Lab 3 – Fusion Compiler – synteza i implementacja top-down

I. Zapoznanie się z 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 &
```

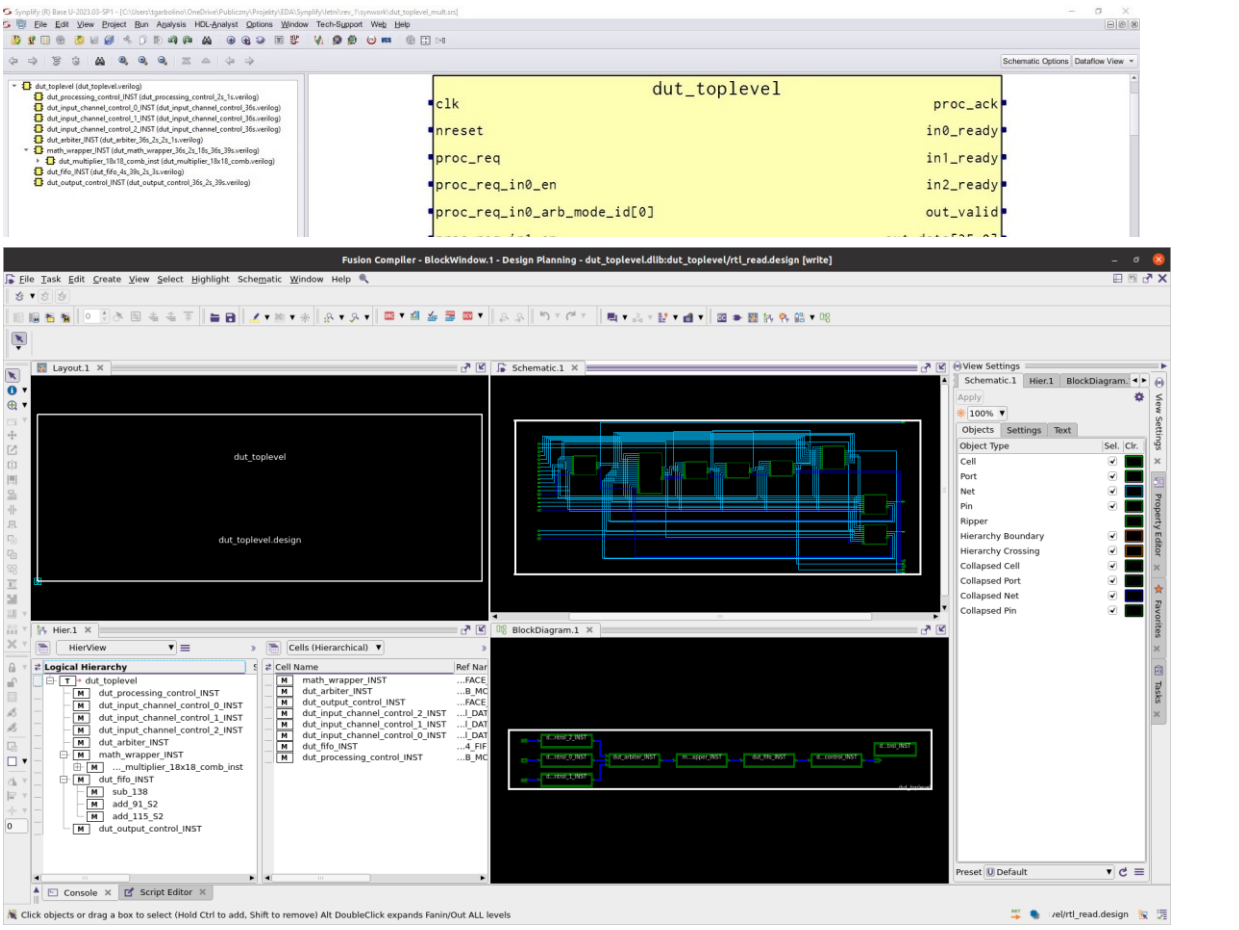
Interfejs graficzny programu można także włączyć i wyłączyć z `fc_shell` używając odpowiednio isntrukcji:

```
gui_start
gui_stop
```

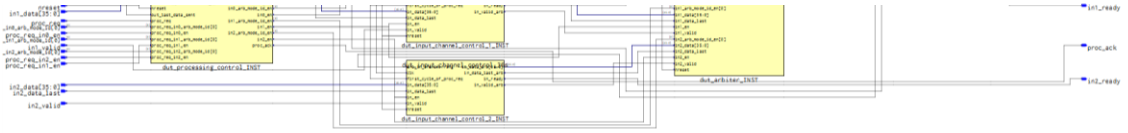
Program Fusion Compiler pozwala także na wykonanie zaraz po uruchomieniu wskazanego skryptu w języku TCL. W takim przypadku program należy wywołać w jednym z dwóch poniższych sposobów:

```
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



Rys. 1. Widok głównego okna programu Fusion Compiler



Rys. 2. Schemat blokowy modelu DUT

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. 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. 3 (jest to widok jednego z okien programu Synplify firmy Synopsys).

Wewnętrzną strukturę blokową modelu pokazano z kolei na rys. 2 (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.

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_lab3/lab/task<task_no>/work
```

gdzie *<lab_root_path>* to ścieżka do miejsca, w którym umieszczono katalog *asic_lab3* 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. Nazwa tego skryptu wraz z pełną ścieżką ma postać:

```
<lab_root_path>/asic_lab3/lab/task<task_no>/scripts/task<task_no>.tcl
```

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 1

W trakcie tego ćwiczenia zostanie przeprowadzona wstępna synteza logiczna i fizyczna układu a także optymalizacja jego parametrów PPA (ang. power, performance, area).

Przed przystąpieniem do wykonania ćwiczenia konieczne jest dostosowanie modelu DUT do potrzeb syntezy i implementacji w programie Fusion Compiler. Należy dodać do interfejsów wszystkich modułów układu DUT wejścia linii zasilania `VDD` i masy `VSS`. Wejścia te powinny być typu `logic` i zaleca się dodanie ich na końcu listy wejść/wyjść modułów. Niezbędne jest również połączenie tych sygnałów na wszystkich poziomach hierarchii. Zmodyfikowane pliki źródłowe, dostarczone w postaci skompresowanego katalogu DUT, należy zapisać w następującej lokalizacji:

```
<lab_root_path>/asic_lab3/src/rtl/sverilog
```

Przed rozpoczęciem implementacji układu DUT trzeba utworzyć bibliotekę projektu. Możemy do tego celu wykorzystać m. in. plik technologiczny lub bibliotekę technologiczną. W tym ćwiczeniu skorzystamy z pierwszego z wymienionych sposobów.

```
create_lib ${ResultsDir}/${DesignLibrary}\
-technology $TechFile -ref_libs ${RefLib}
```

Aby następnie wyświetlić listę bibliotek powiązanych z projektem należy użyć polecenia:

```
report_ref_libs
```

Należy zamieścić wyniki działania tego polecenia w sprawozdaniu.

Program Fusion Compiler akceptuje modele układów opisane na poziomie RTL w jednym z następujących języków opisu sprzętu: Verilog HDL, SystemVerilog HDL lub VHDL. Program najpierw dokonuje analizy modelu, w trakcie której sprawdzana jest jego poprawność syntaktyczna oraz następuje przetworzenie opisu z języka HDL na format pośredni (wewnętrzny):

```
analyze -format sverilog [glob ${SystemVerilogDir}/*.svh]
analyze -format sverilog [glob ${SystemVerilogDir}/*.sv]
```

Proszę zwrócić uwagę, że analiza plików nagłówkowych musi się odbyć przed analizą pozostałych plików.

Kolejnym etapem syntezy jest elaboracja całego projektu, na potrzeby której trzeba wyspecyfikować główny moduł w hierarchii projektu (ang. top-level module).

```
elaborate ${DesignName}
```

Kolejny krok po przeprowadzeniu analizy i elaboracji projektu to utworzenie bloku projektu o nazwie *<nazwa_biblioteki>*:*<nazwa_projektu>*.design i powiązanie go z bibliteką projektu, za co odpowiedzialne jest polecenie:

```
set_top_module ${DesignName}
```

Po jego wykonaniu należy otworzyć interfejs graficzny programu Fusion Compiler.

Proszę zauważyć, że okno layoutu na tym etapie syntezy nie zawiera jeszcze komórek z biblioteki technologicznej. Z kolei schemat układu zbudowany jest z elementów niezależnych o technologii implementacji pochodzących z biblioteki WVGTECH. Warto również zaobserwować powiązania między komponentami projektu widocznymi w oknie hierarchii a odpowiadającymi im blokami na schemacie układu. Zrzuty ekranu ilustrujące opisane powyżej własności interfejsu programu Fusion Compiler należy zamieścić w sprawozdaniu.

Zaleca się aby w tym punkcie zapisać blok projektu:

```
save_block -as ${DesignName}/rtl_read
```

Wstępne odwzorowanie technologiczne układu następuje po wykonaniu instrukcji:

```
compile_fusion -to initial_map
```

Proszę zaobserwować różnice między poprzednim a obecnym schematem układu oraz zamieścić zrzuty ekranu ilustrujące te różnice w sprawozdaniu.

Wydajność, pobór mocy oraz powierzchnia (PPA) to jedne z najistotniejszych parametrów implementowanego układu. W celu ich sprawdzenia trzeba wykonać poniższy zestaw poleceń:

```
report_timing
report_power
report_area
```

Podsumowanie wyników działania tych poleceń proszę uwzględnić w sprawozdaniu.

W celu zapisania netlisty układu w języku Verilog HDL wywołujemy następującą komendę:

```
write_verilog ${ResultsDir}/${DesignName}_initial_synthesis.v
```

Proszę pobieżnie przejrzeć zawartość tego pliku i jego reprezentatywny fragment umieścić w sprawozdaniu.

Przed przejściem do następnego etapu ćwiczenia zapisujemy bieżący blok oraz bibliotekę. W tym celu należy wykonać poniższy zestaw poleceń:

```
current_block
save_block
save_block -as ${DesignName}/inital_synthesis

get_blocks -all
list_blocks

save_lib
```

Następnie kopiujemy blok projektu pod nową nazwą i otwieramy go:

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

Kolejnym zadaniem jest stworzenie skryptów z ograniczeniami projektowymi i scenariuszami optymalizacji Multi Corner Multi Mode. W tym celu w katalogu *<lab_root_path>*/asic_lab3/src/sdc/ proszę stworzyć plik o nazwie dut_toplevel.sdc z następującą zawartością:

```
set PERIOD  3
set CLOCK_NAME clk

create_clock -period $PERIOD \
-name $CLOCK_NAME [get_ports $CLOCK_NAME]
```

Plik ten zawiera jedynie definicję sygnału zegarowego clk o okresie 3 ns.

Kolejny skrypt o nazwie mcmm_setup.tcl należy stworzyć w katalogu *<lab_root_path>*/asic_lab3/setup
Pierwszą operacją, którą wykonujemy w tym skrypcie jest usunięcie wszystkich istniejących definicji tzw. narożników oraz trybów i scenariuszy technologicznych:

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

Następnie definiujemy nowy tzw. narożnik technologiczny określający typowe warunki pracy układu i definiujemy dla niego parametry pasożytnicze.

```
create_corner Typical
set_parasitics_parameters -early_spec nomTLU \
-late_spec  nomTLU -corners {Typical}
```

W kolejnym kroku definiujemy tryb i związany z nim scenariusz pracy układu, dla których następnie określamy warunki pracy układu.

```
create_mode  Normal
current_mode Normal
```

```
create_scenario -mode Normal -corner Typical -name Normal_Typical

current_corner Typical
current_scenario Normal_Typical
set_operating_conditions ${PVT_TT}
```

Przed rozpoczęciem procesu syntezy i implementacji konieczne jest również zdefiniowanie w bibliotece projektowej szeregu parametrów technologiczny, jak m. in. informacji o parametrach pasożytniczych, kierunku prowadzenia ścieżek na poszczególnych warstwach metalizacji, itp. Odpowiada za to skrypt `technology_setup.tcl` dostępny w katalogu *<lab_root_path>*`/asic_lab3/setup .`

Przed przejściem do dalszej realizacji układu wywołujemy trzy wyżej wymienione skrypty:

```
source -echo ${SetupDir}/tech_setup.tcl
read_sdc -echo ${SdcFile}
source -echo ${SetupDir}/mcmm_setup.tcl
```

a ponadto określamy m. in. które z komórek z biblioteki technologicznej zostaną wykorzystane do realizacji układu

```
set_lib_cell_purpose -include none {*/*_AO21* */*V2LP*}
set_app_options -name \
place.coarse.continue_on_missing_scandef -value true
```

Następnie weryfikujemy, czy projekt jest gotowy do dalszej realizacji.

```
compile_fusion -check_only
```

W dalszych krokach realizujemy kolejne etapy syntezy i implementacji .

```
compile_fusion -to initial_map
compile_fusion -from logic_opto -to logic_opto
compile_fusion -from initial_place -to initial_place
compile_fusion -from initial_drc -to initial_drc
compile_fusion -from initial_opto -to initial_opto
compile_fusion -from final_place -to final_place
compile_fusion -from final_opto -to final_opto
```

Po każdy etapie należy wygenerować raporty dotyczące parametrów projektu, co można osiągnąć korzystając np. z przedstawionego poniżej zestawu instrukcji. Przed ich wywołaniem należy zmiennej `StageName` przypisać nazwę aktualnego etapu syntezy projektu, np.: `initial_map`, `logic_opto`, `initial_place`, itd.

```
#echo "Generating reports for the stage: ${StageName} ..."

redirect -file ${ReportsDir}/${StageName}_report_power.rpt\
{report_power -scenarios Normal_Typical}
redirect -file ${ReportsDir}/${StageName}_report_timing_setup.rpt \ {report_timing -
scenarios Normal_Typical -delay_type max}
redirect -file ${ReportsDir}/${StageName}_report_timing_hold.rpt \  {report_timing -
scenarios Normal_Typical -delay_type min}
redirect -file ${ReportsDir}/${StageName}_report_area.rpt {report_area}
redirect -file ${ReportsDir}/${StageName}_report_qor.rpt  {report_qor}
```

```
redirect -file ${ReportsDir}/${StageName}_report_design.rpt {report_design}

#echo "Reports have been generated for the stage: ${StageName}."
```

Powyższy zestaw instrukcji można zawrzeć w procedurze `generateReports` i zapisać ją w pliku *<lab_root_path>*`/asic_lab3/setup/utilities.tcl`
Po każdym etapie syntezy i implementacji należy zaobserwować również zmiany w rozmieszczeniu komórek w layoucie układu. Wyniki tych obserwacji powinny znaleźć odzwierciedlenie w sprawozdaniu w formie zrzutów ekranu opatrzonych krótkim komentarzem.

Po zakończeniu wyżej wymienionych operacji sprawdzamy, czy rozmieszczenie (placement) komórek standardowych jest poprawne a następnie zapisujemy i zamykamy blok i bibliotekę.

```
check_legality

save_block
save_lib

close_bloks
close_lib
```

IV. Sprawozdanie

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