cadence®

Genus Synthesis Solution: Genus RAK for Beginners with Common UI (CUI)

Rapid Adoption Kit (RAK)

Product Version Genus 19.1
June, 2020

Note: The RAK testcase database, scripts, and references can be found in the ‘Attachments’ and ‘Related Solutions’ sections below the PDF. This RAK can also be searched on the support portal (https://support.cadence.com) using the ‘title’ of the RAK.

Contents

Purpose 4
Introduction 4
Lab Directory Structure 4
Design 5
Lab Sequence 5
Basic Genus Commands 6
Lab 1: Base Lab for Genus 10
Starting Genus 10
Loading Libraries and Design 10
Reading SDC Constraints 11
Synthesizing the Design 11
Lab 2: DFT Flow 13
Setting Up the Environment 13
Loading Libraries 13
Loading Design 13
Reading SDC Constraints 14
DFT Setup 14
Synthesizing the Design 17
DFT Reports 18
Lab 3: Running Low-Power Synthesis 19
Loading Libraries 19
Enabling Clock Gating 19
Loading Design 20
Reading SDC Constraints 20
Setting Power Optimization Attributes 20
Running Generic Synthesis 21
Running and Reporting Low-Power Synthesis 21
Support 22
Feedback 22

Purpose

This RAK includes Genus Common User Interface (CUI) adoption kit with a demo design. It provides a detailed overview of Genus features, how to get started, and how to create a simple script quickly.

Introduction

The RAK testcase database, scripts, and references can be found in the 'Attachments' and 'Related Solutions' sections below the PDF. This RAK can also be searched on the support portal (https://support.cadence.com) using the 'title' of the RAK.

Lab Directory Structure

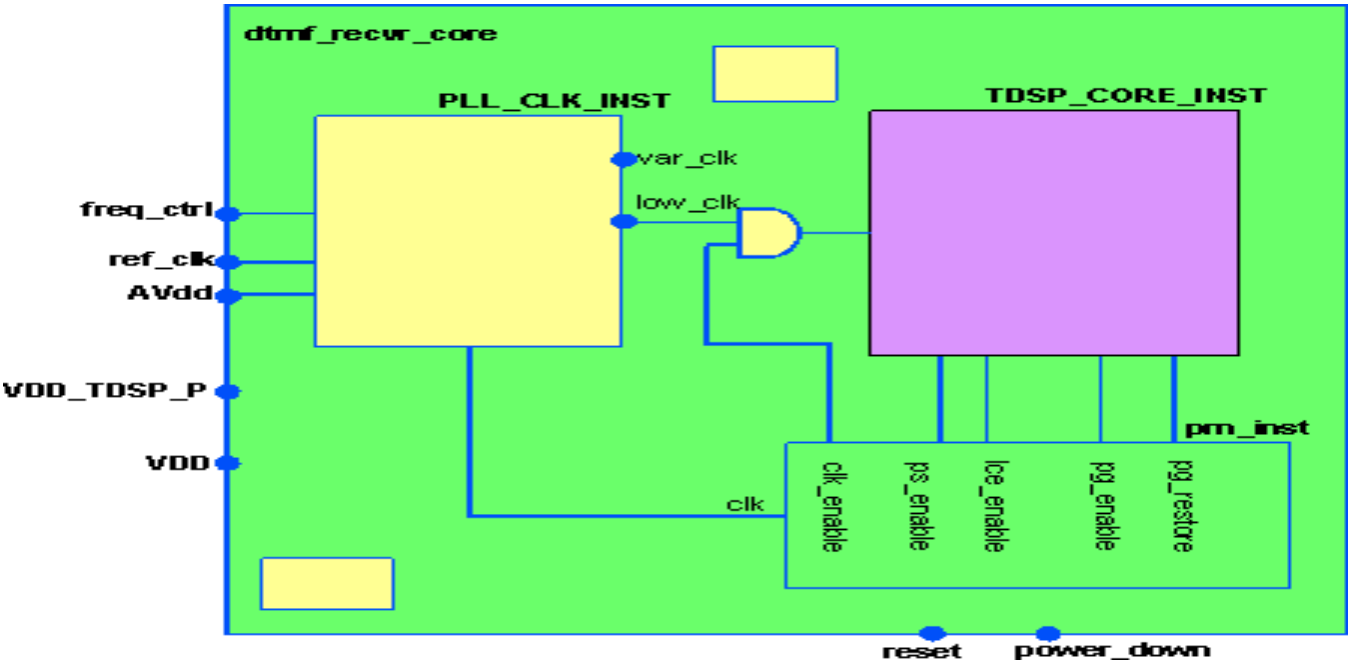
The lab directory structure is as shown below:

```
ls Genus_CUI_RAK/

-constraints          // Contains SDC (constraint) files

-LIBS                 // Contains technology libraries and macro libraries

-LEFS                 // Contains technology lef and macro lefs

-RTL                  // Contains HDL files

-LAB1,LAB2,LAB3       // Lab directories

-README               // Contains info regarding RAK
```

Each lab directory contains a lab-specific script file.

Design

The sample design provided with this RAK is a Verilog description of a dual-tone multi-frequency (DTMF) receiver. In a telephone network, DTMF is a common in-band signaling technique used for transmitting information between network entities. DTMF signals are commonly generated by touch-tone telephones.



Lab Sequence

List of labs:

1. Lab 1 - Basic Genus flow
2. Lab 2 - DFT flow
3. Lab 3 – Low-power flow using clock gating and leakage power opt

For running the labs, "`cd`" into the respective lab directory and run the lab-specific script file. Following are the lab-specific script files:

- Lab 1 - run.tcl
- Lab 2 - run.tcl
- Lab 3 - run.tcl

Basic Genus Commands

Start Genus

```
unix:> genus   [-options]
```

Useful options:

`[-files <string>]`: execute command file

`[-help]`: print this message

`[-log <string>]`: specify log files

`[ lic_startup <string>]`: specify one of the following licenses:

- Genus_Synthesis
- Virtuoso_Digital_Implem
  Virtuoso_Digital_Implem_XL

`[-lic_startup_options <string>]+`: check out an option license at startup:

- Genus_Low_Power_Opt
- Genus_Physical_Opt
- Vdixl_Capacity_Opt

`[-no_gui]`: start with GUI disabled

`[-version]`: return program version information

`[-legacy_ui]`: start Genus with Legacy UI

Genus will run in Common UI through this RAK.

Typical: `unix:> genus –f run.tcl –log base.log`

Run Genus: `unix:> genus`

A Genus prompt, `genus@root:>`, will appear, which indicates that you are running Genus in Common UI.

Create a run script: Use write_template

- Creates a baseline synthesis template for running Genus
- Should be used for each new design and/or tool release
    - Ensures that all attribute/variable settings are for the latest release
    - Ensures use of the latest recommended synthesis flow
- Supports the following:
    - DFT
    - Low Power
    - Multiple Supply Voltage (MSV)
    - Retiming
    - Multimode
    - Area Optimization
    - Netlist-to-Netlist Optimization
    - Genus Physical Flow
    - CPF-Based Low-Power Synthesis

The 'write_template' usage model

```
genus@root:> write_template –outfile<string> [options]
```

Useful options:

`[-split]`: writes out template script with separate file for setup, DFT, and power

`[-no_sdc]`: writes out constraints in Genus format <optional>

`[-dft]`: writes out DFT attributes and commands in the template script

`[-power]`: writes out clock gating, and dynamic and leakage power attributes in the template script [`-cpf`]; writes out a template script for CPF-based flow and a template CPF file (`template.cpf`)

`[-full]`: writes out a template script with all basic commands, DFT, power, and retiming attributes

`[-retime]`: writes out retiming attributes and commands in the template script

`[-n2n]`: writes out a template script for netlist-to-netlist optimization

`[-multimode]`: writes out a template script for multimode analysis

`[-simple]`: writes out a simple template script

`[-area]`: writes out a template script for area-critical designs

`[yield]`: writes out a template script for yield

`[-physical]`: writes out a template script for Genus Physical Flow

To start, use the following command:

```
genus@root:> write_template –outfile run.tcl
```

This will create a script file called `run.tcl`, which you will use to operate Genus.

To open your Genus documents

- Know where Genus's supporting documents are kept.
- They will be useful to explain new and unfamiliar commands, objects, attributes, and variables used in the template.

You can find these documents under the directory where the tool is installed on your system.

```
unix:> which genus
```

The above command shows
"`$path_to_tool_directory/tools.xxx/bin/genus`".

Useful Genus documents

You can find useful documents under the following directory:

```
$path_to_tool_directory/doc/
```

Quick word about Genus run scripts

- Genus is a true Tcl-based tool; its run scripts are all Tcl scripts.
- They utilize variables, lists, objects, attributes, directories, and commands.
- Look for the angle brackets <name> to find what you need to provide.
- Again, use the documentation if something is not clear.

Invoke Genus with run.tcl script:

```
unix:> genus –f run.tcl –log base.log
```

You can use the '`-h`' option to get help for Genus commands.

Examples:

```
genus@root:> write_template –h

genus@root:> write_design -h
```

Completion of running this script will end with the following message:

```
============================

Synthesis Finished .........

============================
```

This will leave you with a Genus prompt as follows:

```
genus@root:>
```

Finally, your REPORT and OUTPUT directories should be populated as well.

To quit the tool, use either of the following commands:

```
genus@root:> quit
```

OR

```
genus@root:> exit
```

Lab 1: Base Lab for Genus

Starting Genus

1. After un-taring the Genus_CUI_RAK, change to the work directory by entering the following commands:

```
cd Genus_CUI_RAK

cd LAB1
```

2. Start the software by entering this command:

```
genus -log synth_flow.log
```

Note: You can type commands interactively at the '`genus@root:>`' shell prompt.

The command shell that starts Genus is dedicated to the Genus Stylus Common UI mode. You must view files in a separate terminal window and not in the Genus shell.

Loading Libraries and Design

Command to read technology libraries:

```
read_libs { ../LIB/slow.lib ../LIB/pll.lib \
../LIB/CDK_S128x16.lib  ../LIB/CDK_S256x16.lib \
../LIB/CDK_R512x16.lib}
```

Command to read the LEF library (which turns PLE on):

```
read_physical -lef { ../LEF/gsclib045_tech.lef
../LEF/gsclib045_macro.lef ../LEF/pll.lef
../LEF/CDK_S128x16.lef  ../LEF/CDK_S256x16.lef
../LEF/CDK_R512x16.lef}
```

Command to read the design:

```
read_hdl " pllclk.v accum_stat.v alu_32.v arb.v
data_bus_mach.v data_sample_mux.v decode_i.v decoder.v \

digit_reg.v conv_subreg.v dma.v dtmf_recvr_core.v
execute_i.v m16x16.v mult_32_dp.v \

port_bus_mach.v prog_bus_mach.v ram_128x16_test.v
ram_256x16_test.v results_conv.v spi.v \
```

```
tdsp_core_glue.v tdsp_core_mach.v tdsp_core.v
tdsp_data_mux.v tdsp_ds_cs.v

test_control.v \

ulaw_lin_conv.v power_manager.v "
```

Command to elaborate the design:

```
elaborate
```

When you elaborate the design, it displays some warning messages. In a real design scenario, you typically attend to the messages in detail.

Also, the elaboration must finish with the "`Done elaborating....`" message.

Command to check for unresolved references:

```
check_design -unresolved
```

Resolving all design references is very important before proceeding to synthesis, so that your synthesis results are accurate. Also, you will not be wasting your time doing hours of synthesis without having all proper design files.

Reading SDC Constraints

```
read_sdc  ../constraints/dtmf_recvr_core_gate.sdc
```

To read the constraints again, reset the timing of the design and read the updated SDC file.

```
reset_design

read_sdc  ../constraints/dtmf_recvr_core_gate.sdc
```

Synthesizing the Design

Set the generic synthesis effort using the `syn_generic_effort` attribute. By default, generic synthesis is run using medium effort. Should you want to change this, use the following attribute:

```
set_db / .syn_generic_effort low|medium|high|express
```

By default, the value is `medium`.

Synthesizes the design to generic gates. It takes a list of top-level designs and synthesizes the RTL blocks to generic gates using the given constraints and performs RTL optimization.

```
syn_generic
```

Set the effort level for mapping to technology library and optimization (defaults are `high`).

```
set_db / .syn_map_effort medium

set_db / .syn_opt_effort medium
```

Synthesize the design to technology gates and optimize it.

```
syn_map

syn_opt
```

The above commands map the design to the cells described in the supplied technology library and perform logic optimization.

When you synthesize the design, the tool shows you informational messages on how your synthesis is being done. You can also view the log file in a separate window to view the same results.

Write out the following:

- The db (database) file

  ```
  write_db dtmf_recvr_core -to_file design.db
  ```

- The netlist

  ```
  write_hdl > design_syn.v
  ```

- The constraints

  ```
  write_sdc > design_syn.sdc
  ```

Lab 2: DFT Flow

In this section, you will set the environment of the synthesis session, load the RTL files, and elaborate the top-level design.

Setting Up the Environment

1. Change to the work directory.

```
cd LAB2
```

2. Start the software by entering the following command:

```
genus
```

Loading Libraries

Command to read technology libraries:

```
read_libs { ../LIB/slow.lib ../LIB/pll.lib \
../LIB/CDK_S128x16.lib  ../LIB/CDK_S256x16.lib \
../LIB/CDK_R512x16.lib}
```

Command to read the LEF library (which turns PLE on):

```
read_physical -lef { ../LEF/gsclib045_tech.lef
../LEF/gsclib045_macro.lef ../LEF/pll.lef
../LEF/CDK_S128x16.lef  ../LEF/CDK_S256x16.lef
../LEF/CDK_R512x16.lef}
```

Loading Design

Read the design and elaborate by entering the following commands:

```
read_hdl " pllclk.v accum_stat.v alu_32.v arb.v
data_bus_mach.v data_sample_mux.v decode_i.v decoder.v \

digit_reg.v conv_subreg.v dma.v dtmf_recvr_core.v
execute_i.v m16x16.v    mult_32_dp.v \

port_bus_mach.v prog_bus_mach.v ram_128x16_test.v
ram_256x16_test.v  results_conv.v spi.v \

tdsp_core_glue.v tdsp_core_mach.v\ tdsp_core.v
tdsp_data_mux.v  tdsp_ds_cs.v test_control.v \
```

```
ulaw_lin_conv.v power_manager.v "


elaborate
```

Reading SDC Constraints

Read the constraint file.

```
read_sdc  ../constraints/dtmf_recvr_core_gate.sdc
```

DFT Setup

- `set_db / .dft_scan_style muxed_scan`

  This attribute specifies the scan style for all designs read in.

- `set_db / .dft_prefix DFT_`

  This attribute specifies the prefix for instances of added control logic, and the base names for any scan-data input, scan-data output, shift-enable, scan_clock_a and scan_clock_b ports, or ports for compression test signals created during test synthesis.

Note: For VDIO customers, it is recommended to set the value of the next two attributes to `false`.

- `set_db / .dft_identify_top_level_test_clocks true`

  This attribute indicates whether the DFT rule checker can automatically assign a "test clock" for each top-level clock pin that is traceable from the clock pin of a flip-flop, and a corresponding "test-clock domain". To prevent auto identification, set this attribute to `false`.

- `set_db / .dft_identify_test_signals true`

  This attribute indicates whether the DFT rule checker can automatically assign a test mode signal for each top-level pin that is traceable from the async set or reset pin of a flip-flop. To prevent auto identification, set this attribute to `false`.

- `set_db / .dft_identify_internal_test_clocks false`

  This attribute indicates whether the DFT rule checker must identify the output pins of multi-input combinational gates and the clock output pins of the clock-gating instances in the clock path as separate test clocks in the same DFT clock domain as its root-level test clock.

- `set_db / .use_scan_seqs_for_non_dft false`

  During synthesis, scan mapping is bypassed. To prevent inferred flip-flops from being mapped to scan flip-flops for functional use, set the `use_scan_seqs_for_non_dft` root attribute to `false` before you run initial synthesis on the design.

- `set_db "design: dtmf_recvr_core" .dft_scan_map_mode tdrc_pass`

  This attribute controls the mapping of the flip-flops to their scan-equivalent flip-flops.

- `set_db "design:dtmf_recvr_core"\ .dft_connect_shift_enable_during_mapping tie_off`

  This attribute controls the connection of the following pins during an initial synthesis run (`syn_map`), when mapping generic flops to their scan-equivalent flops.

- `set_db "design: dtmf_recvr_core"\ .dft_connect_scan_data_pins_during_mapping loopback`

  Starting from RTL, this attribute controls the connections of the scan-data pins during an initial synthesis run (`syn_map`), when mapping generic flops to their scan-equivalent flops.

  Starting from a gate-level netlist, this attribute controls the connections of the scan-data pins during an incremental synthesis run (`syn_opt`), when remapping non-scan flops to their scan-equivalent flops.

- `set_db "design: dtmf_recvr_core" .dft_scan_output_preference auto`

  This attribute controls which scan flip-flop output pin to use for the scan-data path connection.

- `set_db "design: dtmf_recvr_core" .dft_lockup_element_type preferred_level_sensitive`

  This attribute controls the type of component to be used as lockup element when combining scan flops triggered by different clocks or different edges of a test clock in the same chain.

- `set_db "design: dtmf_recvr_core" .dft_mix_clock_edges_in_scan_chains true`

  This attribute controls combining flip-flops from the same DFT domain, which are triggered by either edge of the same test clock, on the same scan chain.

- `define_test_clock -name scanclk -period 18000 scan_clk`

  This command defines a test clock and associates a test-clock waveform with the clock. The test-clock waveform can be different from the system clocks.

- `define_shift_enable -name se -active high scan_en`

  This command specifies the name and active value of the input signal that activates scan shifting. The input signal can be defined on a top-level port or an internal driving pin. This type of input signal is required by the muxed_scan style. The active value of the shift-enable signals is propagated through the design by the `check_dft_rules` command.

- `define_test_mode -name tm  -active high test_mode`

  This command specifies the input signal and constant value that is assigned during a test session. The input signal can be defined on a top-level port or an internal driving pin.

- `define_scan_chain -name top_chain  -sdi scan_in -sdo scan_out -shift_enable se -create_ports`

  This command creates a scan chain or analyzes an existing chain with the specified input and output scan data ports.

Run the DFT rule checks

```
check_dft_rules

report_scan_registers

report_scan_setup
```

Synthesizing the Design

To synthesize your design, perform the following steps:

1. Set the generic synthesis effort using the following attribute. By default, generic synthesis is run using medium effort. Should you want to change this, use the following attribute:

   ```
   set_db / .syn_generic_effort low|medium|high|express
   ```

   By default, the attribute value is set to `medium`.

   ```
   set_db / .syn_generic_effort medium
   ```

2. The following command synthesizes the design to generic gates. It takes a list of top-level designs and synthesizes the RTL blocks to generic gates using the given constraints and performs RTL optimization.

   ```
   syn_generic
   ```

3. Set the effort level for mapping to technology library and optimization (defaults are `high`).

   ```
   set_db / .syn_map_effort medium

   set_db / .syn_opt_effort medium
   ```

4. Synthesize the design to technology gates and optimize it.

   ```
   syn_map

   syn_opt
   ```

   The above commands map the design to the cells described in the supplied technology library and perform logic optimization.

Re-run DFT rule checks

```
check_dft_rules -advanced
```

Build the full scan chains

```
connect_scan_chains  -auto_create_chains

report_scan_chains
```

If your timing is not met, re-run incremental optimization with a different effort level

```
set_db / .syn_opt_effort low

syn_opt -incremental
```

DFT Reports

```
Report_scan_setup

write_scandef -scanDEF

write_dft_abstract_model

write_hdl -abstract

write_script -analyze_all_scan_chains
```

Lab 3: Running Low-Power Synthesis

You run low-power synthesis to insert clock gating for power optimization and to optimize the leakage and dynamic power.

In this section, you will set the environment of the synthesis session, load the RTL files, and elaborate the top-level design.

1. Change to the work directory.

```
cd LAB3
```

2. Start the software by entering the following command:

```
genus
```

Loading Libraries

Command to read technology libraries:

```
read_libs { ../LIB/slow.lib ../LIB/pll.lib \
../LIB/CDK_S128x16.lib  ../LIB/CDK_S256x16.lib \
../LIB/CDK_R512x16.lib}
```

Command to read the LEF library (which turns PLE on):

```
read_physical -lef { ../LEF/gsclib045_tech.lef
../LEF/gsclib045_macro.lef ../LEF/pll.lef
../LEF/CDK_S128x16.lef  ../LEF/CDK_S256x16.lef
../LEF/CDK_R512x16.lef}
```

Enabling Clock Gating

In this section, you will set the variables to enable low-power synthesis and tell Genus the style of clock gating to use before elaboration.

Enable clock-gating features by entering the following command:

```
set_db / .lp_insert_clock_gating true
```

It is important to enable clock gating before elaborating your design.

Loading Design

Read the design and elaborate by entering the following commands:

```
read_hdl " pllclk.v accum_stat.v alu_32.v arb.v
data_bus_mach.v data_sample_mux.v decode_i.v decoder.v \

digit_reg.v conv_subreg.v dma.v dtmf_recvr_core.v
execute_i.v m16x16.v mult_32_dp.v \

port_bus_mach.v prog_bus_mach.v ram_128x16_test.v
ram_256x16_test.v results_conv.v spi.v \

tdsp_core_glue.v tdsp_core_mach.v tdsp_core.v
tdsp_data_mux.v tdsp_ds_cs.v test_control.v \

ulaw_lin_conv.v power_manager.v "



elaborate
```

Reading SDC Constraints

Read the constraint file.

```
read_sdc  ../constraints/dtmf_recvr_core_gate.sdc
```

Setting Power Optimization Attributes

Low-power synthesis is included as part of the synthesis process.

1. Set the leakage power optimization effort.

   ```
   set_db / .leakage_power_effort medium
   ```

2. Specify the optimization weight for dynamic power versus leakage power.

   ```
   set_db design:dtmf_recvr_core
   .lp_power_optimization_weight 0.5
   ```

3. Set the maximum dynamic power attribute.

   ```
   set_db design:dtmf_recvr_core .max_dynamic_power 100
   ```

4. Set the clock-gating style.

```
set_db design:dtmf_recvr_core .lp_clock_gating_style
latch
```

Running Generic Synthesis

Synthesize the design to generic logic.

```
syn_generic
```

Report the power consumption of the design as follows:

```
report_power
```

Running and Reporting Low-Power Synthesis

Synthesize the design as follows:

```
set_db / .syn_map_effort medium

syn_map

set_db / .syn_opt_effort medium

syn_opt
```

Multi-VT optimization and leakage-power optimization are run at this stage.

To get the CPU run time, use the `cpu_runtime` root attribute.

Report the clock gating.

```
report_clock_gating
```

Report the power consumption of the gates.

```
report_gates -power
```

Report the power consumption of the design.

```
report_power
```

You can change the unit of power as follows:

```
set_db / .lp_power_unit mW
```

The following message will show in the Genus session when synthesis is done:

```
============================

Synthesis Finished .........

============================
```

Support

Cadence Support Portal provides access to support resources, including an extensive knowledge base, access to software updates for Cadence products, and the ability to interact with Cadence Customer Support. Visit https://support.cadence.com.

Feedback

Email comments, questions, and suggestions to content_feedback@cadence.com.