IC-Project I-Synthesis

Joachim Rodrigues

Masoud Nouripayam

Objective of the Presentation

- Introduce basic synthesis
- Guide that can be used to create a basic synthesis flow
  - Steps
  - Actual commands

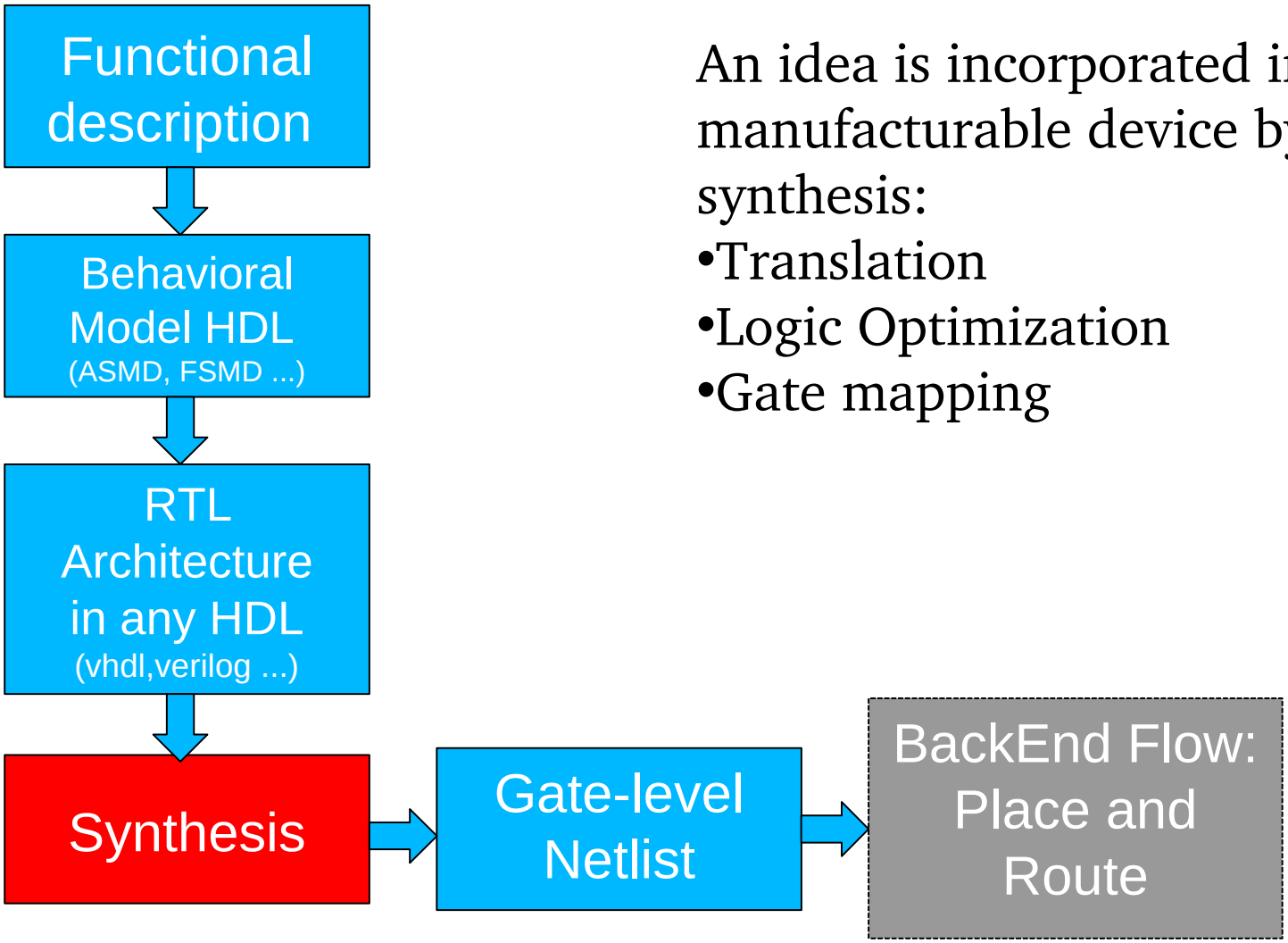
- Getting familiar with the synthesis environment
- Your first ASIC synthesis script

What is Synthesis?

- A process which combines two or more pre-existing elements resulting in the formation of something new.

- Synthesis links the conceptual description of the logic functions needed for the design to their actual physical architecture elements in the underlying device.

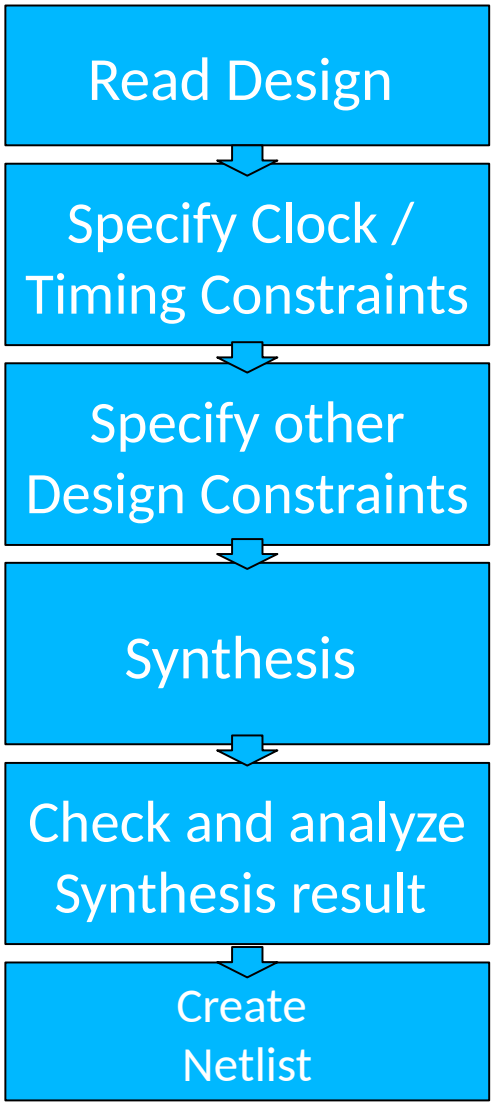
What is Synthesis?



An idea is incorporated into a manufacturable device by doing synthesis:

- Translation
- Logic Optimization
- Gate mapping

Synthesis Flow



Libraries

- Semiconductor manufacturer delivers technology timing libraries as ASCII file (*.lib_)

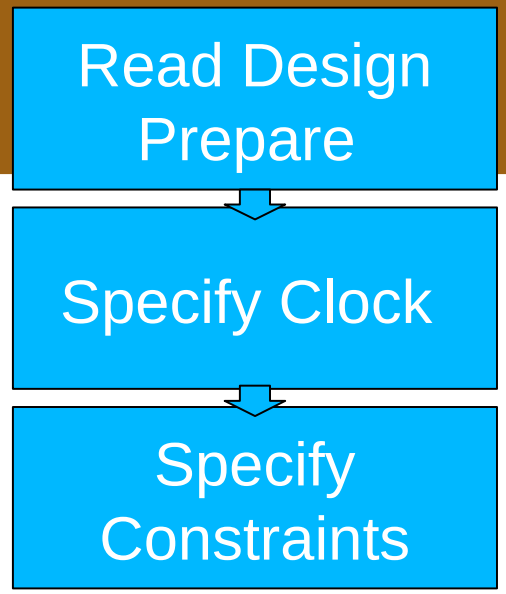
  describes parameters and rules for a particular technology (130nm,90nm, 65nm...).

- Every design kit consists of several logic cells

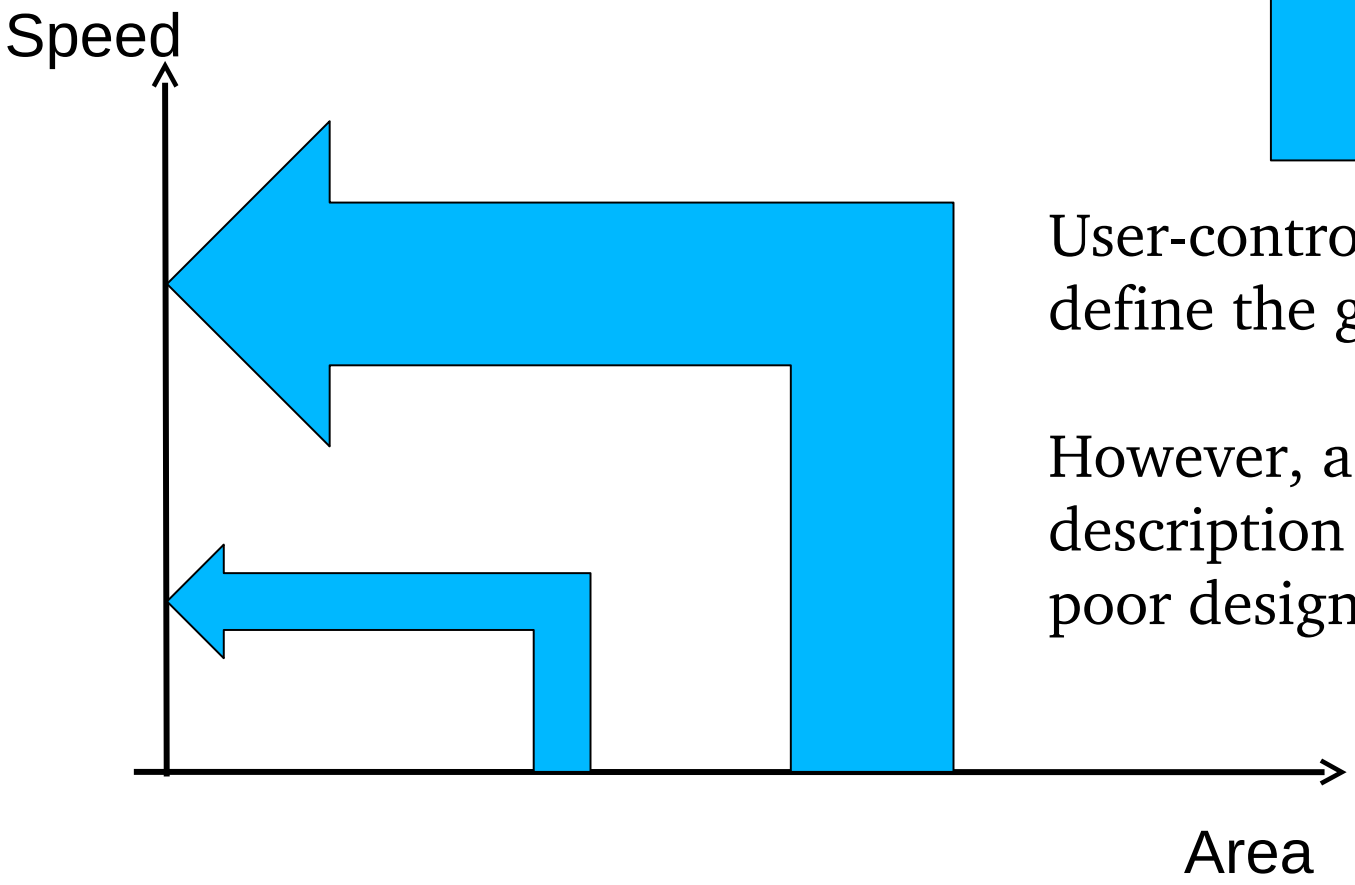
  full adder, multiplexer, flip-flop, XOR, NAND etc

- Various libraries, e.g., low-leakage (LL) or high-speed (HS) are usually available.

Synthesis Constraints



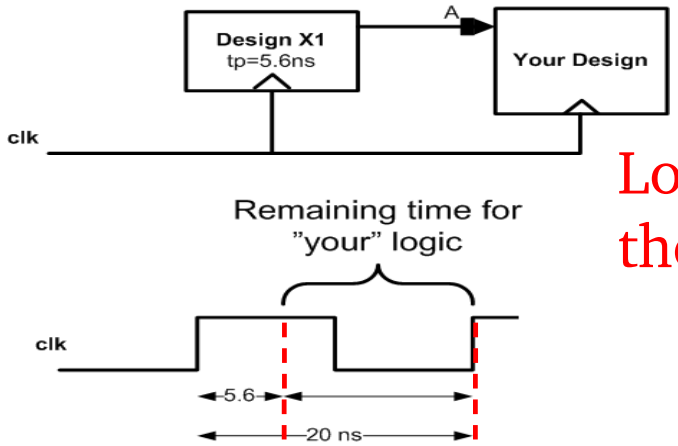
High speed vs low-area



User-controlled constraints define the goal

However, a poor RTL description will result in a poor design

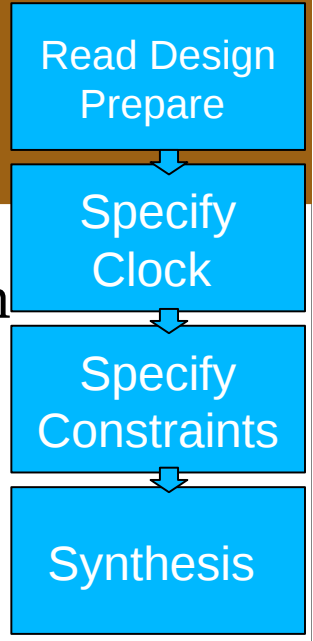
Constraining Input Paths



Logic is triggered by the same clock

Need to Specify propagation delay of external logic that drives your logic

Synthesis and Optimization



performs **logic** and **gate-level** synthesis and optimization on the design

Optimization is controlled by user-specified constraints

- obtain smallest possible circuit
- fastest design
- any other design requirement

The constraints describe:

- goals for the optimization process
- Force specified outputs to meet timing requirements

Values for components' area and speed used during synthesis and optimization are obtained from Manufacturer libraries

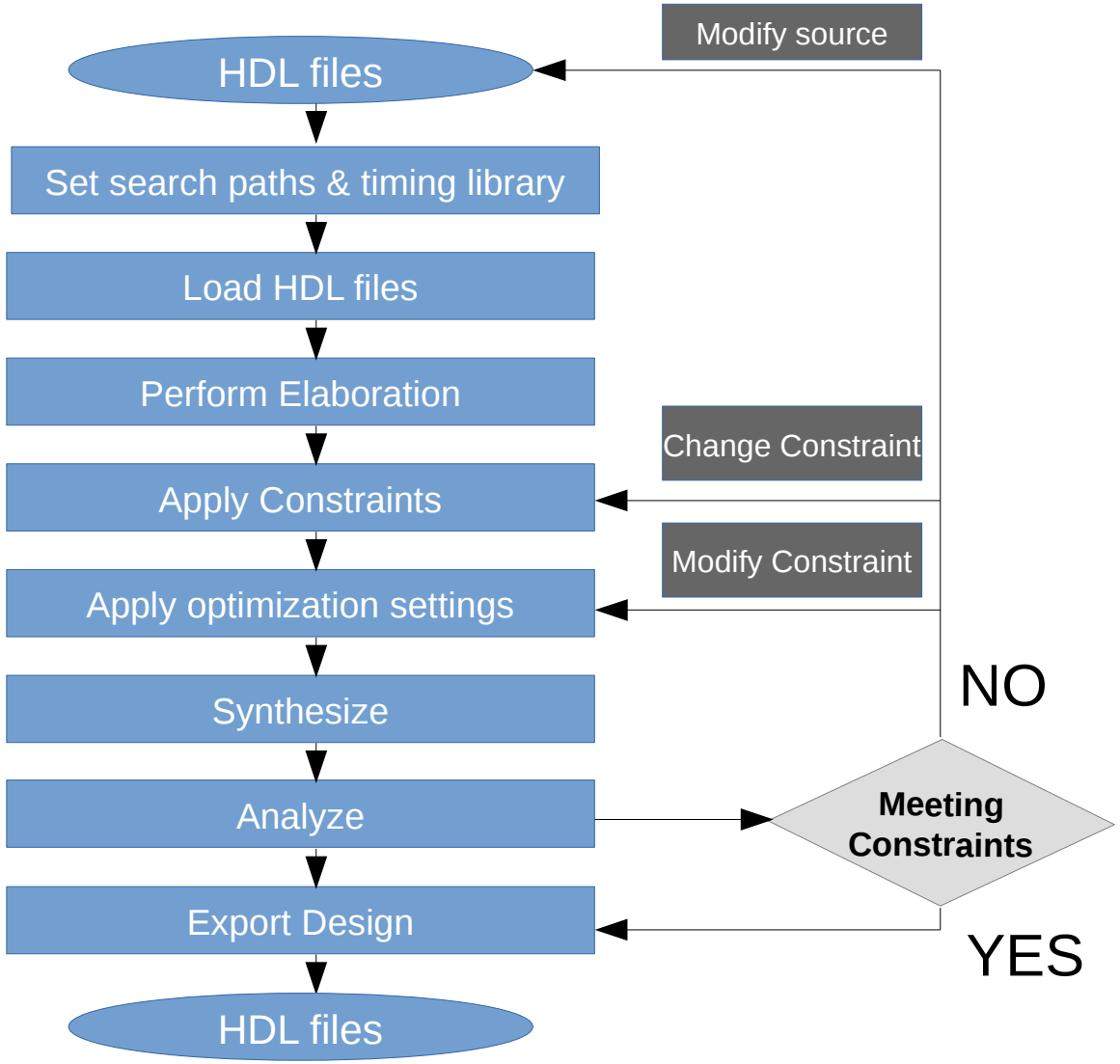
Getting Started with Genus flow

- To run the genus synthesis tool, issue the command 'genus' after you did the initialization which involves loading the licences and envpaths and more necessary files.
- Genus has two user interface:
  - Stylus common UI (you see genus @ root:>) which is a unified interface for genus, innovus and tempus
  - Legacy UI (you see `legacy_genus:/>`) which supports RTL Compiler commands /attributes

- Start Genus directly in legacy UI by

```
genus -legacy_ui
```

Genus flow overview



Getting Started – Extra Help

- To start the gui:

  ```
  gui_show
  ```

- To check what information each message contains, use :

  ```
  vls -a (MESSAGE ID)
  ```

- To check more information about a command or attribute :

  ```
  help (COMMAND)
  ```

- To generate an automatic tcl template file with the very important commands in the flow:

  ```
  legacy_genus:/> write_template outfile TEMPLATE.tcl
  ```

Main steps for performing the sysnthesis

Synthesizing a design generally consists of two main steps:

1- Configure the environment

- Define tool-native variables/attribute and user-global variables
- This step is **design dependent**

  => means for each design with different specifications you may have different attribute and values

2- Carry-out the synthesis flow

- Use some mostly fixed commands
- This step is mostly **design independent**

  => means that for example for any design you synthesize to generic gates using 'syn_generic' disregarding design complexity

Configuring the Environment – Global Parameters

Define the Global variables:

- simpler passing the values to **Genus-native variables** and attributes

- Define the name of the top module of your design to in variable DESIGN.
  - `set DESIGN TOP_MODULE_NAME`

  Access to top_design name everywhere in the whole flow by simply using **$DESIGN**

- Variables passed to Genus-native variable to control the synthesis/Optimization level
  - `set SYN_EFF  {low | medium | high}`
  - `set MAP_EFF  {low | medium | high}`
  - `set OPT_EFF  {low | medium | high}`

- Variables which define Clock constraints
  - `set PERIOD VALUE`
  - `set LATENCY VALUE`

Define all necessary global variables at the very beginning of your script

Configuring the Environment – native parameters

- Using the following attributes you can define the search paths for technology libraries, RTL design files, and flow script paths:

```
set_attribute init_lib_search_path {. ../LIB} /
set_attribute script_search_path {. ../scripts} /
set_attribute init_hdl_search_path {. ../RTL} /
```

- Using the following attributes you can determine the synthesizing level of effort. Notice that you can pass easily the previously define global variables to these attributes:

```
set_attribute syn_generic_effort $SYN_EFF /
set_attribute syn_map_effort $MAP_EFF /
set_attribute syn_opt_effort $OPT_EFF /
```

- Define how detailed information you want to see in the terminal or log files (normally you may have enough information by 5 to 7):

```
set_attribute information_level 9 /
```

Configuring the Environment – native parameters

- Define the attributes which specify the various libraries and lefs for following the flow in the design:

  Notice that these libraries are to be used as the specific target library from the technology to which you decide to map

```
set_attribute library { ../LIB/lib1.lib \
../LIB/lib2.lib} /
set_attribute lef_library { ../LEF/lef1.lef \
 ../LEF/lef2.lef} /
```

- Specify the cap tables:

```
set_attribute cap_table_file capfile /
```

Starting the synthesis flow

- Now it's time to start the first step of the synthesis flow. It loads the hdl files into the genus memory in the defined order:

```
read_hdl -format file_list
```

- If in your design you have multiple hdl formats you can read them into the memory based on their formats:

```
read_hdl -vhdl  "file1.vhd file2.vhd"
read_hdl -v1995 "file3.v file4.v"
read_hdl -sv    "file5.sv file6.sv"
```

- If your design consists of just of one type of HDL files in "Configuring the Environment – native parameters", set your flow default hdl language: (hdl_language {v2001 | v1995 | vhdl | sv} Default: v2001)

```
set_attribute  hdl_language vhdl
```

Starting the synthesis flow

- Notice that you can preform reading the design files two different ways:
  - Define a global variable and put all the files into that as its argument, and then, before issuing read_hdl command source it into the flow. It is absolutely a common practice in a large design when you have so mant design files.

```
set DESIGN_FILES “file1.vhd file2.vhd file3.vhd”

source design_files.tcl

read_hdl $DESIGN_FILES
```

  - If the design is fairly small with a few design files you can pass them directly into the read_hdl argument

```
read_hdl “file1.vhd file2.vhd file3.vhd”
```

Starting the synthesis flow

Create the design from your HDL model =>

```
elaborate $DESIGN
```

Elaboration step does the following:

- Checks the semantics
- Builds data structures
- Infers registers in the design
- Perform higher level HDL optimization (dead-code removal and …)

Notice that you are just passing the top module name from the previously defined variable $DESIGN

Starting the synthesis flow

- With the following command you can check loading the design files and creating the HDL model went.
  - ```
    check_design [-lib_lef_consistency]
    [-undriven [-threshold_fanout integer]]
    [-multidriven] [-unloaded] [-unloaded_comb]
    [-unresolved] [-assigns]
    [-constant [-threshold_fanout] [-through_tie_cell]]
    [-preserved] [-logical_only] [-physical_only]
    [-only_user_hierarchy] [-vname]
    [-report_scan_pins | -skip_scan_pins]
    [-long_module_name] [-status] [-all] [design] [> file]
    ```
- A common practice is check the following commands:

```
check_design -unloaded
check_design -unresolved
```


```

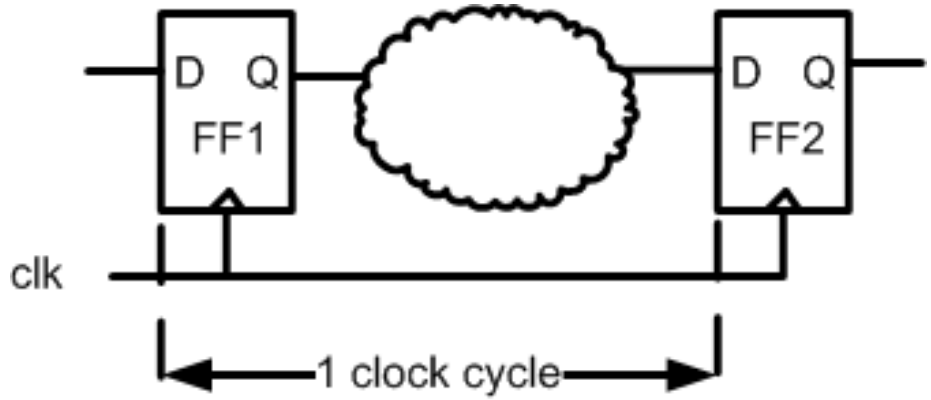
Clock Definition

Read Design
Prepare

Specify Clock

Required Definitions

- clock period
- clock name
- clock source
- Duty cycle (50% default)
- (Offset/skew)



A clock constrains timing paths between registers.
A design may have several clocks.

More detailed clock synthesis happens in *place and route*

Constrain the design – Timing

DEFINE a clock waveform for clocking the design

```
define_clock -name ${ClkName} -period ${PERIOD} -design $
{ClkTop} -domain ${ClkDomain} [find / -pin CLK]
```

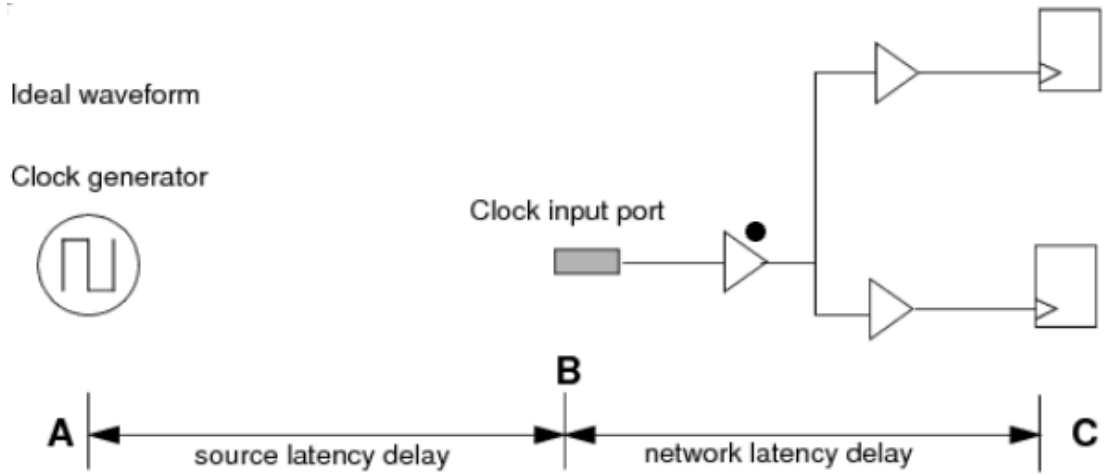
- The command with all necessary arguments is:
  - `define_clock -name string`
    `-period integer [-divide_period integer]`
    `[-rise integer] [-divide_rise integer]`
    `[-fall integer] [-divide_fall integer]`
    `[-domain string] [-mode mode_name]`
    `[-design design] [pin|port]...`

Constrain the design – Timing

- Clock Latency or insertion delay (the sum of clock source and network latency) can be defined using the following commands which basically sets the max latency.

```
set_attribute clock_network_late_latency ${Clk_Latency} ${ClkName}
set_attribute clock_source_late_latency  ${Clk_Latency} ${ClkName}
```

(You can replace 'late' in the command with 'early' and set the min latency)



Ref: Cadence - Genus Timing Analysis Guide for Legacy UI 17.2 (figure1-5)

Constrain the design – Timing

- Clock skew or clock uncertainty is defined as the max difference between the arrival clock at registers clock pins in one clock domain or between different domains. Specifying positive value of uncertainty contributes to a pessimistic timing constraint which is sort of needed in many designs.
  - The following command defines the uncertainty in late mode (setup timing analysis)

```
set_attribute clock_setup_uncertainty ${ClkUncertainty} ${ClkName}
```

  - The following command defines the uncertainty in arrival time of the capturing edge for the clock (hold timing analysis)

```
set_attribute clock_hold_uncertainty ${ClkUncertainty} ${ClkName}
```

Constrain the design – Timing

- Clock Transition defines the amount of time it takes for a signal to change the logic (high-low or low-high)

  the following command sets a 100 minimum rise value, a 110 minimum fall value, a 110 maximum rise value, and a 120 maximum fall value on clock1:

```
set_attribute slew {100 110 110 120} \xs
/designs/example_design/timing/clock_domains/domain_1/clock1
```

- Input delays (delay between a launching clock edge and the time when an input port becomes stable) and output delays (the delay between an output becoming stable and a capturing edge of a clock) can be specified using:

```
external_delay -clock [find / -clock clock1] -input 200 \
    -name in_con [find /des* -port ports_in/*]
```


```
external_delay -clock [find / -clock clock1] -input 200 \
    -name in_con [find /des* -port ports_in/*]
```


Constrain the design – Design Rule Constraints

- When optimizing a design, Genus tries to satisfy all design rule constraints (DRCs) such as:
  - maximum transition
  - Fanout
  - capacitance limits
  - Operating conditions
  - wire-load models
- These constraints are specified using attributes on a module or port, or from the technology library

- To specify a maximum transition limit for all nets in a design or on a port

```
set_attribute max_transition value [design|port]
```

- To specify a maximum fanout limit for all nets in a design or on a port

```
set_attribute max_fanout value [design|port]
```

- To specify a maximum capacitance limit for all nets in a design or on a port

```
set_attribute max_capacitance value [design|port]
```

Constrain the design – Power Optimization

- Leakage power is the power dissipated by current leaks in transistors. The Following commands helps constraining the design to optimize leakage power:

```
set_attribute max_leakage_power VALUE /designs/$DESIGN
set_attribute leakage_power_effort {low | medium | high} /
```

- Design Dynamic power can be constrained by the following commands

```
set_attribute max_dynamic_power constraint /designs/design
set_attribute lp_power_analysis_effort medium /
```

Optimization constraints

- There are several optimization constraints one can apply to the design. However, two of them is worth mentioning here:

- To make Genus work on all the paths to reduce the total negative slack (TNS), instead of just WNS:

```
set_attribute tns_opto true /
```

- By default, Genus tries to fix all DRC errors, but not at the expense of timing. If DRCs are not being fixed, it could be because of infeasible slew issues on input ports or infeasible loads on output ports. You can force Genus to fix DRCs even at the expense of timing (By default, this attribute is false)

```
set_attribute drc_first true
```

Synthesize the design

- Synthesizing the design consists of three steps:
  - Generic synthesis => a technology independent optimization

    `syn_generic` [level of effort is controlled by `syn_generic_effort`]
  - Global Mapping => Genus maps the generic gate-level netlist to the technology library cells

    `syn_map` [level of effort is controlled by `syn_map_effort`]
  - Incremental synthesis (IOPT): This step primarily performs more advanced timing optimization, area optimization and fix DRC violations
    - `syn_opt` [level of effort is controlled by `syn_opt_effort`]

Synthesize the design

- After performing each synthesis stage it is a common practice to use

```
report timing -lint
```

- This command reports possible timing problems in the design such as:
  - clock pins that have no clock waveform driving them
  - Ports that have no external delays
  - Timing exceptions that cannot be satisfied ...

Area Optimization

- The main focus of the tool is to optimize on timing.
- There is no concrete flow or command to tell the tool to only optimize based on area
- However, Genus does offer commands and attributes that let you achieve better area results:

```
set_attr syn_generic_effort high /
set_attr syn_map_effort high /
set_attr syn_opt_effort high /
```

  - Force the tool to remove higher priority of design rule constraints which can burst the area

```
set_attribute drc_first false
```

  - Relax the design constraints like max_transition or max_capacitance
  - Don't use pessimistic segmented wireload models

Exporting the design files

- After Synthesizing the design, write out the design netlist for later stages of ASIC flow:

```
write_hdl     > ${_OUTPUTS_PATH}/${DESIGN}.v
```

- write out the design constraints in SDC format

```
write_sdc     > ${_OUTPUTS_PATH}/${DESIGN}.sdc
```

- write out a Standard Delay Format (SDF) file

```
write_sdf > ${_OUTPUTS_PATH}/${DESIGN}.sdf
```

- You can also generates all the files needed to reload the session in Genus (for example, .g, .v. and .tcl files).

Reporting the design information

- Use the following commands and many other more to check different sorts of information about your design:

```
report qor      > $_REPORTS_PATH/${DESIGN}_qor.rpt
report area     > $_REPORTS_PATH/${DESIGN}_area.rpt
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt
report messages > $_REPORTS_PATH/${DESIGN}_messages.rpt
report gates    > $_REPORTS_PATH/${DESIGN}_gates.rpt
report timing   > $_REPORTS_PATH/${DESIGN}_timing.rpt
```