal punto di vista comportamentale; infatti ad ogni colpo di clock vengono preparati gli ingressi del sommatore ed al colpo di clock successivo vengono portati in uscita.

Inoltre si osservi l'idealità di questo tipo di simulazione; infatti non vengono previsti ritardi di propagazione ed i segnali di conseguenza commutano immediatamente al fronte di salita del clock.

In qualsiasi momento, cliccando sul tasto  (Restart) nella Main window o nella Wave window, è possibile riniziare la simulazione oppure con  (Continue Run) continuarla per un altro intervallo di tempo pari a quello fissato all'inizio della prima simulazione.

Si possono inoltre aggiungere segnali alla simulazione globale, andando sulla Structure window ed evidenziando il progetto che si intende simulare e, di seguito, spostandosi nella Signals window, cliccando col tasto destro sul segnale da aggiungere e selezionando Add to Wave->Selected Signals.

Una volta che si hanno tutti i segnali nella Wave window, si simula con **Continue Run**.

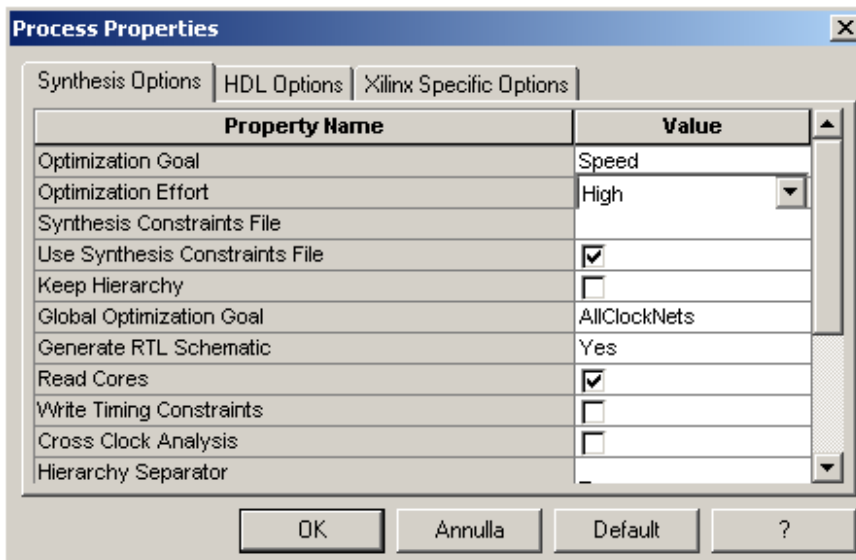


Si può, inoltre, associare ad un segnale un clock “personalizzato”, evidenziandolo nella finestra dei segnali, cliccando su **Edit->Clock** ed andando poi a modificarne il periodo (sempre in pico secondi), il duty cycle, settare i valori logici e fissare la tipologia del primo fronte di clock.

Al solito, poi, sulla Wave window si continua la simulazione tramite l'apposito tasto, oppure la si ricomincia avendo prima aggiunto i segnali nel modo precedentemente descritto.

A questo punto si può procedere con il processo di sintesi, avendone prima scelto la configurazione desiderata.

N.B. Per avere una visualizzazione dell'intero menu di configurazione con tutte le voci personalizzabili, si evidenzia il progetto e sul Project Navigator sotto **Edit->Preferences->Processes->Property Display Level** si sceglie **Advanced**.



Si ritorni al Project Navigator e si evidenzia il file nel progetto più in alto nella gerarchia; cliccando col tasto destro su **Synthesize** e poi **Properties**, si può scegliere

- il tipo ed il livello di ottimizzazione (**Optimization Goal** e **Optimization Effort**);
- un'eventuale file di vincoli di sintesi, che dev'essere scritto con la sintassi appropriata ed ha estensione **xcf**;
- la strategia di ottimizzazione (**Global Optimization Goal**),

ovvero il periodo dell'intero progetto (**AllClockNets**), il massimo ritardo tra pad d'ingresso e pad d'uscita (**Inpad To Outpad**), il massimo ritardo tra pad d'ingresso e clock (**Offset In Before**) o quello tra quest'ultimo e la pad d'uscita (**Offset Out After**) oppure un'insieme dei precedenti (**Maximum Delay**); il tutto se non si specificano vincoli di tempo.

Si impostino le opzioni come in figura sopra e, dopo aver confermato, si avvii il processo di sintesi cliccando sopra a **Synthesize** due volte.

Al termine si visualizzi il report cliccando su **View Synthesize Report** oppure aprendo con un editor di testo il file con estensione **syr** (**Synthesize Report**): all'inizio viene fatto un riepilogo delle

impostazioni scelte e, successivamente sotto la voce final report, si possono analizzare i risultati finali della sintesi.

Si noti come nel report, dopo un'introduzione che riassume le impostazioni scelte, si ha un'idea delle risorse utilizzate su FPGA dal progetto a livello di logica

Cell Usage :

```
# BELS           : 47
#  and2          : 16
#  gnd           : 4
#  or2           : 8
#  vcc           : 3
#  xor2          : 16
# FlipFlops/Latches : 25
#  FD           : 1
#  FDCE         : 24
# Clock Buffers   : 1
#  BUFGP        : 1
# IO Buffers      : 25
#  IBUF         : 16
#  OBUF         : 9
```

nonchè a livello più alto, di dispositivi impiegati

Device utilization summary:

Selected Device : 2s50tq144-5

```
Number of Slices:           16 out of 768  2%
Number of Slice Flip Flops: 25 out of 1536  1%
Number of bonded IOBs:      25 out of 96   26%
Number of GCLKs:            1 out of 4    25%
```

Vengono inoltre date informazioni riguardo ai segnali di clock presenti nel progetto

Clock Information:

```
-----+-----+-----+
Clock Signal | Clock buffer(FF name) | Load |
-----+-----+-----+
clk          | BUFGP                  | 25   |
-----+-----+-----+
```

A concludere, vengono analizzate le tempistiche attraverso il così detto *Timing Summary*,

Timing Summary:

Speed Grade: -5

```
Minimum period: 35.556ns (Maximum Frequency: 28.125MHz)
Minimum input arrival time before clock: 2.827ns
Maximum output required time after clock: 7.999ns
```

Maximum combinational path delay: No path found

in cui si può vedere la massima frequenza di lavoro consentita per un corretto funzionamento, che in questo caso è 28MHz, nonchè il massimo percorso di collegamento tra gli ingressi e gli elementi sequenziali (esso caratterizza il *minimum input arrival time before clock*) ed il massimo percorso di collegamento tra gli elementi sequenziali e le uscite (esso caratterizza il *maximum output required time before clock*).

Di seguito sono dati i dettagli per questi percorsi particolari

Timing Detail:

All values displayed in nanoseconds (ns)

Timing constraint: Default period analysis for Clock 'clk'

Delay: 35.556ns (Levels of Logic = 17)

Source: xlx_i_3/i_q0

Destination: xlx_i_6

Source Clock: clk rising

Destination Clock: clk rising

Data Path: xlx_i_3/i_q0 to xlx_i_6

Gate Net

Cell:in->out fanout Delay Delay Logical Name (Net Name)

FDCE:c->q 2 1.292 1.340 i_q0 (q<0>)

end scope: 'xlx_i_3'

xor2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_8_xlx_i_1_xlx_i_2 (xlx_i_1_xlx_i_1_xlx_i_8_xlx_n_7)

and2:i1->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_8_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_8_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_8_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_7)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_7_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_7_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_7_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_6)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_6_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_6_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_6_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_5)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_5_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_5_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_5_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_4)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_4_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_4_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_4_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_3)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_3_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_3_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_3_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_2)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_2_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_2_xlx_n_15)

or2:i0->o 2 0.653 1.340 xlx_i_1_xlx_i_1_xlx_i_2_xlx_i_3 (xlx_i_1_xlx_i_1_xlx_n_1)

and2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_1_xlx_i_2_xlx_i_1 (xlx_i_1_xlx_i_1_xlx_i_1_xlx_n_15)

or2:i0->o 1 0.653 1.150 xlx_i_1_xlx_i_1_xlx_i_1_xlx_i_3 (somma<8>)

FD:d 0.753 xlx_i_6

Total 35.556ns (13.146ns logic, 22.410ns route)

(37.0% logic, 63.0% route)

Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'

Offset: 2.827ns (Levels of Logic = 1)

Source: b<3>

Destination: xlx_i_4/i_q3

Destination Clock: clk rising

Data Path: b<3> to xlx_i_4/i_q3

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:i->o	1	0.924	1.150	b_3_ibuf (b_3_ibuf)
begin scope: 'xlxi_4'				
FDCE:d		0.753		i_q3

Total		2.827ns (1.677ns logic, 1.150ns route)		(59.3% logic, 40.7% route)

Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'

Offset: 7.999ns (Levels of Logic = 1)

Source: xlx_i_5/i_q2

Destination: sum<2>

Source Clock: clk rising

Data Path: xlx_i_5/i_q2 to sum<2>

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDCE:c->q	1	1.292	1.150	i_q2 (q<2>)
end scope: 'xlxi_5'				
OBUF:i->o		5.557		sum_2_obuf (sum<2>)

Total		7.999ns (6.849ns logic, 1.150ns route)		(85.6% logic, 14.4% route)

=====
CPU : 1.35 / 1.82 s | Elapsed : 1.00 / 1.00 s

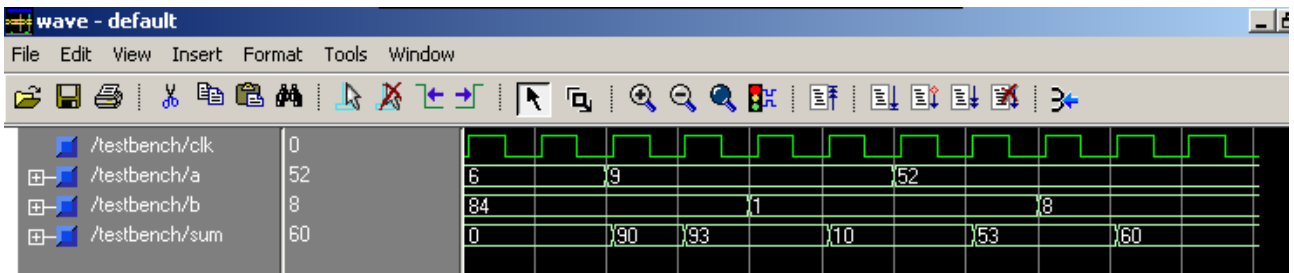
-->

Total memory usage is 48644 kilobytes

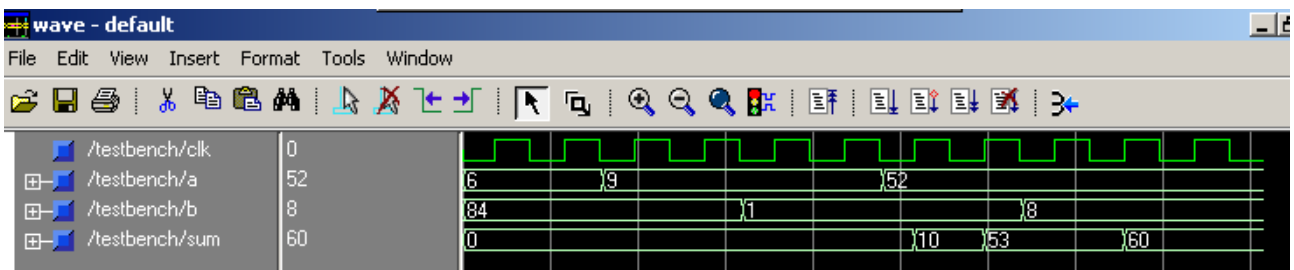
In particolare, per chiarire maggiormente, per la parte evidenziata in grassetto ad esempio si ha una situazione del genere



Si noti allora che per una corretta simulazione post sintesi (Simulate Post-Translate VHDL Model) è necessario tenere conto dei dati precedenti; infatti con un clock a 10MHz ed impostando l'Input Setup Time e l'Output Valid Delay entrambi a 10ns (quindi nei limiti consentiti) si ha una simulazione che produce un risultato corretto



Se ora però si forza un clock superiore (cliccando due volte sul segnale di test ed impostando un periodo di clock di 20ns), si ha un risultato molto diverso, come tra l'altro ci si poteva aspettare dal report di sintesi:



qui la prima e la seconda somma non vengono mai effettuate e la terza si presenta con un ciclo di clock di ritardo.

Se ora si provano a cambiare le opzioni di sintesi, modificando il livello di ottimizzazione o andando ad ottimizzare direttamente secondo lo spazio occupato (Area) e si risintetizza, si può notare che i risultati non cambiano in maniera significativa. Questo è dovuto al fatto che il progetto in questione è stato realizzato sostanzialmente a livello di architetturale, ossia secondo una struttura già ben definita e pertanto scarsamente ottimizzabile a livello di sintesi.

2.2 Sommatore con logic block

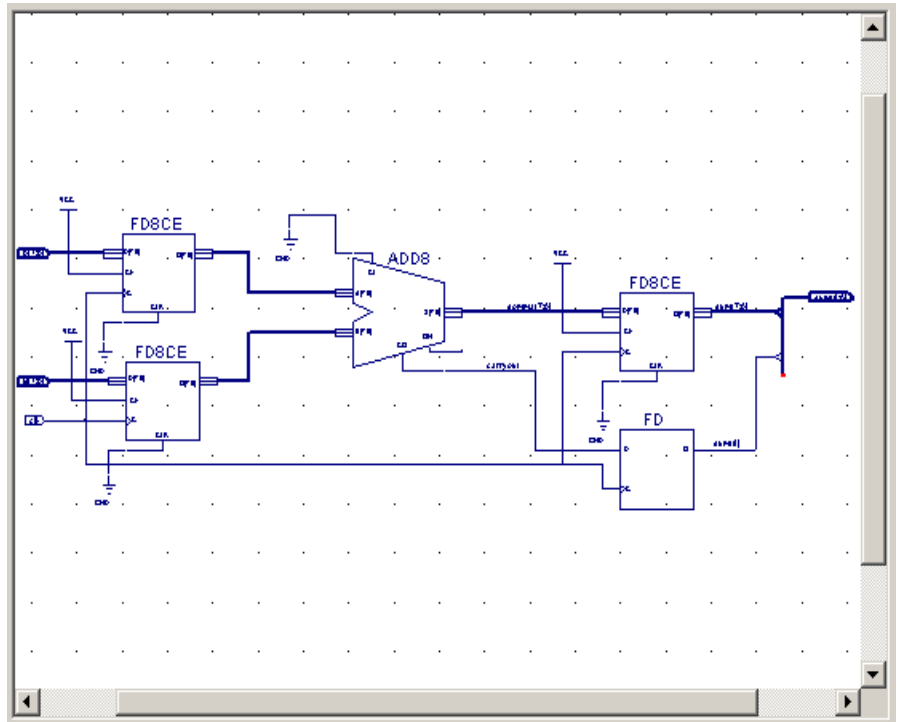
Come già visto diverse volte in precedenza,

- si crei un nuovo progetto dandogli un nome;
- si aggiunga un nuovo schematico, che creerà il file col nome, ad esempio, sommatore_lb.sch;

Nello schematic editor, sotto **Symbols** e **Arithmetic**, si selezioni il sommatore ad 8 bit denominato con *ADD8* e lo si posizioni nell'area di lavoro.

Si colleghi il carry in a massa; si colleghino le pad con gli ingressi e le uscite del blocco; a differenza del progetto precedente, qui si hanno già le uscite riunite in bus, a cui basta solo dare un nome, ricordandosi che dev'essere del tipo *<nome bus>(dim)*.

Come nello schematico precedente si aggiunga il sincronismo mettendo dei flip-flop a 8 bit in ingresso e uno a 8 bit insieme ad uno a 1 bit in uscita, riunendo poi in un unico bus a nove fili, per comodità di visualizzazione in fase di simulazione, tramite gli appositi bus-tap, gli 8 bit di somma ed il carry out, in modo che questo sia il bit meno significativo.



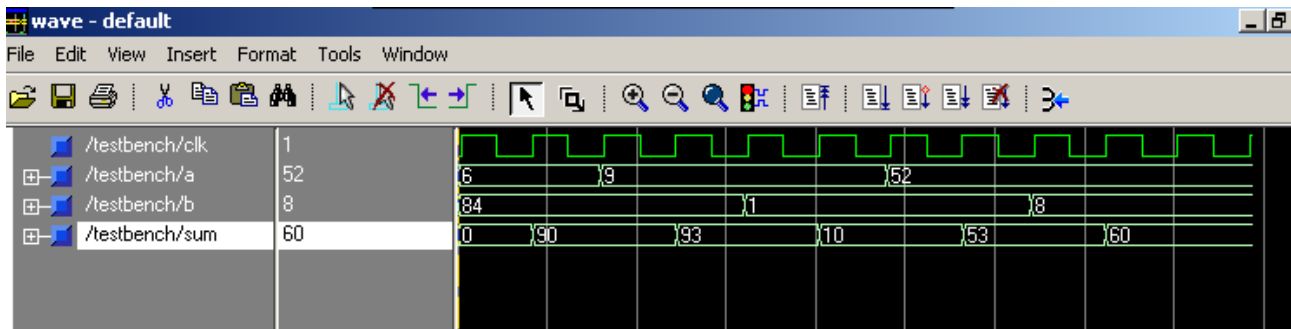
N.B. Si faccia attenzione a quando si danno i nomi ai vari fili che compongono il bus, in quanto, sbagliando le associazioni, la sintassi o dimenticando di mettere tra parentesi la dimensione, si rischia di incorrere facilmente in errori che si ripercuotono poi nel resto del progetto.

Non occorre collegare l'uscita del carry overflow.

Si verifichi la correttezza dello schematico e si salvi il tutto.

Come nel caso del progetto precedente, si associ allo schematico un nuovo segnale di test che, per comodità sia identico a quello utilizzato nel sommatore con i fulladder.

Al termine si salvi e si lanci la simulazione comportamentale: anche in questo caso si può verificare che il modello funziona come dovrebbe e peraltro il risultato è identico.



Si reimpostino le opzioni di sintesi con ottimizzazione secondo il criterio della velocità ed il livello sia alto. Si lanci quindi la sintesi e si analizzi il report.

Si noti come questa volta si abbia un numero di dispositivi utilizzati diverso

Cell Usage :

```
# BELS : 33
# GND : 5
# muxcy : 1
# muxcy_d : 1
# muxcy_l : 6
# vcc : 3
# xor2 : 9
# xorcy : 8
# FlipFlops/Latches : 25
# FD : 1
# FDCE : 24
# Clock Buffers : 1
# BUFGP : 1
# IO Buffers : 25
# IBUF : 16
# OBUF : 9
# Others : 8
# fmap : 8
```

Device utilization summary:

Selected Device : 2s50tq144-5

```
Number of Slices:          20 out of 768  2%
Number of Slice Flip Flops: 25 out of 1536  1%
Number of bonded IOBs:    25 out of 96  26%
Number of GCLKs:          1 out of 4  25%
```

l'occupazione è leggermente superiore al precedente progetto e questo è dovuto alla diversa tecnica impiegata per il sommatore.

Fatto rilevante è dato dalla frequenza di lavoro: il dispositivo, essendo esso stesso un elemento di libreria e quindi ottimizzato nel suo complesso, risulta molto più veloce di quello impiegato nel progetto precedente; in questo caso infatti ci si può permettere un clock che può arrivare ai 130MHz! In particolare l'elevata frequenza è sostanzialmente legata al fatto che il sommatore così realizzato sfrutta quelle risorse integrate nei dispositivi Xilinx per una veloce propagazione del Carry.

Timing Summary:

Speed Grade: -5

Minimum period: 7.732ns (Maximum Frequency: 129.333MHz)

Minimum input arrival time before clock: 2.827ns

Maximum output required time after clock: 7.999ns

Maximum combinational path delay: No path found

Nuovamente, come nel caso precedente, se si modifica il tipo di ottimizzazione e si risintetizza, i risultati non cambiano in maniera sostanziale.

Suggerimento:

Si suggerisce al lettore di provare a realizzare un sommatore a livello puramente "comportamentale" ad esempio realizzando in VHDL un sommatore descritto dal processo:

```
sum <= a + b
```

ed a ripetere su di esso le fasi di sintesi modificando opportunamente i vincoli e le specifiche.