Lab 4 – Fusion Compiler – synteza i implementacja top-down

I. Oprogramowaniem Fusion Compiler

Fusion Compiler firmy Synopsys to oprogramowanie realizujące ujednoliconą syntezę logiczną i fizyczną od poziomu RTL do poziomu masek technologicznych w formacie GDSII. Fusion Compiler łączy tradycyjna syntezę logiczną z tworzeniem floorplanu, rozmieszczaniem komórek i połączeń (ang. place and route) i optymalizacją.

1. Uruchamianie oprogramowania Fusion Compiler

Fusion Compiler to oprogramowanie pracujące pod kontrolą systemów Unix/Linux (wybranych dystrybucji). Przed uruchomieniem programu Fusion Compiler konieczne jest ustawienie odpowiednich zmiennych i ścieżek systemowych. W tym celu należy wykonać przygotowany uprzednio skrypt, wywołując z linii komend następujące polecenie:

```
add-synopsys-IMP-all
```

Następnie należy przejść od katalogu roboczego i uruchomić program Fusion Compiler wpisując w linii komend polecenie:

```
fc_shell
```

Zaleca się aby przed uruchomieniem programu Fusion Compiler uruchomić w terminalu nową powłokę poprzez wykonanie polecenia:

```
bash
```

W razie wystąpienia problemów we współpracy z omawianym programem powyższy zabieg ułatwi powrót do pierwotnego środowiska pracy.

Samo polecenie `fc_shell` uruchamia program Fusion Compiler w trybie terminalowym. Aby uruchomić ten program w trybie graficznym konieczne jest użycie przełącznika linii komend `-gui`, jak to pokazano poniżej:

```
fc_shell -gui &
```

Interfejs graficzny programu można także włączyć i wyłączyć z `fc_shell` używając odpowiednio isntrukcji:

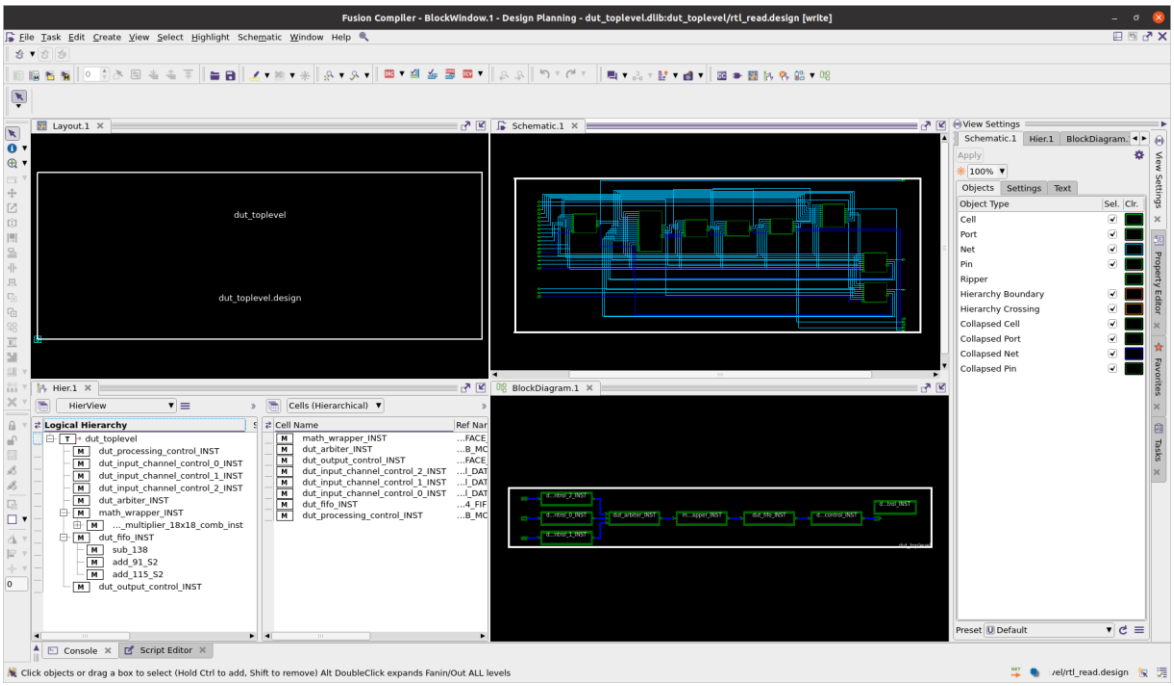
```
gui_start
gui_stop
```

Program Fusion Compiler pozwala także na wykonanie zaraz po uruchomieniu wskazanego skryptu w języku TCL. W takim przypadku program należy wywołać w jednym z dwóch poniższych sposobów:

```
fc_shell -f <ścieżka_do_skryptu>/<nazwa_skryptu>.tcl
fc_shell -f <ścieżka_do_skryptu>/<nazwa_skryptu>.tcl -gui
```

2. Interfejs graficzny programu Fusion Compiler

Widok głównego okna programu Fusion Compiler pokazano na rys. 1. U góry okna widoczny jest pasek menu i pasek narzędzi. Poniżej otwartych jest kilka okien podrzędnych, m. in. okno przedstawiające layout układu (tu zawiera tylko jedną komórkę), okna zawierające schemat logiczny oraz schemat blokowy układu a także okno prezentujące hierarchię projektu. Po prawej stronie głównego okna znajduje się pasek zakładek.



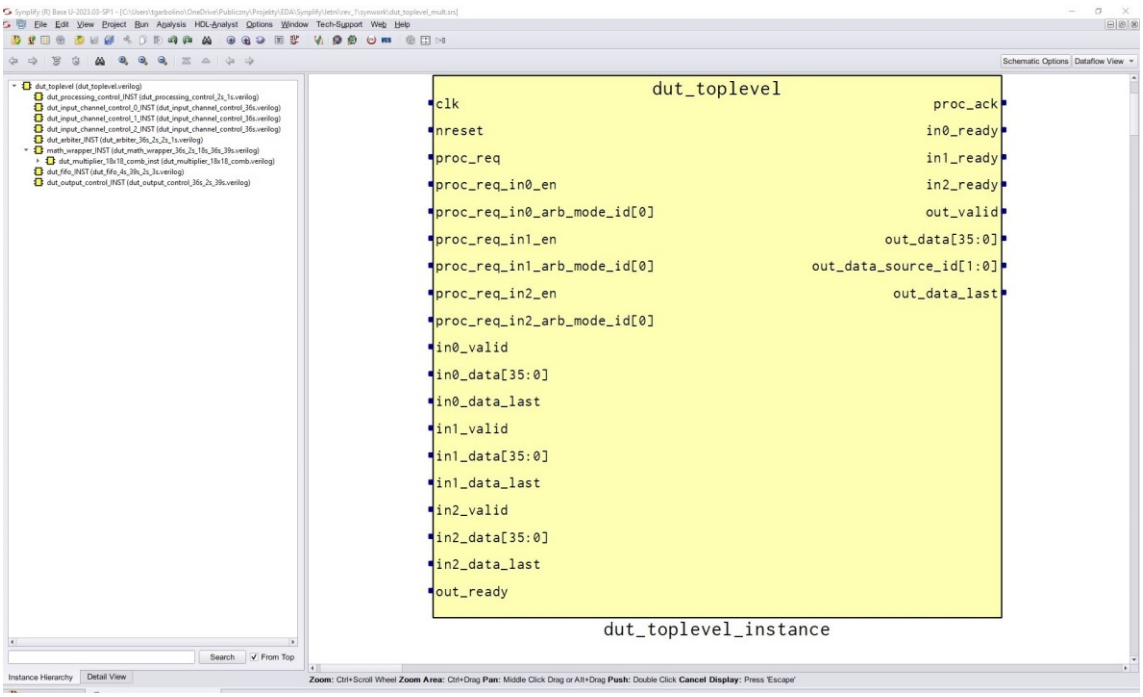
Rys. 1. Widok głównego okna programu Fusion Compiler

Na rys. 1 widoczna jest akurat zakładka ustawień wyświetlania.

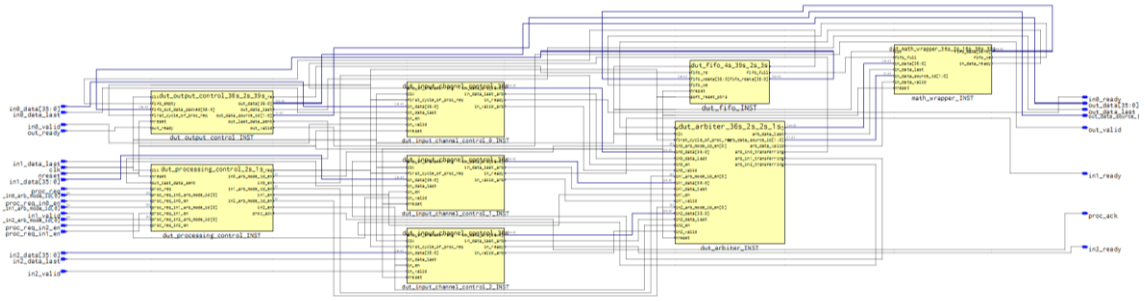
II. Zapoznanie się z realizowanym projektem

W trakcie zajęć laboratoryjnych zostanie wykonana synteza i implementacja modelu układu DUT (ang. Design Under Test) udostępnionego przez firmę Intel. Model został napisany całkowicie w języku opisu sprzętu SystemVerilg HDL. Interfejs i strukturę hierarchiczną modelu pokazano na rys. 2 (jest to widok jednego z okien programu Synplify firmy Synopsys).

Wewnętrzną strukturę blokową modelu pokazano z kolei na rys. 3 (ten schemat także został wygenerowany przez program Synplify). DUT zawiera trzy kanały wejściowe oraz kanał wyjściowy wraz z układami zarządzającymi transmisją danych i kolejką FIFO, moduł arytmetyczny, układ arbitrażu a także układ sterujący całością przetwarzania danych.



Rys. 2. Struktura hierarchiczna i interfejs modelu DUT



Rys. 3. Schemat blokowy modelu DUT

III. Przebieg ćwiczenia

W ramach zajęć laboratoryjnych zostanie wykonanych kilka ćwiczeń prezentujących przebieg syntezy logicznej i implementacji układu cyfrowego do postaci układu ASIC.

Każde ćwiczenie rozpoczynamy od przejścia do katalogu:

`<lab_root_path>/asic_lab4/lab/task<task_no>/work`

gdzie `<lab_root_path>` to ścieżka do miejsca, w którym umieszczono katalog `asic_lab4` wraz z podkatalogami, natomiast `<task_no>` to numer aktualnie wykonywanego zadania.

W katalogu `work` znajduje się skrypt `Makefile` zawierający listę operacji dla polecenia systemowego `make`. Wpisanie w linii komend polecenia:

```
make clean
```

powoduje doprowadzenie katalogów związanych z danym ćwiczeniem do stanu początkowego. Zaleca się wykonanie powyższego polecenia przed rozpoczęciem i powtórzeniem każdego ćwiczenia. Z kolei polecenie:

```
make run
```

powoduje uruchomienie skryptu TCL tworzonego przez użytkownika w trakcie zajęć.

Każde z ćwiczeń należy rozpocząć od wykonania w *fc_shell* skryptów inicjujących odpowiednie zmienne potrzebne w dalszej realizacji ćwiczenia:

```
source -echo ../../setup/main_setup.tcl
source -echo ${SetupDir}/design_setup.tcl
```

Ćwiczenie 2

Podobnie jak na poprzednich zajęciach laboratoryjnych implementację układu DUT rozpoczniemy od utworzenia biblioteki projektu. Tym razem wykorzystamy jednak do tego celu bibliotekę komórek typu `NDM frame_only`, czyli niezawierającą szczegółowych parametrów czasowych komórek. Wobec tego konieczne będzie wykorzystanie dodatkowych bibliotek w formacie `Liberty` (pliki z rozszerzeniem `.lib` lub `.db`) oraz pliku technologicznego. W tym przypadku użytkownik przed utworzeniem biblioteki projektu musi wskazać odpowiednie pliki bibliotek w formacie `Libery` (tj. pliki z rozszerzeniem `.db`). Sekwencja poleceń tworząca bibliotekę projektu wygląda następująco:

```
set_app_var link_library "${DB_FF} ${DB_TT} ${DB_SS}"
create_lib ${ResultsDir}/${DesignLibrary} \
     -technology ${TechFile} -ref_libs ${RefLib}
```

Aby sprawdzić poprawność wykonania powyższej operacji proszę skorzystać z polecenia `report_ref_libs` i zamieścić wyniki jego działania w sprawozdaniu.

Następnie należy dokonać kompilacji i elaboracji plików projektowych oraz wskazać poleceniem `set_top_module` główny moduł układu. Po tych operacjach zapisujemy bieżący blok projektu pod nazwą `rtl_read`, po czym zamykamy wszystkie bloki projektu (uprzednio można jeszcze wyświetlić listę dostępnych bloków w bibliotece projektu) oraz zapisujemy i zamykamy bibliotekę projektu.

Na zakończenie tego etapu proszę przekopiować bibliotekę projektu do katalogu *<lab_root_path>/asic_lab4/lab/solution*, dzięki czemu będzie ją można łatwo odtworzyć w wypadku pojawienia się problemów na dalszych etapach projektu. W tym celu należy wykorzystać polecenie `copy_lib -to_lib`.

Ćwiczenie 3

Przedmiotem ćwiczenia jest opracowanie scenariuszy syntezy wielokryterialnej (ang. Multi Corner Multi Mode setup) a także opanowanie metod kontrolowania, który z parametrów układu jest głównym celem optymalizacji.

Przyjmijmy, że układ DUT ma dwa tryby pracy: praca normalna oraz tryb o zredukowanym poborze mocy. Pierwszy z trybów nazwiemy `Normal` a drugi `LowPower`. Dla każdego z tych trybów zostaną zdefiniowane trzy tzw. narożniki technologiczne (ang. process corner, PVT corner): `Slow`, `Typical` oraz `Fast`. Dla każdej kombinacji trybu i narożnika technologicznego zostanie utworzony osobny scenariusz. Ponadto dla każdego z w/w trybów zostanie przygotowany osobny plik z ograniczeniami projektowymi w formacie SDC: `dut_toplevel_normal.sdc` oraz `dut_topelevel_power_save.sdc`.

Definicje narożników technologicznych a także trybów oraz scenariuszy zostaną umieszczone w osobnym skrypcie TCL o nazwie `mcmm_setup.tcl`, tak żeby można się było do nich wielokrotnie łatwo odwoływać w przyszłości.

Skrypt rozpoczynamy od usunięcia wszystkich aktualnie zdefiniowanych narożników technologicznych, trybów i scenariuszy.

```
remove_corners   -all
remove_modes     -all
remove_scenarios -all
```

Kolejny krok to utworzenie trzech narożników technologicznych: `Fast`, `Typical` oraz `Slow` oraz wyspecyfikowanie dla każdego z narożników parametrów pasożytniczych.

```
create_corner Fast
create_corner Typical
create_corner Slow

set_parasitics_parameters -early_spec minTLU \
     -late_spec  minTLU  -corners {Fast}
set_parasitics_parameters -early_spec nomTLU \
     -late_spec  nomTLU -corners {Typical}
set_parasitics_parameters -early_spec maxTLU \
     -late_spec  maxTLU -corners {Slow}
```

Następnie definiujemy tryby pracy układu i dla każdej kombinacji trybu i narożnika technologicznego tworzymy osobny scenariusz. W ten sposób łącznie otrzymujemy sześć scenariuszy: `Normal_Fast`, `Normal_Typical`, `Normal_Slow`, `PowerSave_Fast`, `PowerSave_Typical`, `PowerSave_Slow`. Dla każdego scenariusza wczytujemy osobno ograniczenia projektowe zdefiniowane w formacie SDC.

Poniżej przedstawiono przykładowy szablon pliku zawierającego ograniczenia projektowe w formacie SDC, który będzie używany w przypadku tego projektu.

```
set ClockName clk

set Period <okres>
set RiseTransition <czas_narastania>
set FallTransition <czas_opadania>
set Uncertainty <niepewność>

create_clock -period $Period \
  -name $ClockName [get_ports $ClockName]
set_clock_uncertainty \
  -setup $Uncertainty [get_clocks $ClockName]
set_clock_transition \
  -rise $RiseTransition [get_clocks $ClockName]
set_clock_transition \
  -fall $FallTransition [get_clocks $ClockName]
```

Wartości jakie należy przypisać poszczególnym parametrom sygnału zegarowego dla różnych trybów pracy układu podano w poniższej tabeli.

Parametr / Tryb	Normal	PowerSave
Okres	2	10
Czas narastania	0.1	0.3
Czas opadania	0.2	0.5
Niepewność	0.3	0.7

Ponadto dla każdego narożnika technologicznego i odpowiadającego mu scenariusza definiujemy warunki pracy określone przez proces technologiczny, napięcie zasilania oraz temperaturę. Każdy scenariusz musi zostać także odpowiednio skonfigurowany pod kątem różnego typu analiz.

Ponieważ wszystkie wymienione wyżej operacje wykonujemy dla każdego z trybów pracy układu, wskazane jest użycie pętli języka TCL `foreach`, tak jak to zilustrowano na poniższym listingu.

```
foreach mode_name {Normal PowerSave} {

  echo "Current mode: ${mode_name}"

  # Create Mode
  create_mode  ${mode_name}
  current_mode ${mode_name}

  # Create Scenario(s) for current mode
  create_scenario -mode ${mode_name} -corner Fast \
       -name ${mode_name}_Fast
  create_scenario -mode ${mode_name} -corner Typical \
       -name ${mode_name}_Typical
  create_scenario -mode ${mode_name} -corner Slow \
       -name ${mode_name}_Slow

  # Read Constraints for each scenario
  set scenarios [get_attribute [get_scenarios \
                                  -modes ${mode_name}] name]

  echo "Current mode scenarios: ${scenarios}"

  foreach scenario ${scenarios} {
       echo "Current scenario: ${scenario}"
       current_scenario ${scenario}
       read_sdc [set ${mode_name}ModeSdcFile]
  }

  # Set operating conditions for each corner and scenario
  current_corner Fast
  current_scenario ${mode_name}_Fast
  set_operating_conditions ${PVT_FF}

  current_corner Typical
  current_scenario ${mode_name}_Typical
  set_operating_conditions ${PVT_TT}

  current_corner Slow
  current_scenario ${mode_name}_Slow
  set_operating_conditions ${PVT_SS}

  # Scenario configuration
  set_scenario_status {mode_name}_Fast -setup false \
    -hold true -leakage_power false -dynamic_power true \
    -max_transition false -max_capacitance true -active true
  set_scenario_status {mode_name}_Typical -all -active true
  set_scenario_status {mode_name}_Slow -setup true \
    -hold false -leakage_power true -dynamic_power true \
    -max_transition true -max_capacitance false -active true

}
```

Po zdefiniowaniu ograniczeń oraz strategii projektowych możemy przejść do syntezy logicznej i wstępnej optymalizacji układu. W tym celu należy wykonać polecenie:

```
compile_fusion -to logic_opto
```

W trakcie etapu syntezy o nazwie `logic_opto` przeprowadzana jest optymalizacja opóźnień w układzie a także tworzony jest automatycznie floorplan.

Przykładowy przebieg syntezy układu do etapu `logic_opto` ilustruje poniższy listing.

```
# Open the design library
open_lib ${ResultsDir}/${DesignLibrary}

# Copy and open block
copy_block -from ${DesignName}/rtl_read \
     -to ${DesignName}/mcmm_and_logic_opto_general
open_block ${DesignName}/mcmm_and_logic_opto_general

# Source tech setup script
source -echo ${SetupDir}/technology_setup.tcl


# Read the constraints
read_sdc -echo ${SdcFile}

# MCMM setup
source -echo ${SetupDir}/mcmm_setup.tcl

# Initial mapping
compile_fusion -to logic_opto

# Collecting the reports
set TargetName "logic_opto_general"
generateReports ${TargetName}

# Gate level netlist generation
write_verilog ${ResultsDir}/${DesignName}_${TargetName}.v

# Saving block and library
current_block
get_blocks -all
list_blocks
save_block
save_lib
close_blocks
close_lib
```

Proces syntezy logicznej można ukierunkować na optymalizację:

- parametrów czasowych układu (ang. timing)
- powierzchni układu (ang. area)
- mocy (ang. power)

W tym celu należy ustawić odpowiednie opcje programu bezpośrednio przed wykonaniem komendy `fusion_compile`.

W celu optymalizacja parametrów czasowych używamy opcji:

```
set_app_options -name compile.flow.high_effort_timing -value 2
```

Powyższa opcja może przyjmować wartości od 0 do 2. Wyższe wartości powoduję położenie większego nacisku na uzyskanie jak najkrótszych czasów propagacji.

Natomiast aby położyć nacisk na redukcję powierzchnię układu i wydzielanej przez niego mocy należy użyć odpowiednio opcji:

```
set_app_options -name compile.flow.high_effort_area -value true
set_app_options -name compile.flow.enable_power -value true
```

Projekt

- Przygotuj skrypty optymalizujące parametry czasowe, powierzchnię i pobór mocy układu DUT. Nazwij je odpowiednio *task2_timing.tcl*, *task2_area.tcl* oraz *task2_power.tcl* .
- Przeanalizuj i porównaj w postaci tabelarycznej/graficznej parametry układu DUT zawarte w raportach.

Ćwiczenie 4
Tematem tego ćwiczenia jest tworzenie floorplanu układu. Najważniejsze parametry floorplanu wymieniono poniżej.

Aspect ratio – stosunek wysokości do szerokości rdzenia układu. Jeżeli ten parametr nie zostanie jawnie zdefiniowany przez użytkownika, to narzędzie stworzy kwadratowy rdzeń układu.
Row / Core ratio – przyjmuje wartości do 0.0 do 1.0. Określa wielkość przestrzeni pozostawionej na prowadzenie połączeń między komórkami. Mniejsze wartości tego parametru oznaczają więcej przestrzeni na prowadzenie połączeń.
Core width: szerokość rdzenia układu
Core height: wysokość rdzenia układu
Number of rows: liczba wierszy przeznaczona na rozmieszczenie komórek
Block shape: dostępne są różne kształty rdzenia układu – m. in w kształcie prostokąta, liter L, T i U.
Core Utilization: stopień zapełnienia rdzenia układu komórkami
Die width: szerokość układu
Die height: wysokość układu

1. Automatyczne generowanie floorplanu

Oprogramowanie Fusion Compiler jest w stanie wygenerować floorplan automatycznie co zilustrowano na poniższym listingu.

```
open_lib ${ResultsDir}/${DesignLibrary}

copy_block -from ${DesignName}/rtl_read \
     -to ${DesignName}/auto_floorplan
open_block ${DesignName}/auto_floorplan

source -echo ${SetupDir}/technology_setup.tcl
```

```
read_sdc -echo ${SdcFile}

source -echo ${SetupDir}/mcmm_setup.tcl

# Setup application options
set_app_options \
     -name place.coarse.continue_on_missing_scandef \
     -value true
set_app_options \
     -name compile.auto_floorplan.enable \
     -value true

compile_fusion -check_only

compile_fusion -to logic_opto
```

Po wykonaniu powyższych poleceń włącz interfejs graficzny programu i zaobserwuj wyniki automatycznego generowania floorplanu. Rdzeń układu powinien mieć kształt kwadratu o stopniu wypełnienia 0.7. Charakterystyczny jest brak odstępu między brzegami rdzenia a brzegami całego układu. Wszystkie piny we/wy powinny być utworzone i rozmieszczone.

Parametry tego typu floorplanu można dostroić poleceniem:

```
set_auto_floorplan_constraints \
    -control_type core \
    -core_utilization 0.5 \
    -core_offset 10 \
    -shape R \
    -side_ratio {1 3} \
    -flip_first_row true


report_auto_floorplan_constraints
```

W wyniku jego wykonania zostanie stworzony floorplan rdzenia układu w kształcie prostokąta o stopniu wypełnienia komórkami 0.5. Wysokość prostokąta jest 3 razy większa od jego szerokości. Proszę też zwrócić uwagę na dostęp między brzegami rdzenia i całego układu wynoszący 10.

Istnieje również możliwość kontrolowania rozmieszczenia pinów we/wy poprzez przypisanie ich do wskazanych krawędzi układu oraz warstw metalizacji. Ponadto można również określić rozmiary i wzajemne odstępy pinów. Temu celowi służy polecenie `set_block_pin_constraints`, którego przykładowe użycie zilustrowano poniżej.

```
set_block_pin_constraints -self \
    -allowed_layers {M2 M3} \
```

```
    -sides {1 2 3} \
    -pin_spacing_distance 1 \
    -width 0.19 \
    -length 0.19
```

Piny zostaną rozłożone na wszystkich krawędziach układu oprócz dolnej i będą przypisane do warstw metalizacji M2 i M3.

Oprogramowanie Fusion Compiler pozwala również na kontrolowanie parametrów pojedynczych pinów za pośrednictwem polecenia `set_individual_pin_constraints`. Przykład zastosowania tego polecenia w stosunku do pinu wejścia zegarowego `clk` podano poniżej.

```
set_individual_pin_constraints \
    -ports [get_ports clk] \
    -sides 4 \
    -allowed_layers M4
```

Pin wejściowy sygnału zegarowego zostanie umieszczony na dolnej krawędzi układu i przypisany do warstwy metalizacji M4.

Do wyświetlenia informacji o ograniczeniach nałożonych na piny we/wy służy polecenie:

```
report_block_pin_constraints -self
```

Aby wygenerować nowy floorplan zgodnie z powyższymi ograniczeniami trzeba ponownie wykonać polecenie:

```
compile_fusion -to logic_opto
```

Po tej operacji zaleca się uruchomienie polecenia:

```
report_congestion -rerun_global_router
```

w celu sprawdzenia i usunięcia ewentualnych nadmiernych zagęszczeń komórek (ang. congestion). Wskazane jest również zapisanie raportów poleceniem oraz ich późniejsze przeanalizowanie:

```
generateReports tuned_auto_floorplan
```

Ten etap ćwiczenia kończymy zapisując stworzony floorplan:

```
write_floorplan -output ${ResultsDir}/auto_floorplan_files \
```

```
  -exclude {cells nets}
write_def -exclude {cells nets} \
  ${ResultsDir}/auto_floorplan.def
```

Proszę również nie zapomnieć o zapisaniu i zamknięciu bloku i biblioteki projektu.

2. Ręczne tworzenie floorplanu

Zawartość skryptu przeznaczonego do ręcznego stworzenia floorplanu podano poniżej:

```
# Open the design library
open_lib ${ResultsDir}/${DesignLibrary}

# Copy and open block
copy_block -from ${DesignName}/auto_floorplan \
-to ${DesignName}/manual_floorplan
open_block ${DesignName}/manual_floorplan

#### Initialize the floorplan
initialize_floorplan  \
     -control_type core \
     -core_utilization 0.55 \
     -core_offset 5 \
     -shape R \
     -side_ratio {2.5 1.5} \
     -flip_first_row true

#### Set pin placement constraints
set ports [remove_from_collection [get_ports] {VDD VSS}]

set_block_pin_constraints -self \
     -allowed_layers {M3 M4} \
     -sides {1 2 3} \
     -pin_spacing_distance 1 \
     -width 0.11 \
     -length 0.11
```

```
set_individual_pin_constraints \
     -ports [get_ports clk] \
     -sides 4 \
     -allowed_layers M5

place_pins -self -ports ${ports}

#### Insert Boundary and TAP cells in the design
source -echo ../scripts/insert_special_physical_cells.tcl

#### Write out the created floorplan
write_floorplan -output \
${ResultsDir}/manual_floorplan_files -exclude {cells nets}

#### Wrtl_readrite DEF for created floorplan
write_def -exclude {cells nets} \
${ResultsDir}/manual_floorplan.def

generateReports manual_floorplan

get_blocks -all
save_block
save_lib

close_blocks
close_lib
```

Należy zwrócić uwagę na polecenie `initialize_floorplan`, które inicjalizuje floorplan od nowa, definiując jego nowe parametry. Dodano również specjalne komórki brzegowe i TAP, których zadaniem jest poprawa własności elektrycznych układu.

3. Stworzenie sieci zasilania

Wykonany ręcznie floorplan uzupełnimy o siatkę sieci zasilania, za co odpowiedzialny jest skrypt `create_pg_network.tcl` na poniższym listingu.

```
# Open the design library
open_lib ${ResultsDir}/${DesignLibrary}

# Copy and open block
```

```
copy_block -from ${DesignName}/rtl_read \
  -to ${DesignName}/final_floorplan
open_block ${DesignName}/final_floorplan

# Source tech setup script
source -echo ${SetupDir}/technology_setup.tcl

# Read the constraints
read_sdc -echo ${SdcFile}

# MCMM setup
source -echo ${SetupDir}/mcmm_setup.tcl

#### Read floorplan from DEF
read_def ${ResultsDir}/manual_floorplan.def

#### Insert Boundary and TAP cells in the design
source -echo ../scripts/insert_special_physical_cells.tcl

#### Create Power/Ground Network
source -echo ../scripts/create_pg_network.tcl

save_block
save_lib
close_blocks
close_lib
```

Sprawozdanie

W sprawozdani należy zamieścić zawartość opracowanych skryptów. Ponadto należy przenalizować i porównać dane ze wszystkich wygenerowanych raportów i na tej podstawie wyciągnąć wnioski.