

TUTORIAL PARA SÍNTESE STANDARD-CELLS UTILIZANDO CADENCE

Matheus Moreira – Ricardo Guazzelli – Leonardo Rezende- Fernando Moraes
Atualizado em - **10/junho/2021**

Arquivos do projeto (detector de padrão) com ambiente de síntese e simulação

Passos para gerar o ambiente de trabalho para esse tutorial:

- ▲ Conectar-se ao servidor **kriti**
“ssh -X <usuário>@kriti.inf.pucrs.br”
- ▲ Baixar o arquivo de distribuição:
wget http://www.inf.pucrs.br/moraes/testa_padrao.tar --no-check-certificat
- ▲ Descompactar o arquivo: **“tar -xvf testa_padrao.tar”**
- ▲ Ir para a raiz do projeto : **“cd testa_padrao”**
- ▲ Carregar as ferramentas necessárias: **“source /soft64/source_gaph”**
“module load xcelium genus innovus”

Abaixo está a estrutura da distribuição, a qual contém quatro diretórios:

- ▲ constraint – diretório que contém as restrições de projeto
- ▲ rtl – diretório que contém a descrição em VHDL do projeto
- ▲ sim – diretório que contém os ambientes de simulação para as diferentes etapas do projeto
- ▲ synthesis – diretório que contém o ambiente de síntese do projeto

Para um novo circuito, deve-se editar os *scripts* de síntese e simulação. Apenas os arquivos marcados com “não alterar” são independentes do circuito a ser sintetizado.

Arquivos contidos na distribuição:

| | |
|---------------------------|---|
| -- constraint | |
| `-- busca_padrao.sdc | Restrições de timing particulares a cada circuito |
| -- rtl | |
| `-- busca_padrao.vhd | Código VHDL do circuito |
| -- sim | |
| -- rtl | |
| `-- file_list.f | Script de simulação RTL |
| -- sdf | |
| `-- file_list.f | Script de simulação com atraso de fio |
| `-- sdf_cmd.cmd | Script que diz qual o arquivo com os atrasos |
| -- synth | |
| `-- file_list.f | Script de simulação com atraso unitários após síntese lógica |
| `-- tb | |
| `-- tb_padrao.vhd | Test bench utilizado pelas 3 simulações |
| -- synthesis | |
| -- Clock.ctstch | Definições do <i>clock</i> para a geração da árvore de <i>clock</i> |
| -- comandos_gennus.txt | Script para síntese lógica |
| -- load.tcl | Não alterar, configuração da tecnologia – não depende do circuito |
| `-- physical | |
| `-- 1_init.tcl | Alterar a 1ª linha para apontar para o arquivo de configuração |
| `-- 2_power_plan.tcl | Definição do roteamento de alimentação e polarização - não alterar |
| `-- 3_pin_clock.tcl | Posicionamento dos pinos de E/S e geração da árvore de clock |
| `-- 4_nano_route.tcl | Roteamento - não alterar |
| `-- 5_fillers_reports.tcl | Células de preenchimento - não alterar |
| `-- 6_netlist_sdf.tcl | Alterar o nome de arquivo de saída |

ETAPA 1 – Simulação RTL com o *xrun*

Ir para o ambiente de simulação rtl: **cd sim/rtl**

Observar o script de simulação fornecido: **cat file_list.f**:

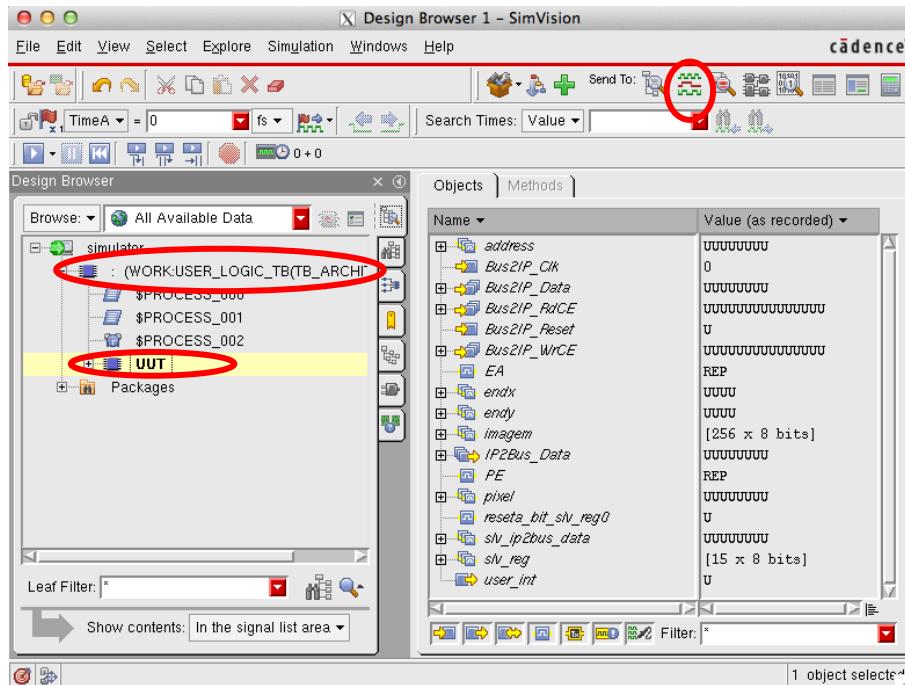
```
-smartorder -work work -V93 -top user_logic_tb -notimingchecks -gui -access +rw  
../../rtl/busca_padrao.vhd  
../../tb/tb_padrao.vhd
```

onde:

- ▲ -smartorder – indica que o compilador deve reconhecer a ordem hierárquica das descrições fornecidas
- ▲ -work – define o nome da biblioteca onde serão armazenados os módulos compilados
- ▲ -V93 – habilita características do VHDL93, como evitar a declaração de componentes
- ▲ -top – topo da hierarquia do projeto (*user_logic_tb* – entidade do test_bench)
- ▲ -notimingchecks – desabilita verificações de timing
- ▲ -gui – habilita modo gráfico
- ▲ -access +rw – acesso aos sinais internos do circuito para exibição

Executar o seguinte comando: **xrun -f file_list.f**. A ferramenta xrun irá compilar e elaborar o projeto.

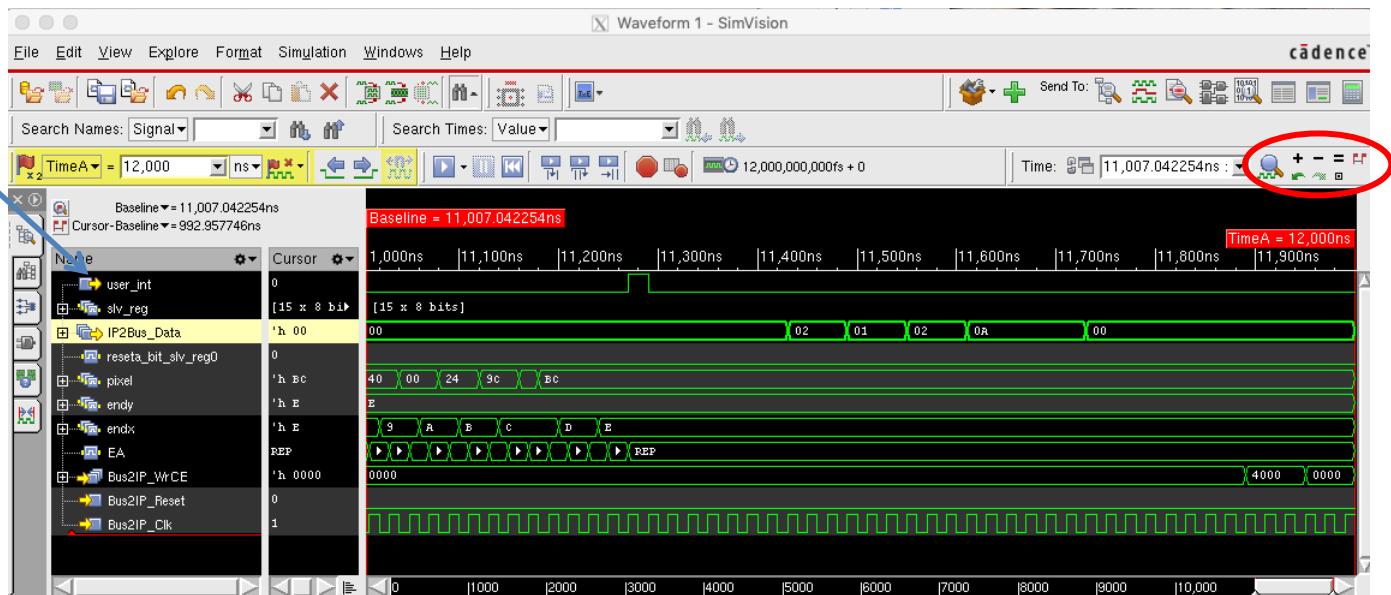
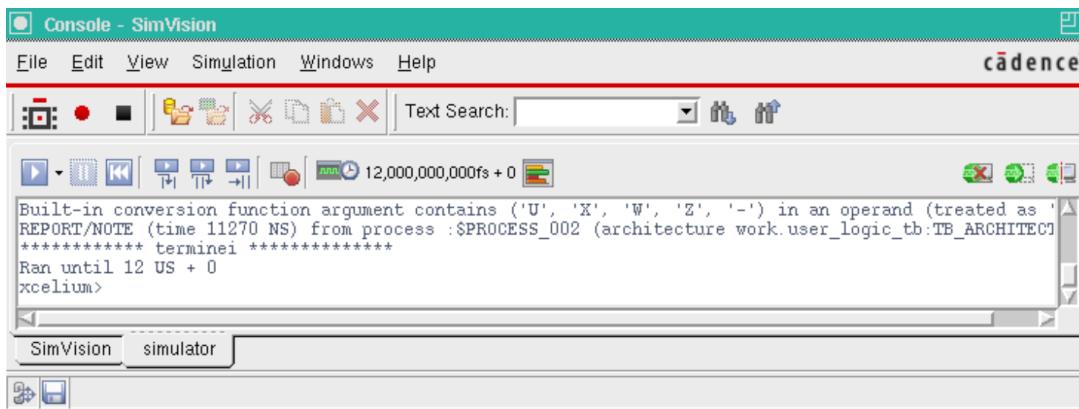
A interface do simulador é aberta. Selezionando-se o *top* (USER_LOGIC_TB) tem-se os sinais da entidade, os quais podem ser enviados para uma *waveform*, clicando no local indicado.



Inserir os seguintes sinais do **UUT** na waveform (clicar em UUT, e para cada sinal clicar no símbolo de *waveform*): **user_int, slv_reg, IP2Bus_Data, reseta_bit_slv_reg0, pixe, endy, endx, EA, Bus2IP_WrCE, Bus2IP_Reset, Bus2IP_Clk**.

Executar por 12 microssegundos, digitando **reset** e **run 12 us** no console.

Clicar no sinal de “=” para fazer um zoom dos 12 microssegundos, e depois com as barras verticais de “Baseline” e “TimeA” fazer um zoom entre 11,2 microssegundos e 11,8 microssegundos:



Para este circuito deve-se observar o sinal *user_int* (interrupção gerada pelo UUT), e depois os dados em *IP2bus_Data*. A interpretação destes dados são: 2 matches do padrão a ser pesquisado em uma dada imagem, nos endereços (1,2) e (A,A).

Para sair, menu *File* → *Exit SimVision*

Observação: ao relançar uma simulação executar antes **xrun -clean**

ETAPA 2 - Síntese Lógica

- Ir para o diretório de síntese: **cd ../../synthesis**
- Para a síntese lógica será utilizada a ferramenta Genus da CADENCE. Para abrir a ferramenta digite: **genus -gui** (não executar com a opção '&', pois a ferramenta tem um *shell* interno).

Na interface gráfica poderão ser acompanhados as respostas dos comandos inseridos no *shell* do Genus. Os comandos necessários para a correta síntese do projeto estão disponíveis no arquivo “*comandos_genus.txt*” e deverão ser inseridos sequencialmente no *shell* do *genus*.

A lista de comandos está dividida em **5 grupos distintos** (copie e cole os comandos não comentados em cada grupo):

- 1) **Configuração do ambiente de síntese e compilação e elaborar o projeto.** Abrir o arquivo *load.tcl* e entender os comandos definidos nesse script.

- Os três primeiros comandos definem o nível de informação e os diretórios onde serão buscados scripts e descrições RTL:

```
set_db script_search_path ./
set_db hdl_search_path ../rtl
set_db information_level 9
```

- Os dois comandos seguintes (**MUITO IMPORTANTES**), definem as bibliotecas que serão usadas na síntese do projeto. Nesse caso, optamos pela biblioteca de standard-cells CORE65GPSVT, fornecida pela *foundry* (STMicroelectronics). Além dessa biblioteca, também usaremos uma biblioteca de células físicas (PRHS65), que será explicada na próxima etapa de projeto.

```
set_db library ...
set_db lef_library ...
```

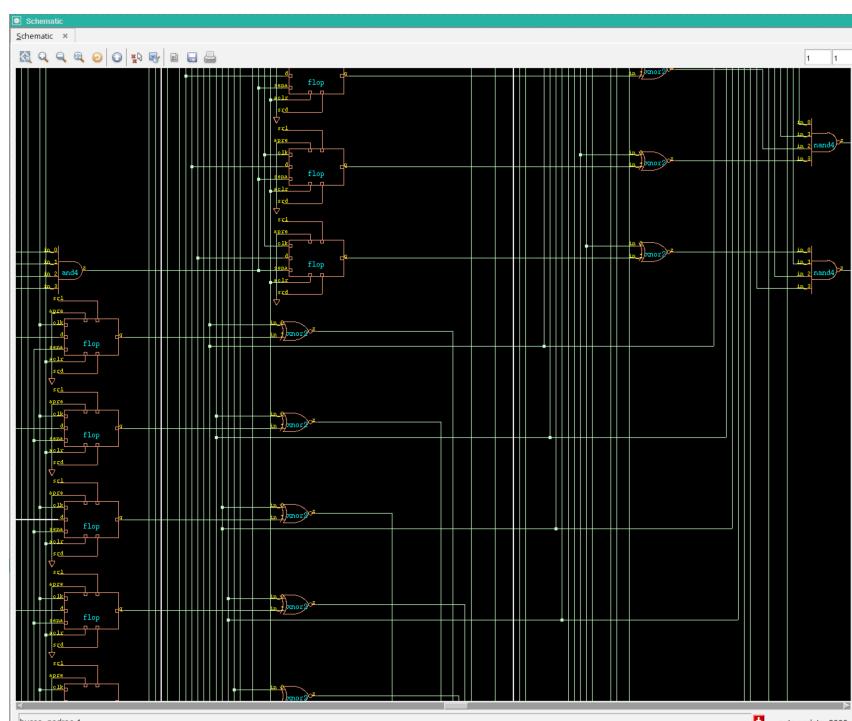
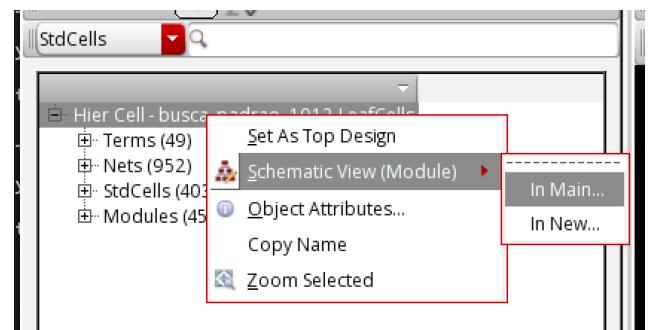
- OBS: O arquivo cmos065_7m4x0y2z_AP_Worst.lef é um arquivo comum para todas as bibliotecas dessa tecnologia e deve ser sempre utilizado em projetos implementados na mesma. Ele define parâmetros para as ferramentas de síntese.
- Os dois últimos comandos, definem a *captable* que será utilizada e a condição operacional. A *captable* é um arquivo que contém valores de resistência e capacitância que serão usados para modelar as interconexões do design. Essa informação será usada quando a ferramenta de síntese extrair os fios do projeto, para realizar análises de timing e power.

```
set_db cap_table_file ....
```

Digite os três comandos do item 1 do comandos_rc.txt no shell do *genus*:

```
include load.tcl
read_hdl -vhdl busca_padrao.vhd
elaborate busca_padrao
```

O resultado após executar esse bloco de comandos é dado na janela gráfica do *genus*. Navegar pelo visualizador de esquemáticos. Conforme pode ser observado, o projeto foi elaborado para funções definidas nas bibliotecas instanciadas na descrição, *ands*, *ors*, etc. Seleciona “*schematic view (Module)* → *in Main...*”



- 2) Restrições geradas para esse projeto. Abrir o arquivo de restrições - constraints (“`./constraint/busca_padrao.sdc`”) e entender seus comandos.
- ↳ Os dois primeiros comandos definem variáveis internas da ferramenta.
 - ↳ O comando `create_clock` define quem é o clock do circuito e o período desejando (2.0 ns)
 - ↳ O comando `set_false_path` evita que o reset seja utilizado na análise de atraso
 - ↳ O comando “`set_input_transition`” define a rampa nas entradas do circuito para transições de descida e subida. Esses valores foram obtidos de informações contidas na biblioteca utilizada. Foi definido que o menor valor de transição é 0.003ns (correspondente ao melhor caso de um inverter de alto ganho) e o maior valor de transição é 0.16ns (correspondente ao pior caso de um inverter de baixo ganho).
 - ↳ O comando “`set_load`” define a carga nas saídas do circuito. Os valores foram obtidos utilizando-se o mesmo método descrito acima. A carga mínima foi definida para 0.0014pF (capacitância do pino de entrada de um inverter pequeno) e a carga máxima para 0.32pF (capacitância do pino de entrada de um inverter grande).

Com essas informações, a ferramenta de síntese pode escolher o ganho/tamanho das células que serão instanciadas no projeto. No shell do `genus` digite o segundo comando:

```
read_sdc ./constraint/busca_padrao.sdc
```

Como resultado desse bloco, deve-se obter a seguinte saída:

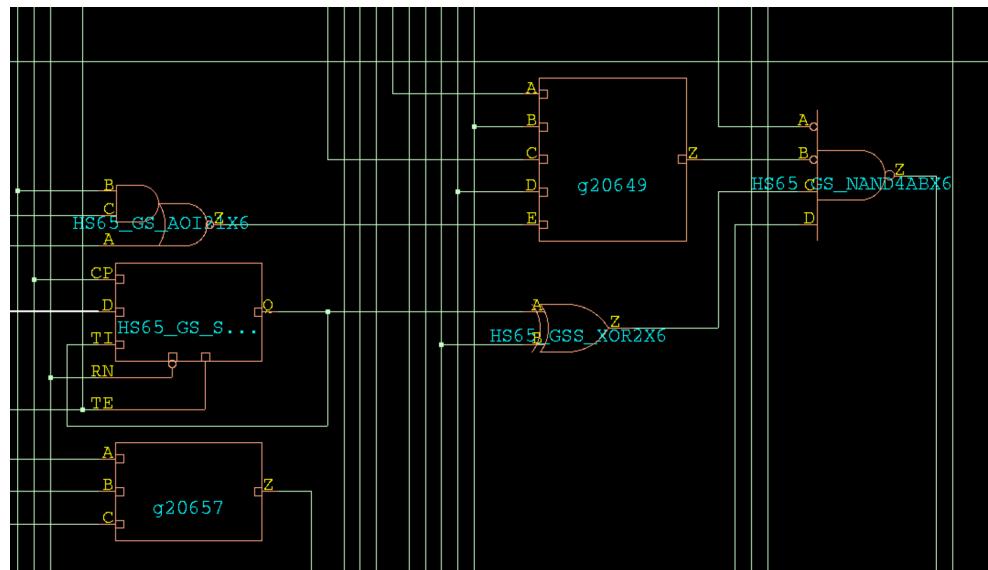
```
Statistics for commands executed by read_sdc:
"all_inputs"           - successful      5 , failed      0 (runtime  0.00)
"all_outputs"          - successful     2 , failed      0 (runtime  0.00)
"create_clock"         - successful     1 , failed      0 (runtime  0.00)
"get_ports"            - successful     2 , failed      0 (runtime  0.00)
"set_false_path"       - successful     1 , failed      0 (runtime  0.00)
"set_input_delay"      - successful     1 , failed      0 (runtime  0.00)
"set_input_transition" - successful    4 , failed      0 (runtime  0.00)
"set_load"              - successful     2 , failed      0 (runtime  0.00)
"set_load_unit"         - successful     1 , failed      0 (runtime  0.00)
Total runtime 0
```

Que indica que as *constraints* foram geradas corretamente.

- 3) Síntese lógica otimizada. No shell do `genus` digite os comandos abaixo para a realização da síntese lógica:

```
synthesize -to_mapped -eff high -no_incr
syn_opt
syn_opt -incremental
```

Esse passo realiza a síntese **optimizada** para o projeto, otimizando o projeto elaborado, que conta com elementos genéricos (ands, ors, etc.) para implementar a lógica descrita.



4) Relatórios:

report area

Onde a coluna “*Instance*” indica o circuito, a coluna “*Cells*” o número de células utilizadas, a coluna “*Cell Area*” a área dessas células e a coluna “*Net Area*” uma estimativa da área de fios que será necessária. Notar que neste circuito forma utilizadas 760 células (portas lógicas).

| Instance | Module | Cell Count | Cell Area | Net Area | Total Area |
|--------------|--------|------------|-----------|----------|------------|
| busca_padrao | | 760 | 3450.200 | 1983.462 | 5433.662 |

report gates

Apresenta todas as portas lógicas utilizadas no projeto.

| | | | |
|------------------|-----|----------|-------------|
| HS65_GS_OR3X9 | 2 | 7.280 | CORE65GPSVT |
| HS65_GS_OR4X4 | 2 | 10.400 | CORE65GPSVT |
| HS65_GS_SDFPRQX9 | 101 | 1260.480 | CORE65GPSVT |
| total | 760 | 3450.200 | |

| Library | Instances | Area | Instances % |
|--------------|-----------|----------|-------------|
| CLOCK65GPSVT | 8 | 12.480 | 1.1 |
| CORE65GPSVT | 752 | 3437.720 | 98.9 |

| Type | Instances | Area | Area % |
|----------------|-----------|----------|--------|
| sequential | 116 | 1418.560 | 41.1 |
| inverter | 83 | 134.160 | 3.9 |
| buffer | 4 | 11.440 | 0.3 |
| logic | 557 | 1886.040 | 54.7 |
| physical_cells | 0 | 0.000 | 0.0 |
| total | 760 | 3450.200 | 100.0 |

report timing

Este relatório informa o atraso de cada célula no caminho crítico, e principalmente se a síntese atendeu à restrição de timing. Dado que o *clock* é de 2000 ps, há uma sobra de 2 ps (*slack*) – destacado em amarelo.

```
report timing
  Tracing clock networks.
  Levelizing the circuit.
  Computing delays.
  Computing arrivals and requireds.
Warning : Possible timing problems have been detected in this design. [TIM-11]
  : The design is 'busca_padrao'.
  : Use 'report timing -lint' for more information.
```

```
=====
Generated by:          Genus(TM) Synthesis Solution 18.14-s037_1
Generated on:         Jun 10 2021 11:58:19 am
Module:               busca_padrao
Operating conditions: _nominal_
Interconnect mode:   global
Area mode:            physical library
=====
```

Path 1: MET (2 ps) Setup Check with Pin EA_reg[1]/CP->D

```
  Group: Bus2IP_Clk
  Startpoint: (R) address_reg[7]/CP
    Clock: (F) Bus2IP_Clk
  Endpoint: (F) EA_reg[1]/D
    Clock: (R) Bus2IP_Clk

      Capture           Launch
  Clock Edge:+ 2000        1000
  Src Latency:+ 0           0
  Net Latency:+ 0 (I)       0 (I)
```

```

Arrival:= 2000 1000

      Setup:= 62
Required Time:= 1938
Launch Clock:= 1000
Data Path:= 935
Slack:= 2

#
#-----#
#   Timing Point   Flags   Arc   Edge   Cell           Fanout Load Trans Delay Arrival Instance
#                                         (fF)   (ps)   (ps)   (ps)   Location
#-----#
address_reg[7]/CP -      -     R   (arrival)          16   -    0   -  1000  (-,-)
address_reg[7]/QN -      CP->QN R   HS65_GS_DFPRQNX9  10 37.2  87  110  1110  (-,-)
g21430/z -             A->Z  F   HS65_GS_IVX9       5 21.0   45   50  1160  (-,-)
g21305/z -             B->Z  F   HS65_GS_OR2X9     10 38.6   56   78  1238  (-,-)
...
g20617/z -             D->Z  R   HS65_GS_OAI112X5   1  5.0   44   42  1898  (-,-)
g20612/z -             C->Z  F   HS65_GS_NOR4ABX2   1  5.0   38   37  1935  (-,-)
EA_reg[1]/D <<< -     F   HS65_GS_DFPRQX9     1  -    -    -   0  1935  (-,-)
#-----#

```

report power

```

@genus:root: 12> report power
=====
Generated by:          Genus (TM) Synthesis Solution 18.14-s037_1
Generated on:         Jun 10 2021 12:01:16 pm
Module:               busca_padrao
Operating conditions: _nominal_
Interconnect mode:   global
Area mode:            physical library
=====

          Leakage      Dynamic      Total
Instance Cells Power(nW) Power(nW) Power(nW)
-----
busca_padrao    760 40010.992 1877530.402 1917541.395

Info : Time taken to report power. [RPT-7]
      : 0.00 cpu seconds

```

5) Exportação para a síntese física. Digite:

“write_design -innovus -base_name layout/busca_padrao”.

Esse passo gera o ambiente a ser utilizado pela ferramenta de síntese física e exporta o *netlist* mapeado para a tecnologia.

Para sair do *genus* digite “**exit**”.

Para executar via script: **genus -f comandos_genus.txt**

ETAPA 3 - Simulação pós síntese (com atraso unário)

Deve-se garantir que o *netlist*, gerado no passo anterior, implementa a funcionalidade desejada. Para tanto, ir para o ambiente de simulação pós síntese:

cd ..sim/synth

Observar o *netlist* gerado em: **more ../../synthesis/layout/busca_padrao.v**

Notar que este *netlist* corresponde ao circuito original, mapeado para as portas lógicas da tecnologia.

O *script* desse ambiente é similar ao de verificação RTL, porém agora também será compilada a descrição funcional das bibliotecas utilizadas. Isso é necessário pois o *netlist* representa a interconexão de células específicas da tecnologia. Executar **more file_list.f**:

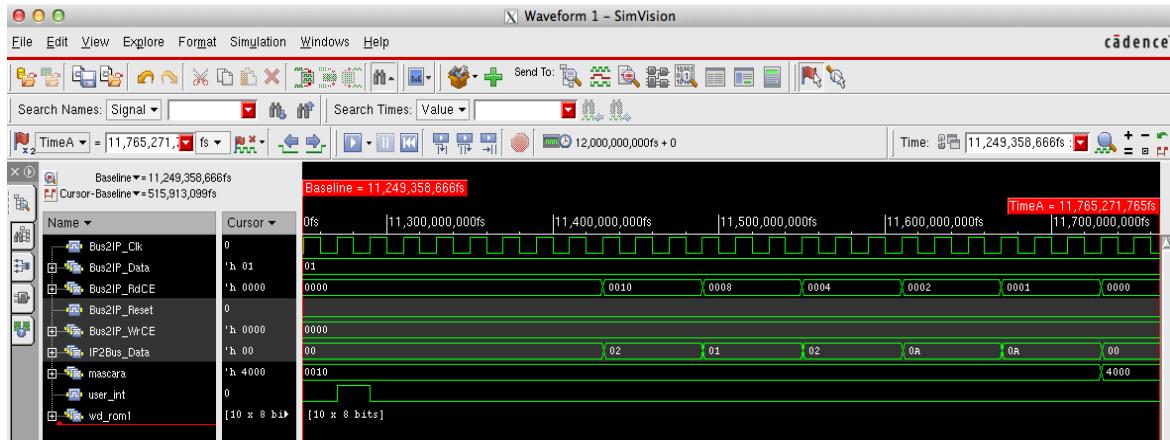
```

-smartorder -work work -V93 -top user_logic_tb -gui -access +rw
/soft64/design-kits/stm/65nm-cmos065_536/CORE65GPSVT_5.1/behaviour/verilog/CORE65GPSVT.v
/soft64/design-kits/stm/65nm-cmos065_536/CLOCK65GPSVT_3.1/behaviour/verilog/CLOCK65GPSVT.v
../../synthesis/layout/busca_padrao.v
./tb/tb_padrao.vhd

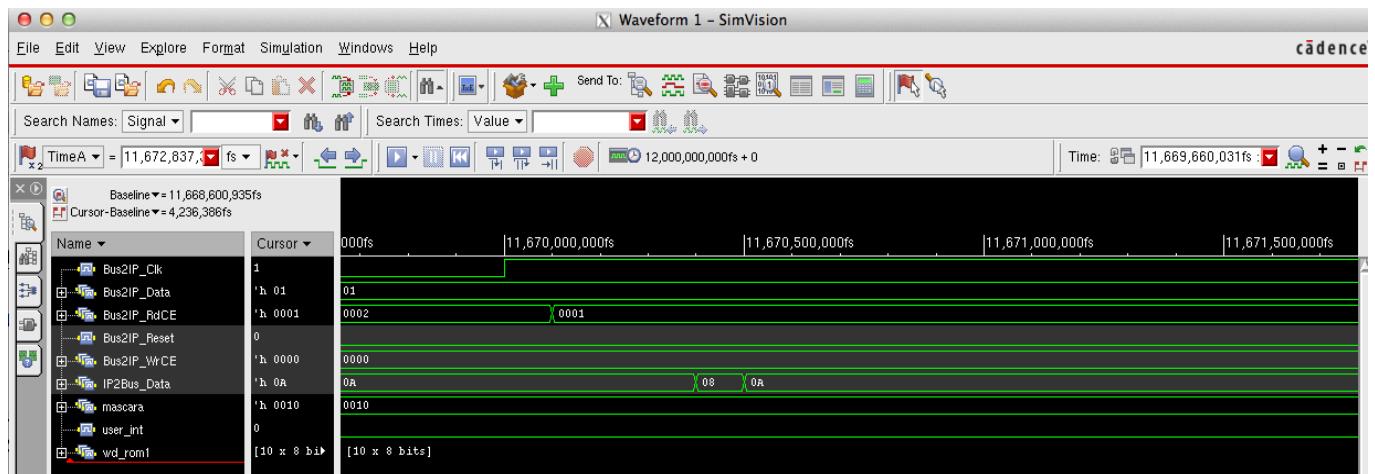
```

Executar o comando **xrun -f file_list.f**

Enviar os sinais do top para uma *waveform* e simular o circuito por 12us (reset; run 12 us)



Notar que fazendo um zoom no **0A** podemos visualizar o atraso no sinal IP2Bus_Data:



O que representa esse atraso? Notar que o valor é sempre múltiplo de 100ps. Isso se deve ao fato de o atraso unário considerar o atraso de todas standard-cells da biblioteca fixo em 100ps.

ETAPA 4 – SÍNTESE FÍSICA

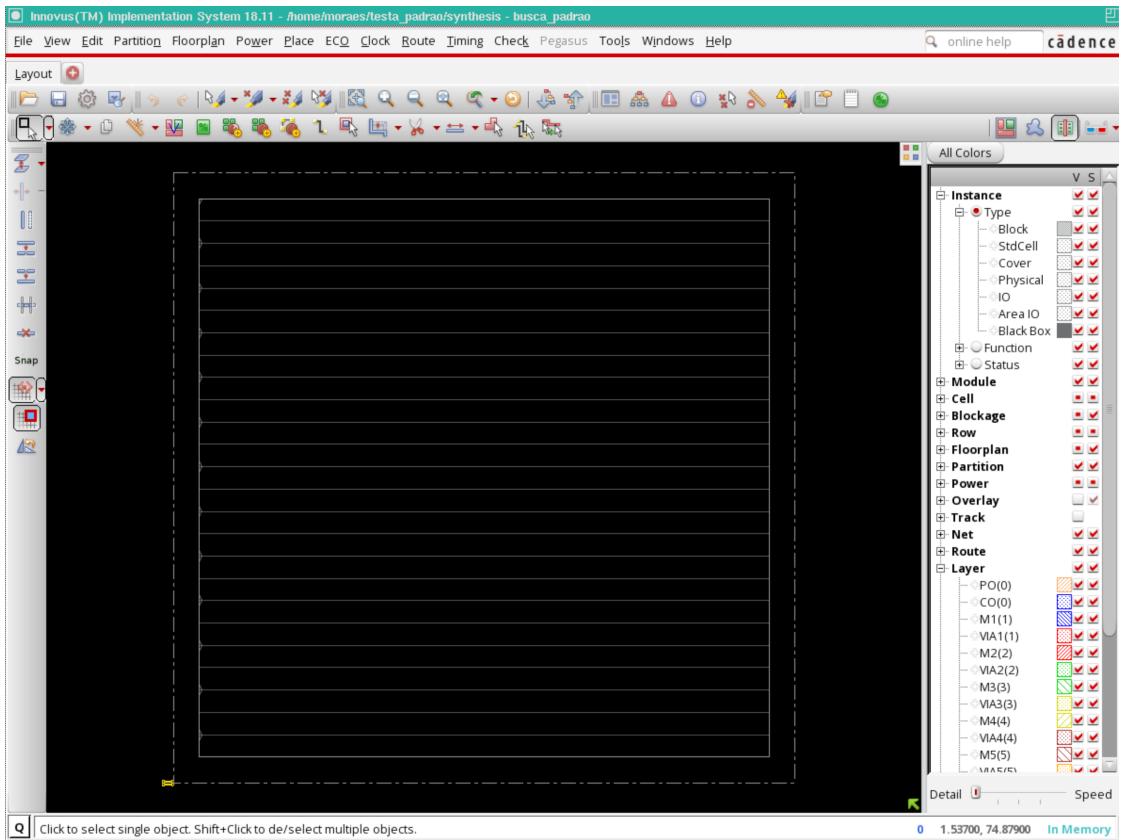
Uma vez que a síntese lógica do projeto foi validada, deve ser feita a síntese física. Para isto iremos utilizar os arquivos gerados na ferramenta anterior e a ferramenta *Innovus* da CADENCE.

Ir para a pasta **cd ../../synthesis**

Executar o comando **innovus -common_ui**

Carregar a configuração inicial da síntese física, digitando o seguinte comando no *shell* do Innovus:

source physical/1_init.tcl

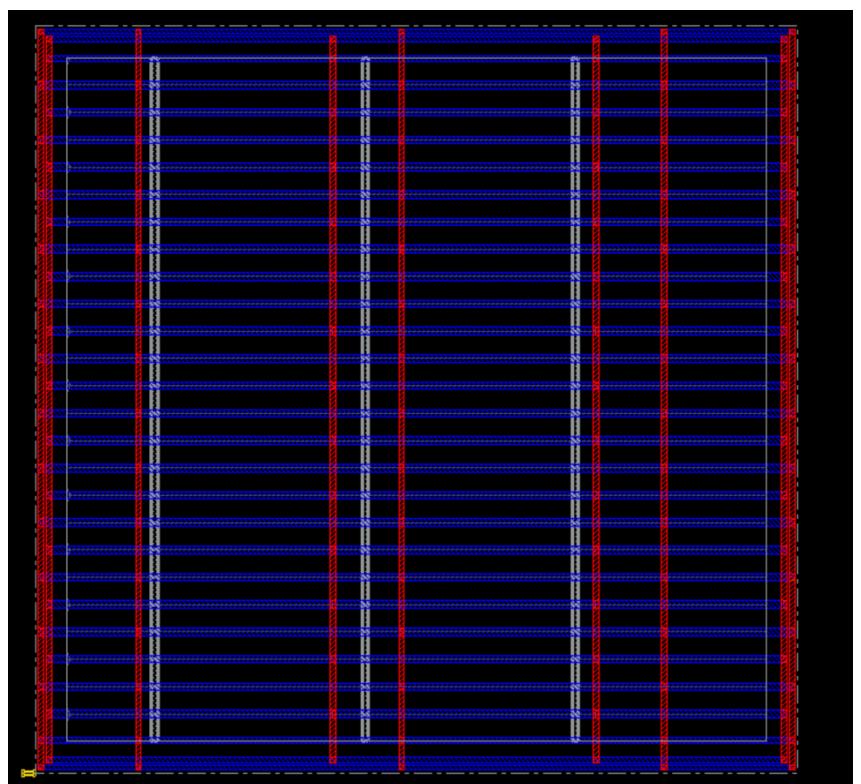


Abrir o arquivo *physical/1_init.tcl* e entender os comandos passados para o Innovus (ELES ESTÃO COMENTADOS).

Carregar a configuração de power planning no shell do Innovus:

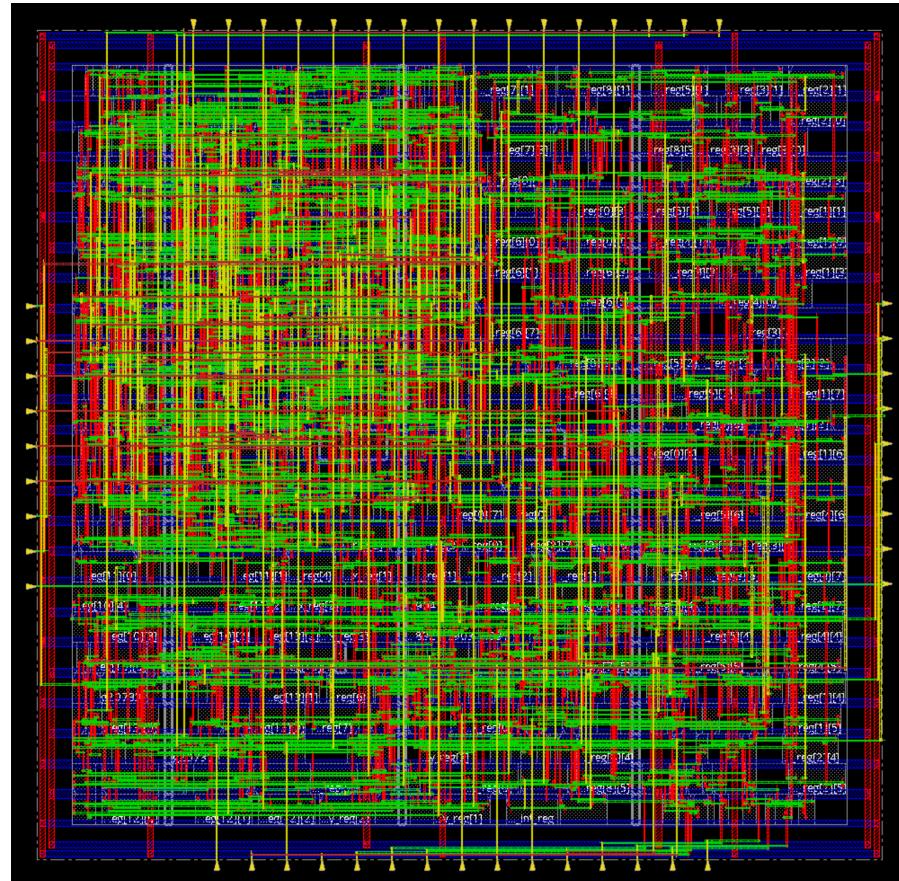
```
source physical/2_power_plan.tcl
```

Abrir o arquivo de configuração de *power planning* e entendê-lo. Notar que foi gerado um anel e linhas de alimentação, que serão utilizadas para posicionar as células lado a lado. A simetria dessas linhas (mesma altura) facilita o algoritmo de posicionamento e a instanciação das células físicas. Essas células serão posicionadas entre as linhas de alimentação (retângulos azuis) dentro do bloco definido no floorplan. Além disso, foram posicionadas colunas de *tap cells*. Estas células garantem a polarização da difusão, já que para essa biblioteca, as células lógicas não possuem conexão com bulk. Essas células devem ser posicionadas no máximo 30 µm de distância uma da outra, para garantir polarização da difusão (informação obtida na documentação da biblioteca). Também foram inseridos reforços para a alimentação.

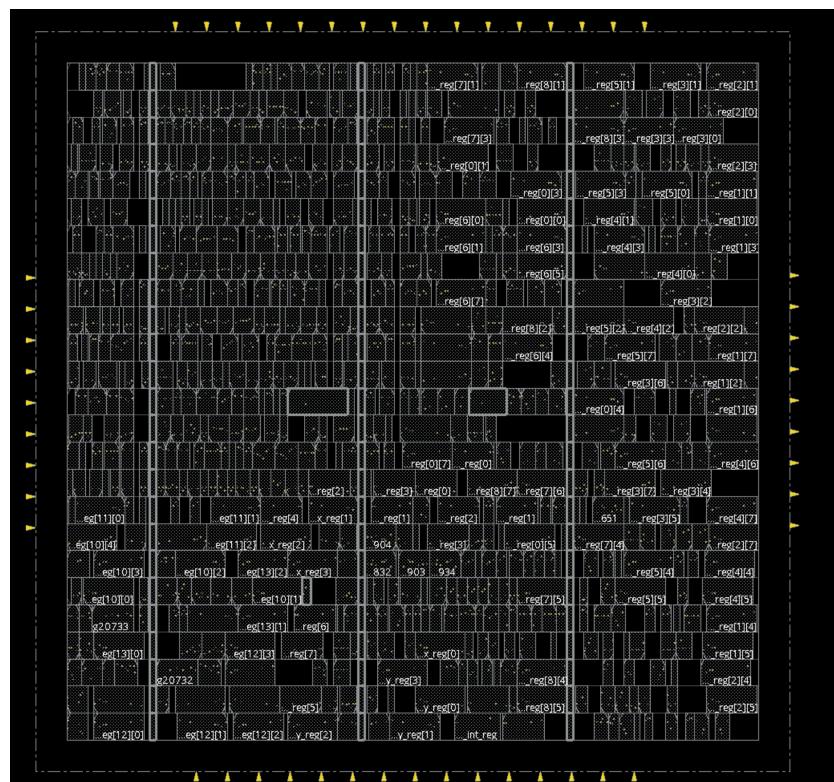


Instanciar as células físicas no projeto e realizar um roteamento inicial. Executar o seguinte comando no Innovus: **source physical/3_pin_clock.tcl**

(demora um pouco)



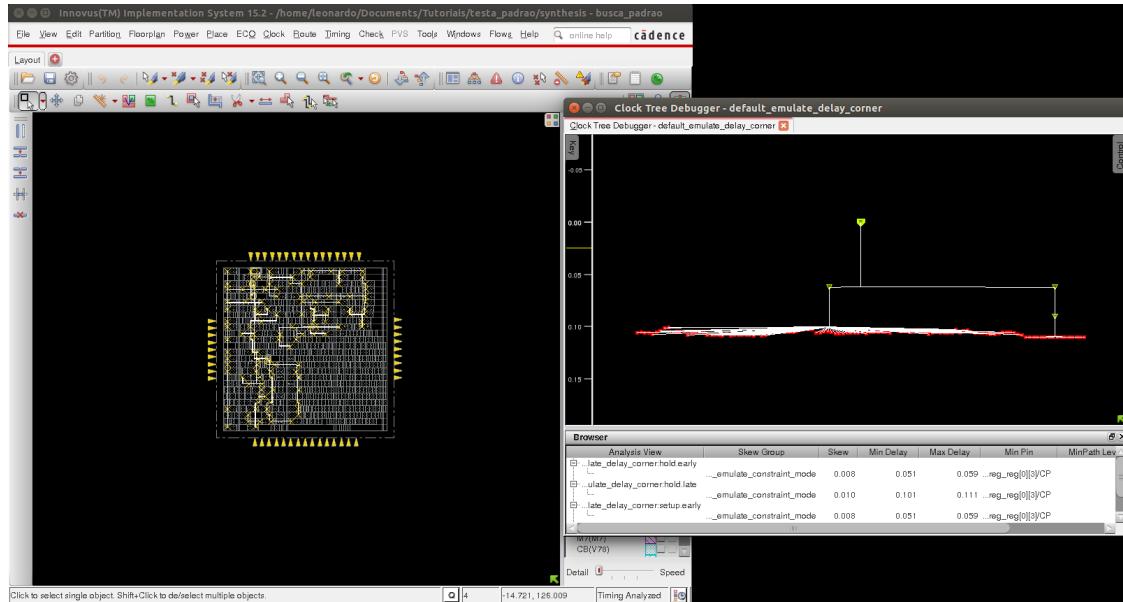
- observar que há um menu à direita – se as camadas não aparecerem, selecionar como visível (V) as instâncias, fios, etc. Desmarcar “layer” para visualizar apenas as instâncias das células.



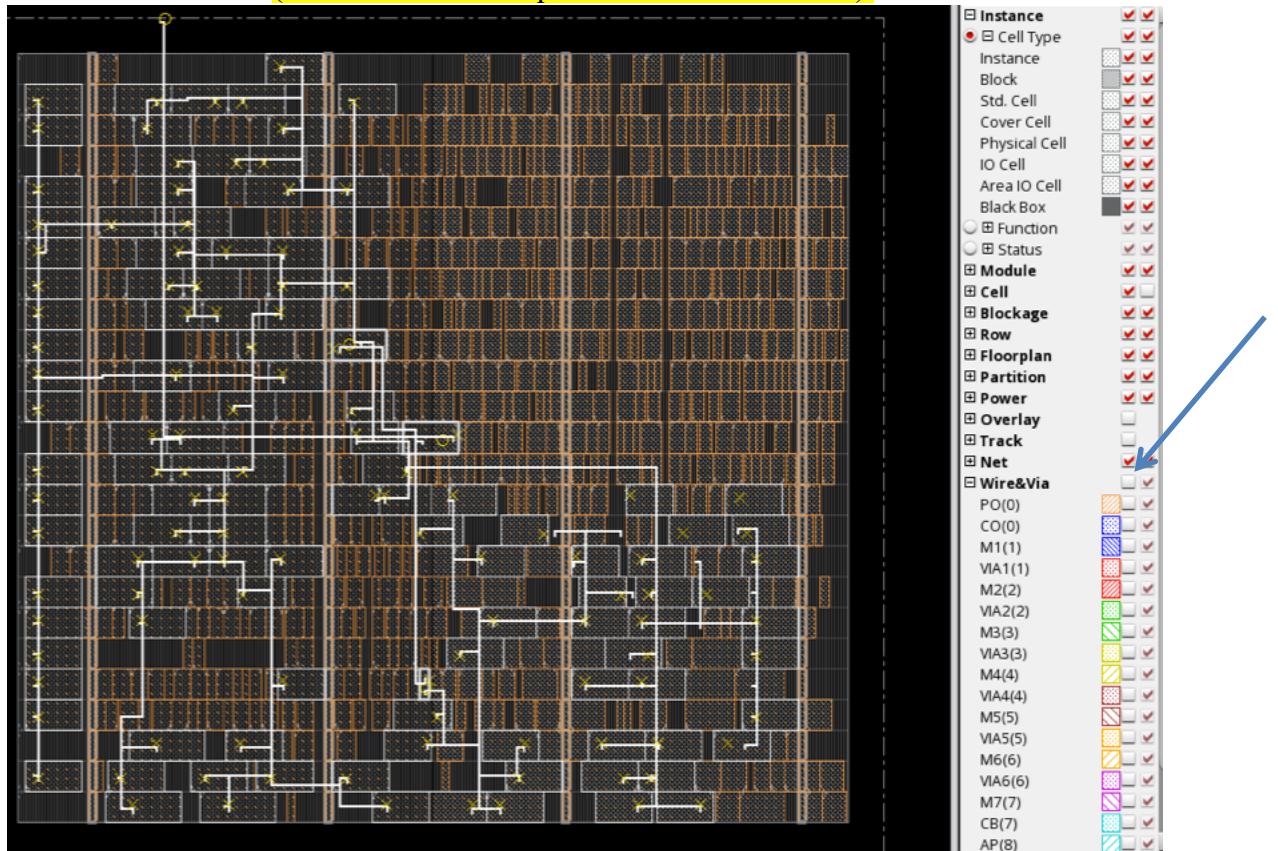
Este terceiro script tem 2 objetivos:

1. posicionar os pinos de entrada e saída na periferia do circuito – olhar os 4 comandos `editPin`;
2. gerar a árvore de clock- a arquivo Clock.ctstch é utilizado neste passo

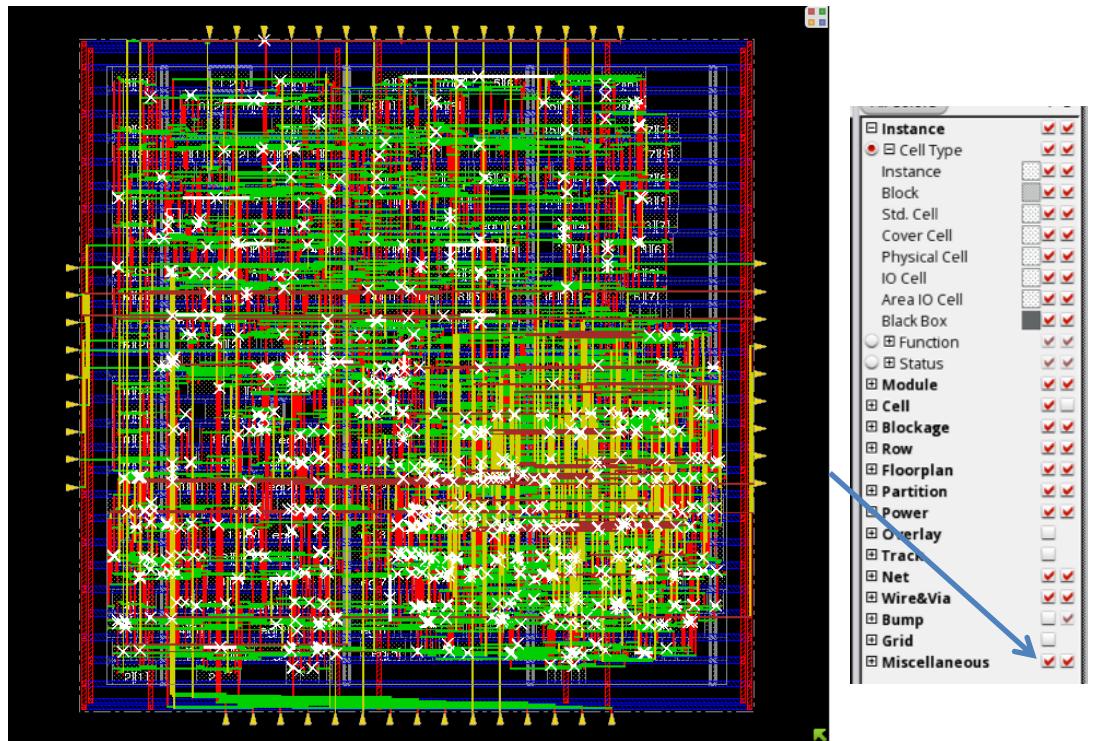
No menu **clock → CCOpt Clock Debugger** selecionar apenas os fios da árvore de *clock*. Na sequência, desabilitar a visualização das camadas de metal. Percebe-se o sinal de *clock* entrando na parte superior do circuito, e sendo distribuído uniformemente pelo layout.



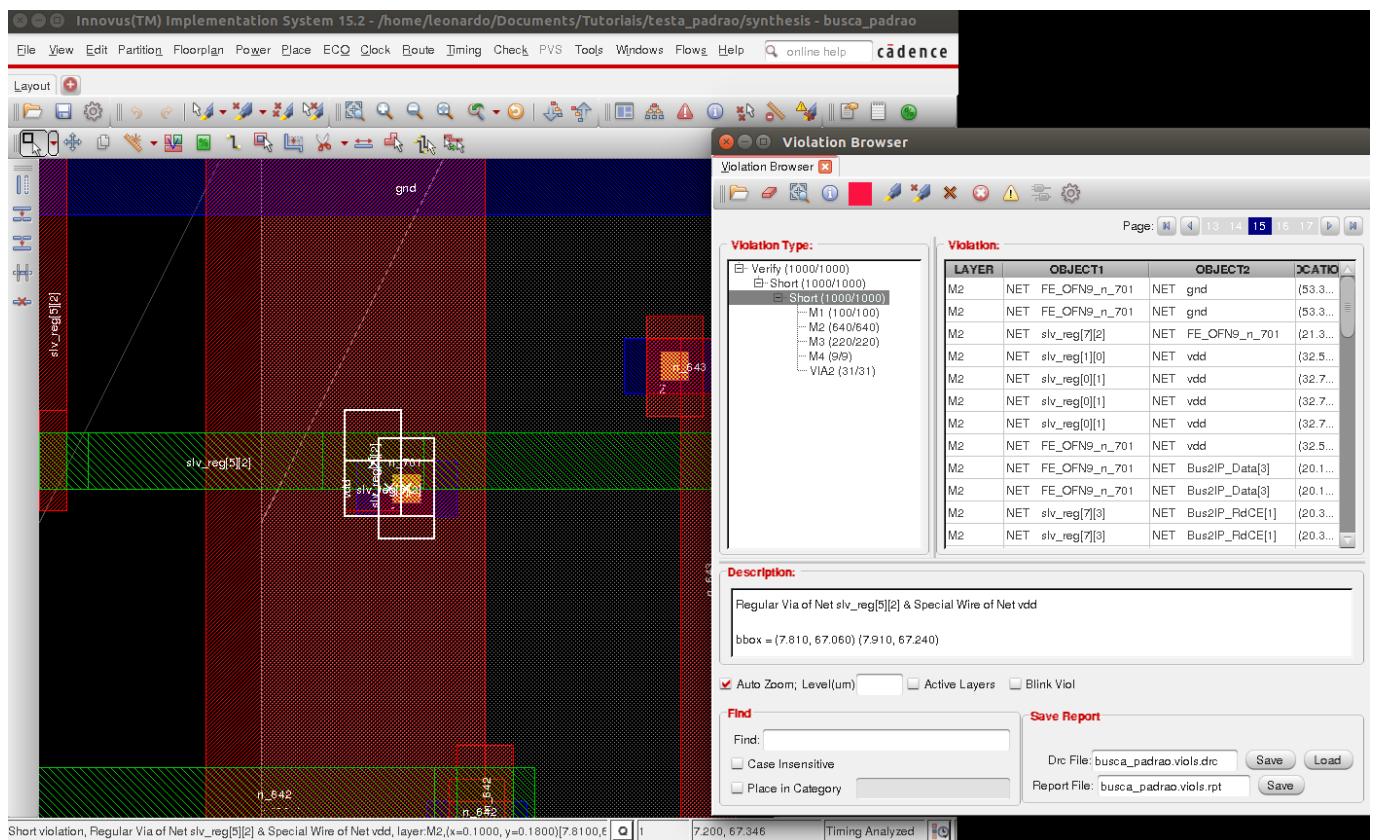
Para selecionar toda a árvore de *clock*, ir na janela *Clock Tree Debugger* e com o mouse selecionar tudo (arrastando canto esquerdo até o canto direito).



O roteamento feito por esse passo é um roteamento **inicial** e pode **violar regras** de DRC, por exemplo duas linhas de metal de mesmo nível podem estar muito próximas. Portanto, verificar o projeto. Clicar em “**Check → Check Geometry**” na janela gráfica do *Innovus* e clicar em “OK”. **Para os erros aparecerem deve estar visível “miscellaneous”**.



Nesse exemplo, foram geradas muitas violações, representadas por polígonos brancos com um “X” no meio. Clicar duas vezes em uma violação. Uma nova janela deve abrir com um detalhamento dessa. No caso representado abaixo, duas nets diferentes foram sobrepostas em um mesmo nível de metal, gerando um “short” (curto) um curto entre elas (se a janela na abrir, ir em *tools* → *Violation Browser*).



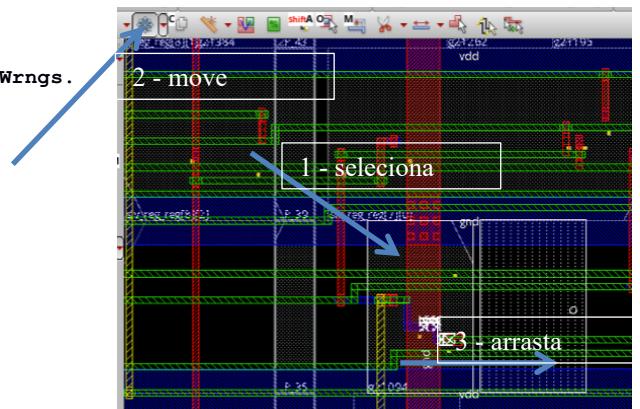
Esses erros podem ser evitados ao utilizarmos uma ferramenta de roteamento mais poderosa. Para tanto, executar o seguinte comando no Innovus: [source physical/4_nano_route.tcl](#)

Executar uma nova verificação. Notar que um roteamento bem elaborado foi suficiente para evitar violações.

```
innovus 5> *** Starting Verify Geometry (MEM: 1441.2) ***
VERIFY GEOMETRY ..... Starting Verification
VERIFY GEOMETRY ..... Deleting Existing Violations
VERIFY GEOMETRY ..... Creating Sub-Areas
..... bin size: 1440
**WARN: (IMPVFG-198): Area to be verified is small to see any runtime gain from multi-cpus. Use
set_multi_cpu_usage command to adjust the number of CPUs.
VERIFY GEOMETRY ..... SubArea : 1 of 1
VERIFY GEOMETRY ..... Cells : 0 Viols.
VERIFY GEOMETRY ..... SameNet : 0 Viols.
VERIFY GEOMETRY ..... Wiring : 0 Viols.
VERIFY GEOMETRY ..... Antenna : 0 Viols.
VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 1.00
Begin Summary ...
Cells : 0
SameNet : 0
Wiring : 0
Antenna : 0
Short : 0
Overlap : 0
End Summary

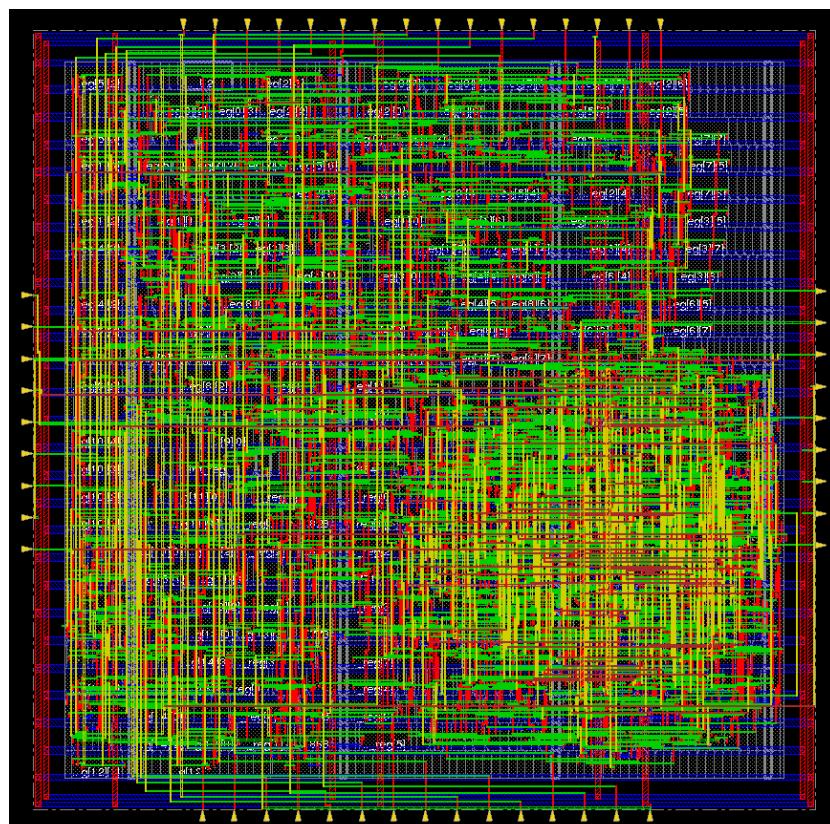
Verification Complete : 0 Viols. 0 Wrngs.

*****End: VERIFY GEOMETRY*****
*** verify geometry (CPU: 0:00:00.4 MEM: 10.9M)
```



→ Minha execução inicial mostrou 2 erros. Estes ocorreram sobre uma célula específica. Manualmente movi a célula para o lado (selecionar a célula e depois a opção de mover – canto superior esquerdo), e executei este passo novamente. Os erros foram resolvidos.

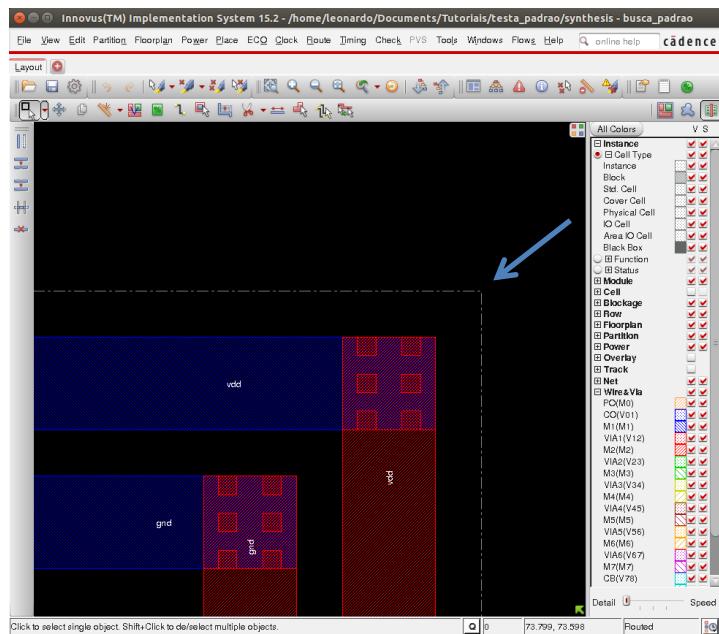
Uma vez que o projeto está validado, passar para o último passo. Note que o projeto contém “buracos” entre instâncias de células. Tal condição pode representar uma violação nas regras de manufatura, definidas pela *foundry*. Portanto, devem ser incluídas *filler cells*, que preencherão todos espaços entre células e garantirão que o projeto não violará regras pelo motivo descrito. Para tanto, executar o seguinte comando no Innovus: [source physical/5_fillers_reports.tcl](#)



Os seguintes comandos podem ser executados: **report_clocks** / **report_timing** / **report_area**

Notem que o slack diminui para 0,04 ns, devido aos atrasos dos fios.

Posicionar o mouse no canto superior do circuito como abaixo. O circuito tem uma área de 72,4 microns por 70,978 microns, o que resulta em um área de silício de 5.145 microns². O *report_area* só relata área de células.



Notar que o projeto está agora completamente preenchido. O script executado também gerou um relatório geral do projeto “*summaryReport*”. Executar o seguinte comando **no terminal do Innovus**:

firefox summaryReport/busca_padrao.main.htm

Neste relatório temos por exemplo a área do *core* (área sem o anel de alimentação) e a área total do circuito, que confere com a medida realizada acima:

```
Total area of Core 4316 um^2
Total area of Chip 5140 um^2
```

Explorar informação como, células instanciadas, área do core, comprimentos de fios, níveis de metal utilizados, etc.

Finalmente, gerar o netlist do projeto físico e um arquivo com o atraso de cada fio desse netlist. Para tanto, executar o seguinte comando no Innovus:

source physical/6_netlist_sdf.tcl

O arquivo de atrasos conta com o atraso de cada fio do circuito gerado, que representa os atrasos de portas lógicas somado aos atrasos de fios.

Sair do Innovus digitando **exit**

ETAPA 5 – SIMULAÇÃO COM ATRASO DE ROTEAMENTO

Neste passo iremos simular o circuito com atraso de portas e fios. O arquivo que contém estes atrasos é o *busca_padrao.sdf*. A descrição *sdf* é um formato de VHDL, e significa *standard delay format*. O primeiro

passo para a simulação com atraso pós-síntese física é compilar o arquivo de atrasos. Para tanto executar o seguinte comando no diretório *synthesis*: **ncsdfc busca_padrao.sdf**

Verificar que foi gerado o arquivo **busca_padrao.sdf.X**

Ir para o ambiente de simulação com atraso de roteamento: **cd ./sim/sdf**

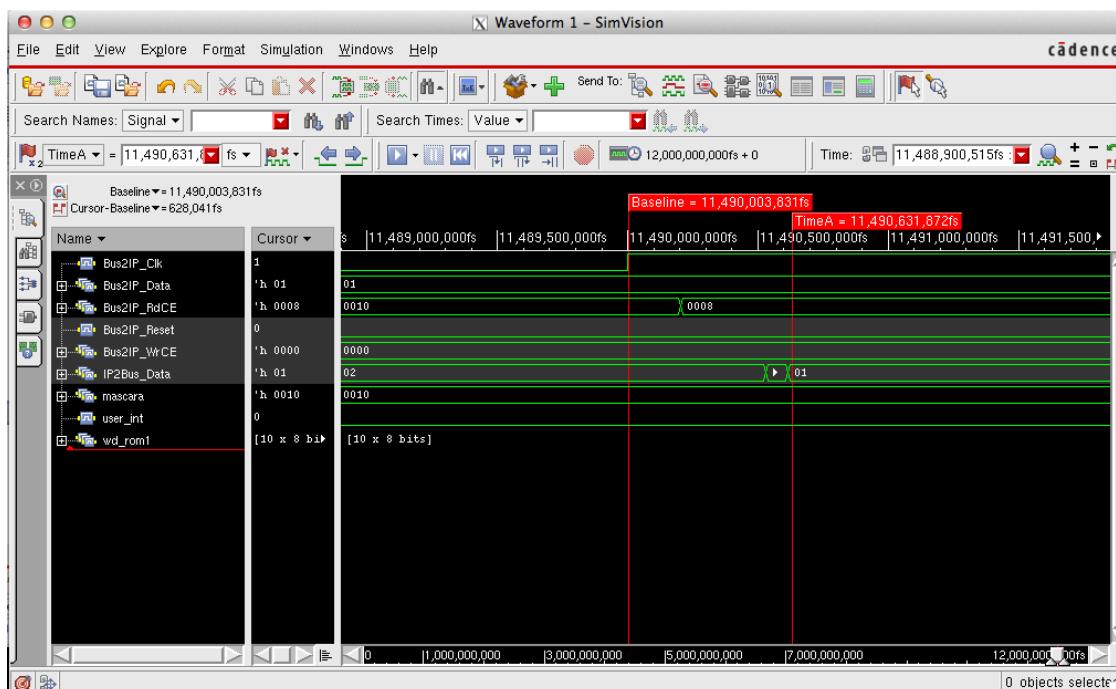
O script desse ambiente é similar ao de verificação pós síntese, porém agora é dado um parâmetro a mais (-sdf_cmd_file), o script de configuração de atraso. Ver **more sdf.cmd.cmd**:

```
COMPILED_SDF_FILE = "../../synthesis/busca_padrao.sdf.X",
LOG_FILE = "./sdf_log.log",
SCOPE = :UUT;
MTM_CONTROL = "MAXIMUM",
SCALE_FACTORS = "1.0:1.0:1.0",
SCALE_TYPE = "FROM_MAXIMUM";
```

Executar o comando **xrun -f file_list.f**

```
-smartorder -work work -v93 -top user_logic_tb -gui -access +rw -maxdelays -
sdf_cmd_file sdf.cmd
/soft64/design-kits/stm/65nm-
cmos065_536/CORE65GPSVT_5.1/behaviour/verilog/CORE65GPSVT.v
/soft64/design-kits/stm/65nm-
cmos065_536/CLOCK65GPSVT_3.1/behaviour/verilog/CLOCK65GPSVT.v
../../synthesis/busca_padrao.v
./tb/tb_padrao.vhd
```

Enviar os sinais do top para uma *waveform* e simular o circuito por 12 us:



Notar que agora o atraso não é mais múltiplo de 100ps, em comparação com a simulação pós síntese lógica. O que acontece é que agora esse atraso é um valor muito aproximado da realidade. Esse valor é baseado em modelos definidos pela fabricante. Também observar o arquivo **sdf_log.log**, o qual indica que diversas células não tiveram o atraso anotado.