

LABORATÓRIO 6 –SÍNTESE STANDARD-CELLS UTILIZANDO CADENCE (28 nm)

Prof. Fernando Gehm Moraes – Revisão: 14/maio/2025

Objetivo deste laboratório:

Compreender o fluxo de projeto *standard cells*

Simulação RTL, simulação com *netlist*, simulação com anotação de atraso de roteamento

Executar a síntese lógica e física

Após executar o laboratório, pesquise e responda (entregar as respostas no dia da segunda avaliação – P2):

- Qual a finalidade das três simulações executadas no fluxo *standard-cells* (RTL, *netlist*, com SDF)?
- Qual a função dos arquivos com extensão **LEF**, e qual o significado desta sigla?
Qual a função dos arquivos com extensão **LIB**, e qual o significado desta sigla?
- Modifique a frequência utilizada na síntese lógica (**constraint/restrictions.sdc**) e preencha a tabela abaixo:

Frequência	Nº de portas lógicas	Slack (ps)
250 MHz		
500 MHz		
1 GHz		
1.5 GHz		

Com o aumento da frequência, como se comporta o número de portas lógicas (aumenta ou diminui? Por quê?). Com o aumento da frequência, como se comporta o slack time (aumenta ou diminui? Por quê?). E o tempo de síntese?

- A síntese lógica utiliza três comandos de síntese: **syn_generic / syn_map / syn_opt**. Qual o objetivo de cada comando? (dica para iniciar a resposta: **man syn_generic** ou **help syn_generic** no terminal do genus).
- No arquivo *load.tcl* (síntese lógica) há o comando para ativar o modo PLE (*physical layout estimator*):

```
#Set PLE
set_db interconnect_mode ple
```

Execute ao final da síntese lógica: *report_ple*. O que ele indica? Porque é importante ativar o modo PLE durante a síntese lógica?

- Modifique o arquivo de síntese lógica, de forma que estes três comandos sejam como abaixo. Preencher a tabela abaixo e discutir os novos resultados.

```
syn_generic -create_floorplan -physical
syn_map -physical
syn_opt -incremental
```

Frequência	Nº de portas lógicas	Slack (ps)
500 MHz		
1.0 GHz		

- Porque na síntese física a geração do *floorplan* deve-se ter uma densidade inferior a 100%, como no comando utilizado neste laboratório?

```
##Generating square floorplan (1) with 85% of density (0.85) with 3um margins
create_floorplan -site CORE12T -core_density_size 1 0.85 3 3 3 3
```

- Qual a função da etapa *clock tree synthesis* na síntese física (comando **ccopt_design**)? Por que esta etapa é importante no fluxo de projeto?
- Qual a função das *filler cells* no projeto físico?
- Qual a tendência do *slack time* após a síntese física? Explicar este comportamento.

Arquivos do projeto (nanoCPU)

- ▲ Conectar-se ao servidor **paxos**: ssh -X <usuário>@paxos.inf.pucrs.br
- ▲ Baixar o arquivo de distribuição:

```
wget https://fgmoraes.github.io/microel/nanoCPU_lab6.zip
unzip nanoCPU_lab6.zip
cd nanoCPU_28
```

Abaixo está a estrutura da distribuição, a qual contém três diretórios principais:

```
-- limpa                                // script de limpeza do laboratório
|-- rtl
|   `-- nanoCPU.sv                      // código SystemVerilog do circuito - nanoCPU
|-- sim
|   |-- rtl
|   |   |-- file_list.f                // script de simulação RTL
|   |   |-- wave.tcl
|   |   `-- wave.tcl.svcf
|   |-- sdf
|   |   |-- file_list.f                // script de simulação com atraso de fios
|   |   `-- sdf_cmd.cmd
|   |-- synth
|   |   `-- file_list.f                // script de simulação com o netlist da síntese física
|   `-- tb
|       `-- nanoTB.sv                 // test bench
`-- synthesis
    |-- comandos_genus.tcl           // script de síntese lógica
    |-- comandos_innovus.tcl         // script de síntese física
    |-- constraint
    |   |-- load.tcl                 // parâmetros para a síntese lógica (tecnologia de 28 nm)
    |   `-- restrictions.sdc        // restrições de temporização
    '-- physical
        |-- 1_init.tcl              // primeira linha aponta para o arquivo de configuração do netlist
        |-- 2_power_plan.tcl         // definição do roteamento de alimentação e polarização das células
        |-- 3_pin_clock.tcl          // posicionamento dos pinos de E/S e geração da árvore de clock
        |-- 4_nano_route.tcl         // roteamento
        `-- 5_fillers_reports.tcl    // inserção das células de preenchimento
```

ETAPA 1 – Simulação RTL com o simulador xrun

Configurar o ambiente e ir para o diretório de simulação RTL:

```
module load xcelium
cd ~/nanoCPU_28/sim/rtl
```

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

```
-smartorder -work work -top tb -notimingchecks -gui -access +rw
../../rtl/nanoCPU.sv
./tb/nanoTB.sv
```

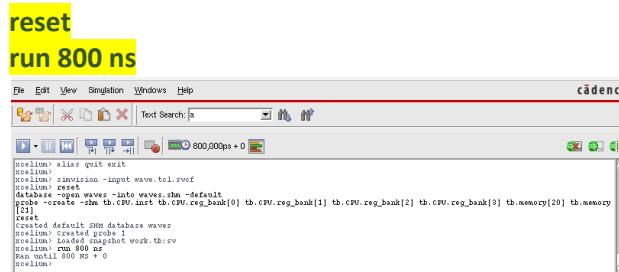
onde:

- ▲ -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
- ▲ -top – topo da hierarquia do projeto (*tb* – entidade do *test_bench*)
- ▲ -notimingchecks – desabilita verificações de timing
- ▲ -gui – habilita modo gráfico
- ▲ -access +rw – acesso aos sinais internos do circuito para exibição

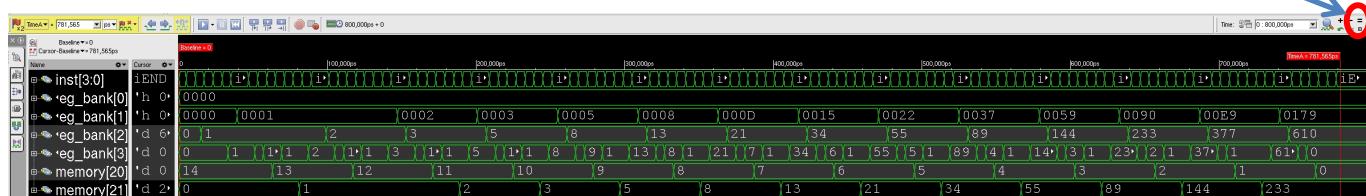
Executar o seguinte comando para inicializar a simulação: `xrun -f file_list.f -input wave.tcl`

A ferramenta *xrun* irá compilar e elaborar o projeto. A interface do simulador é aberta.

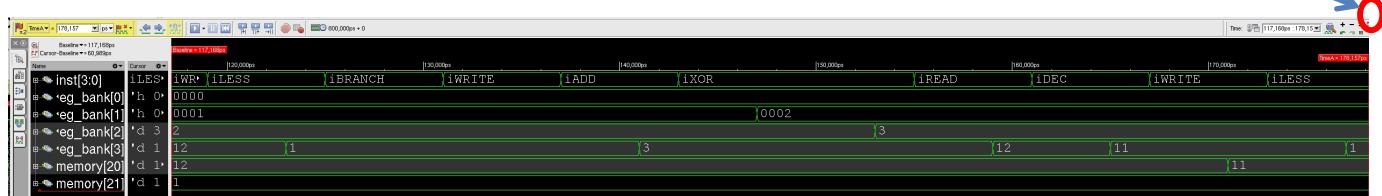
No terminal executar por 800 ns:



Clicar no '=' para zoom. Observar que em **memória[20]** temos um contador de vai de 14 à 0, e em **memória[21]** os primeiros 14 elementos da série de Fibonacci. Estamos simulando uma CPU muito simples, com 4 registradores.



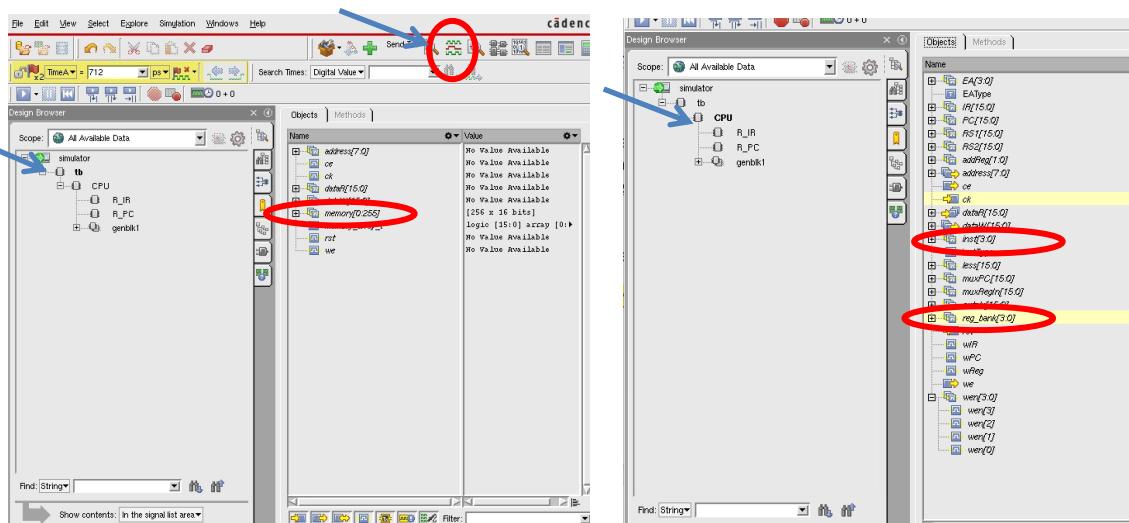
Com as barras verticais de “Baseline” e “TimeA” fazer um zoom no símbolo mais à direita das opções de zoom. Observa-se assim as instruções sendo executadas.



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

Ao sair do simulador recomenda-se apagar arquivos temporários: `xrun -clean`

O **wave.tcl** é gerado selecionando-se sinais do *top* (*tb - memory[20] e memory[22]*) e na CPU os sinais *int/reg bank[3:0]*. Uma vez selecionados salvar o script **wave.tcl**.



ETAPA 2 - Síntese Lógica

Para a síntese lógica é utilizada a ferramenta *genus* da CADENCE. Não executar o *genus* com a opção ‘&’, pois a ferramenta tem um *shell* interno. Configurar o ambiente e ir para o diretório de síntese:

```
cd ../../synthesis
module load ddi
genus -gui
```

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.tcl*” e deverão ser inseridos sequencialmente no *shell* do *genus*.

A lista de comandos está dividida em **5 grupos distintos** (copie e cole os comandos no *shell* do *genus*).

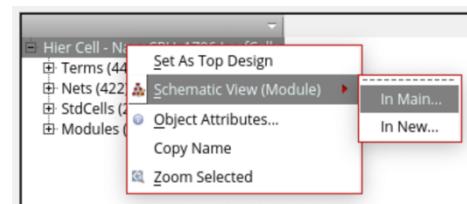
1. Configuração do ambiente de síntese e compilação do projeto

Abrir o arquivo *load.tcl*. Este é o arquivo que define os LIB e LEF files, e definições gerais para a síntese lógica. Usarem os a biblioteca 28 nm da STMicroelectronics.

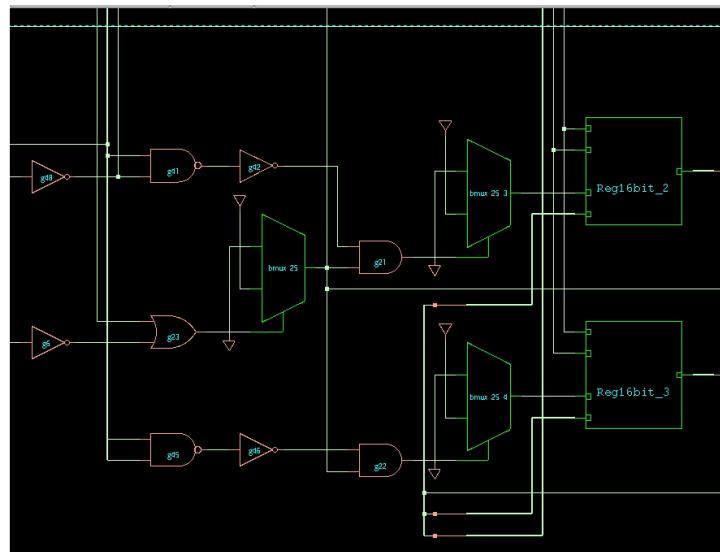
A *capturable* define a condição operacional. A *capturable* é um arquivo que contém valores de resistência e capacidade usadas para modelar as interconexões do projeto. Essa informação é usada quando a ferramenta de síntese extrair os fios de roteamento, para realizar análises de *timing* e *power*.

Digite os três abaixo no *shell* do *genus*:

```
include ./constraint/load.tcl
read_hdl -sv nanoCPU.sv
elaborate NanoCPU
set_db [current_design].dft_dont_scan true
```



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. Selecionar “*schematic view (Module)* → *in Main*”. **Faça um zoom em uma região para visualizar as portas lógicas.**



2. Restrições do projeto

Abrir o arquivo de restrições “*../constraint/restrictions.sdc*” e observar 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 desejado (**2.0 ns**)
- ▲ O comando *set_false_path* evita que o reset seja utilizado na análise de atraso

- ▲ O comando `set_load` define a carga nas saídas do circuito

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/restrictions.sdc
```

Como resultado desse comando, deve-se obter a seguinte saída, a qual indica que as *constraints* foram geradas corretamente.

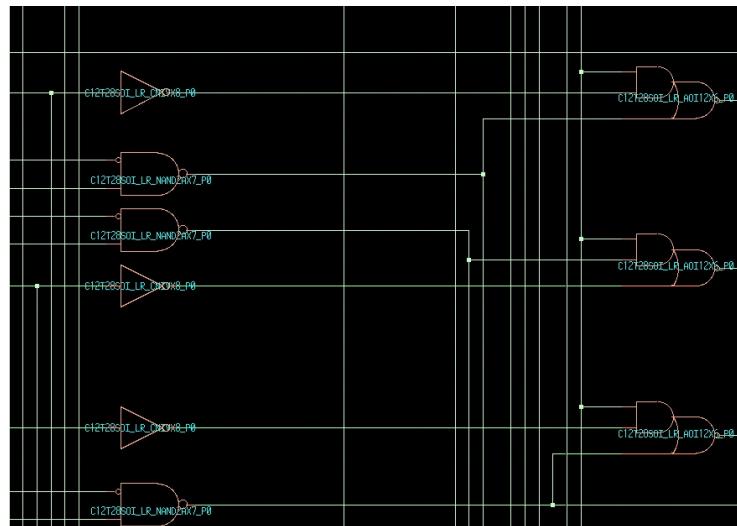
```
Statistics for commands executed by read_sdc:  
"all_inputs"           - successful      1 , failed      0 (runtime 0.00)  
"all_outputs"          - successful      2 , failed      0 (runtime 0.00)  
"create_clock"         - successful      1 , failed      0 (runtime 0.00)  
"get_ports"            - successful      2 , failed      0 (runtime 0.01)  
"set_false_path"       - successful      1 , failed      0 (runtime 0.00)  
"set_input_delay"      - successful      1 , failed      0 (runtime 0.01)  
"set_load"              - successful      1 , failed      0 (runtime 0.00)  
"set_output_delay"     - successful      1 , failed      0 (runtime 0.00)  
"set_units"             - successful      1 , failed      0 (runtime 0.00)
```

3. Síntese lógica otimizada

No *shell* do *genus* digite os comandos abaixo para a realização da síntese lógica:

syn_generic
syn_map
syn_opt

Esse passo realiza a síntese **optimizada** para o projeto, otimizando o projeto elaborado, para a biblioteca de células de 28 nm.



4. Relatórios

report_area: a coluna “*Instance*” indica o circuito, a coluna “*cell count*” 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 **774** células (portas lógicas).

Instance	Module	Cell Count	Cell Area	Net Area	Total Area
NanoCPU		774	1116.451	476.886	1593.338
R_IR	Reg16bit_98	15	62.179	0.000	62.179
R_PC	Reg16bit_97	9	35.578	0.000	35.578
genblk1_0.reg_inst	Reg16bit	17	70.829	0.000	70.829
genblk1_1.reg_inst	Reg16bit_101	17	70.829	0.000	70.829
genblk1_2.reg_inst	Reg16bit_100	17	70.829	0.000	70.829
genblk1_3.reg_inst	Reg16bit_99	17	70.829	0.000	70.829

report_gates: apresenta todas as portas lógicas utilizadas no projeto.

```
...
C12T28SOI_LR_SDFPRQX8_P0      85   374.544  C28SOI_SC_12_CORE_LR
C12T28SOI_LR_XOR2X8_P0        1    1.469  C28SOI_SC_12_CORE_LR
C12T28SOI_LR_XOR3X8_P0        6   13.709  C28SOI_SC_12_CORE_LR
-----
total                           774  1116.451
```

Library	Instances	Area	Instances %
C28SOI_SC_12_CLK_LR	74	24.480	9.6
C28SOI_SC_12_CORE_LR	700	1091.971	90.4

Type	Instances	Area	Area %
sequential	90	394.128	35.3
inverter	73	23.827	2.1
buffer	1	0.653	0.1
logic	610	697.843	62.5
physical_cells	0	0.000	0.0
total	774	1116.451	100.0

report_timing: 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. Observar que temos o tempo de setup de 198ps, o atraso CP-Q de 106 ps, e depois o atraso combinacional.

```
Path 1: MET (2 ps) Setup Check with Pin genblk1_3.reg_inst/Q_reg_15/CP->TI
  Group: ck
  Startpoint: (R) R_IR/Q_reg_12/CP
  Clock: (R) ck
  Endpoint: (F) genblk1_3.reg_inst/Q_reg_15/TI
  Clock: (R) ck
```

	Capture	Launch
Clock Edge:+	2000	0
Src Latency:+	0	0
Net Latency:+	0 (I)	0 (I)
Arrival:=	2000	0
Setup:-	198	
Required Time:=	1802	
Launch Clock:-	0	
Data Path:-	1799	
Slack:=	2	

#	Timing Point	Flags	Arc	Edge	Cell	Fanout	Load	Trans	Delay	Arrival	Instance
#						(fF)	(ps)	(ps)	(ps)		Location
R_IR/Q_reg_12/CP	-	-	R	(arrival)		90	-	0	0	0	(-, -)
R_IR/Q_reg_12/Q	-	CP->Q	R	C12T28SOI_LR_SDFPRQX8_P0	8	13.3	80	106	106	106	(-, -)
g4891_4733/z	-	B->Z	R	C12T28SOI_LRS_XOR2X6_P0	2	4.5	70	81	186	186	(-, -)
g4849_3680/z	-	B->Z	R	C12T28SOI_LR_AND2X8_P0	1	2.2	20	61	247	247	(-, -)
g4926/z	-	D->Z	F	C12T28SOI_LR_A0113X5_P0	1	3.2	41	24	271	271	(-, -)
g4923/z	-	C->Z	R	C12T28SOI_LR_OA121X11_P0	17	22.9	146	68	339	339	(-, -)
g4799_5115/z	-	B->Z	R	C12T28SOI_LR_AND2X8_P0	16	20.2	123	142	482	482	(-, -)
g4783_2398/z	-	B->Z	R	C12T28SOI_LR_A012X8_P0	1	3.0	28	83	564	564	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1248_8246/z	-	B->Z	R	C12T28SOI_LRS_XOR2X6_P0	2	5.1	82	67	632	632	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1263_5477/z	-	B->Z	R	C12T28SOI_LR_OA12X8_P0	2	4.6	32	65	696	696	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1262_6417/z	-	B->Z	R	C12T28SOI_LR_OA12X8_P0	2	4.6	32	48	744	744	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1261_7410/z	-	B->Z	R	C12T28SOI_LR_OA12X8_P0	2	4.6	32	48	792	792	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1260_1666/z	-	B->Z	R	C12T28SOI_LR_OA12X8_P0	2	4.6	32	48	840	840	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1189_2346/CO	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	1	3.0	24	71	1562	1562	(-, -)
sub_97_34_Y_sub_94_34_Y_add_99_35_Y_add_96_34_g1188_2883/z	-	C->Z	F	C12T28SOI_LR_XOR2X8_P0	1	2.0	25	85	1648	1648	(-, -)
g4653_6417/z	-	S1->Z	F	C12T28SOI_LR_MX41X7_P0	2	3.2	35	80	1727	1727	(-, -)
g4627_2398/z	-	D0->Z	F	C12T28SOI_LR_MUX21X8_P0	4	4.3	28	72	1799	1799	(-, -)
genblk1_3.reg_inst/Q_reg_15/TI	<<	-	F	C12T28SOI_LR_SDFPRQX8_P0	4	-	-	0	1799	1799	(-, -)

report_power -unit mW

Category	Leakage	Internal	Switching	Total	Row%
memory	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
register	2.28972e-03	1.92963e-01	1.77485e-02	2.13001e-01	30.56%
latch	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
logic	6.18002e-03	1.73144e-01	2.71757e-01	4.51080e-01	64.73%
bbox	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
clock	0.00000e+00	0.00000e+00	3.28050e-02	3.28050e-02	4.71%
pad	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
pm	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
Subtotal	8.46974e-03	3.66107e-01	3.22310e-01	6.96887e-01	100.00%
Percentage	1.22%	52.53%	46.25%	100.00%	100.00%

5. Exportação para a síntese física

```
write_netlist [current_design] > nano.v
write_snapshot -innovus -directory layout -tag logical [current_design]
```

Para sair do *genus* digite **exit**

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

Os relatórios estão no arquivo: **nano.txt**.

ETAPA 3 - Simulação pós síntese com o netlist nano.v

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

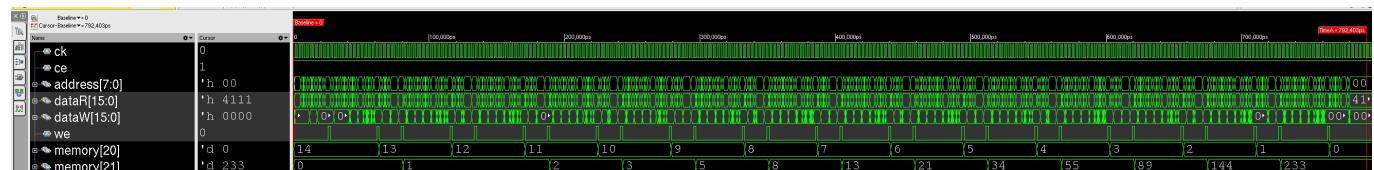
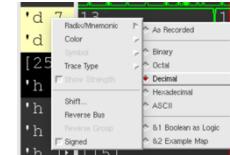
```
module purge
module load xcelium
cd ../../sim/synth
```

Observar o *netlist* gerado: **more ../../synthesis/nano.v**. Este *netlist*, com 2192 linhas, corresponde ao circuito original, mapeado para as portas lógicas da tecnologia.

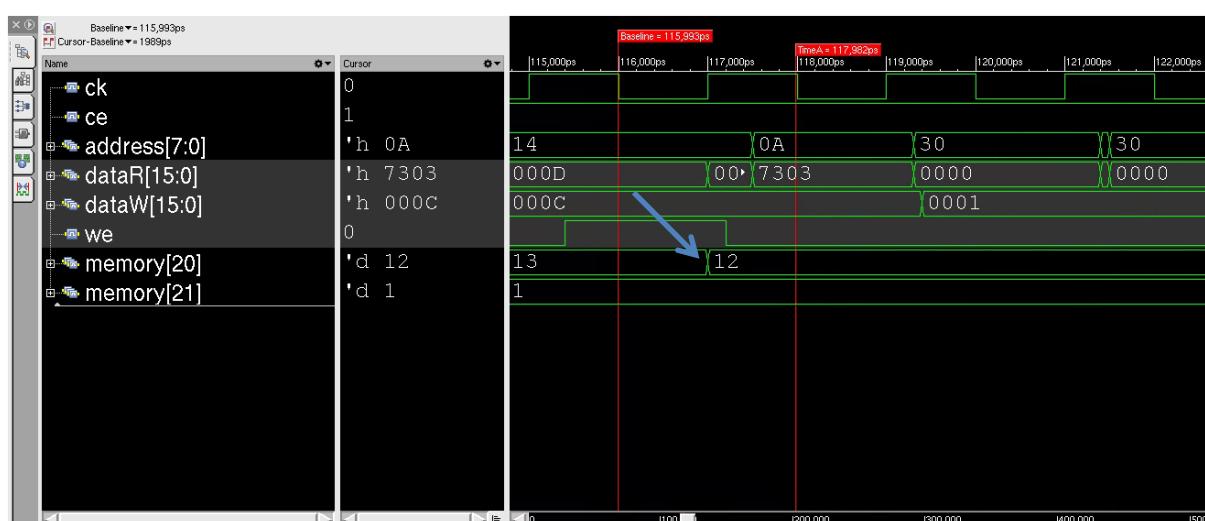
O script desse ambiente é similar ao de verificação RTL, porém agora também será compilada a descrição funcional das bibliotecas utilizadas. Isso é necessário pois o *netlist* contém as células específicas da tecnologia. Executar: **more file_list.f**

```
-smartorder -work work -top tb -gui -access +rw
/soft64/design-kits/stm/28nm-cmos28fdsoi_25d/C28SOI_SC_12_CLK_LR@2.1@20130621.0/behaviour/verilog/C28SOI_SC_12_CLK_LR.v
/soft64/design-kits/stm/28nm-cmos28fdsoi_25d/C28SOI_SC_12_CORE_LR@2.0@20130411.0/behaviour/verilog/C28SOI_SC_12_CORE_LR.v
../../synthesis/nano.v
../../tb/nanoTB.sv
```

Executar o comando **xrun -f file_list.f** - enviar os sinais do top para uma *waveform* e simular o circuito por 800 ns (reset; run 800ns).



Fazer um zoom entre ao redor do valor de *memory*[20]=12. Observar que agora temos atrasos em relação ao sinal *ck*, dado que estamos simulando o *netlist* no nível de portas lógicas, e não mais o projeto no nível RTL (SystemVerilog).



A importância desta simulação é a demonstração que o código HDL (VHDL ou SystemVerilog) está correto, e o circuito opera como o esperado no nível de portas lógicas.

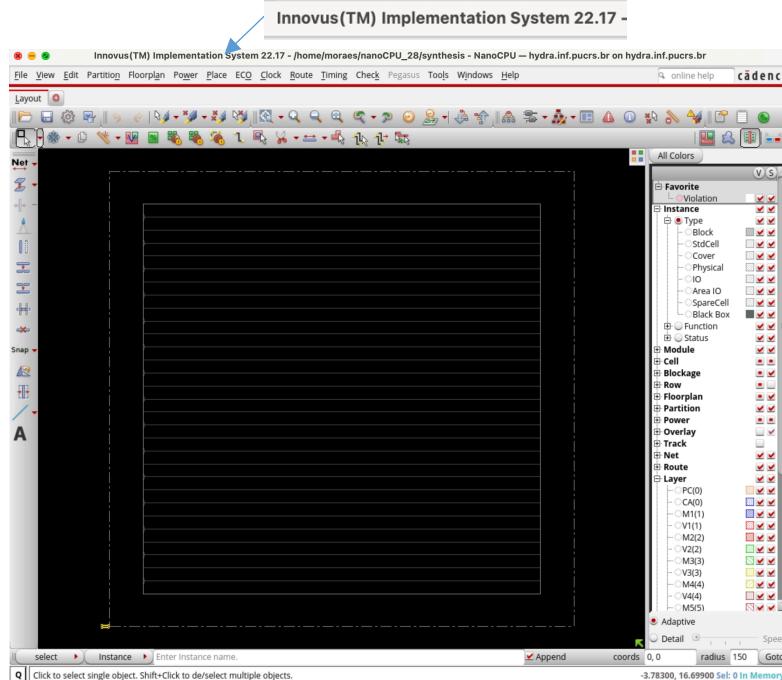
Ao sair do simulador recomenda-se apagar arquivos temporários: `xrun -clean`

ETAPA 4 – SÍNTSE 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 (*genus*) e a ferramenta *innovus* da CADENCE:

```
module load ddi
cd ../../synthesis
innovus -common_ui
```

`source physical/1_init.tcl` – carrega a configuração inicial da síntese física, digitando o seguinte comando no *shell* do *innovus*:



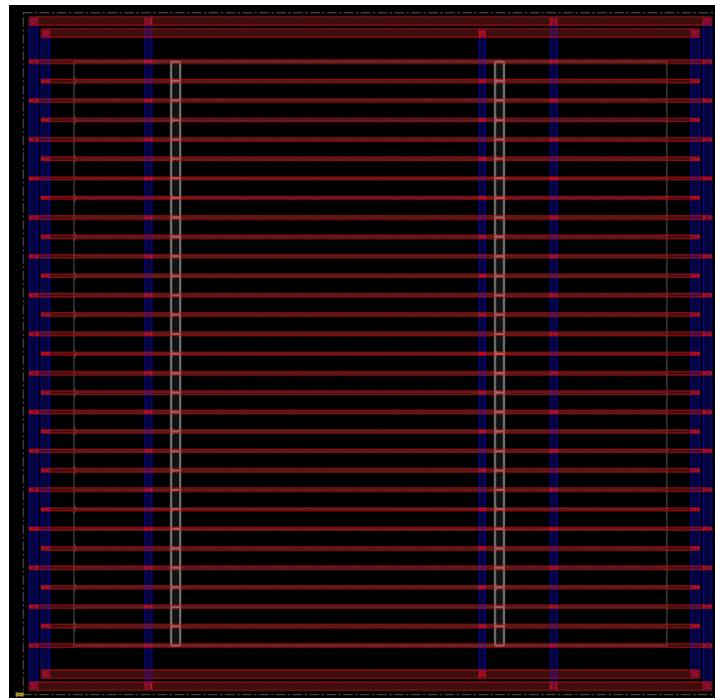
Abrir o arquivo *physical/1_init.tcl* e entender os comandos passados para o *innovus*. Dentre os comandos, os 2 principais são:

```
##Load the circuit configuration from genus
source layout/logical_NanoCPU.invs_setup.tcl

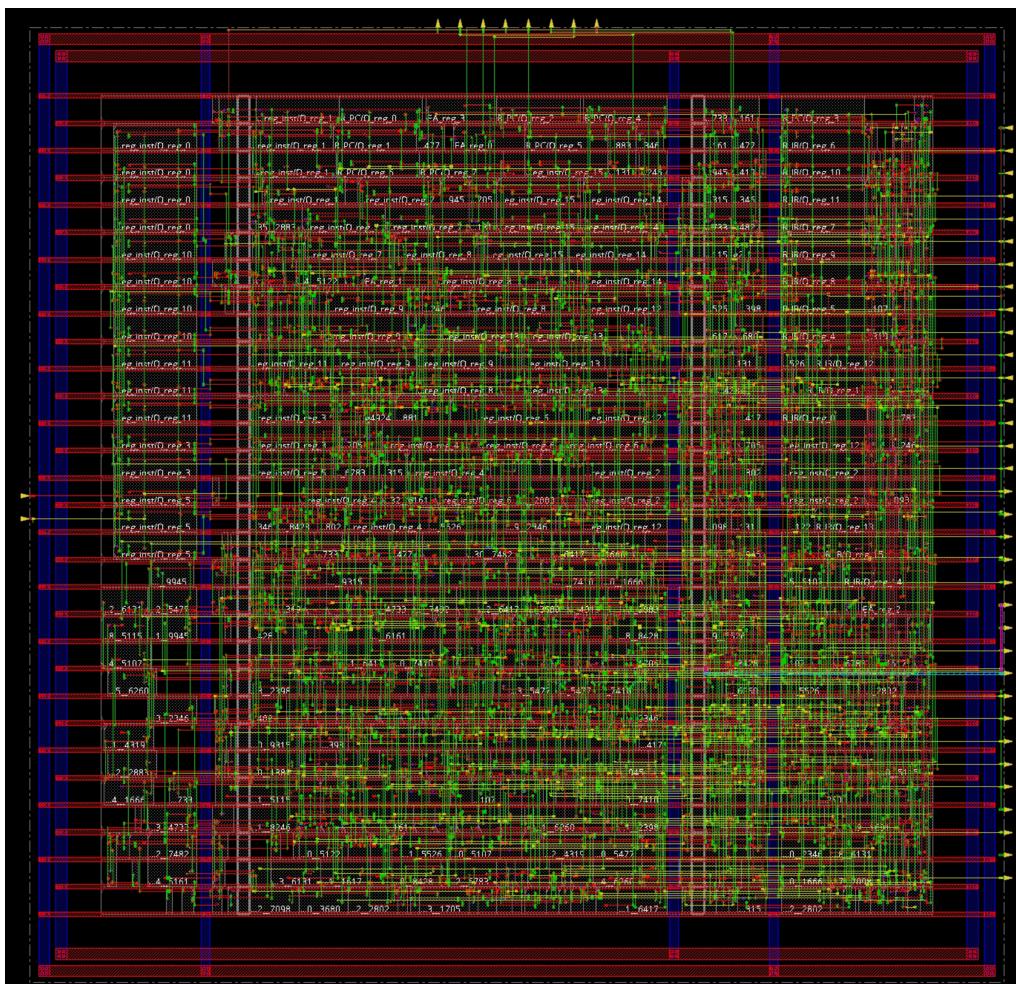
##Generating square floorplan (1) with 85% of density (0.85) with 3um margins
create_floorplan -site CORE12T -core_density_size 1 0.85 3 3 3 3
```

`source physical/2_power_plan.tcl` – carreg a configuração de *power planning*

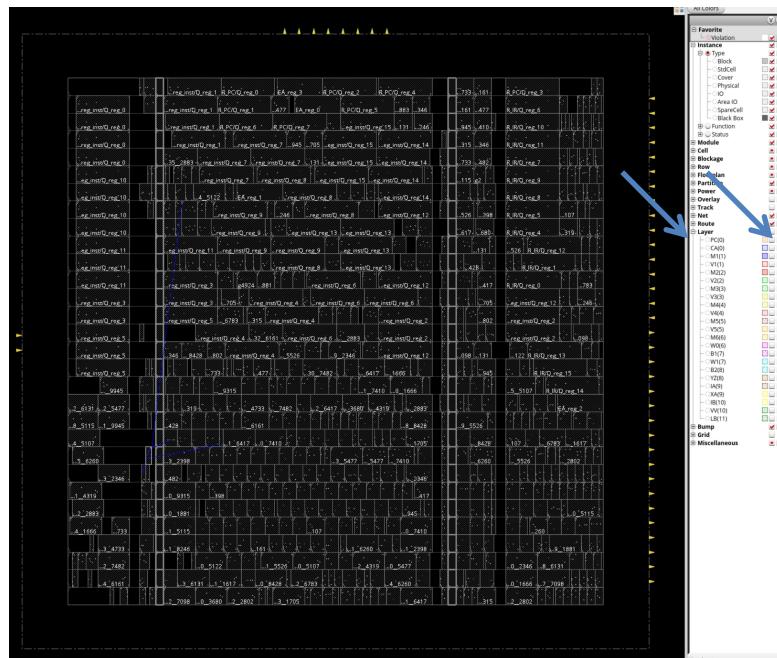
Abrir o arquivo *2_power_planning.tcl*. Notar que foi gerado um anel e linhas de alimentação, que serão utilizadas para posicionar as células lado a lado (*add_rings*). A simetria dessas linhas (mesma altura) facilita o algoritmo de posicionamento e a instanciação das células físicas. As linhas de alimentação correspondem aos retângulos vermelhos (metal 2 - *route_special*). Além disso, foram posicionadas colunas de *tap cells* (*add_well_taps*). 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 20 µm de distância uma da outra, para garantir polarização da difusão (informação obtida na documentação da biblioteca). Também foram inseridos reforços para a alimentação (*add_stripes*)



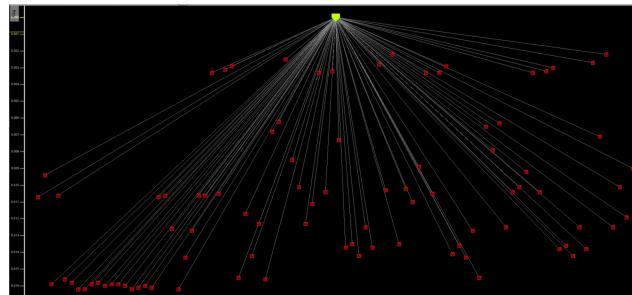
source physical/3_pin_clock.tcl - Instancia as células físicas no projeto, posiciona os pinos na periferia do circuito, e realizar a árvore de *clock*.



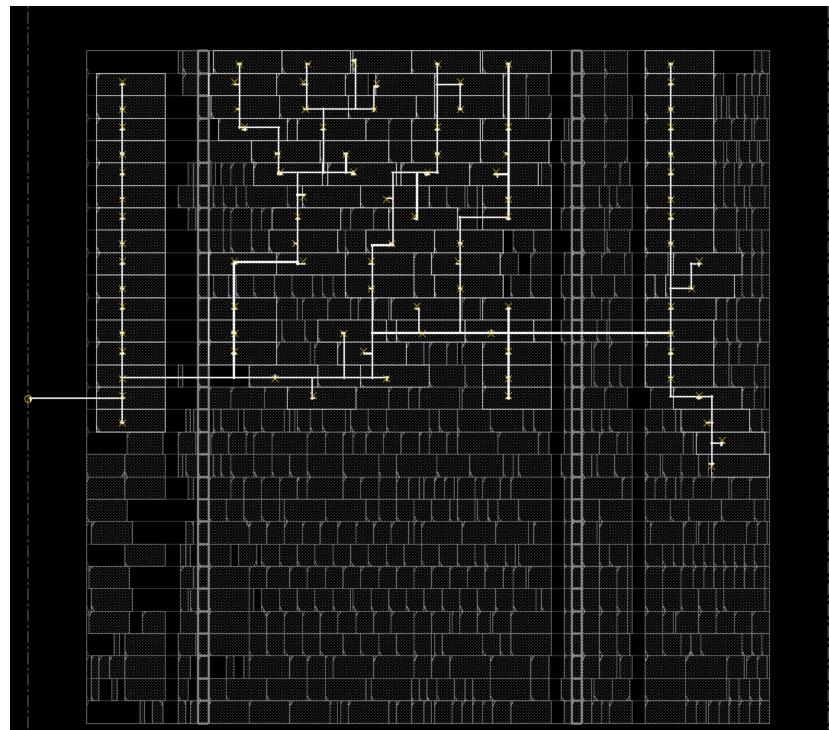
- observar que há um menu à direita – se as camadas não aparecerem, selecionar como visível (V) as instâncias, fios, etc. Desmarcar “layer” para visualizar apenas as instâncias das células.



No topo da tela selecionar **clock → CCOpt Clock Debugger** e OK. Esta ação abre uma janela, com os buffers de *clock*:



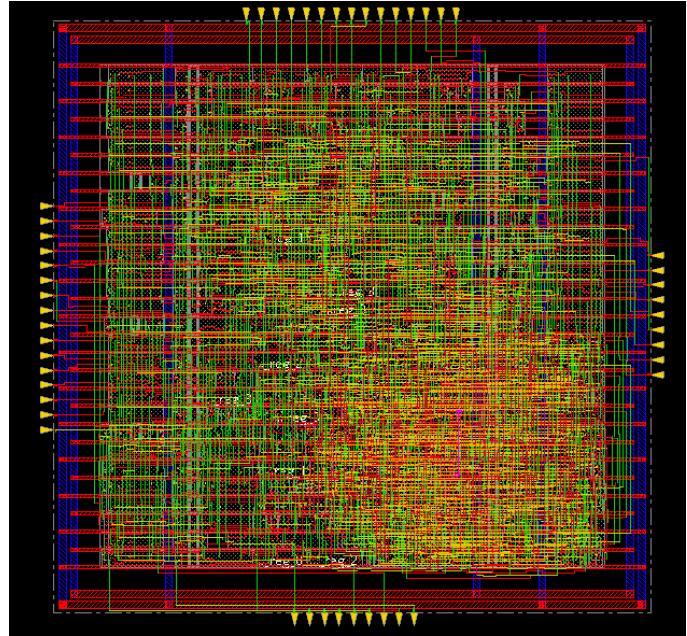
Na figura acima selecionei todo os sinais de *clock* e desmaquei “micellaeous” na pallete. O resultado é a visualização da árvore de *clock*, e de todos os registradores alimentados pelo *clock*



source physical/4_nano_route.tcl - executa o roteamento, ou seja, conectar as células e os pinos de entrada/saída.

Este script contém 4 comandos:

```
route_design
route_design -global_detail -wire_opt
set_db timing_analysis_type ocv
opt_design -post_route
```



source 5_fillers_reports.tcl. Notar que a figura acima possui “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 os espaços entre células e garantirão que o projeto não violará regras pelo motivo descrito.

```
** Starting Verify DRC (MEM: 3224.6) **

VERIFY DRC ..... Starting Verification
VERIFY DRC ..... Initializing
VERIFY DRC ..... Deleting Existing Violations
VERIFY DRC ..... Creating Sub-Areas
VERIFY DRC ..... Sub-Area: {0.000 0.000 42.840 42.000} 1 of 1
VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

Verification Complete : 0 Viols.
```

- Os comandos de extração de parasitas servem para gerar o verilog e o SDF para simulação com atraso de roteamento.

report_timing → observar agora que o slack **diminui** devido ao roteamento

```
Path 1: MET (0.070 ns) Setup Check with Pin genblk1_0.reg_inst/Q_reg_15/CP->TI
      View: default_emulate_view
      Group: ck
      Startpoint: (R) R_IR/Q_reg_1/CP
      Clock: (R) ck
      Endpoint: (F) genblk1_0.reg_inst/Q_reg_15/TI
      Clock: (R) ck

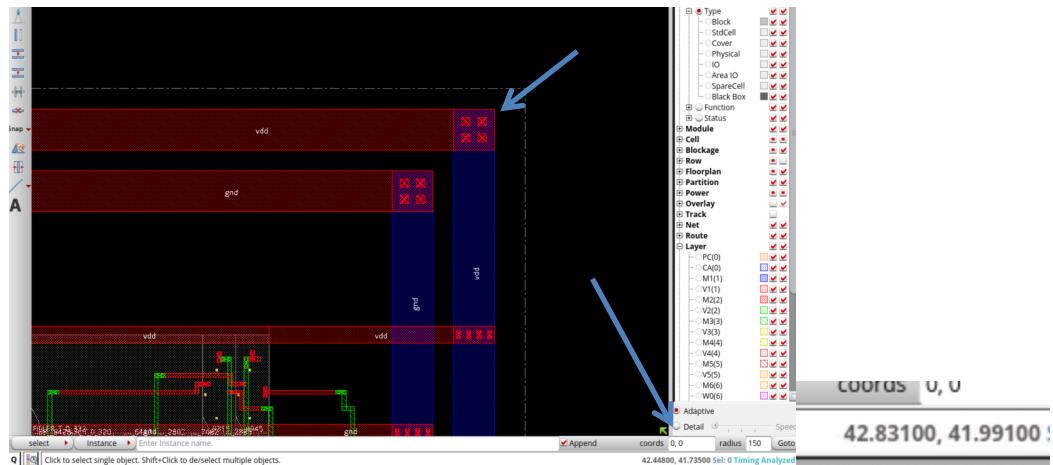
      Capture           Launch
      Clock Edge:+    2.000    0.000
      Src Latency:+   0.000   -0.011
      Net Latency:+   0.008 (P)  0.012 (P)
      Arrival:=       2.008    0.001

      Setup:-         0.184
      Required Time:= 1.823
      Launch Clock:=  0.001
      Data Path:+     1.752
      Slack:=        0.070
```

report_area

Hinst Name	Module Name	Inst Count	Total Area
NanoCPU		767	1113.677

Observar que esta é a área de células. Posicionar o mouse no canto superior do circuito como abaixo. O circuito tem uma área aproximada de $42.83\mu\text{m} \times 42 \mu\text{m}$, o que resulta em **$1798 \mu\text{m}^2$** .



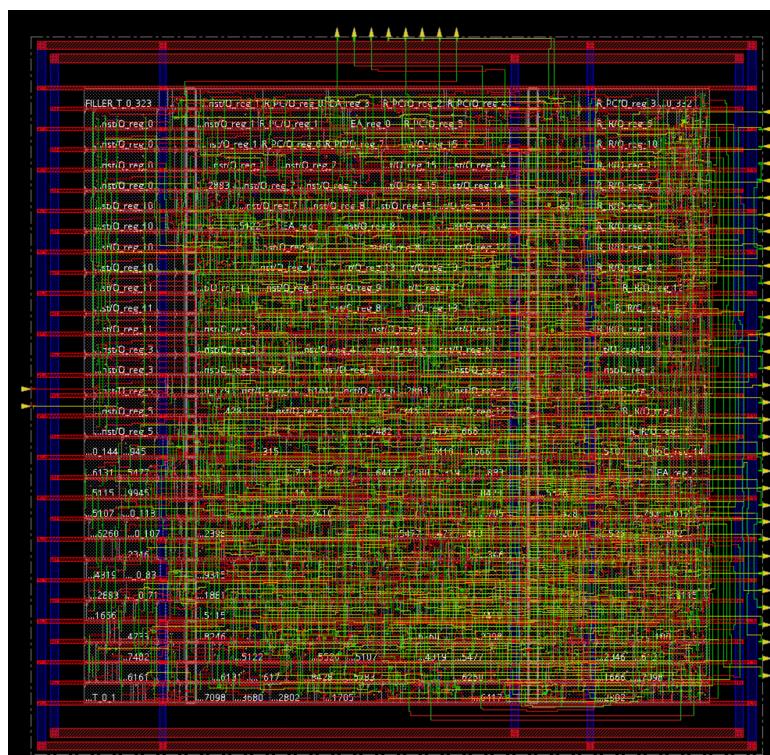
Observar que foi gerado um relatório do projeto, no diretório “*summaryReport*”. Executar o seguinte na linha de comando (pode ser dentro do *innovus*):

```
firefox summaryReport/NanoCPU.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:

Floorplan/Placement Information	
Total area of Standard cells	1317.024 um ²
Total area of Standard cells(Subtracting Physical Cells)	1113.677 um²
Total area of Core	1317.024 um ²
Total area of Chip	1799.280 um²

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



Para sair: **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 **nano_pr.sdf**. A descrição **sdf** é um formato de VHDL, e significa *Standard Delay Format*.

Executar:

```
module purge
module load xcelium
cd ..//sim/sdf
```

(pasta de simulação com atraso de roteamento)

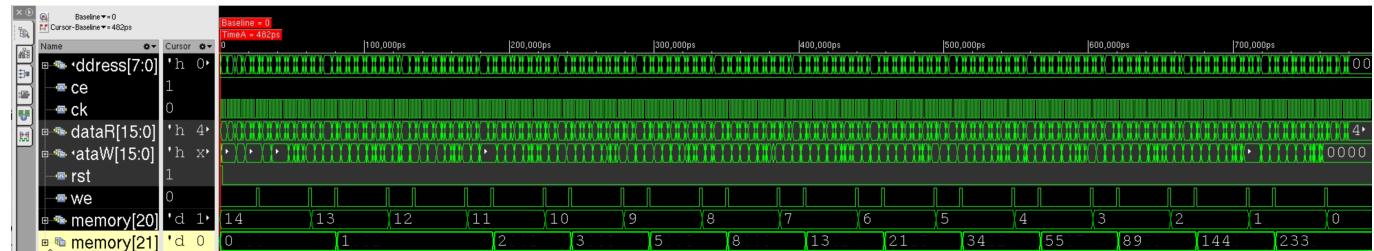
O script para esta simulação é similar ao de verificação pós-síntese, porém agora é dado um parâmetro a mais (**-sdf_cmd_file**), o script de configuração de atraso. Ver **more sdf_cmd.cmd**:

```
SDF_FILE = "../../synthesis/nano_pr.sdf",
LOG_FILE = "./sdf_log.log",
SCOPE = :CPU;
MTM_CONTROL = "MAXIMUM",
SCALE_FACTORS = "1.0:1.0:1.0",
SCALE_TYPE = "FROM_MAXIMUM";
```

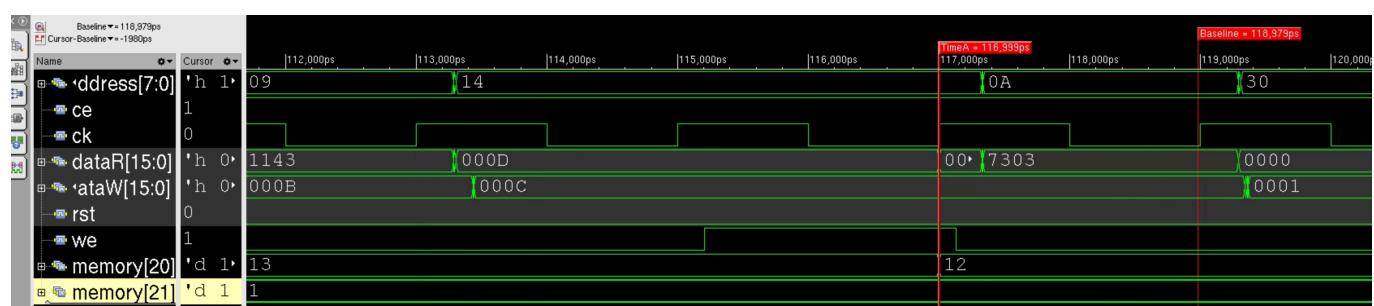
Arquivo: **file_list.f**

```
-smartorder -work work -top tb -gui -access +rw -maxdelays -sdf_cmd_file sdf_cmd.cmd
/soft64/design-kits/stm/28nm-cmos28fdsoi_25d/.../behaviour/verilog/C28SOI_SC_12_CLK_LR.v
/soft64/design-kits/stm/28nm-cmos28fdsoi_25d/.../behaviour/verilog/C28SOI_SC_12_CORE_LR.v
../../synthesis/nano_pr.v
./tb/nanoTB.sv
```

Executar o comando **xrun -f file_list.f** - enviar os sinais do top para uma *waveform* e simular o circuito por 800 ns (reset; run 800ns).



Visualizando atrasos:



Conferir o **sdf.log**, não deve ter erro neste arquivo.

Observar o arquivo **xrun.log** (parte transcrita abaixo). Indica que todas os caminhos foram “anotados”.

SDF statistics:

	No. of Pathdelays = 8460	No. of Disabled Pathdelays = 0	Annotated = 100.00% (8460/8460)
No. of Tchecks	= 1056	No. of Disabled Tchecks = 0	Annotated = 100.00% (1056/1056)
	Total(T)	Disabled(D)	Anotated(A)
Path Delays	8460	0	8460
\$width	442	0	442
\$recrrem	90	0	90
\$setuphold	524	0	524

Building instance overlay tables: Done

Abra o arquivo nano_pr.txt, e observe o caminho crítico. É possível navegar na hierarquia do projeto e visualizar a saída Q do IR, assim como a entrada D do Q_reg_15

Slack:= 0.070

#	Timing Point	Flags	Arc	Edge	Cell	Fanout	Trans (ns)	Delay (ns)	Arrival (ns)
<hr/>									
#	R_IR/Q_reg_1/CP	-	CP	R	(arrival)	90	0.023	-	0.001
#	R_IR/Q_reg_1/Q	-	CP->Q	R	C12T28SOI_LR_SDFFPR0X8_P0	2	0.023	0.084	0.085
#	FE_OFc4_IR_1/Z	-	A->Z	R	C12T28SOI_LR_CNBFX15_P0	20	0.023	0.067	0.152
#	g4835_2883/Z	-	S0->Z	F	C12T28SOI_LR_MUX41X8_P0	24	0.077	0.257	0.408
#	mul_98_34_g2387_4319/Z	-	A->Z	R	C12T28SOI_LRS_XOR2X6_P0	13	0.172	0.204	0.613
#	FE_OFc9_mul_98_34_n_36/Z	-	A->Z	F	C12T28SOI_LR_CNIVX8_P0	3	0.201	0.076	0.688
#	mul_98_34_g2275_7410/Z	-	A->Z	F	C12T28SOI_LR_AND2X8_P0	12	0.053	0.085	0.774
#	mul_98_34_g2208_5115/Z	-	A->Z	F	C12T28SOI_LR_A022X8_P0	1	0.049	0.088	0.862
#	mul_98_34_cdinfadd_004_0_2346/S0	-	B0->S0	F	C12T28SOI_LR_FA1X8_P0	1	0.024	0.071	0.933
#	mul_98_34_g2178_6131/C0	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	1	0.020	0.067	1.000
#	mul_98_34_g2177_7098/C0	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	1	0.023	0.068	1.068
#	mul_98_34_g2172_2802/C0	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	1	0.022	0.070	1.138
#	mul_98_34_g2164_6260/C0	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	4	0.025	0.079	1.217
#	mul_98_34_g2148_7482/Z	-	C->Z	R	C12T28SOI_LR_NAND4ABX6_P0	1	0.032	0.027	1.244
#	mul_98_34_g2140_8246/Z	-	D->Z	R	C12T28SOI_LR_0A112X8_P0	3	0.019	0.069	1.313
#	mul_98_34_g2134_6783/Z	-	B->Z	F	C12T28SOI_LR_NOR3AX6_P0	1	0.039	0.023	1.336
#	mul_98_34_g2133_5526/Z	-	D->Z	F	C12T28SOI_LR_A0112X8_P0	1	0.015	0.085	1.422
#	mul_98_34_g2131_4319/C0	-	CI->CO	F	C12T28SOI_LR_FA1X8_P0	1	0.026	0.071	1.492
#	mul_98_34_g2418_5115/Z	-	C->Z	F	C12T28SOI_LRS_XNOR3X4_P0	1	0.023	0.079	1.571
#	g4653_6417/Z	-	S0->Z	F	C12T28SOI_LR_MX41X7_P0	2	0.052	0.107	1.678
#	g4627_2398/Z	-	D0->Z	F	C12T28SOI_LR_MUX21X8_P0	4	0.043	0.076	1.753
#	genblk1_0.reg_inst/Q_reg_15/TI	-	TI	F	C12T28SOI_LR_SDFFPR0X8_P0	4	0.026	0.000	1.753



FINAL DO TUTORIAL