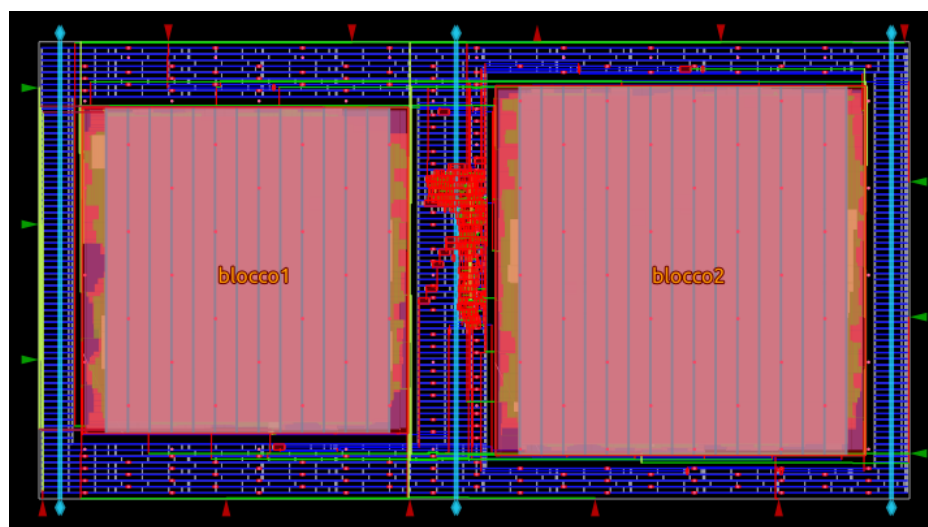


Struttura del chip

Organizzeremo il chip nel modo seguente: i blocchi *iir* ed *enfasi* verranno implementati autonomamente, in modo da ottenere due macroblocchi (più sinteticamente, macro). Le due macro saranno poi utilizzate nel chip complessivo in cui saranno anche presenti le celle relative al blocco *detector*. La figura seguente mostra il risultato finale cui vogliamo pervenire, in cui blocco1 si riferisce alla macro *iir*, mentre blocco2 alla macro *enfasi*:



Fasi di backend

Partiamo dalla descrizione del backend per la macro *iir* (per la macro *enfasi* i passi sono analoghi). I passi che seguiremo sono elencati di seguito:

1. Sintesi
2. Floorplan
3. Placement
4. Clock tree
5. Routing
6. Controlli finali

Tecnologia e libreria di celle standard

Il progetto verrà realizzato utilizzando una tecnologia a 45nm denominata FreePDK45. Questa tecnologia non corrisponde ad un processo costruttivo commerciale, ma è un “kit di progettazione” (*process design kit*, PDK) sviluppato in ambito universitario che consente a ricercatori e studenti di sperimentare la progettazione su un nodo tecnologico moderno senza dover firmare accordi di non divulgazione restrittivi o pagare per le licenze. La tecnologia prevede 10 livelli di metal. Tutte le informazioni sono disponibili al sito: <https://eda.ncsu.edu/freepdk/freepdk45/>

La Tabella seguente riporta dimensione minima e spaziatura minima dei vari livelli di metal

Metal	Dimensione minima	Spaziatura minima
metal1	65nm	65nm
metal2, metal3	70nm	70nm
metal4, metal6	65nm	65nm
metal7, metal8	140nm	140nm
metal9, metal10	400nm	400nm

La libreria di celle standard è denominata FreePDK45 ed è anch'essa disponibile liberamente; uno dei siti che ne ospita i files è:

<https://github.com/JulianKemmerer/Drexel-ECE575/tree/master/Encounter/NangateOpenCellLibrary>

Il databook che riporta le caratteristiche delle celle standard è disponibile nel file: `/vlsi/tech/stdcells-databook-1.pdf` (potete visualizzare il file pdf con il programma *evince*). Il file di libreria (con le indicazioni di timing, power ecc.) è:

`/vlsi/tech/nangate45/lib/NangateOpenCellLibrary_typical.lib`

Aprire il file con visual studio code, facendo attenzione a NON MODIFICARLO. Il file è diviso in più parti: una prima parte dichiarativa, ed una seconda parte contenente la descrizione delle celle standard. Nella parte dichiarativa vengono fissate le unità di misura, i valori di tensione, temperatura e processo con i quali sono state ottenute le prestazioni delle celle, e le tabelle del wireload model.

La seconda parte del file, invece, contiene una lista di tutte le celle standard che possono essere utilizzate nella sintesi. Utilizzando il menù "Edit -> Find" (o semplicemente Ctrl+F) cercate nel file la parola "NAND2". Troverete diverse celle: NAND2_X1, NAND2_X2 ecc. ecc. Tutte realizzano la funzione nand a due ingressi; le celle si differenziano per le dimensioni dei dispositivi (driving strength): le celle con drive strength maggiore sono in grado di pilotare carichi più elevati con minori tempi di commutazione, al prezzo di maggiore area e potenza. Potete notare che la libreria utilizza il modello non-lineare (tabellare) per la caratterizzazione dei ritardi.

Sintesi del blocco iir

Posizioniamoci nella cartella backend del blocco iir; vi troveremo alcune sottocartelle: **src** **scripts** **results**. Nella cartella **src** dovete copiare i files systemverilog del blocco (i testbench NON DEVONO ESSERE COPIATI, NON SERVONO in questa fase – non sono sintetizzabili). La cartella include anche il file con i constraints, denominato *constraints.sdc*. Apritelo con visual studio code. Il contenuto è il seguente:

```

1  # file di vincoli
2
3  set_cmd_units -time ns -capacitance fF
4
5  create_clock -name my_clock -period 10 clk
6  set_clock_transition 0.050 my_clock
7  # jitter: 100ps
8  set_clock_uncertainty -setup -hold 0.100 my_clock
9
10 set_input_delay -clock my_clock 0.500 [all_inputs -no_clocks]
11 set_output_delay -clock my_clock 0.500 [all_outputs ]
12
13 set_load 3.9 [all_outputs]
14 set_driving_cell -lib_cell BUF_X1 [all_inputs]

```

Abbiamo avuto modo di descrivere i vari comandi che compaiono in questo file durante le lezioni del corso (set di slides: Lez012-Vincoli-temporizzazione).

Nella cartella **results** salveremo i risultati delle varie fasi del backend.

Nella cartella **scripts** sono contenuti i comandi che servono durante le varie fasi.

Per effettuare i vari step richiameremo i tools [rimando nella directory backend](#). Partiamo dalla sintesi. Apriamo con visual studio code il file *scripts/sintesi.tcl* in modo da avere sott'occhio i comandi da inviare al sintetizzatore. I comandi sono molto semplici: dapprima si legge il contenuto della libreria di celle standard (*read_liberty*), poi si leggono i files systemverilog (*read_slang*), quindi si effettua una sintesi generica (*syn_generic* non riferita ad una particolare tecnologica) infine si effettua il *mapping* utilizzando le celle di libreria (*syn_map*). I nomi dei vari files da utilizzare sono definiti mediante variabili, cui si attribuisce un valore con il comando **set**

```

1  # sintesi.tcl
2  # Script per la sintesi
3
4  set VERILOG_FILES {src/iir_section1.sv src/iir_section2.sv src/iir.sv}
5  set TOP_MODULE iir
6  set LIB_FILE /vlsi/tech/nangate45/lib/NangateOpenCellLibrary_typical.lib
7  set CONSTRAINTS_FILE ./src/constraints.sdc
8
9  read_liberty $LIB_FILE
10 read_slang {*} $VERILOG_FILES
11
12 # Sintesi generica. Il comando richiede un solo parametro:
13 # il nome del modulo top-level
14 syn_generic $TOP_MODULE
15
16 # Standard-cell mapping ed ottimizzazione.
17 # Il comando richiede un solo parametro:
18 # il nome del file di constraints
19 syn_map $CONSTRAINTS_FILE
20
21 report_area
22 report_timing
23 write_netlist results/iir_netlist.v
24 report_area results/area_report.txt
25 report_timing results/timing_report.txt

```

I comandi in linea 21-25 mostrano a video o salvano su file dei report relativi alle celle standard utilizzate ed al timing. Il comando in linea 23 salva la netlist sintetizzata nella cartella **results**.

Richiamate il sintetizzatore digitando nella finestra di comandi **myosys**. Inviare quindi i vari comandi come riportati nel file (a tal fine potete copiare riga per riga il contenuto del file dall'editor ed incollarlo nella finestra di comandi – ricordate che selezionando del testo esso viene automaticamente copiato, e che per incollarlo è sufficiente premere il tasto centrale del mouse).

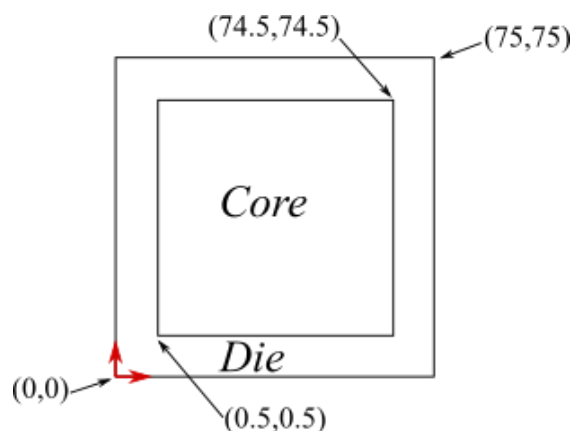
Al termine della sintesi, prendete nota dell'*area complessiva delle celle standard* – ci servirà come punto di partenza per il floorplan.

Floorplanning

I comandi per il floorplanning sono contenuti nel file **scripts/floorplan.tcl**. Aprite il file con visual studio code in modo da avere i vari comandi sottomano. Nella finestra del terminale invocate il programma che ci consentirà di completare il progetto del chip: **openroad**

Bisogna preliminarmente leggere alcuni files: la netlist sintetizzata in precedenza, il file di libreria delle celle standard, il file di constraints. Bisogna inoltre caricare due files con estensione .lef che riportano informazioni circa la dimensione e la posizione dei pin delle celle standard ed alcune caratteristiche della tecnologia, relativamente ai livelli di metal ed alle vias (distanza minima fra due linee di metal, resistenza per unità di lunghezza ecc. ecc)

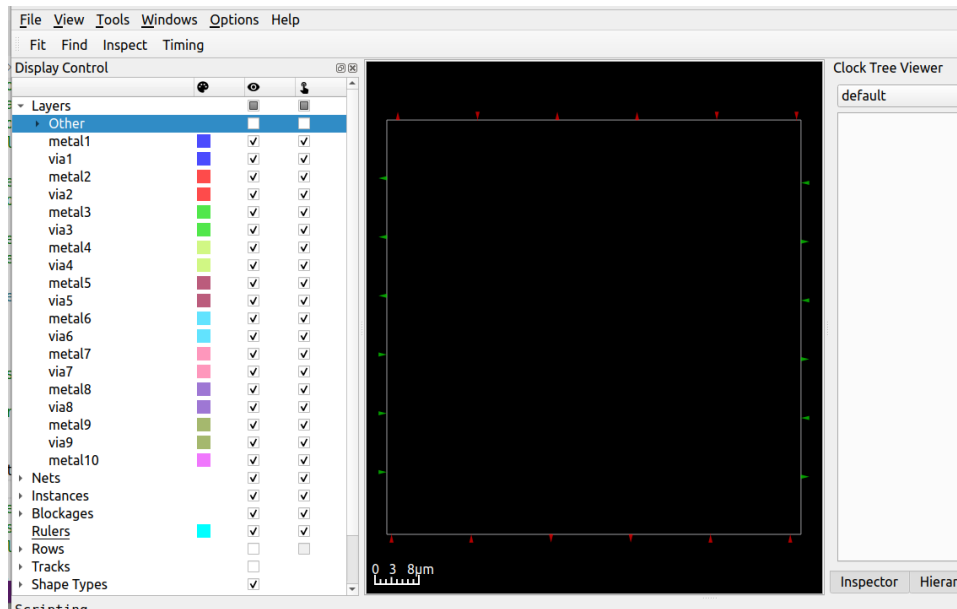
Dopo questa fase preliminare, si effettua il comando **initialize_floorplan** che definisce la regione in cui allocare il chip. In particolare, è necessario definire le coordinate di quello che viene definito il *core* (dove verranno alloggiare le celle standard) ed il *die* (dove arriveranno i terminali del circuito). Il die è più grande del core, includendo una zona "cuscinetto", priva di celle standard, attorno al core. La Figura seguente mostra i parametri inseriti nel file, che prevedono la realizzazione di un macroblocco quadrato (le coordinate sono espresse in micron).



Prendete nota della *percentuale di utilizzo*, data dal rapporto fra $\text{Area_Celle_Standard} / \text{Area_Core}$.

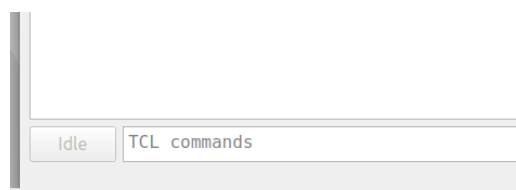
Si procede quindi al piazzamento dei pin sul bordo del die, con il comando **place_pin**. Lasciamo al programma la libertà di piazzare casualmente i pin. Utilizziamo metal3 per i pin posti sui lati est ed ovest del die, e metal2 per quelli posti sui lati nord e sud.

A questo punto potete attivare la finestra grafica del programma, con il comando **gui::show**. Compare una finestra come quella in Figura, che evidenzia al momento solo il contorno del die e la posizione dei pin (in rosso i pin di ingresso, in verde quelli di uscita)



La parte a sinistra della finestra grafica consente di evidenziare gli oggetti che si intende visualizzare. Provate ad attivare la visione delle *rows*, cliccando sul corrispondente quadratino: si evidenzieranno le righe in cui verranno allocate le celle standard. Le *tracks*, invece, sono “i binari” su cui verranno alloggiati i collegamenti orizzontali e verticali. Potete usare il tasto destro del mouse per fare uno zoom, mentre con il menù View=>Fit vediamo l’intero die.

Possiamo inserire i successivi comandi all’interno della finestra “*TCL commands*” dell’interfaccia grafica:



Con il comando *tapcell* inseriamo su ogni riga delle celle particolari, che non hanno funzione logica, ma sono indispensabili per realizzare i collegamenti di substrato.

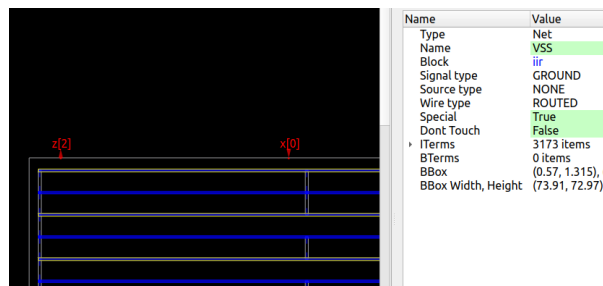
Passiamo quindi a definire la griglia di alimentazione. Alcuni comandi preliminari servono a definire i nomi dei segnali di alimentazione e massa ed il nome che si intende assegnare alla griglia di alimentazione. Dopo averli eseguiti, eseguiamo il comando:

add_pdn_stripe -grid power_grid -layer metal1 -followpins

Eseguiamo quindi il comando:

pdngen

Vengono generate delle linee orizzontali in metal1 (azzurro) che rappresentano i collegamenti di alimentazione e massa delle celle standard. Provate a fare uno zoom su una parte del chip ed a selezionare una delle linee – noterete che vengono selezionate anche altre linee collegate elettricamente, mentre nella finestra a destra compariranno le indicazioni relative al nome della linea selezionata:



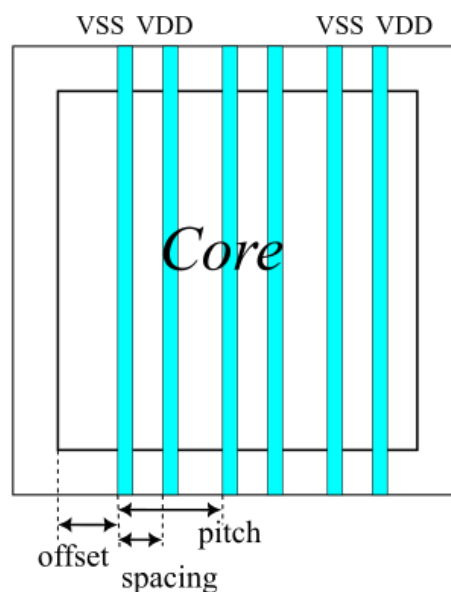
Aggiungiamo ora delle strisce verticali di alimentazione, che serviranno a collegare fra loro le alimentazioni delle righe delle celle, riducendo la resistenza complessiva della rete di alimentazione. Poiché le linee orizzontali di alimentazione sono in metal1, quelle verticali dovranno utilizzare un livello di metal differente. Usiamo metal6 (in modo da consentire l'utilizzo indisturbato dei livelli di metal da 2 a 5 per completare i collegamenti). I comandi sono i seguenti:

```
add_pdn_stripe -grid power_grid -layer metal6 -width 0.56 -pitch 10 -offset 5 -extend_to_boundary
```

```
add_pdn_connect -grid power_grid -layers {metal1 metal6}
```

pdngen

Il comando `add_pdn_stripe` definisce la geometria delle strisce verticali di alimentazione secondo la Figura seguente. Nel nostro caso non abbiamo specificato lo *spacing*, assunto di default uguale alla metà del *pitch*:



Gli ultimi comandi del file ci forniscono una indicazione dell'occupazione di area, mentre il comando **write_db** consente di salvare nella directory dei risultati il floorplan appena realizzato.

Placement

I comandi per il placement sono contenuti nel file **scripts/placement.tcl**. Anche in questo caso – come faremo in seguito – conviene editare il file per vedere i comandi da eseguire di volta in volta. Dalla finestra del terminale invochiamo nuovamente **openroad**

Procediamo alla lettura dei files di libreria ed anche al caricamento del floorplan salvato in precedenza (comando **read_db results/floorplan.odb**).

Il placement si sviluppa in due step. Il primo è il global placement, che definisce una posizione iniziale, di tentativo, per le varie celle. Dopo aver eseguito il comando corrispondente:

global_placement -density 0.75

attiviamo la finestra grafica con **gui::show**. Noterete che le celle sono posizionate in maniera ancora approssimativa, senza rispettare l'allocazione sulle righe e sovrapponendosi parzialmente. Questo piazzamento rozzo consente comunque di avere una prima stima della lunghezza delle interconnessioni che andremo a realizzare in seguito. Eseguiamo quindi i comandi:

estimate_parasitics -placement

set_propagated_clock clk

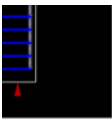
Possiamo ora richiamare il programma di analisi statica dei ritardi, per vedere se i vincoli di timing sono rispettati:

report_checks -path_delay max -digits 3 -format full_clock_expanded -field capacitance

report_checks -path_delay min -digits 3 -format full_clock_expanded

Il primo report fornisce l'analisi per il ritardo massimo (vincolo di setup). Abbiamo un notevole margine e quindi non avremo problemi a rispettare questo vincolo anche dopo la fase di routing.

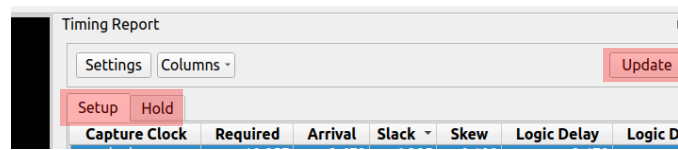
Il secondo report_checks si riferisce invece al ritardo minimo (vincolo di hold). In questo caso il margine (ammesso sia positivo) è molto ridotto. Conviene pertanto inserire dei buffer in modo da avere un slack maggiore, che ci rassicuri sul fatto che il vincolo di hold venga rispettato anche dopo il routing. Prima di effettuare questa operazione usiamo la finestra grafica di openroad per vedere la posizione delle celle che compongono i vari cammini. Nella parte in basso a destra della gui, selezionate "Timing report"



3065/ZN (XNOR2_X1)	4	↑	0.547	0.055
3066/B (XNOR2_X1)		↑	0.547	0.000
3066_/ZN (XNOR2_X1)	2	↑	0.598	0.051

Inspector Hierarchy Browser Timing Report Charts

Quindi, nella parte a destra in alto, cliccate "Update"



Cliccando a questo punto sul pulsante “Setup” vengono riportati i vari cammini, mentre nella schermata centrale della gui si evidenziano le celle attraverso cui si instrada il cammino selezionato.

Passiamo ora a migliorare il vincolo sul tempo di hold. Il comando proposto è:

repair_timing -hold -hold_margin 0.10 -verbose

L’esecuzione del comando richiede qualche secondo, dopo i quali il tool informa che sono stati inseriti numerosi buffer. Eseguendo la verifica sul tempo di hold, osserviamo che effettivamente lo slack è aumentato, portandosi a non meno di 100ps.

Dobbiamo ora provvedere alla seconda fase del piazzamento delle celle, denominata *detailed placement*. A tal fine, eseguiamo il comando:

detailed_placement -max_displacement 20

con il quale si richiede di effettuare il piazzamento, spostando le celle entro un raggio di 20um rispetto alla posizione approssimativa stabilita dal global placement. Notate il sintetico report fornito dal tool, che riporta un parametro denominato HPWL (half-perimeter wirelength) che rappresenta una stima di quella che sarà la lunghezza complessiva dei futuri collegamenti (tanto minore HPWL, tanto migliore il placement). Il programma evidenzia che, avendo “legalizzato” il placement (posizionando le celle nelle righe, evitando sovrapposizioni delle celle) si è evidenziato un certo aumento del HPWL. È possibile migliorare il placement con i due comandi successivi:

optimize_mirroring

improve_placement -max_displacement 10

Il primo comando prova a ruotare di 180 gradi alcune celle e riesce a migliorare solo di qualche punto percentuale il valore di HPWL. Il secondo comando, invece, prova a spostare nuovamente le celle al fine di individuare un miglior piazzamento complessivo. L’**improve_placement** produce risultati migliori – in alcuni casi aiuta a legalizzare un piazzamento quando il *detailed_placement* fallisce. Provate ad eseguire nuovamente l’*improve_placement*, noterete che i successivi miglioramenti di HPWL divengono marginali.

Procediamo a questo punto al salvataggio del progetto, da cui ripartiremo per effettuare la sintesi dell’albero di clock:

write_db results/placement.oddb

Generazione dell'albero di clock

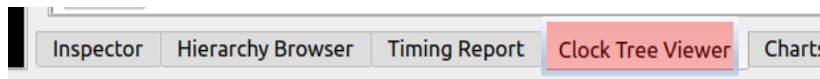
I comandi sono nel file **scripts/cts.tcl**. Dopo il caricamento dei files di libreria si procede al caricamento del floorplan salvato in precedenza (comando **read_db results/placement.odt**).

Invochiamo quindi il comando che genera l'albero di clock:

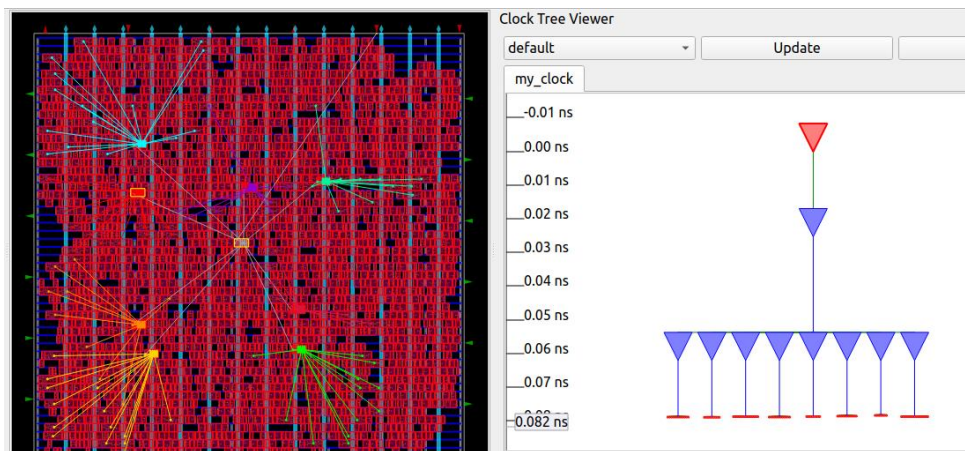
```
clock_tree_synthesis -buf_list {BUF_X8 BUF_X4 BUF_X2 BUF_X1} -root_buf BUF_X8
```

Il comando successivo **report_cts** ci fornisce alcune informazioni sull'albero di clock (come il numero di buffer inseriti ed il numero di "Sinks", ovvero di punti in cui il clock deve essere distribuito che, nel nostro caso, coincide con il numero complessivo di flip-flop). I comandi **report_clock_latency** e **report_clock_skew** ci danno invece indicazioni sul ritardo complessivo e sullo skew.

Attiviamo la gui e, nella parte destra della finestra grafica, in basso, evidenziamo "Clock Tree Viewer"



Clicchiamo quindi il tasto "Update" nella parte a destra in alto. Compare una vista grafica dell'albero di clock, mentre nella parte centrale della finestra grafica viene evidenziato il piazzamento dei buffer



In questo semplice circuito il generatore dell'albero di clock ha suddiviso gli 88 flip flop in 8 gruppi, ognuno dei quali è pilotato da un buffer. L'insieme degli 8 buffer è a sua volta pilotato da un altro buffer, posizionato all'incirca al centro del chip. In questo modo (mancando ancora informazioni relative ai ritardi introdotti dalle interconnessioni) l'albero di clock è perfettamente bilanciato e lo skew che viene riportato è esclusivamente quello legato alla *clock uncertainty* definita nel file di constraints.

Con i successivi comandi provvediamo nuovamente a verificare lo slack sul tempo di hold, che è cambiato rispetto al controllo precedente, sia per la presenza dell'albero di clock, sia perché il placement è stato legalizzato. Il comando **repair_timing -hold** aggiunge ulteriori buffer per riportare nuovamente lo slack ad almeno 100ps.

Si deve ora legalizzare nuovamente il piazzamento (avendo aggiunto i buffer del clock e quelli per il vincolo di hold) per poi salvare il progetto in vista della fase successiva di routing.

Routing

I comandi sono nel file ***scripts/route.tcl***. Si procede, come di consueto, al caricamento dei files di libreria e del progetto salvato in precedenza (comando **`read_db results/cts.odb`**).

Con il comando **`set_routing_layers -signal metal1-metal5 -clock metal1-metal4`** definiamo i livelli di metal da usare per completare il routing del progetto. Anche la fase di routing (come il placement) si compone di due step: *global routing* e *detailed routing*. Nella fase di global routing di definiscono dei collegamenti di tentativo fra i vari pin, senza preoccuparsi di eventuali violazioni di regole di progetto (due linee dello stesso livello di metal troppo vicine fra loro ecc. ecc.). Al termine del global routing si ha una indicazione abbastanza attendibile della lunghezza delle interconnessioni ed è possibile stimare con buona accuratezza i ritardi che introdurranno. La fase di detailed routing provvede a legalizzare il routing, in modo da ottenere il risultato finale.

Dopo il global routing controlliamo nuovamente i vincoli di setup e di hold – eventualmente si potrebbe effettuare in questa fase un ultimo aggiornamento del progetto per rispettare i vincoli assegnati. Nel nostro caso questo non è necessario e possiamo quindi passare alla fase di detailed routing. Questa è la fase più lunga del backend e può richiedere alcuni minuti; noterete che il programma effettua alcune iterazioni durante le quali riduce le violazioni fino a portarle a 0.

Completato il routing rimangono degli spazi vuoti nei canali delle celle che vanno riempiti con opportune celle “filler” (comando **`filler_placement`**). Dopo alcuni controlli finali, il layout viene salvato per le successive analisi.

Controlli finali

I comandi sono nel file ***scripts/final.tcl***. Dopo aver caricato i vari files iniziali, si valutano i parametri parassiti delle interconnessioni con il comando **`extract_parasitics`**. Viene quindi generato un file che riporta resistenze e capacità delle interconnessioni (comando **`write_spef`**); i parametri parassiti vengono quindi re-importati (**`read_spef`**). Viene poi effettuata un’ultima analisi dei ritardi che, grazie all’inserimento dei buffer effettuato in precedenza, non mostra alcuna violazione; viene inoltre realizzata una stima della dissipazione di potenza.

Apriamo la gui ed analizziamo l’albero di clock. Noterete che i tempi di arrivo del clock ai vari flip-flop mostrano una certa variabilità, introdotta dai parametri parassiti delle interconnessioni.

Lo script provvede, infine, a salvare i risultati finali del progetto: i files **`lef`** e **`def`** riportano delle informazioni geometriche sulle interconnessioni presenti nel chip, mentre il file **`lib`** riporta i dati relativi al timing. L’ultimo comando serve per modificare il file **`lef`**, in modo da renderlo compatibile per il successivo utilizzo del sistema appena progettato come macro nel chip complessivo.

Blocco enfasi.

Il secondo macroblocco del progetto è realizzato seguendo esattamente gli stessi passi visti per il blocco iir. Al termine, anche per il blocco enfasi dovranno essere disponibili i files *lef*, *def*, e *lib*.

Top level del progetto.

Per completare il progetto, posizioniamoci nella cartella backend del blocco *detector*; nella cartella **src** dovete copiare i files systemverilog del blocco.

In questa cartella troverete anche un file denominato **top.sv**. Dovete completare il file inserendo l'istanza del blocco *detector* e dei suoi componenti, dovete inoltre istanziare anche i due macroblocchi progettati in precedenza (*iir* ed *enfasi*). **NOTA:** i due macroblocchi devono essere istanziati senza utilizzare la sintassi sintetica *.** del systemverilog ma utilizzando invece la sintassi estesa, ad esempio: **iir blocco1 (.clk(clk),)** L'editor mostrerà degli errori in corrispondenza delle istanze di *iir* ed *enfasi* - questo non è un problema, in quanto noi non intendiamo utilizzare una descrizione systemverilog di questi blocchi ma intendiamo invece istanziarli nel progetto finale come macro.

Sempre nella cartella backend troverete altre due sottocartelle (**iir** ed **enfasi**): copiate in queste due cartelle i files *lef def* e *lib* delle due macro.

Procediamo quindi alla sintesi del top level. Il file di comandi *scripts/sintesi.tcl* è riportato di seguito:

```
1  # sintesi.tcl
2  # Script per la sintesi
3
4  set VERILOG_FILES {src/top.sv src/detector.sv src/fsm.sv src/ser_par.sv}
5  set TOP_MODULE top
6  set LIB_FILE /vlsi/tech/nangate45/lib/NangateOpenCellLibrary_typical.lib
7  set CONSTRAINTS_FILE ./src/constraints.sdc
8
9  read_liberty $LIB_FILE
10 read_liberty iir/iir.lib
11 read_liberty enfasi/enfasi.lib
12
13 # Notare l'opzione aggiuntiva, necessaria per i blocchi iir ed enfasi
14 read_slang --ignore-unknown-modules {*} $VERILOG_FILES
15
16 # Sintesi generica. Il comando richiede un solo parametro:
17 # il nome del modulo top-level
18 syn_generic $TOP_MODULE
19
20 # Standard-cell mapping ed ottimizzazione.
21 # Il comando richiede un solo parametro:
22 # il nome del file di constraints
23 syn_map $CONSTRAINTS_FILE
24
25 report_area
26 report_timing
27 write_netlist results/top_netlist.v
28 report_area results/area_report.txt
29 report_timing results/timing_report.txt
```

Rispetto ai files di sintesi utilizzati in precedenza notiamo che si procede alla lettura dei file *lib* delle due macro:

```
read_liberty iir/iir.lib
read_liberty enfasi/enfasi.lib
```

Inoltre, nel richiamare i files systemverilog del progetto, si utilizza una opportuna opzione relativa ai due moduli che utilizzeremo come macro:

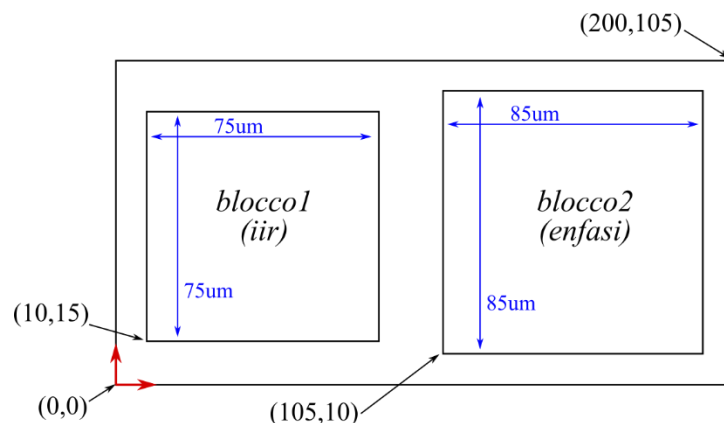
```
read_slang --ignore-unknown-modules {*} $VERILOG_FILES
```

Gli altri comandi sono analoghi a quelli utilizzati in precedenza. La sintesi si conclude con la generazione della netlist che viene salvata nella cartella *results*.

Floorplanning – prima parte

La fase di floorplan del top level è un po' più complessa a causa della presenza delle macro – la dividiamo pertanto in due parti. I comandi relativi alla prima parte sono nel file *scripts/floorplan1.tcl*

In openroad, dopo la lettura dei vari files (fra cui quelli *lib* e *lef* delle due macro) si provvede ad inizializzare il floorplan. Considerate le dimensioni delle due macro, la struttura cui si vuole pervenire è riportata in Figura (è mostrato solo il *die*, per semplicità – il *core* è più piccolo di 0.5um):



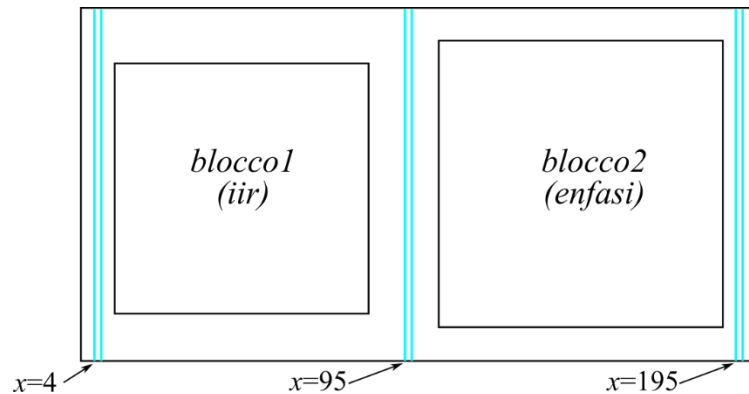
I comandi relativi sono:

```
initialize_floorplan -die_area {0 0 200 105} -core_area {0.5 0.5 199.5 104.5} \
                    -site FreePDK45_38x28_10R_NP_162NW_340
source $TRACK_SCRIPT
place_pins -hor_layer metal3 -ver_layer metal2 -random
place_macro -macro_name blocco1 -location {10 15}
place_macro -macro_name blocco2 -location {105 10}
```

Procediamo quindi al posizionamento delle tapcell ed al salvataggio di questa prima parte del floorplan.

Floorplanning – seconda parte

I comandi sono nel file ***scripts/floorplan2.tcl*** e servono alla realizzazione della power grid. Realizzeremo una power grid un po' più elaborata rispetto agli esempi precedenti. Innanzitutto, utilizziamo delle stripes in metal 6 per assicurare il collegamento all'alimentazione delle righe che si trovano a sinistra del primo blocco, a destra del secondo e nella zona compresa fra i due blocchi. Da notare che non possiamo utilizzare metal6 per passare sopra le due macro, dato che le macro utilizzano esse stesse il metal6. La situazione cui si vuol pervenire è mostrata in Figura:



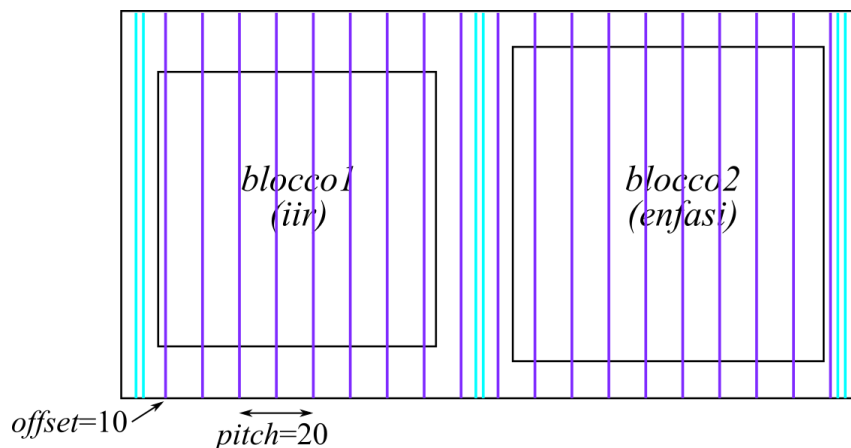
Allo scopo, utilizziamo i comandi seguenti in cui si include un pitch molto grande, in modo che ogni comando generi una sola coppia di stripes:

```
add_pdn_stripe -grid power_grid -layer metal6 -width 0.28 -spacing 0.28 -offset 4 -pitch 200 -extend_to_boundary
add_pdn_stripe -grid power_grid -layer metal6 -width 0.28 -spacing 0.28 -offset 95 -pitch 200 -extend_to_boundary
add_pdn_stripe -grid power_grid -layer metal6 -width 0.28 -spacing 0.28 -offset 195 -pitch 200 -extend_to_boundary
```

Procediamo aggiungendo ulteriori stripes verticali in metal8. Dato che il metal8 non è utilizzato nelle macro, queste stripes possono estendersi su tutto il chip senza alcun vincolo particolare:

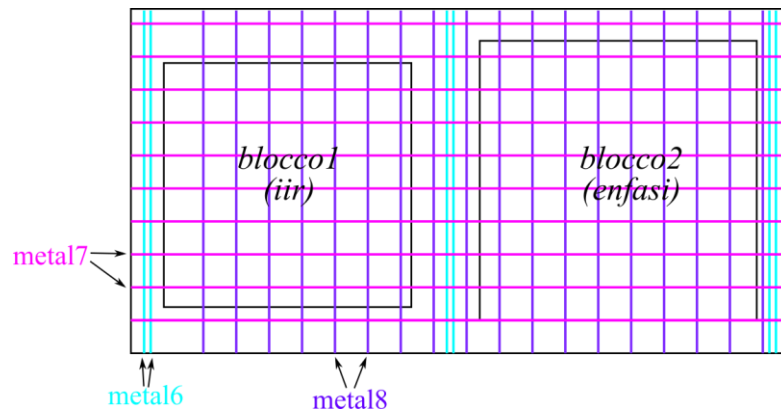
```
add_pdn_stripe -grid power_grid -layer metal8 -width 1.2 -pitch 20 -offset 10 -extend_to_boundary
```

Otteniamo così la situazione in Figura:



Infine, aggiungiamo delle stripes orizzontali in metal7. Anche queste strisce possono estendersi sull'intero chip:

```
add_pdn_stripe -grid power_grid -layer metal7 -width 1.2 -pitch 20 -offset 10 -extend_to_boundary
```



I comandi successivi (**add_pdn_connect**) servono a collegare le strisce orizzontali con quelle verticali.

Dobbiamo ora collegare la griglia principale di alimentazione con quella delle due macro. Allo scopo possiamo utilizzare le stripes orizzontali in metal7 che possono essere collegate con quelle in metal6 delle macro. I comandi da utilizzare sono:

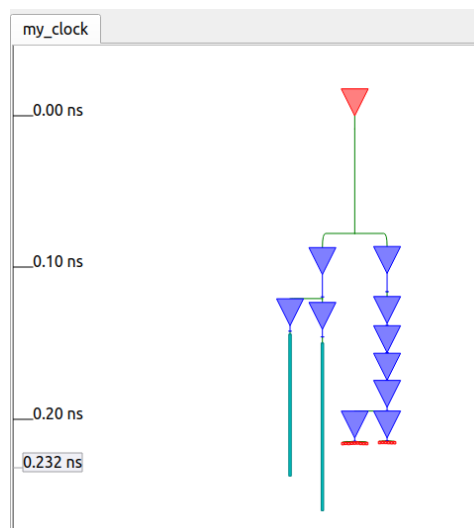
```
define_pdn_grid -name macro_grid -voltage_domains CORE -macro -cells {.*}  
add_pdn_connect -grid macro_grid -layers {metal6 metal7}
```

Completata la griglia di alimentazione con il comando **pdngen**, provvediamo a salvare il risultato del floorplan ed a verificare con la gui il risultato ottenuto.

Placement, clock tree, routing, controlli finali

Questi passi sono del tutto analoghi a quelli effettuati per i due macroblocchi.

Al termine del progetto è interessante verificare il clock tree totale, mostrato nella Figura seguente:



Per pilotare i pochi flip-flop del top level, il tool ha inserito una cascata di sei buffer al fine di rendere il ritardo complessivo sul clock paragonabile a quello dei due macroblocchi.