High-level Design Beginner Training Course
Part 3 High-level Design Synthesis Exercises

Renesas Electronics Corporation
Design Automation Department

2014/2/3 Rev. 1.0

RENESAS Group CONFIDENTIAL

LLWEB-00018077

Export Control No. LLWEB-00018077

Table of Contents

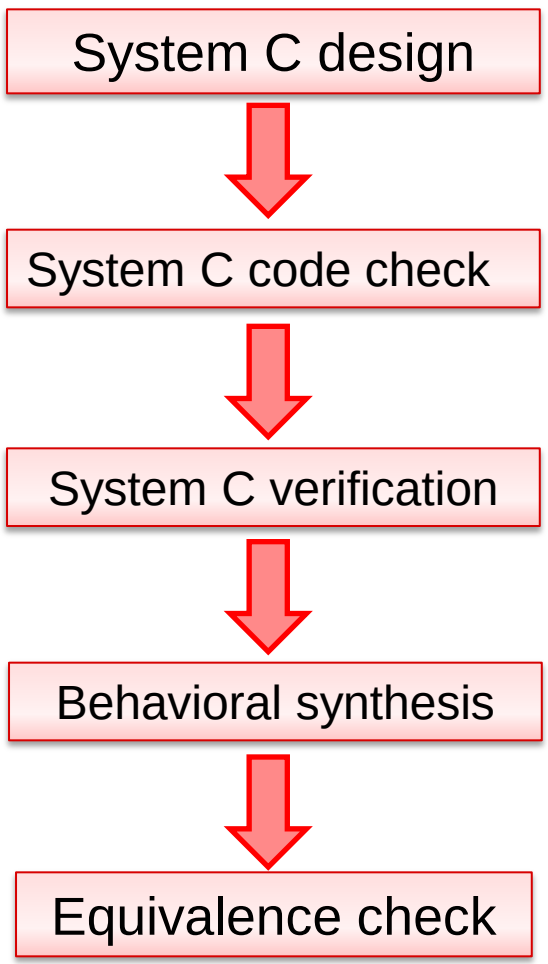
Course Prerequisites

- The high-level design beginner training course assumes basic knowledge of the topics listed below. If you are not knowledgeable about these topics, you may find this course difficult to understand. We recommend that you attend this course after obtaining necessary knowledge through the learning materials shown below.
  - We also recommend that before carrying out the synthesis exercises, you finish the “Part 1 High-level Design Modeling Exercises”.

Basic knowledge required	Learning material
Logic design methodology	•Intranet: **設計の杜** (Japanese web site) http://ppweb01.mu.renesas.com/knowledge/soc/designhome/ •Renesas technical course: RTL logic design exercises
C language programming	•Books (Japanese only) - **明解C言語入門編** (SB Creative Corp.) - C**の絵本** (SHOEISHA.Co.,Ltd.) •Web sites (Japanese web site) 42827198 -C language http://www.c-lang.org/ -Points of Learning C Language for Beginners http://www9.plala.or.jp/sgwr-t/
Overview of High-level Design	•LiveLink: “High-level Design Methodology and Application Examples” http://livelink.renesas.com/Livelink/livelink.exe/open/42827198 •【System-Level & High-Level Design/Verification】Training/Seminar Materials http://eda.develop.renesas.com/lv1ww/REL/training/en/System_High_level_Design_training.html

Purposes

- The High-level Design Beginner Training Course is divided into three parts. The purposes of each part are shown below in relation to the high-level design flow.



Part 1 Modeling Exercises
To help you learn the SystemC syntax and coding styles and create/code basic SystemC models for high-level design.
To help you detect and correct coding problems with SystemC code check tools (such as 1Team).

Part 2 Verification Exercises (Based on This Teaching Document)
To help you create a test bench and verify the SystemC model (i.e., perform simulation-based debugging).
To help you obtain code coverage.

Part 3 Synthesis Exercises
To help you generate various RTLs from SystemC models with a behavioral synthesis tool (CtoS).
To help you check the equivalency between SystemC and RTL with an equivalence checker (SLEC).

Exercise Data

- Obtain exercise data for this training course from the following:

  Training materials for System-Level High-level Design/Verification

  http://eda.develop.renesas.com/lv1ww/REL/training/en/System_High_level_Design_training.html

  High-level Design Beginner Training Course

  3. High-level Design Synthesis Exercises (Exercise Data)

- Reuse SystemC descriptions created in Exercise 2 of Part1 Modeling Exercise.
  - If you have not taken the modeling exercise course, or if you have deleted data after taking the course, use the answer data included in the modeling exercise data.
  - Obtain the modeling exercise data from the following:

  [System-level/high-level design/verification] Training/Seminar Materials

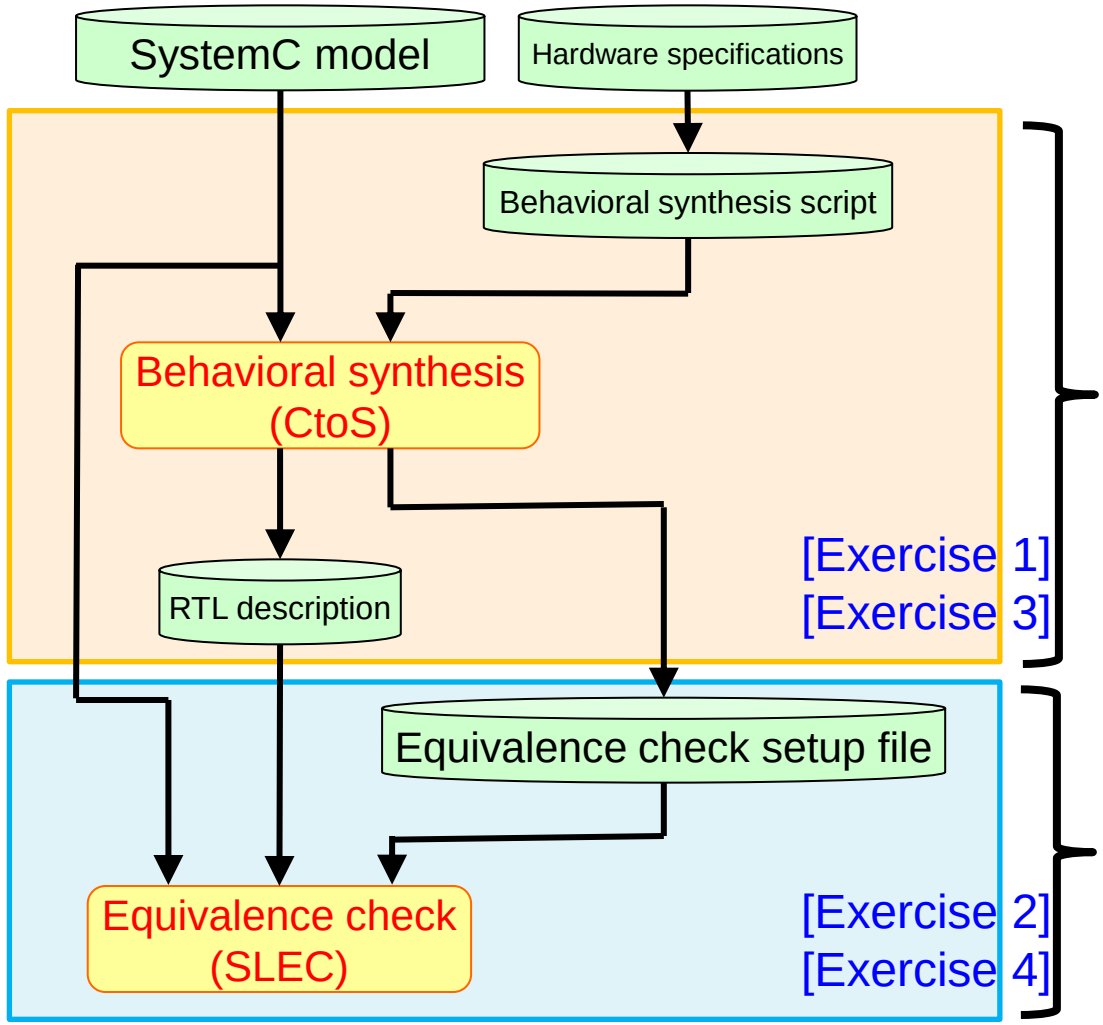
  http://eda.develop.renesas.com/lv1ww/REL/training/en/System_High_level_Design_training.html

  High-level Design Beginner training course

  1. High-level Design Modeling Exercises (Exercise Data)

Behavioral Synthesis/Equivalence Check Flow

- The correspondence between the behavioral synthesis/equivalence check flow and exercises is as follows:

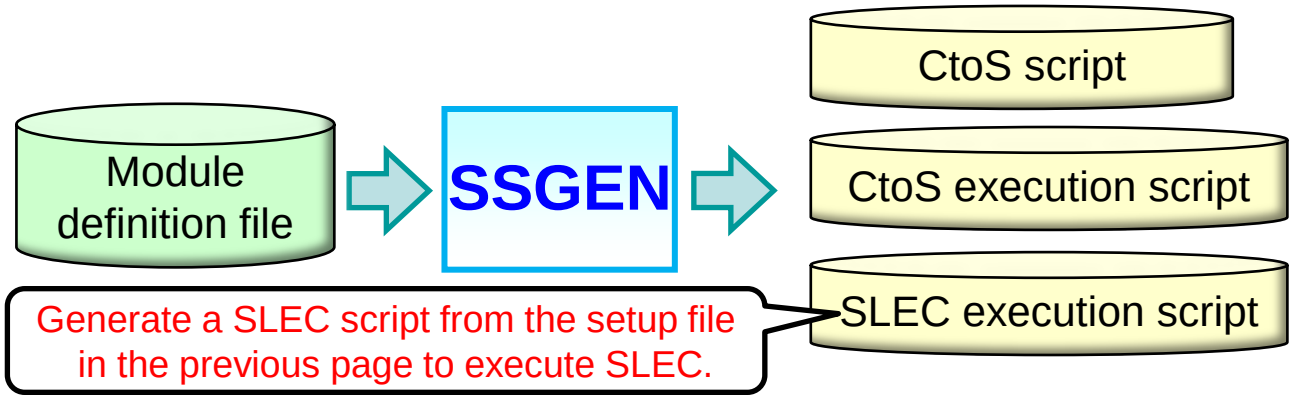


Behavioral synthesis
Automatically generate an RTL description from a SystemC model. Describe behavioral synthesis constraints in a behavioral synthesis script.

Equivalence check
Check the equivalency between SystemC and RTL data.

Generating a Tool Execution Script using SSGEN

- You can generate a behavioral synthesis script and an equivalence check script using SSGEN.
  - ssgen.pl -in *module definition file* -subdir -only_script -ctos -slec

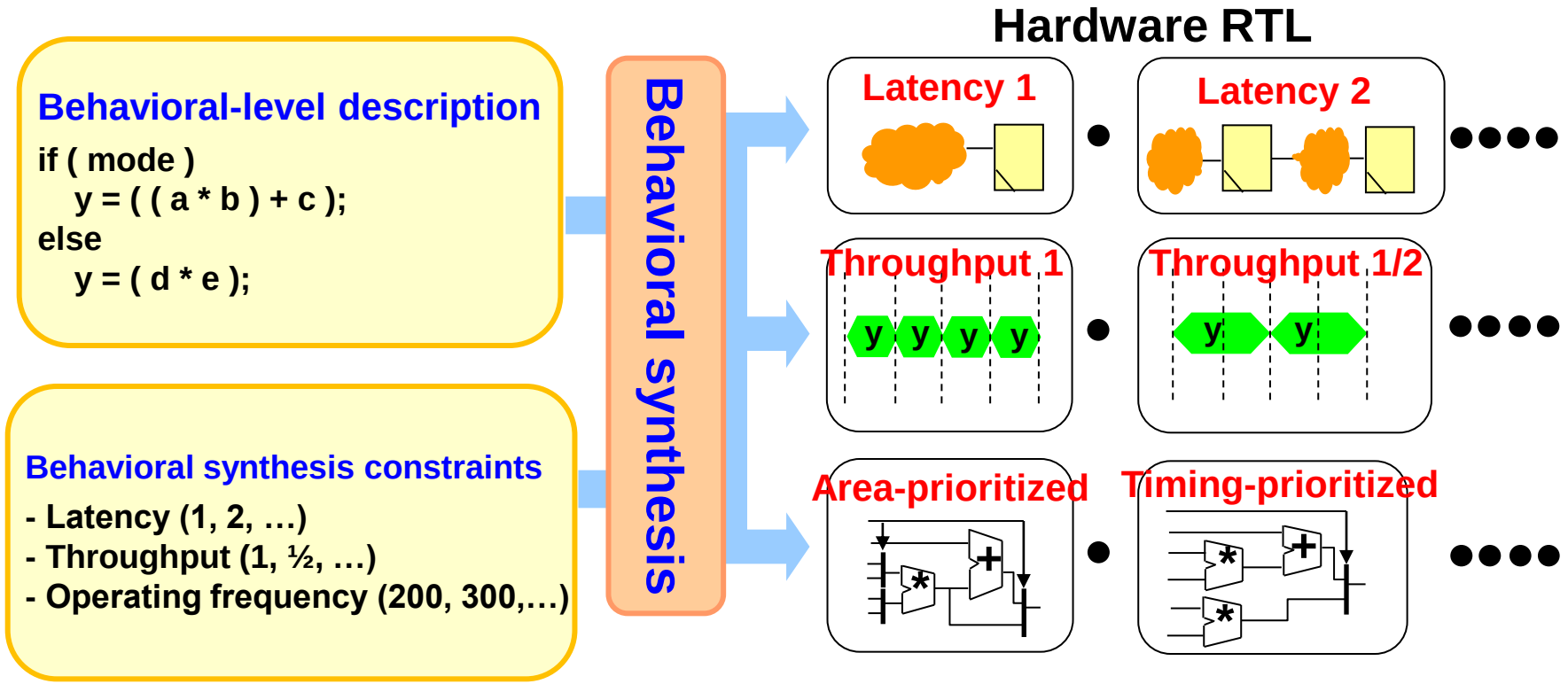


List of command line options (used in this exercise)

Option	**Description**
-in *filename*	Specifies an input file (module definition or hierarchy definition). (Required)
-subdir	Separates generated files in a subdirectory. (If this option is not used, all files are generated on the same path.)
-only_script	Generates a tool execution script only.
-ctos	Generates a behavioral synthesis (CtoS) execution script.
-slec	Generates an equivalence check (SLEC) execution script.

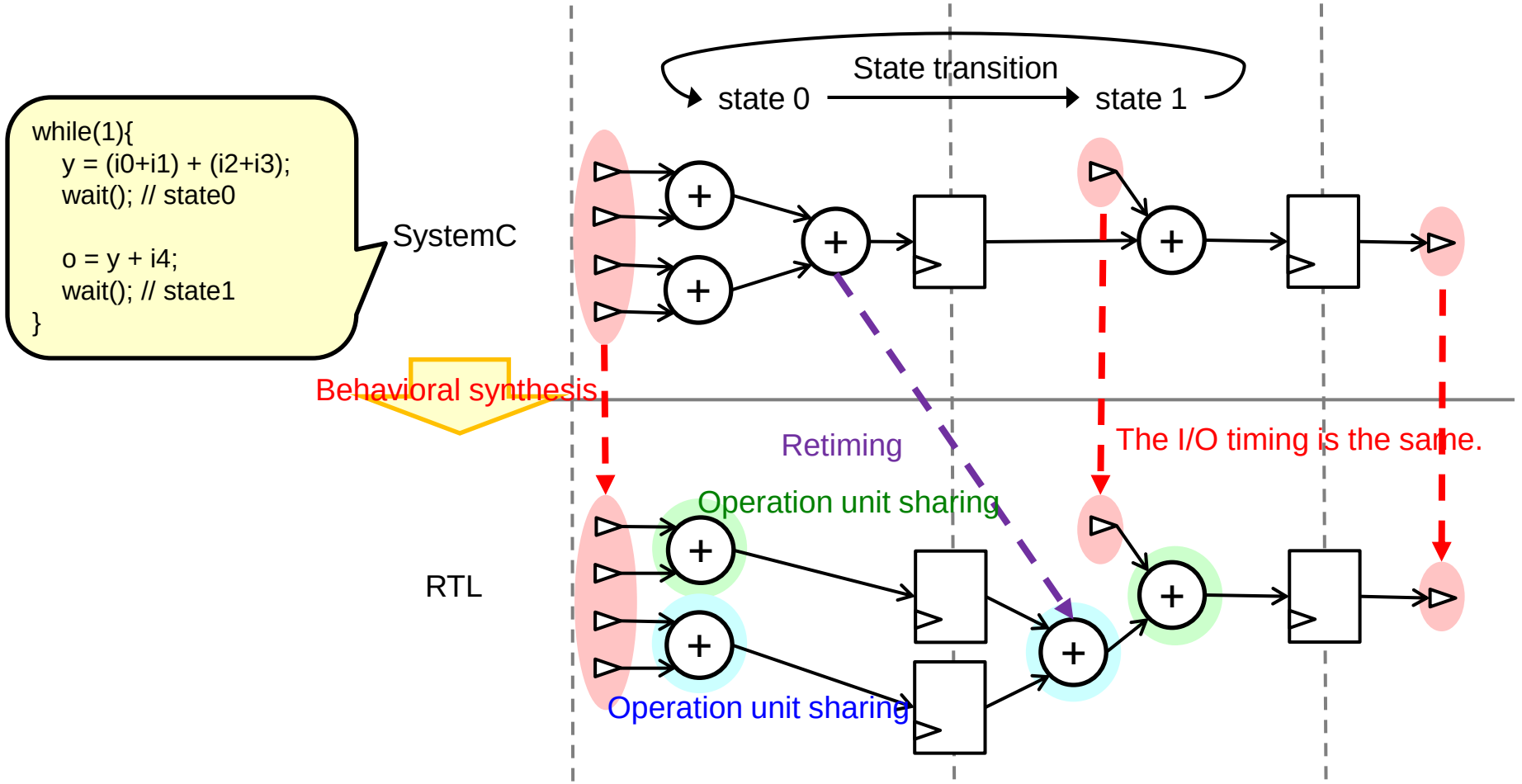
Behavioral Synthesis

- Generate an RTL description from a SystemC model.
  - You can generate various architectures by changing behavioral synthesis constraints.
    - Use CtoS.
    - This exercise shows cycle accurate synthesis and pipeline synthesis.



Cycle Accurate Synthesis

- Generate an RTL description equivalent to the I/O timing of a SystemC model.
  - The simplest way of behavioral synthesis.
  - Optimize operation unit sharing, operation cycle retiming, etc.



Exercise 1: Behavioral Synthesis (Cycle Accurate Synthesis)

- Use CtoS to generate an RTL description from a SystemC model.

Exercise directory: synthesis/ex1

Exercise 1-1. Generating a behavioral synthesis script

- Generate a CtoS script.

Exercise 1-2. Behavioral synthesis

- Execute CtoS.

Exercise 1: Behavioral Synthesis (Cycle Accurate Synthesis)

Exercise 1-1. Generating a behavioral synthesis script

- Use SSGEN to generate a behavioral synthesis script.

1. Move to the exercise directory.

   ```
   %> cd synthesis/ex1
   ```

2. Copy data of exercise 2 of the modeling exercise.
   - Copy the SSGEN input file and the SystemC description that were created in exercise 2 of the modeling exercise.

     ```
     %> cp ../../modeling/ex2/dut.in .
     %> cp -r ../../modeling/ex2/src .
     ```

   - If you have not carry out the modeling exercise, copy answer data. The answer data is stored in modeling/.answer/ex2.

3. Execute SSGEN.
   - When SSGEN is executed as shown below, only scripts for behavioral synthesis can be generated.

     ```
     %> ssgen.pl -in dut.in -subdir -only_script -ctos
     ```

     The following files are generated.

ctos/ctos_dut.tcl	CtoS script (behavioral synthesis constraints)
ctos/run_ctos_dut.sh	CtoS execution script

Exercise 1: Behavioral Synthesis (Cycle Accurate Synthesis)

- Exercise 1-2. Behavioral synthesis
  - Execute CtoS.

1. Execute CtoS.

   %> cd ctos

   %> run_ctos_dut.sh

   - Check that dut.v is generated and that no error is output in ctos_dut.log.
   - If an error is output in ctos_dut.log, an RTL description may not have been generated. In this case, contact hld_support_m@lm.renesas.com.

Checking a Behavioral Synthesis Script Generated by SSGEN

- Check the behavioral synthesis script generated in exercise 1.
  - Here, understand the contents of the behavioral synthesis script generated by SSGEN.
  - Refer to the appendix for information on how to tune RTL data by changing the generated script.

Checking a Behavioral Synthesis Script Generated by SSGE

- ctos/ctos_dut.tcl (1/3)

```
# parameters
set PERIOD 5000
set INPUT_DELAY 0
set TARGET_LIB tutorial.lbr

# set variables
set NAME dut
set MODULE /designs/$NAME/modules/$NAME
set ARRAY $MODULE/arrays
set BEHAVIOR $MODULE/behaviors
set PORT $MODULE/terms

# preparation
new_design $NAME
set_attr source_files "../src/dut.cpp" [get_design]
set_attr compile_flags " -w -D_MEM_MODEL -I/common/appl/Renesas/SystemC/utility/ssgen" [get_design]
set_attr top_module_path $NAME [get_design]
set_attr verilog_rtl_model_suffix "" [get_design]
set_attr auto_write_models false [get_design]
set_attr low_power_clock_gating true [get_design]
set_attr tech_lib_names $TARGET_LIB [get_design]
set_attr reset_all_registers true [get_design]
set_attr verilog_pragma_keyword "synopsys" [get_design]
#set_attr enable_resource_sharing false [get_design]
#set_attr default_speed_grade 90 [get_design]
define_clock -name clk -period $PERIOD
build
```

Operating frequency constraint: 5000 ps

Input delay constraint: 0 ps

Target technology library.
The .lib file of the target library is specified in an actual design.

Variable definitions.

Specifies a SystemC model.

Compilation flags.

Various attribute settings.

Clock definition.

Checking a Behavioral Synthesis Script Generated by SSGEN

- ctos/ctos_dut.tcl (2/3)

```
# flatten_array
set a_list [ls $ARRAY]
set b_list {}
foreach a $a_list {
    set readOnly [get_attr read_only $ARRAY/$a]
    if {$readOnly==0} {
        lappend b_list $ARRAY/$a
    }
}
if {$b_list != ""} {
    flatten_array $b_list
}
```

Flatting arrays. This does not need to be changed. (*)

```
# inline_function
inline_calls -all
```

Inline functions. This should be changed for tuning. (Refer to the appendix.)

```
# loop_unroll
if {[find_combinational_loops]!=""} {
    unroll_loop [find_combinational_loops]
}
```

Unrolling of combinational loops. This does not need to be changed.

```
# input_delay
set port_list [ls $PORT]
foreach i_port $port_list {
    if { [get_attr is_clock $PORT/$i_port]==0
        && [regexp [get_attr direction $PORT/$i_port] "in"]==1 } {
        if {[regexp "rst" $i_port] == 0} {
            external_delay -input $INPUT_DELAY -clock clk -edge rise $PORT/$i_port
        }
    }
}
```

Sets up input delay for all input ports other than those for reset. This should be changed if input delay is to be set up for only specific ports.

(*)It is possible to allocate memory to arrays for synthesis. In this case, this needs to be changed.

Checking a Behavioral Synthesis Script Generated by SSGEN

- ctos/ctos_dut.tcl (3/E)

```
# synthesis
schedule
allocate_registers

# write files
write_rtl -slec slec_${NAME}.tcl -o ${NAME}.v $MODULE

# make reports
file mkdir ./reports_${NAME}
report_resources -detail > reports_${NAME}/report_resources.log
report_schedule > reports_${NAME}/report_schedule.log
report_timing > reports_${NAME}/report_timing.log
report_area -detail > reports_${NAME}/report_area.log
report_registers -detail > reports_${NAME}/report_registers.log
report_summary > reports_${NAME}/report_summary.log

#save_design -dir SAVE_DESIGN
exit
```

Scheduling.
Register allocation.

Generates RTL description.
Generates an equivalence check script.

Generates various report files.

Saves the database.

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - The following report files are generated in reports_dut.
    report_timing.log: Reports on a critical path
    report_resources.log: Reports on resources used
    report_schedule.log: Results of scheduling
    report_registers.log: Results of register binding
    report_area.log: Detailed area report
    report_summary.log: Summary report
    - Later, key points of each report will be described.

  - To evaluate the final QoR (results such as area and delay), examine logic synthesis reports. Timing reports and area reports of CtoS are for reference only.
    - For example, even if an area report of CtoS shows a large decrease in areas, a logic synthesis report may show few changes (the opposite case is also possible).

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - report_timing.log
    - Critical paths are reported.

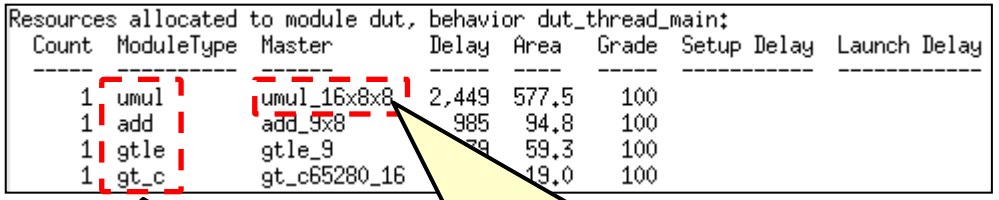
```
Timing report for most critical path in module dut
  Terminal              Type              Fanout  Delay (ps)  Arrival (ps)
  --------              ----              ------  ----------  ------------
  in_d0[0]              in port                7           0             0
  umul_16x8x8/A[0]      umul_16x8x8                       75            75
  umul_16x8x8/Z[0]      umul_16x8x8            1       2,449         2,524
  mux_out_ln17/A_02[0]  mux_7_16                          0         2,524
  mux_out_ln17/Z[0]     mux_7_16               2         504         3,028
  gt_c65280_16/A[0]     gt_c65280_16                      26         3,054
  gt_c65280_16/GT       gt_c65280_16           1         529         3,583
  if_ln49/A             if_then_else                      0         3,583
  if_ln49/Z[0]          if_then_else           1          0         3,583
  mux_out_ln49/C[0]     mux_2_16                          0         3,583
  mux_out_ln49/Z[0]     mux_2_16               1         278         3,861
  out_data_buf/A[0]     buf_16                            0         3,861
  out_data_buf/Z[0]     buf_16                 1          0         3,861
  out_data_reg/D[0]     flipflop_16_r1_0                  0         3,861
  out_data_reg/CLK      flipflop_16_r1_0                320         4,181
-------------------------------------------------------------------------
Timing slack : 819ps
```

A critical path is a path on which multiplication is performed.

An operating frequency constraint of 5000 ps is not violated.

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - report_resources.log
    - Resources used are reported.



Resources allocated to module dut, behavior dut_thread_main:

Count	ModuleType	Master	Delay	Area	Grade	Setup Delay	Launch Delay
1	umul	umul_16x8x8	2,449	577.5	100		
1	add	add_9x8	985	94.8	100		
1	gtle	gtle_9	79	59.3	100		
1	gt_c	gt_c65280_16		19.0	100		

umul_16x8x8: 8 bits * 8 bits = 16 bits

umul: Multiplication
add: Addition
gtle: Comparator (variable vs variable)
gt_c: Comparator (constant vs variable)

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - report_schedule.log
    - Scheduling results

```
Schedule for behavior dut_thread_main:
  Edge            Operation               Resource          Tag
  ----            ---------               --------          ---
  origin          write_dut_in_rdy_ln47   (write in_rdy)    -
  origin          write_dut_out_vld_ln48  (write out_vld)   -
  origin          write_dut_out_data_ln49 (write out_data)  -
  origin          write_dut_pre_d0_ln50   (write pre_d0)    -
  origin          write_dut_pre_d1_ln51   (write pre_d1)    -
  while_ln11      ctrlOr_ln11             (reset only)      -
  xformState_ln11 read_dut_in_d0_ln14     (read in_d0)      -
  xformState_ln11 read_dut_in_d1_ln15     (read in_d1)      -
  xformState_ln11 read_dut_in_op_ln17     (read in_op)      -
  xformState_ln11 read_dut_pre_d0_ln37    (read pre_d0)     -
  xformState_ln11 read_dut_pre_d1_ln41    (read pre_d1)     -
  xformState_ln11 write_dut_pre_d0_ln52   (write pre_d0)    -
  xformState_ln11 write_dut_pre_d1_ln53   (write pre_d1)    -
  xformState_ln11 switch_ln17             switch_ln17       -
  xformState_ln11 gt_ln19                 gtle_9            switch_ln17_0
  xformState_ln11 add_ln25                add_9x8           switch_ln17_1
  xformState_ln11 mul_ln29                umul_16x8x8       switch_ln17_1_0
  xformState_ln11 if_ln19                 if_ln19           switch_ln17_0
  xformState_ln11 mux_out_ln19            mux_out_ln19      switch_
  xformState_ln11 mux_out_ln17            mux_out_ln17      -
  xformState_ln11 gt_ln49                 gt_c65280_16      -
  xformState_ln11 if_ln49                 if_ln49           -
  xformState_ln11 mux_out_ln49            mux_out_ln49      -
  xformState_ln11 write_dut_out_data_ln55 (write out_data)  -
```

Binds umul_16x8x8 to the multiplication in the 29th line.

Checking the Results of Behavioral Synthesis

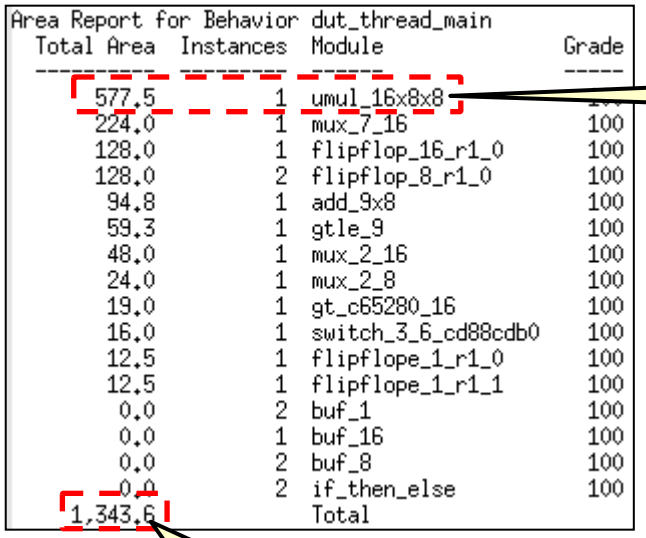
- Check the results of the behavioral synthesis in exercise 1.
  - report_registers.log
    - Results of resister binding.

```
Behavior 'dut_thread_main' does not have any register bindings.
```

Reports on generated registers other than sc_out and sc_signal.
There are no registers related to above descriptions in this exercise.

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - report_area.log
    - Detailed area report.



Area Report for Behavior dut_thread_main

Total Area	Instances	Module	Grade
577.5	1	umul_16x8x8	
224.0	1	mux_7_16	100
128.0	1	flipflop_16_r1_0	100
128.0	2	flipflop_8_r1_0	100
94.8	1	add_9x8	100
59.3	1	gtle_9	100
48.0	1	mux_2_16	100
24.0	1	mux_2_8	100
19.0	1	gt_c65280_16	100
16.0	1	switch_3_6_cd88cdb0	100
12.5	1	flipflope_1_r1_0	100
12.5	1	flipflope_1_r1_1	100
0.0	2	buf_1	100
0.0	1	buf_16	100
0.0	2	buf_8	100
0.0	2	if_then_else	100
1,343.6		Total	

The area of umul_16x8x8 is 577.5CA.

Total 13436CA

Checking the Results of Behavioral Synthesis

- Check the results of the behavioral synthesis in exercise 1.
  - report_summary.log
    - Summary report.

```
Summary report for complete Design /designs/dut
  Name                                  Count
  ----                                  -----
  Overview
    Name                                  dut
    'clk' Clock Period (ps)             5,000
    Minimum Slack after Schedule (ps)     664
    Minimum Slack (ps)                    819
    Area                              1,343.6
    Power                                 N/A
  Behavior
    Modules                                 1
    Processes                               1
    Functions                               0
    Arrays                                  0
    Loops                                   1
    States                                  1
    Edges                                   4
    Total Ops                              24
    Shareable Ops                           4
    Values                                117
    Tags                                    5
  Structure
    Memories (bits)                         0
    Flip Flops (bits)                      34
    Muxes (bits)                           40
    Shareable Resources                     4
    Non Shareable Resources                 8
    Input Terminals (bits)                 23
    Output Terminals (bits)                18
    Nets (bits)                           172
    Instance Terminals (bits)             470
```

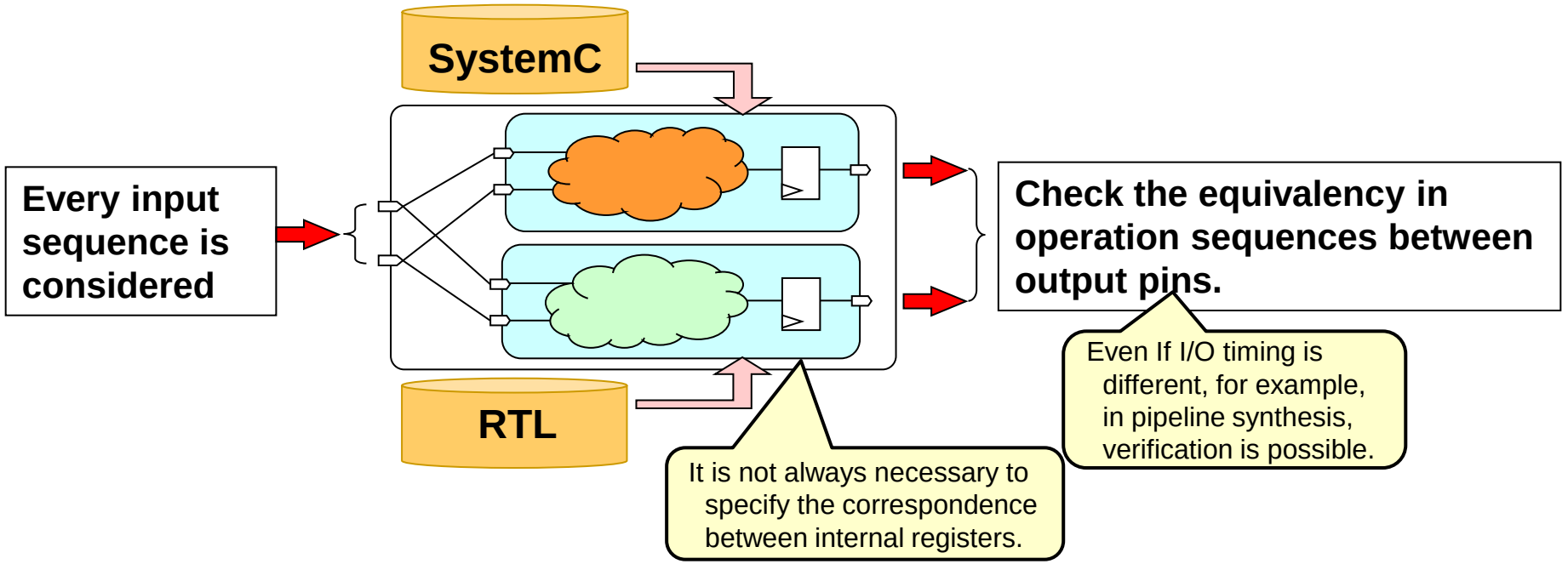
Timing/area report.

The number of states.

The number of bits of F/Fs and multiplexers.

Equivalence Check

- Verify the equivalency between a SystemC model and an RTL description.
  - Verify that the RTL description after behavioral synthesis is equivalent to a SystemC model.
  - Use SLEC.
    - No pattern is required. Static verification in which every input sequence is considered.
    - Faster than RTL simulation.



Exercise 2: Equivalence Check (Cycle Accurate Synthesis)

- Use SLEC to check the equivalency between a SystemC model and an RTL description.

Exercise directory: synthesis/ex2

Exercise 2-1. Generating an equivalence check script.

- Generate a SLEC script.

Exercise 2-2. Equivalence check.

- Execute SLEC.

Exercise 2: Equivalence Check (Cycle Accurate Synthesis)

Exercise 2-1. Generating an equivalence check script

- Use SSGEN to generate an equivalence check script.

1. Move to the exercise directory.

   ```
   %> cd synthesis/ex2
   ```

2. Copy the data of exercise 1.
   - Copy the SSGEN input file, SystemC descriptions, and the CtoS directory for exercise 1.

     ```
     %> cp ../ex1/dut.in .
     %> cp -r ../ex1/src .
     %> cp -r ../ex1/ctos .
     ```

3. Execute SSGEN.
   - After SSGEN is executed as shown below, only scripts for equivalence check are generated.

     ```
     %> ssgen.pl -in dut.in -subdir -only_script -slec
     ```

     The following files are generated.

slec/run_slec_dut_eq.sh	Equivalence check execution script.
slec/slec_dut_sc.tcl slec/run_slec_dut_sc.sh	SystemC verification script (not used in this exercise).

Exercise 2: Equivalence Check (Cycle Accurate Synthesis)

Exercise 2-2. Equivalence Check

- Execute SLEC.

1. Execute SLEC.
   %> cd slec
   %> run_slec_dut_eq.sh
   slec_dut_eq.log is output.

2. Check the results of verification.
   - Check that all the items other than Proven show “0” at the end of slec_dut_eq.log.

```
================================================================================
********** Equivalence Results (calypto_dut_eq/results.log) **********
================================================================================
                          Proven Cond-Proven Bounded-Proven Falsified Unresolved
Output-Maps                    3           0             0         0          0
Active Intermediate-Maps       5           0             0         0          0
Properties                     0           0             0         0          0
```

Check that equivalence is ensured at all the verification points.

   - If there is any difference (if Falsified is not “0”), contact hld_support_m@lm.renesas.com.

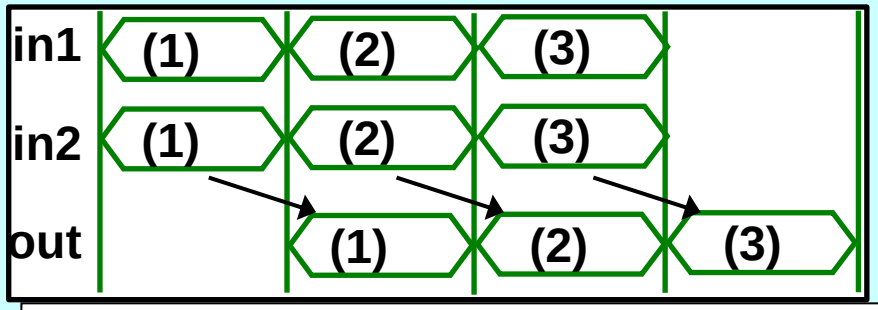
Pipeline Synthesis

- Generate an RTL description with pipeline according to behavioral synthesis constraints.
  - Divide one-cycle operations into multiple stages according to the operating frequency constraint.
  - Perform optimization such as sharing operation units and retiming operation cycles in the same way as for cycle accurate synthesis.

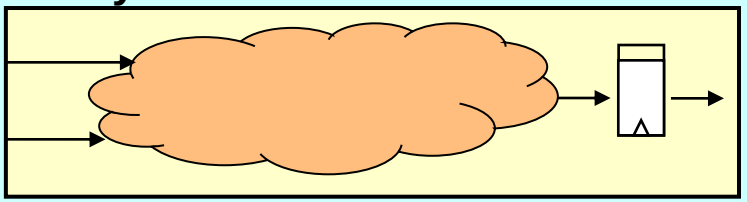
Cycle accurate synthesis

SystemC

```
while(1){
    wait();
}
```

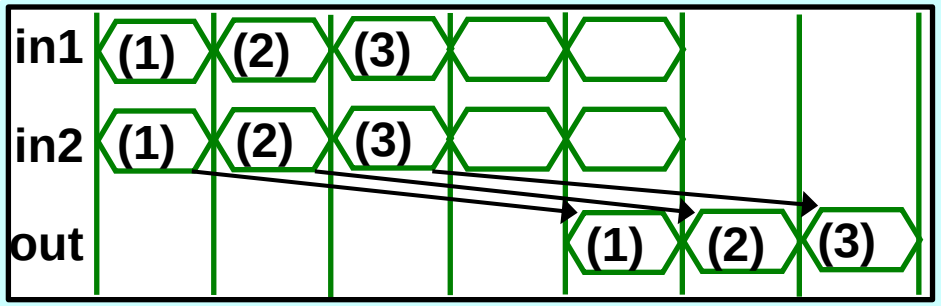


Synthesized RTL schematic

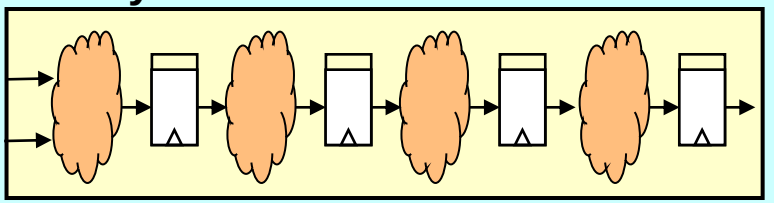


- Data input in each cycle (Throughput = 1 data/1 cycle)
- Results are output after 1 cycle (Latency = 1 cycle)

Pipeline synthesis



Synthesized RTL schematic



You can increase the operating frequency.

- Data input in each cycle (Throughput = 1 data/1 cycle)
- Results are output after 4 cycles (Latency = 4 cycles)

Pipeline Synthesis

- CtoS can pipeline a sequential loop (loop that includes wait()).
  - Steps for pipelining a loop through CtoS. (Details are given in exercise 3.)

    (1) Add a label to the beginning of the target loop in a SystemC model.

    (2) Specify the label in (1) as an argument of the CtoS pipeline command.

    (3) Specify the assignment operation on sc_out and sc_signal to be bound to the last stage of the pipeline.
    - It can be specified by using a CtoS command or adding a label to the SystemC model.
    - In this exercise, specify it using a CtoS command.

Pipeline Synthesis

- SSGEN can generate a sequence (from (1) to (3) above) for pipelining a while(1) loop of SC_CTHREAD by using the -pipe option for the cthread command.

dut.in

```
cthread thread_main -pipe
```

dut.cpp

```
void dut::thread_main() {
  reset_thread_main();
  wait();
 CtoS_MAIN_LOOP(1)
  while (1) {
    // please write here!
    my_wait();
  }
}
```

ctos_dut.tcl

```
# input_delay
...

# pipeline
set LATENCY_THREAD_MAIN 3
set EXPAND_BEFORE_NET # specify net name here
pipeline_loop -init_interval 1 ¥
  -min_lat_interval 2 ¥          (2)
  -max_lat_interval ${LATENCY_THREAD_MAIN} ¥
  -expand_before [find -net $EXPAND_BEFORE_NET] ¥(3)

${BEHAVIOR}/${NAME}_main_th/nodes/CtoS_MAIN_LOOP_while_begin

# synthesis
...
```

-init_interval: Specifies a throughput and does not need to be changed.
-min_lat_interval: The minimum number of stages for pipelining. The default is 2.
-max_lat_interval: The maximum number of stages for pipelining. The default is LATENCY_THREAD_MAIN (=3).
You can change the range of CtoS pipelining by changing the values of -min_lat_interval and -max_lat_interval. (By default, CtoS selects 2 stages or 3 stages.)

For (3), specify the output port name using the variable $EXPAND_BEFORE_NET in -expand_before. (For details, refer to the next page.)

Pipeline Synthesis

- Specify an output port name in -expand_before.
  - Specify the output port name for which assignment operation is to be bound to the last stage of the pipeline.
  - If there are multiple ports for which assignment operation is to be bound to the final stage, only specify the one for which assignment operation is performed first in the pipeline loop of the SystemC model.

```
void dut::thread_main () {
    reset_thread_main () ;
    wait () ;
 CtoS_MAIN_LOOP:
    while (1) {
        ...
        out1.write (x) ;
        out2.write (y) ;
        out3.write (z) ;

        my_wait () ;
    }
}
```

These are all bound to the last stage.

```
# input_delay
...

# pipeline
set LATENCY_THREAD_MAIN 3
set EXPAND_BEFORE_NET out1
pipeline_loop -init_interval 1 ¥
  -min_lat_interval 2 ¥
  -max_lat_interval ${LATENCY_THREAD_MAIN} ¥
  -expand_before [find -net $EXPAND_BEFORE_NET] ¥
  ${BEHAVIOR} /${NAME}_main_th/nodes/CtoS_MAIN_LOOP_while_begin

# synthesis
...
```

Select out1, for which assignment operation is performed in the pipeline loop.
(This binds all the code following out1.write(x) to the last stage.)

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Use CtoS to generate an RTL description from a SystemC model.

Exercise directory: synthesis/ex3

Exercise 3-1. Generating a behavioral synthesis script.

- Generate a CtoS script.

Exercise 3-2. Preparing for pipeline synthesis.

- Add a label to the SystemC model and modify commands for pipeline synthesis.

Exercise 3-3. Behavioral synthesis part 1.

- Execute CtoS.

Exercise 3-4. Modifying behavioral synthesis constraints.

- Modify constraints on the operating frequency.

Exercise 3-5. Behavioral synthesis part 2.

- Execute CtoS.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

Exercise 3-1. Generating a behavioral synthesis script

- Use SSGEN to generate a behavioral synthesis script.

1. Move to the exercise directory.

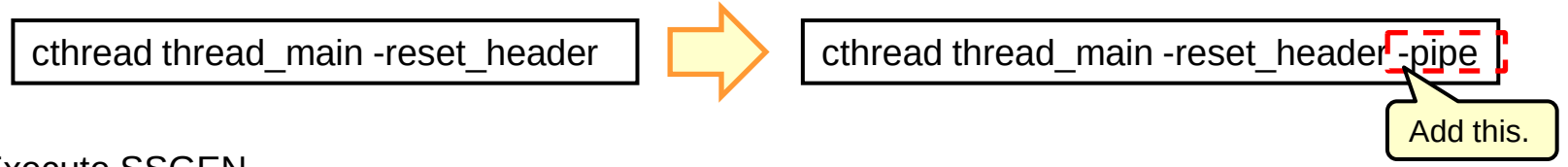
   %> cd synthesis/ex3

2. Copy data of exercise 1.
   - Copy the SSGEN input file and SystemC descriptions of exercise 1.

     %> cp ../ex1/dut.in .

     %> cp -r ../ex1/src .

3. Modify dut.in for use in pipeline synthesis (*).
   - To generate a CtoS pipeline synthesis command, dut.in needs to be modified as shown below:



4. Execute SSGEN.
   - After SSGEN is executed as shown below, only scripts for behavioral synthesis are generated.

     %> ssgen.pl -in dut.in -subdir -only_script -ctos

     The following files are generated.

     ctos/ctos_dut.tcl — CtoS script (behavioral synthesis constraints)

     ctos/run_ctos_dut.sh — CtoS execution script

(*) Use the -pipe option in advance for designs for which pipeline synthesis is to be used.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-2. Preparing for pipeline synthesis (1/2)
  - Add a label for pipeline synthesis to src/dut.cpp (*).

dut.cpp

```
#include "dut.h"

void dut::thread_main() {
    reset_thread_main();
    wait();
    while (1) {

        ...

        out_data.write( out );

        wait();
    }
}
```

```
#include "dut.h"

void dut::thread_main() {
    reset_thread_main();
    wait();
 CtoS_MAIN_LOOP:
    while (1) {

        ...

        out_data.write( out );

        wait();
    }
}
```

Selects the while(1) loop of SC_CTHREAD as the pipeline target loop.

(*) Required in this exercise because data of the modeling exercise is reused. If SSGEN is executed in advance with the intention of pipeline synthesis, SSGEN will automatically generate a label.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

Exercise 3-2. Preparing for pipeline synthesis (2/E)

- Specify the name of the output port for which assignment operation is to be bound to the last stage.

ctos_dut.tcl

```
# pipeline
set LATENCY_THREAD_MAIN 3
set EXPAND_BEFORE_NET # specify net name here
pipeline_loop -init_interval 1 ¥
  -min_lat_interval 2 ¥
  -max_lat_interval ${LATENCY_THREAD_MAIN} ¥
  -expand_before [find -net $EXPAND_BEFORE_NET] ¥

${BEHAVIOR}/${NAME}_main_th/nodes/CtoS_MAIN_LOOP_while_begin
```

```
# pipeline
set LATENCY_THREAD_MAIN 3
set EXPAND_BEFORE_NET out_data
pipeline_loop -init_interval 1 ¥
  -min_lat_interval 2 ¥
  -max_lat_interval ${LATENCY_THREAD_MAIN} ¥
  -expand_before [find -net $EXPAND_BEFORE_NET] ¥

${BEHAVIOR}/${NAME}_main_th/nodes/CtoS_MAIN_LOOP_while_begin
```

Binds the assignment operation on the output port out_data to the last stage of the pipeline.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-3. Behavioral synthesis part 1 (1/2)
  - Execute CtoS.

1. Execute CtoS.

   %> cd ctos

   %> run_ctos_dut.sh

   - Make sure that dut.v is generated and that there are no errors in ctos_dut.log.
   - If there is an error in ctos_dut.log, an RTL description may not have been generated. In this case, contact hld_support_m@lm.renesas.com.

2. Check the results of pipeline synthesis.
   - The results of pipeline synthesis is output to reports_dut/report_schedule.log.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-3. Behavioral synthesis part 1 (2/E)
  - The results of pipeline synthesis (through the use of the SystemC model in modeling/.answer/ex2 in exercise 1.)

report_schedule.log

Stage	Operation
CtoS_MAIN_LOOP_while_begin	1 read_dut_in_d0_ln15
CtoS_MAIN_LOOP_while_begin	1 read_dut_in_d1_ln16
CtoS_MAIN_LOOP_while_begin	1 read_dut_in_op_ln18
CtoS_MAIN_LOOP_while_begin	1 read_dut_pre_d0_ln38
CtoS_MAIN_LOOP_while_begin	1 read_dut_pre_d1_ln42
CtoS_MAIN_LOOP_while_begin	1 switch_ln18
CtoS_MAIN_LOOP_while_begin	1 gt_ln20
CtoS_MAIN_LOOP_while_begin	1 add_ln26
CtoS_MAIN_LOOP_while_begin	1 mul_ln30
CtoS_MAIN_LOOP_while_begin	1 if_ln20
CtoS_MAIN_LOOP_while_begin	1 mux_out_ln20
CtoS_MAIN_LOOP_while_begin	1 mux_out_ln18
CtoS_MAIN_LOOP_while_begin	1 gt_ln50
CtoS_MAIN_LOOP_while_begin	1 if_ln50
CtoS_MAIN_LOOP_while_begin	1 mux_out_ln50
CtoS_MAIN_LOOP_while_begin	2 if_CtoS_MAIN_LOOP_while_begin_stage2
CtoS_MAIN_LOOP_while_begin	2 write_dut_out_data_ln56
CtoS_MAIN_LOOP_while_begin	1 write_dut_pre_d0_ln53
CtoS_MAIN_LOOP_while_begin	1 write_dut_pre_d1_ln54

The assignment operation on out_data is scheduled at stage 2.

```
void dut::thread_main() {
   reset_thread_main();
   wait();
 CtoS_MAIN_LOOP:
   while (1) {
      ...

      out_data.write( out );

      wait();
   }
}
```

Synthesis through 2-stage pipeline

Stage 1

Stage 2

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-4. Modifying behavioral synthesis constraints.
  - Modify the operating frequency constraint to generate a differently structured pipeline.
    - For example, change it to 3300 ps.
    - Refer to the section “Checking a Behavioral Synthesis Script Generated by SSGEN” in this document to check where the operating frequency is specified in the CtoS script.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-5. Behavioral synthesis part 2 (1/2)
  - Execute CtoS.

1. Save the report in exercise 3-3.
   %> mv reports_dut reports_dut.5000

2. Execute CtoS.
   %> run_ctos_dut.sh
   - Make sure that dut.v is generated and that there are no errors in ctos_dut.log.
   - If there is an error in ctos_dut.log, an RTL description may not have been generated. In this case, contact hld_support_m@lm.renesas.com.

3. Check the results of pipeline synthesis.
   - The results of pipeline synthesis are output in reports_dut/report_schedule.log.

Exercise 3: Behavioral Synthesis (Pipeline Synthesis)

- Exercise 3-5. Behavioral synthesis part 2 (2/E)
  - The results of pipeline synthesis (through the use of the SystemC model modeling/.answer/ex2 in exercise 1)

report_schedule.log

	Stage	Operation
CtoS_MAIN_LOOP_while_begin	1	read_dut_in_d0_ln15
CtoS_MAIN_LOOP_while_begin	1	read_dut_in_d1_ln16
CtoS_MAIN_LOOP_while_begin	1	read_dut_in_op_ln18
CtoS_MAIN_LOOP_while_begin	1	read_dut_pre_d0_ln38
CtoS_MAIN_LOOP_while_begin	1	read_dut_pre_d1_ln42
CtoS_MAIN_LOOP_while_begin	1	switch_ln18
CtoS_MAIN_LOOP_while_begin	1	gt_ln20
CtoS_MAIN_LOOP_while_begin	1	add_ln26
CtoS_MAIN_LOOP_while_begin	1	if_ln20
CtoS_MAIN_LOOP_while_begin	1	mux_out_ln20
CtoS_MAIN_LOOP_while_begin	3	if_CtoS_MAIN_LOOP_while_begin_stage3
CtoS_MAIN_LOOP_while_begin	3	mux_out_ln18
CtoS_MAIN_LOOP_while_begin	3	gt_ln50
CtoS_MAIN_LOOP_while_begin	3	if_ln50
CtoS_MAIN_LOOP_while_begin	3	mux_out_ln50
CtoS_MAIN_LOOP_while_begin	2	mul_ln30
CtoS_MAIN_LOOP_while_begin	3	write_dut_out_data_ln56
CtoS_MAIN_LOOP_while_begin	1	write_dut_pre_d0_ln53
CtoS_MAIN_LOOP_while_begin	1	write_dut_pre_d1_ln54

Operations other than those for assignment of out_data are also scheduled in Stage 2 and Stage 3.

```
void dut::thread_main() {
  reset_thread_main();
  wait();
CtoS_MAIN_LOOP:
  while (1) {
    ...
    switch ( in_op.read() ) {
    ...

    case 2:
      out = d0 * d1;
      break;

    ...
    }

    if ( out > 0xFF00 )
      out = 0xFF00;

    pre_d0.write( d0 );
    pre_d1.write( d1 );

    out_data.write( out );

    wait();
  }
}
```

Synthesis through 3-stage pipeline

Stage 1

Stage 2

Stage 1

Stage 3

Stage 1

Stage 3

Exercise 4: Equivalence Check (Pipeline Synthesis)

- Use SLEC to verify the equivalence between a SystemC model and an RTL description.

Exercise directory: synthesis/ex4

Exercise 4-1. Generating an equivalence check script

- Generate a SLEC script.

Exercise 4-2. Verifying equivalence

- Execute SLEC.

Exercise 4: Equivalence Check (Pipeline Synthesis)

Exercise 4-1. Generating an equivalence check script

- Use SSGEN to generate an equivalence check script.

1. Move to the exercise directory.

   %> cd synthesis/ex4

2. Copy data of exercise 3.
   - Copy the SSGEN input file, SystemC descriptions, and CtoS directory of exercise 3.

     %> cp ../ex3/dut.in .

     %> cp -r ../ex3/src .

     %> cp -r ../ex3/ctos .

3. Execute SSGEN.
   - After SSGEN is executed as shown below, only scripts for equivalence check are generated.

     %> ssgen.pl -in dut.in -subdir -only_script -slec

     The following files are generated:

slec/run_slec_dut_eq.sh	Equivalence check execution script.
slec/slec_dut_sc.tcl	SystemC verification script (not used in this exercise).
slec/run_slec_dut_sc.sh	

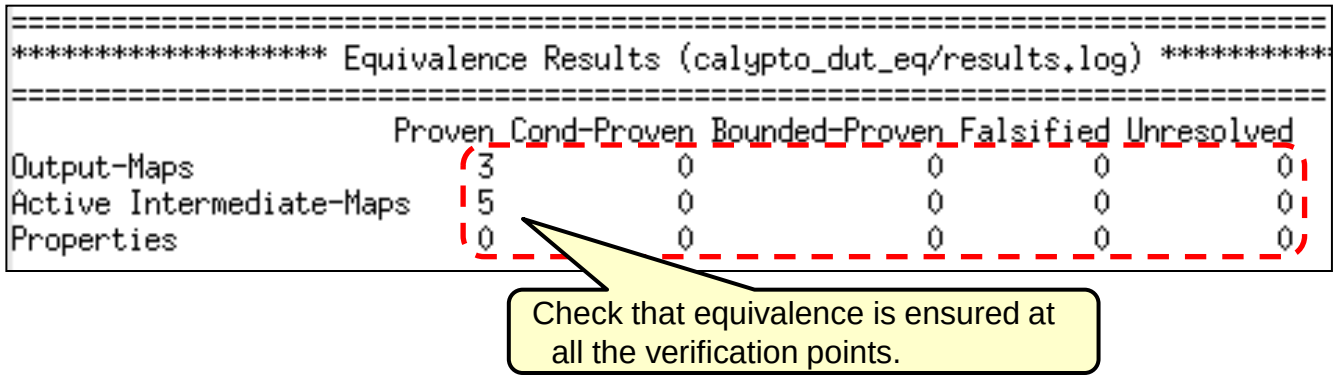
Exercise 4: Equivalence Check (Pipeline Synthesis)

Exercise 4-2. Equivalence check

- Execute SLEC.

1. Execute SLEC.
   %> cd slec
   %> run_slec_dut_eq.sh
   slec_dut_eq.log is output.

2. Check the results of verification.
   - Check that all the items other than Proven show “0” at the end of slec_dut_eq.log.



   - If there is any difference (if Falsified is not “0”), contact hld_support_m@lm.renesas.com.

Summary

- You have learned the following in the high-level design synthesis exercise.
  - Behavioral synthesis
    - Cycle accurate synthesis
    - Pipeline synthesis
  - Equivalence check

Please take advantage of high-level design.
For questions, contact
hld_support_m@lm.renesas.com.

Answers to Exercises

- Answers are stored on the following path in exercise data.
  If you cannot find an answer on your own, use them as a reference.
  synthesis/.answer/ex*

Appendix: How to Tune RTL Data Using CtoS

QoR tuning flow with CtoS

- QoR here means the quality of design such as areas or delays resulting from logic synthesis.
  - Although CtoS makes a report on areas and timings, the report should be used as reference. For example, even if an area report of CtoS shows a large decrease in areas, a logic synthesis report may show few changes (the opposite case is also possible).
- If the desired results are not achieved from (1) to (4), modify the SystemC model in (5).
- Following each of the steps does not always improve QoR. Make a comparison between before and after each step, and select the best results.
- The following assume that settings are changed based on a script generated by SSGEN.

(1) Change scheduling setting.

[Criteria for taking the step]
During logic synthesis, delay constraints are not met, or even though they are met, an increase in delay is a possible cause of an increase in areas.

(2) Change inline function setting.

[Criteria for taking the step]
None in particular. (This step is often effective in a design where the same function is called multiple times.)

(3) Restrict resource sharing.

[Criteria for taking the step]
An increase in the number of multiplexers due to operation unit sharing is a possible cause of an increase in areas.

(4) Change register binding setting.

[Criteria for taking the step]
None in particular. (It is impossible to predict the results.)

(5) Modify the SystemC model.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS

(1) Changing scheduling setting (1/2)

➢Changing the operating frequency constraint

set PERIOD 5100

Change the operating frequency constraint for CtoS. If the operating frequency constraint for logic synthesis is 5000 ps, for example, change it in increments of 100 ps in the range from 4500 to 5500 ps each time generating an RTL description, and select the RTL that produces the best results of logic synthesis.

➢Setting up input delay

set INPUT_DELAY 100

Set up input delay. For example, equalize the input delay in behavioral synthesis with that given as a logic synthesis constraint.

➢Using the useBirthday option for the schedule command.

schedule -useBirthday

Option to restrict operation retiming. The results of scheduling are achieved as described in the SystemC model.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS
  - (1) Changing scheduling setting (2/E)
    - ➢Changing the speed grade of operation units.

      set_attr default_speed_grade 50 [get_design] (50 for example)

      Change the speed grade of operation units. For scheduling, 100 and 0 are assumed for high-speed and low-speed operation units, respectively. The default is 100.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS

(2) Changing inline function setting.

- Functions which are allowed to be non-inlined in behavioral synthesis are non-inlined all at once.

inline_calls

Delete the option “-all” from the SSGEN-generated script. Combinational functions where there is no assignment or reference to ports or signals are non-inlined.

- Specify non-inline functions individually.

```
set func_list [ls $BEHAVIOR]
foreach a $func_list {
    if { $a == "func0" || $a == "func1" } {
      puts "not inline $a"
    } else {
      puts "inline $a"
      inline ${BEHAVIOR}/$a
    }
}
```

func0 and func1 are non-inlined.

Combinational or sequential functions which are specified, for example, as above descriptions are non-inlined.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS
  - (3) Restricting resource sharing.
    - ➢Adding a setting to fully restrict resource sharing.

      set_attr enable_resource_sharing false [get_design]

      Option to prevent timing violation due to resource sharing or to prevent an increase in areas due to an increase in delays.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS
  - (4) Changing register binding setting.
    - Using options for the allocate_registers command.

      allocate_registers –min regs

      Minimizes the number of register bits.

      allocate_registers –min muxes

      Minimizes the number of MUX bits.

      allocate_registers –min regs_and_muxes

      Minimizes the total number of register and MUX bits.

      allocate_registers –min share

      Restricts register sharing more severely than -min muxes.

Appendix: How to Tune RTL Data Using CtoS

- QoR tuning flow with CtoS
  - (5) Modifying the SystemC model.
    - ➢For know-how about modifying a SystemC model, refer to the following:
      - High-level Design SystemC coding style guide
        http://livelink.renesas.com/Livelink/livelink.exe/open/40076986
      - CtoS Know-how
        [REL EDA Tools Information] -> [High-Level Design] -> [C2Silicon] -> [FAQ/Know-how]

Renesas Electronics Corporation

Revision History

	Date of Issue	Description of Revision	Approved by	Checked by	Created by
Rev.1.0	Feb. 3, 2014	Newly created	SIDA Asano Feb.3, 2014	-	SIDA Imamura Feb. 3, 2014