EEDG/CE 6370
Design and Analysis of Reconfigurable Systems
Homework 6
High-Level Synthesis

1. Laboratory Objectives

- Learn how to use a commercial HLS tool.
- Perform the 3 main HLS steps using the commercial HLS tool:
    - Resource allocation
    - Resource Scheduling
    - Binding
- Verify that the design works as expected.
- Learn how to use the automatic testbench generator.
- Create input stimuli files.
- Simulate using the cycle-accurate model generator.
- Perform RTL simulation

2. Summary

- In this lab you will learn to use a commercial HLS tool synthesizing a design from C into RTL and verifying it.

3. Tool Requirement

- NEC CyberWorkbench

Connect to your UTD Nomachine account and edit your ~/.bashrc file in your home directory adding the following:

```
# NEC CWB
export CYBER_PATH=/proj/cad/cwb-6.1
export CYBER_ADMIN_PATH=${CYBER_PATH}/linux_x86_64/admin
export CYBER_SYSTEMC_HOME=${CYBER_PATH}/osci
export CYBER_LIB=${CYBER_PATH}/lib
export LD_LIBRARY_PATH=${CYBER_PATH}/lib:${LD_LIBRARY_PATH}
export CYLMD_LICENSE_FILE=27010@legolas1.utdallas.edu
export PATH=$PATH:${CYBER_PATH}/bin
export PATH=$PATH:${CYBER_ADMIN_PATH}/bin
# Siemens Modelsim (RTL simulator)
source /proj/cad/startup/profile.mentor_2022
```

Test that it works by entering the following command at the terminal window:

```
% cwb &
```

This should launch CyberWorkBench (CWB) OR type:

```
%which cwb
```

Note: We do only have 8 CWB GUI licenses. When you start CWB the GUI is launched and a license is being used. Once you are done, please close the GUI and make sure that the GUI process is not running in the background. If it is → kill it as follows:

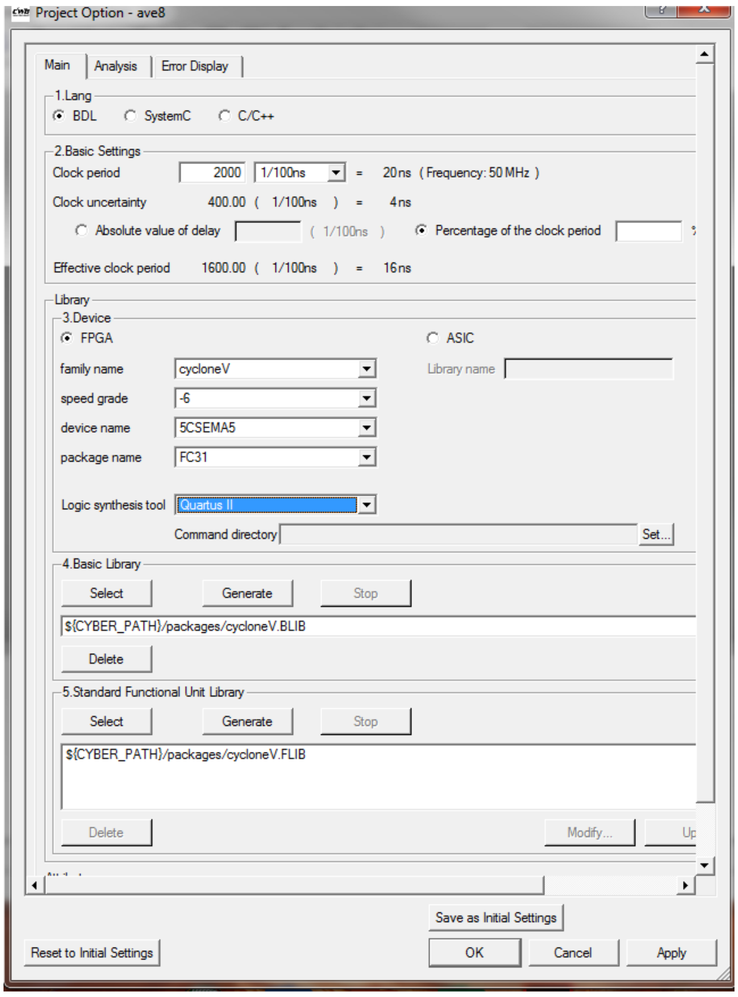
```
% ps -aux | grep <username>
% kill 9 <PID GUI>
```

If no GUI license is available you can execute the synthesis process by calling the individual tools from them command prompt as shown in the appendix.

4. Average of 8 Project

Creating a new empty project

1. Open CWB (either clicking on the CWB icon on the desktop or on linux %cwb &) - /proj/cad/cwb-6.1/bin/cwb
2. Create a new project and workspace called ave8 (File→New)
3. Set the project parameters as shown in the figure:



1.Lang: [✓ BDL] SystemC C/C++

2.Basic Settings

Clock period 2000 1/100ns = 20ns (Frequency: 50 MHz)

Clock uncertainty 400.00 (1/100ns) = 4ns

Absolute value of delay (1/100ns) [✓ Percentage of the clock period]

Effective clock period 1600.00 (1/100ns) = 16ns

Library

3.Device: [✓ FPGA] ASIC

family name: cycloneV

speed grade: -6

device name: 5CSEMA5

package name: FC31

Library name:

Logic synthesis tool: Quartus II

Command directory: Set...

4.Basic Library: Select Generate Stop

${CYBER_PATH}/packages/cycloneV.BLIB

Delete

5.Standard Functional Unit Library: Select Generate Stop

${CYBER_PATH}/packages/cycloneV.FLIB

Delete Modify... Up

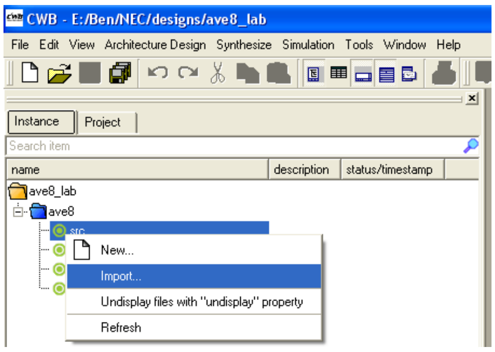
The main items that need to be specified are:

- Clock period (2000 x 1/100ns = 20ns ⬄ 50 MHz)
- Target device Cyclone V 5CSEMA5F31C6 device
- Logic Synthesis tool → Quartus II
- Select the Cyclone V libraries which contain the area and delay information of the basic operations
  - BLIB (Basic Library): contains the delay and area information of basic logic gates (AND, OR,etc..) and muxes of different sizes (cycloneV.BLIB)
  - FLIB (Functional Unit Libirary): Contains the area and delay information of Functional units of different sizes (e.g. 2-4-6-8-12 bit adder) (cycloneV.FLIB)

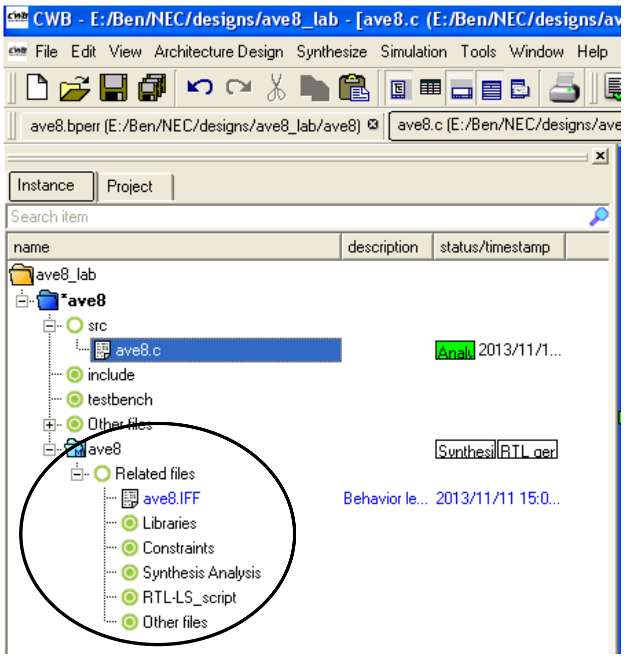
4. Click on Apply to generate the empty project

Editing or Inserting a ANSI-C description for synthesis

1. Right click on source → Import → ave8.c

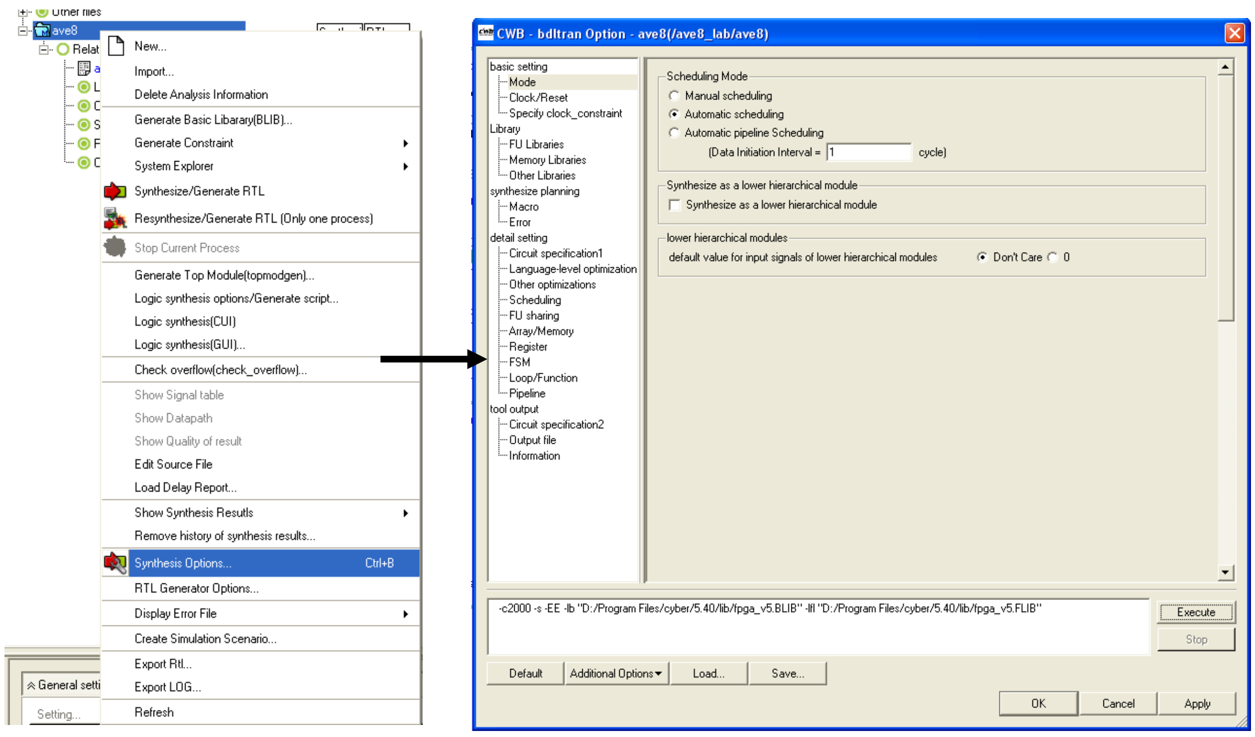

CWB - E:/Ben/NEC/designs/ave8_lab
File Edit View Architecture Design Synthesize Simulation Tools Window Help
Instance
Project
Search item
name
description
status/timestamp
ave8_lab
ave8
src
New...
Import...
Undisplay files with "undisplay" property
Refresh

2. Parse source code to check if there are any syntax errors. The console window will indicate if any errors have occurred and where. A 'light blue' folder with an .IFF file will be generated if the parsing was successful



Setting up Synthesis mode and other constraints

1. Right click on 'light-blue' folder → Synthesis options



Scheduling Mode
Manual scheduling [✓ Automatic scheduling] Automatic pipeline Scheduling
(Data Initiation Interval = 1 cycle)

Synthesize as a lower hierarchical module
☐ Synthesize as a lower hierarchical module

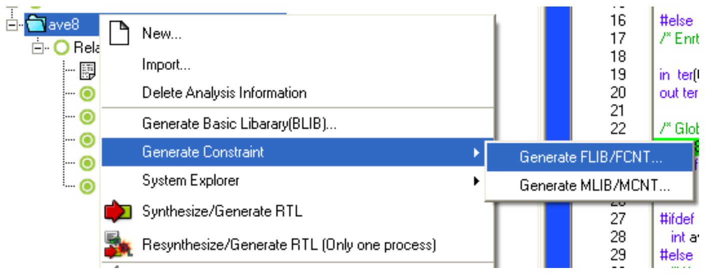
lower hierarchical modules
default value for input signals of lower hierarchical modules [✓ Don't Care] 0

```
-c2000 -s -EE -lb "D:/Program Files/cyber/5.40/lib/fpga_v5.BLIB" -lfl "D:/Program Files/cyber/5.40/lib/fpga_v5.FLIB"
```

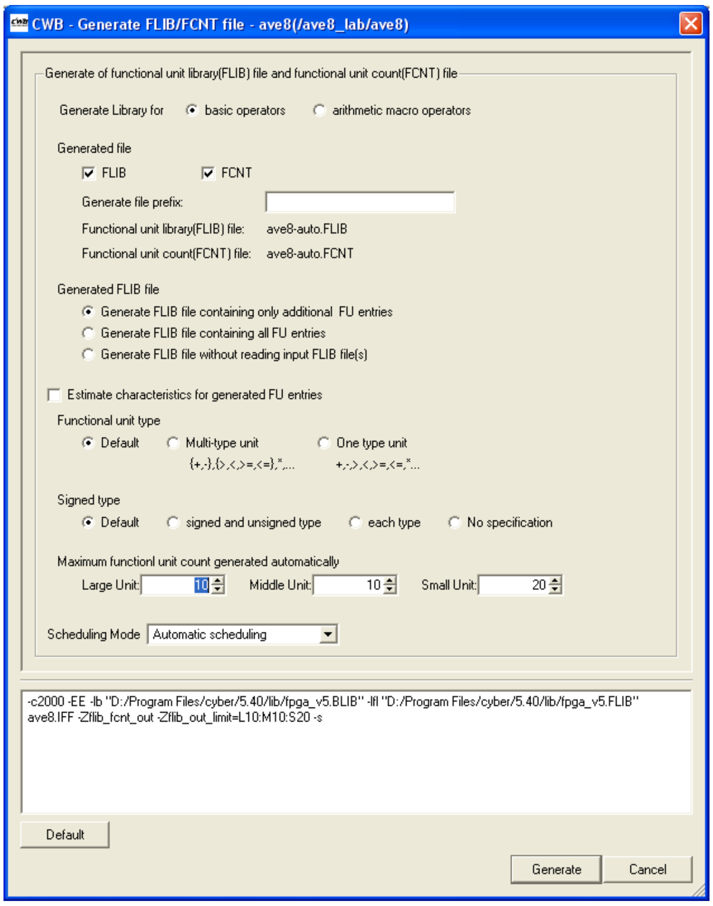
2. Set the scheduling mode to 'Automatic scheduling'. This implies that CWB will automatically time the C description at the scheduling phase. Manual scheduling implies that the user will manually time the description using the '$' sign as a clock boundary.
3. Other synthesis options that control the synthesis process can also be set here as well as the target clock frequency.
4. Click on 'Apply' and OK to accept the changes.

Create the FU constraint files (Resource Allocation)

1. Righ-click on 'light-blue' folder → Generate Constraint → Generate FLIB/FCNT to open a dialog window that allows the generation of the resource constraint file



2. Increase the 'Large Unit' field to 10 as shown in the dialog window.


CWB - Generate FLIB/FCNT file - ave8(/ave8_lab/ave8)

Generate of functional unit library(FLIB) file and functional unit count(FCNT) file

Generate Library for [✓ basic operators] arithmetic macro operators

Generated file

[✓ FLIB] [✓ FCNT]

Generate file prefix:

Functional unit library(FLIB) file: ave8-auto.FLIB

Functional unit count(FCNT) file: ave8-auto.FCNT

Generated FLIB file

[✓ Generate FLIB file containing only additional FU entries]

Generate FLIB file containing all FU entries

Generate FLIB file without reading input FLIB file(s)

Estimate characteristics for generated FU entries

Functional unit type

[✓ Default] Multi-type unit {+,-},{>,<,>=,<=},*,... One type unit +,-,>,<,>=,<=,*...

Signed type

[✓ Default] signed and unsigned type each type No specification

Maximum functionl unit count generated automatically

Large Unit: 10 Middle Unit: 10 Small Unit: 20

Scheduling Mode: Automatic scheduling

```
-c2000 -EE -lb "D:/Program Files/cyber/5.40/lib/fpga_v5.BLIB" -lfl "D:/Program Files/cyber/5.40/lib/fpga_v5.FLIB"
ave8.IFF -Zflib_fcnt_out -Zflib_out_limit=L10:M10:S20 -s
```

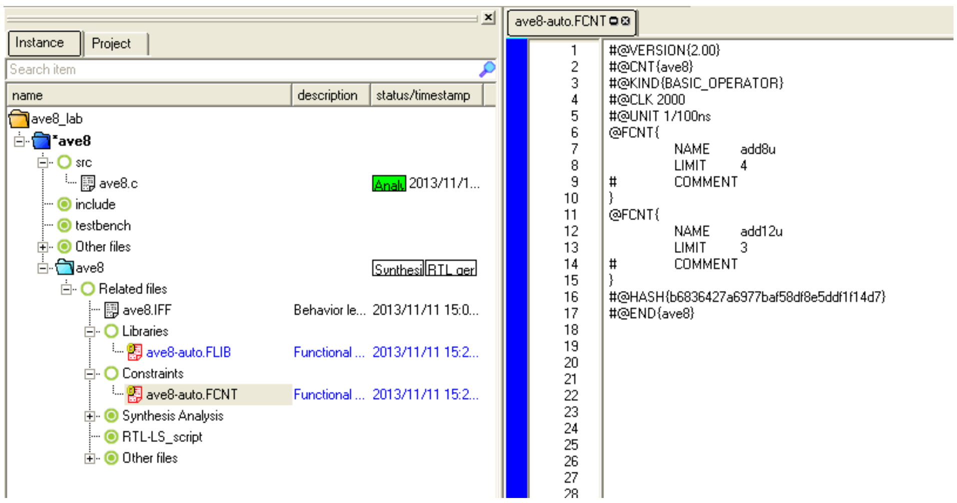
CWB considers FUs of different bws as Large (<=8 bits) , middle (8bits<=4 bits) and small (>4 bits). By default it limits the number of FUs allocated as indicated in the dialog window to 5:10:20

3. Click 'Generate' . Two new files will be generated:

   ave8-auto.FLIB: FU Library file containing FUs area and delay of FUs not contained in the initial FLIB/BLIB files (empty in most cases like in this)

   ave8-auto.FCNT: Constraint file limiting the number of FUs tha the synthesizer can instantiate in paralle.

Review the contents of the files by 'right-click → open by internal editor



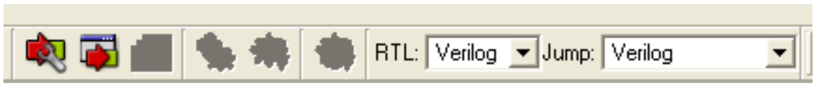
```
name                                  description     status/timestamp
ave8_lab
  *ave8
    src
      ave8.c                                          Analy 2013/11/1...
    include
    testbench
    Other files
    ave8                                              Synthesi RTL ger
      Related files
        ave8.IFF                      Behavior le...  2013/11/11 15:0...
        Libraries
          ave8-auto.FLIB              Functional ...  2013/11/11 15:2...
        Constraints
          ave8-auto.FCNT              Functional ...  2013/11/11 15:2...
        Synthesis Analysis
        RTL-LS_script
        Other files
```

```
#@VERSION{2.00}
#@CNT{ave8}
#@KIND{BASIC_OPERATOR}
#@CLK 2000
#@UNIT 1/100ns
@FCNT{
          NAME      add8u
          LIMIT     4
#         COMMENT
}
@FCNT{
          NAME      add12u
          LIMIT     3
#         COMMENT
}
#@HASH{b6836427a6977baf58df8e5ddf1f14d7}
#@END{ave8}
```

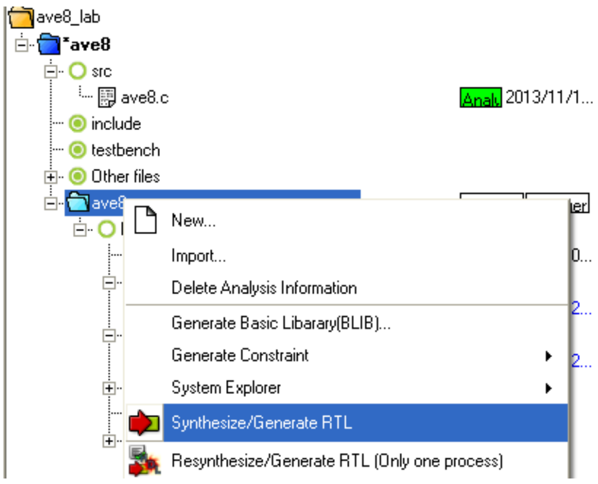
High-Level Synthesis

The conversion of the C program into RTL (Verilog or VHDL) can now take place:

1. Select Verilog at the toolbar

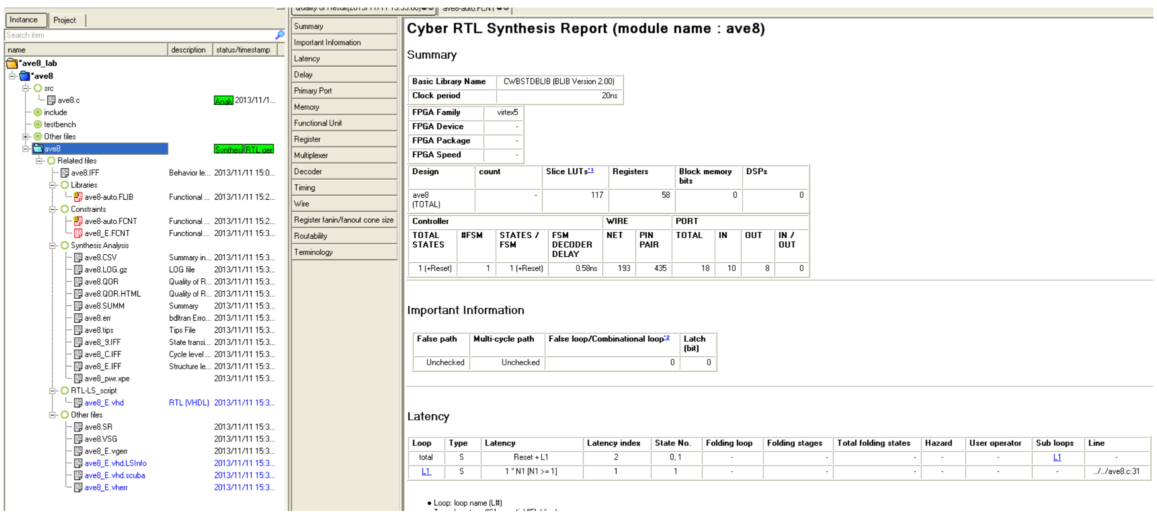


2. Righ-click on 'light-blue'folder → Synthesize/Generate RTL (or toolbar)



A report file (Quality of Results-QoR) will be generated and appears automatically. It contains all the synthesis information: FPGA resources used, critical path, design latency, etc…

NOTE: The information reported is based on the FLIB and BLIB file provided. It is therefore important to provide the correct library files or to re-generat these if a different device is targeted.
Also, the reported data should be confirmed performing a logic synthesis and a simulation.



name	description	status/timestamp
*ave8_lab		
*ave8		
src		
ave8.c		Analy 2013/11/1...
include		
testbench		
Other files		
ave8		Synthesi RTL ger
Related files		
ave8.IFF	Behavior le...	2013/11/11 15:0...
Libraries		
ave8-auto.FLIB	Functional ...	2013/11/11 15:2...
Constraints		
ave8-auto.FCNT	Functional ...	2013/11/11 15:2...
ave8_E.FCNT	Functional ...	2013/11/11 15:3...
Synthesis Analysis		
ave8.CSV	Summary in...	2013/11/11 15:3...
ave8.LOG.gz	LOG file	2013/11/11 15:3...
ave8.QOR	Quality of R...	2013/11/11 15:3...
ave8.QOR.HTML	Quality of R...	2013/11/11 15:3...
ave8.SUMM	Summary	2013/11/11 15:3...
ave8.err	bdltran Erro...	2013/11/11 15:3...
ave8.tips	Tips File	2013/11/11 15:3...
ave8_9.IFF	State transi...	2013/11/11 15:3...
ave8_C.IFF	Cycle level ...	2013/11/11 15:3...
ave8_E.IFF	Structure le...	2013/11/11 15:3...
ave8_pwr.xpe		2013/11/11 15:3...
RTL-LS_script		
ave8_E.vhd	RTL (VHDL)	2013/11/11 15:3...
Other files		
ave8.SR		2013/11/11 15:3...
ave8.VSG		2013/11/11 15:3...
ave8_E.vgerr		2013/11/11 15:3...
ave8_E.vhd.LSInfo		2013/11/11 15:3...
ave8_E.vhd.scuba		2013/11/11 15:3...
ave8_E.vherr		2013/11/11 15:3...

Cyber RTL Synthesis Report (module name : ave8)

Summary

Basic Library Name	CWBSTDBLIB (BLIB Version 2.00)
Clock period	20ns

FPGA Family	virtex5
FPGA Device	-
FPGA Package	-
FPGA Speed	-

Design	count	Slice LUTs*1	Registers	Block memory bits	DSPs
ave8 (TOTAL)	-	117	58	0	0

Controller				WIRE		PORT			
TOTAL STATES	#FSM	STATES / FSM	FSM DECODER DELAY	NET	PIN PAIR	TOTAL	IN	OUT	IN / OUT
1 (+Reset)	1	1 (+Reset)	0.58ns	193	435	18	10	8	0

Important Information

False path	Multi-cycle path	False loop/Combinational loop*2	Latch (bit)
Unchecked	Unchecked	0	0

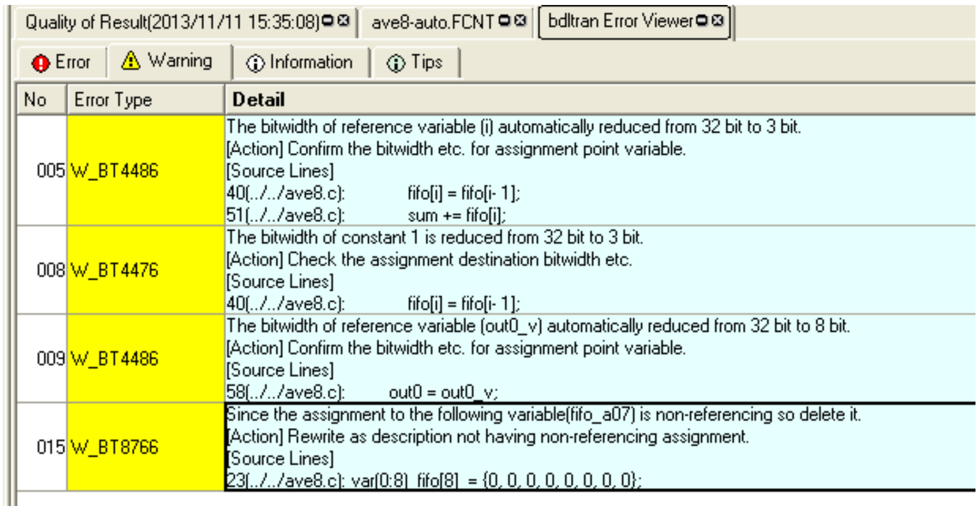
Latency

Loop	Type	Latency	Latency index	State No.	Folding loop	Folding stages	Total folding states	Hazard	User operator	Sub loops	Line
total	S	Reset + L1	2	0, 1	-	-	-	-	-	L1	-
L1	S	1 * N1 (N1 >= 1)	1	1	-	-	-	-	-	-	../../ave8.c:31

- Loop: loop name (L#)

Other report files and information files are also created (.err, .SUMM, .tips). The .err files contains any possible synthesis error, warnings and tips to improve the synthesis results.

3. Open ave8.err file and review the warning, information and tips messages

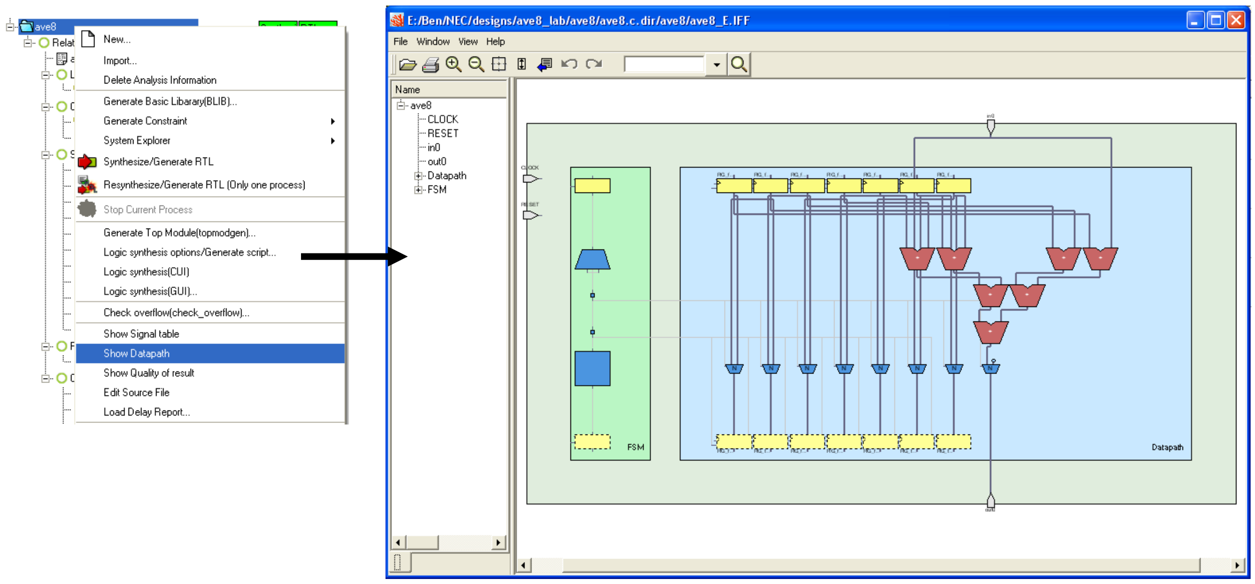


No	Error Type	Detail
005	W_BT4486	The bitwidth of reference variable (i) automatically reduced from 32 bit to 3 bit. [Action] Confirm the bitwidth etc. for assignment point variable. [Source Lines] 40(../../ave8.c): fifo[i] = fifo[i- 1]; 51(../../ave8.c): sum += fifo[i];
008	W_BT4476	The bitwidth of constant 1 is reduced from 32 bit to 3 bit. [Action] Check the assignment destination bitwidth etc. [Source Lines] 40(../../ave8.c): fifo[i] = fifo[i- 1];
009	W_BT4486	The bitwidth of reference variable (out0_v) automatically reduced from 32 bit to 8 bit. [Action] Confirm the bitwidth etc. for assignment point variable. [Source Lines] 58(../../ave8.c): out0 = out0_v;
015	W_BT8766	Since the assignment to the following variable(fifo_a07) is non-referencing so delete it. [Action] Rewrite as description not having non-referencing assignment. [Source Lines] 23(../../ave8.c): var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};

Review of Synthesis Result

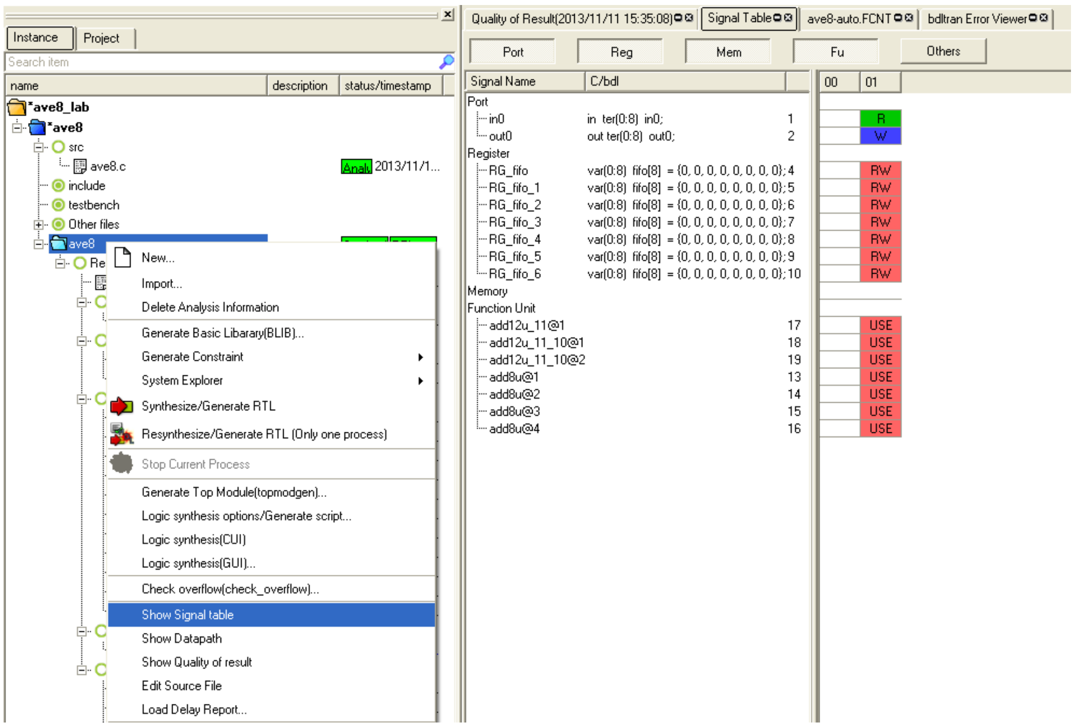
The synthesis result can be reviewed in different ways.

1. Schematic View: Right-click 'light-blue' folder→ Show Datapath



This opens the CWB's datapath viewer. Clicking on any part of the schematic will also highlight the source code in CWB that corresponds to it (cross referencing)

2. Signal Table: Right-click 'light-blue' folder→ Show Signal Table

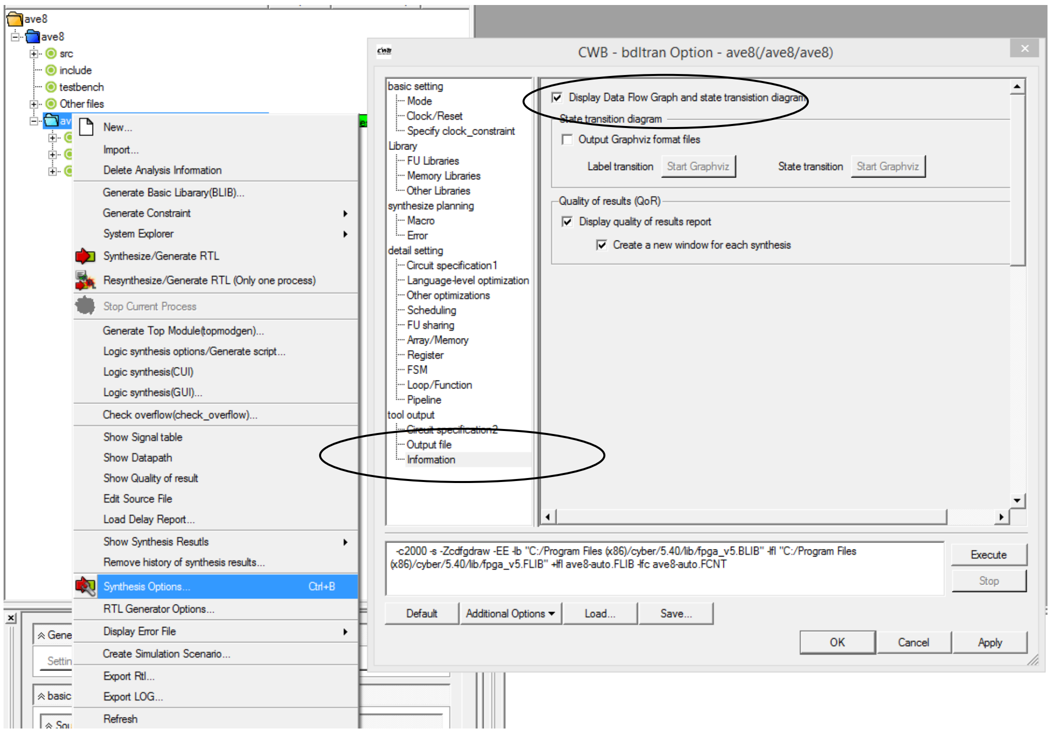


Signal Name	C/bdl		00	01
Port				
in0	in ter(0:8) in0;	1		R
out0	out ter(0:8) out0;	2		W
Register				
RG_fifo	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	4		RW
RG_fifo_1	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	5		RW
RG_fifo_2	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	6		RW
RG_fifo_3	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	7		RW
RG_fifo_4	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	8		RW
RG_fifo_5	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	9		RW
RG_fifo_6	var(0:8) fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};	10		RW
Memory				
Function Unit				
add12u_11@1		17		USE
add12u_11_10@1		18		USE
add12u_11_10@2		19		USE
add8u@1		13		USE
add8u@2		14		USE
add8u@3		15		USE
add8u@4		16		USE

The Signal table shows the timing diagram of the synthesis. When inputs and outputs are being read or written and when registers accessed. State 00 = reset state and state 01 = stating doing the computation.

Scheduling Result

You can review the scheduling result, by enabling a synthesis option to show the scheduled of the synthesized circuit. In Information →select Display Data Flow Graph



CWB - bdltran Option - ave8(/ave8/ave8)

[✓] Display Data Flow Graph and state transition diagram

State transition diagram

[ ] Output Graphviz format files

Label transition

State transition

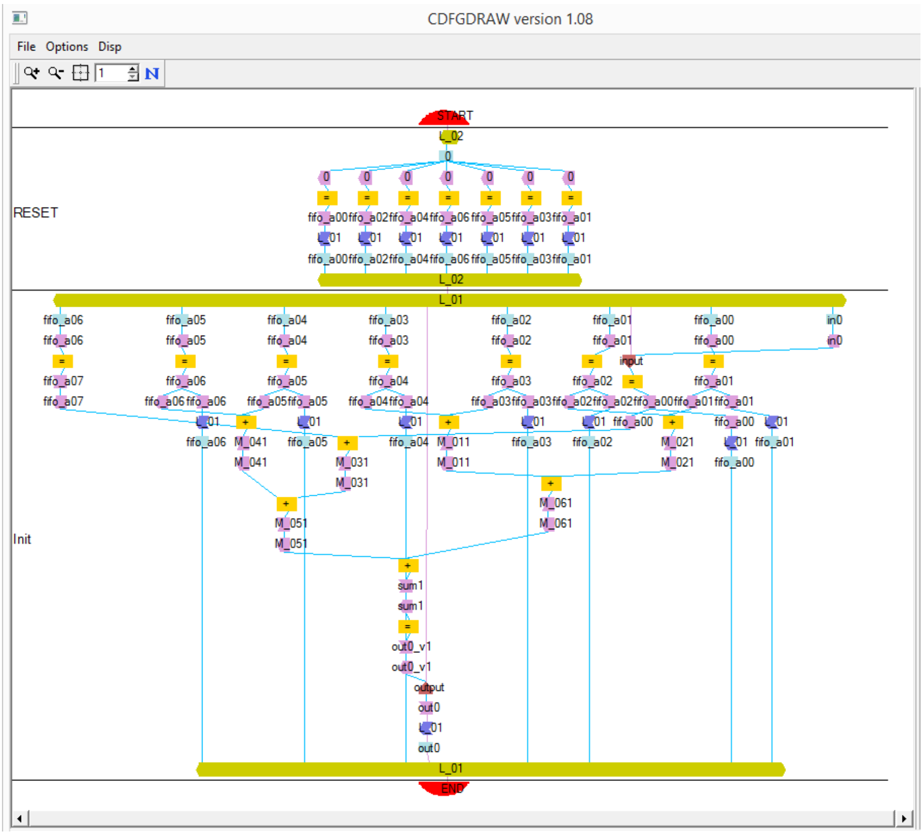
Quality of results (QoR)

[✓] Display quality of results report

[✓] Create a new window for each synthesis

```
-c2000 -s -Zcdfgdraw -EE -lb "C:/Program Files (x86)/cyber/5.40/lib/fpga_v5.BLIB" -lfl "C:/Program Files
(x86)/cyber/5.40/lib/fpga_v5.FLIB" +lfl ave8-auto.FLIB -lfc ave8-auto.FCNT
```

Re-synthesize the circuit. The CDFG will be displayed. The synthesis will continue after the window is closed.



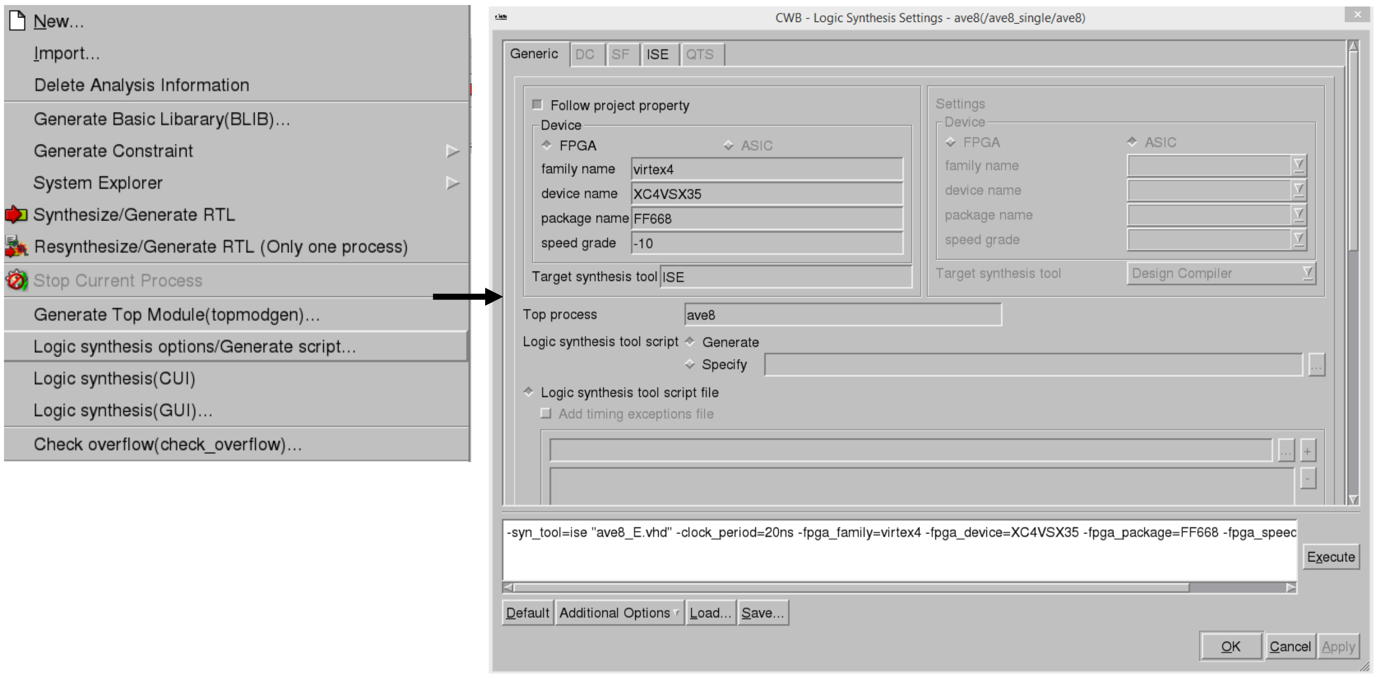
Logic Synthesis

It is now possible to call Intel/Altera's or AMD/Xilinx's logic synthesizers from within CWB. For this, first we need to create a synthesis script for the target FPGA vendor. Then the logic synthesizer can be executed by either opening its IDE (GUI) or from the command line interface (CUI).

NOTE: Make sure that CWB has the path to Quartus setup in CWB's GUI toolbar: Tools→Options→Tool Path

1. Create Logic synthesis script: Right click on light blue process folder → Logic synthesis options/Generate script →Execute

   This creates


New...
Import...
Delete Analysis Information
Generate Basic Libarary(BLIB)...
Generate Constraint
System Explorer
Synthesize/Generate RTL
Resynthesize/Generate RTL (Only one process)
Stop Current Process
Generate Top Module(topmodgen)...
Logic synthesis options/Generate script...
Logic synthesis(CUI)
Logic synthesis(GUI)...
Check overflow(check_overflow)...
CWB - Logic Synthesis Settings - ave8(/ave8_single/ave8)
Generic
DC
SF
ISE
QTS
Execute
Default
Additional Options
Load...
Save...
OK
Cancel
Apply

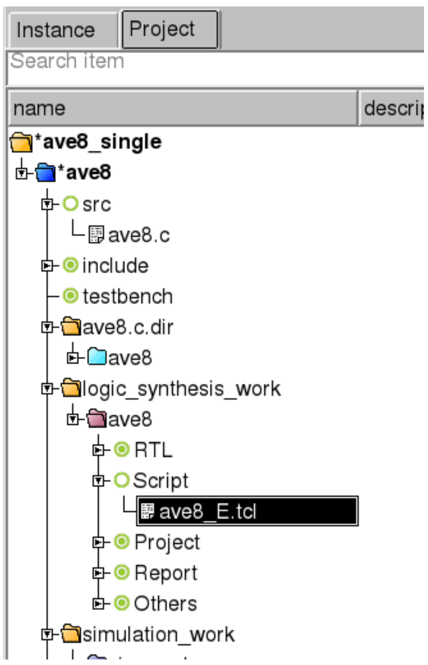
Follow project property
Device
FPGA / ASIC
family name: virtex4
device name: XC4VSX35
package name: FF668
speed grade: -10
Target synthesis tool: ISE

Settings
Device
FPGA / ASIC
family name
device name
package name
speed grade
Target synthesis tool: Design Compiler

Top process: ave8
Logic synthesis tool script: Generate / Specify
Logic synthesis tool script file
Add timing exceptions file

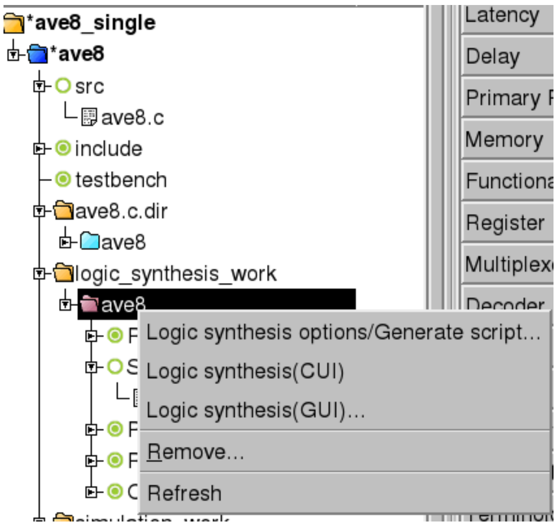
```
-syn_tool=ise "ave8_E.vhd" -clock_period=20ns -fpga_family=virtex4 -fpga_device=XC4VSX35 -fpga_package=FF668 -fpga_speec
```

This will create a .tcl script file for Quartus. The newly generated red folder includes all the information regarding the logic synthesis



Note: You might need to set up the path for the Altera tools in Tools→ Options→Tool Path

2. Logic Synthesis: Right clicking on the red folder → Logic synthesis (CUI) will launch Quartus in command mode. Quartus will start and the Quartus outputs will be displayed on CWB's console window



Once Quartus finishes, the logic synthesis results can be observed at the report folder.

Note: If Quartus is not installed on the Linux machine that you are using, then you will have to copy the RTL code generated by CWB and manually generate a Quartus project on your local machine.

To see if Quartus is installed type on the terminal window:

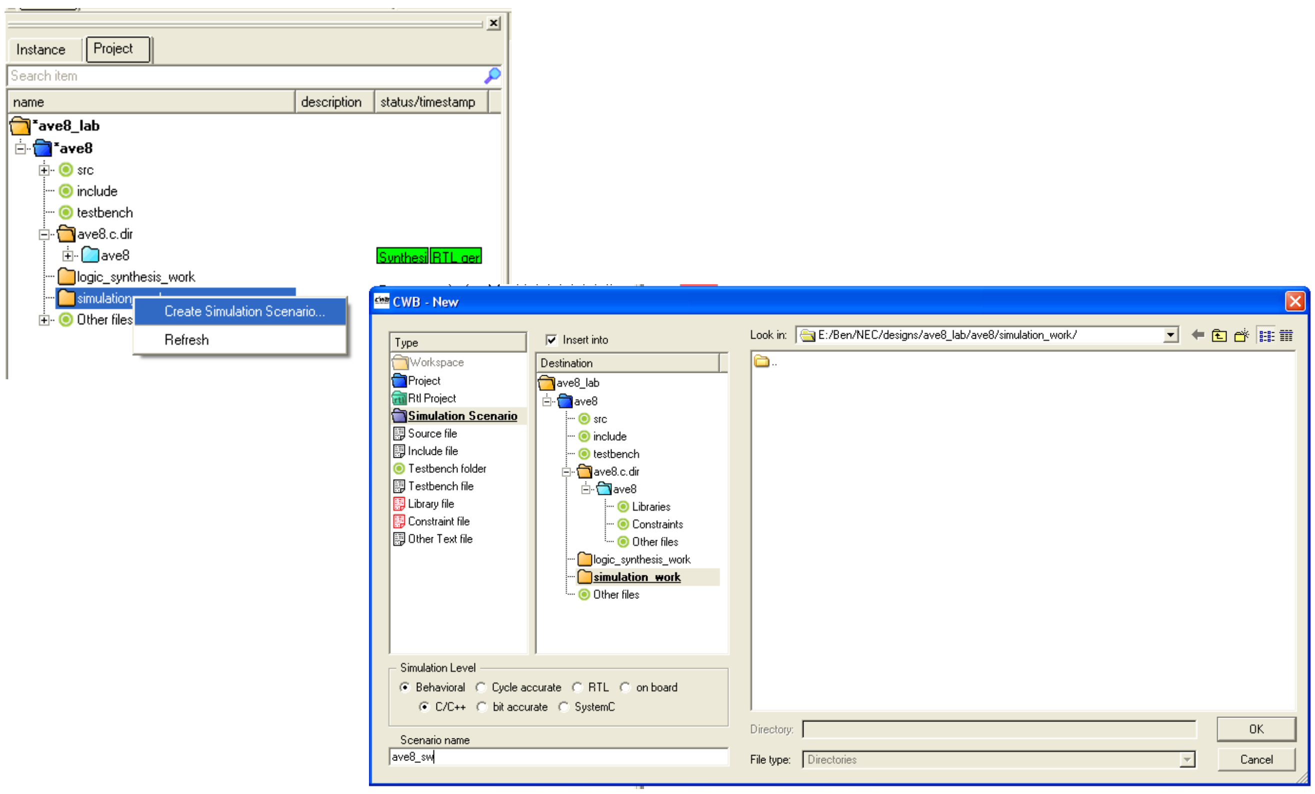
```
%which quartus
```

Design Verification: Software Simulation

There are different levels of simulations that can be used to verify the design:

- Pure SW simulation: normal SW simulation
- Behavioral simulation: untimed simulation (like SW), but considering the HW data types (ter, var, reg)
- Cycle-accurate simulation : timed simulation of the scheduled synthesis results
- RTL simulation

1. The first step is to do a normal SW simulation from within CWB. Chane to the ‘Project’ tab → Create Simulation Scenario.



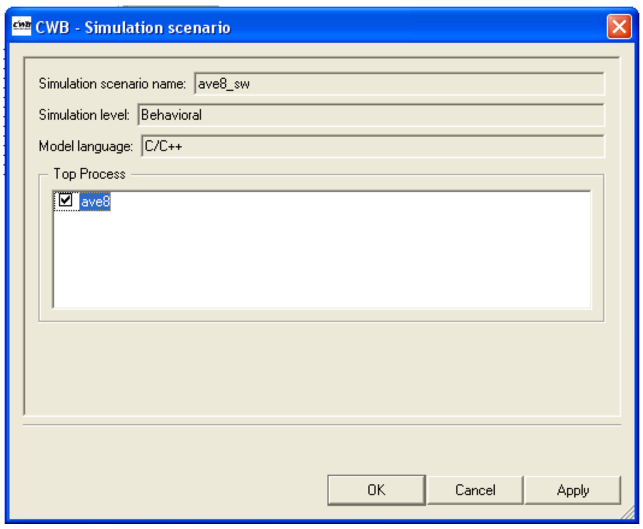
Look in: E:/Ben/NEC/designs/ave8_lab/ave8/simulation_work/

Simulation Level: (● Behavioral) ○ Cycle accurate ○ RTL ○ on board

(● C/C++) ○ bit accurate ○ SystemC

Scenario name: ave8_sw

2. Select “Behavioral” and C/C++ and call the scenario name ‘ave8_sw’
3. Select the top process (check the check box)



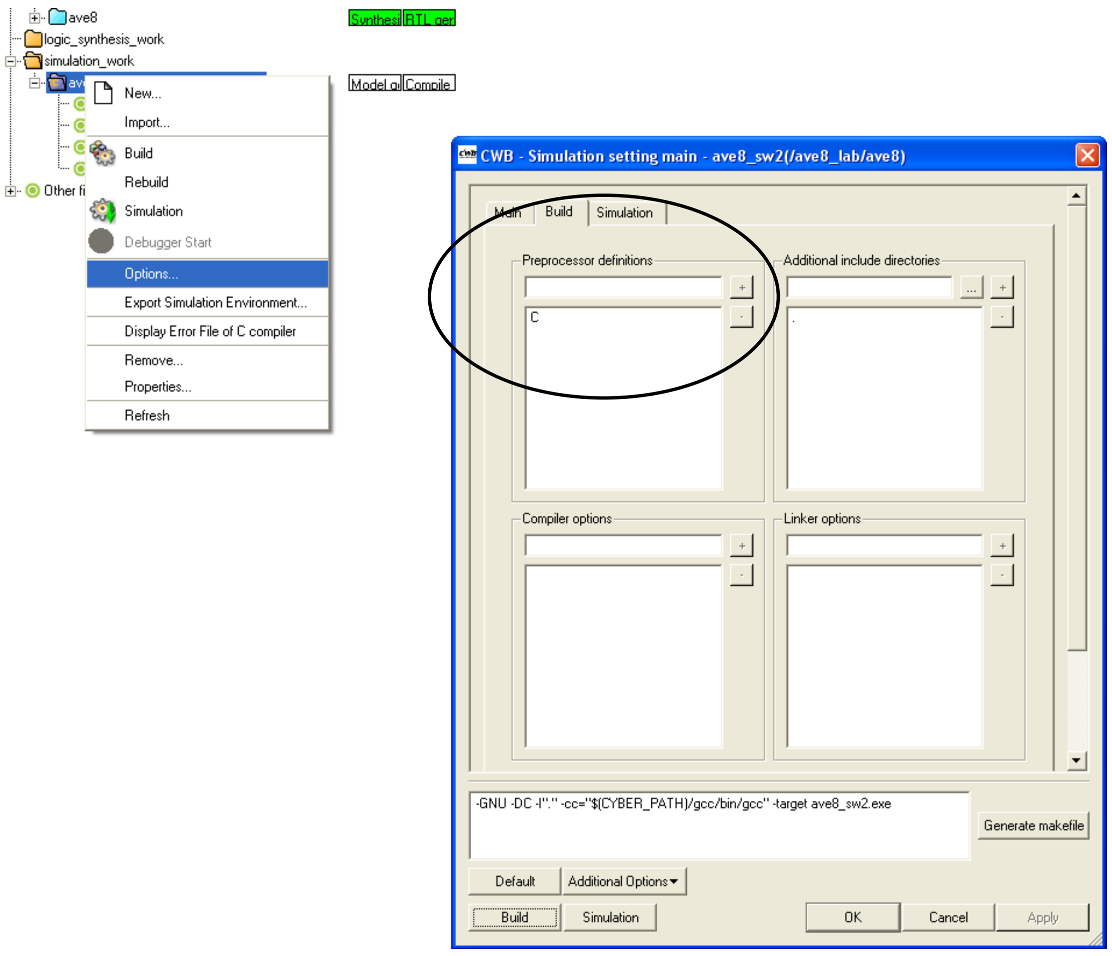
Simulation scenario name: ave8_sw

Simulation level: Behavioral

Model language: C/C++

Top Process: ✓ ave8

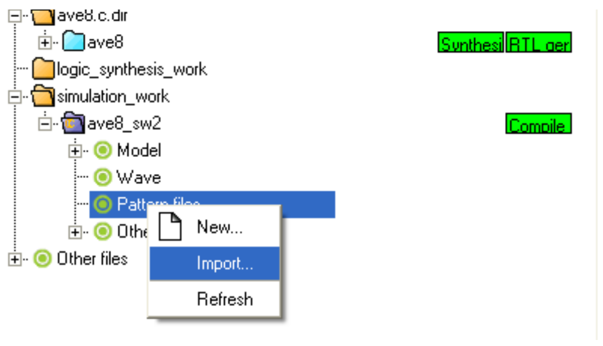
4. Right-click on ‘ave8_sw” folder → options→ Build tab -> Add C to the preprocessor definitions


CWB - Simulation setting main - ave8_sw2(/ave8_lab/ave8)

Preprocessor definitions: C

```
-GNU -DC -I"." -cc="$(CYBER_PATH)/gcc/bin/gcc" -target ave8_sw2.exe
```

5. Add the indata.txt input stimuli file to the simulation environment



6. Build the binary (.exe file) and simulate it

The outdata.txt file with the SW simulation values will be generated. This will be our 'golden' output against which the simulation results throughout the different synthesis stages will be compared against.

Note: Based on the operating system used by the Linux machine that you use, the GUI might give you an error like:

'awk: error while loading shared libraries: libgmp.so.3: cannot open shared object file: No such file or directory'

In this case you can compile the code following these steps at the terminal window:

1.) Go to the folder where you created the project
2.) Inside you will find a 'simulation_work' folder
3.) Go there and into the actual simulation that you want to execute. SW, cycle-accurate, etc…
4.) Compile the code from there by executing the Makefile generated by the GUI:

```
% make -f Makefile.GNU
```

This generates the .exe file

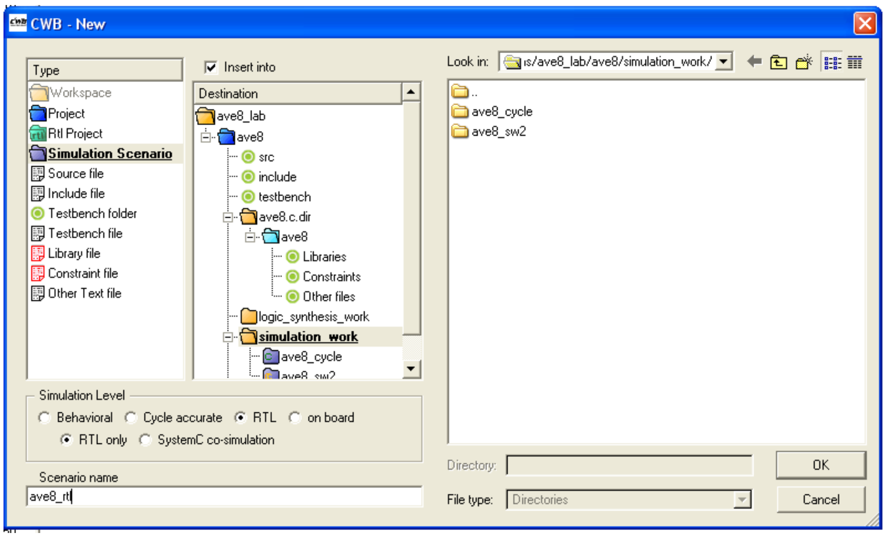
5.) Run the compiled program

```
%./name.exe
```

RTL simulation

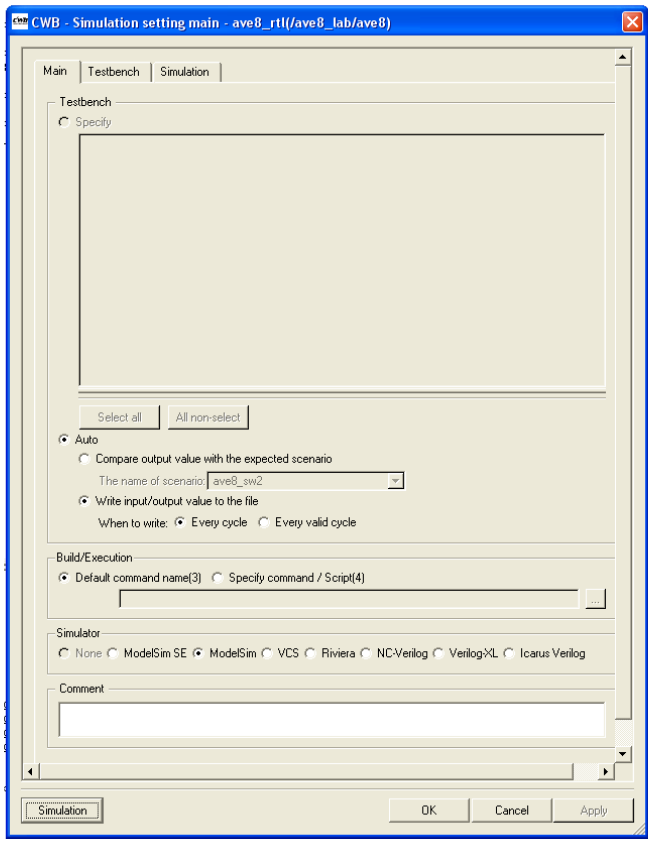
The last verification step is to simulate if the RTL generated also matches the 'golden' output.

1. Create a new simulation scenario and select RTL. Name the scenario ave8_rtl



Insert into: ☑
Look in: ıs/ave8_lab/ave8/simulation_work/
Simulation Level: ( ) Behavioral ( ) Cycle accurate (● RTL) ( ) on board
(● RTL only) ( ) SystemC co-simulation
Scenario name: ave8_rtl
File type: Directories

2. Copy the indata.txt and outdata.txt file into the new simulation folder and rename as in0.clv and out0.clv. CWB needs an input and output file with the same name as the ports as input stimuli.
3. Generate the testbench : Select RTL simulator (Modelsim) and input stimuli every cycle, Input data from file and radix as decimal. Click OK and run the simulation



CWB - Simulation setting main - ave8_rtl(/ave8_lab/ave8)

Main | Testbench | Simulation

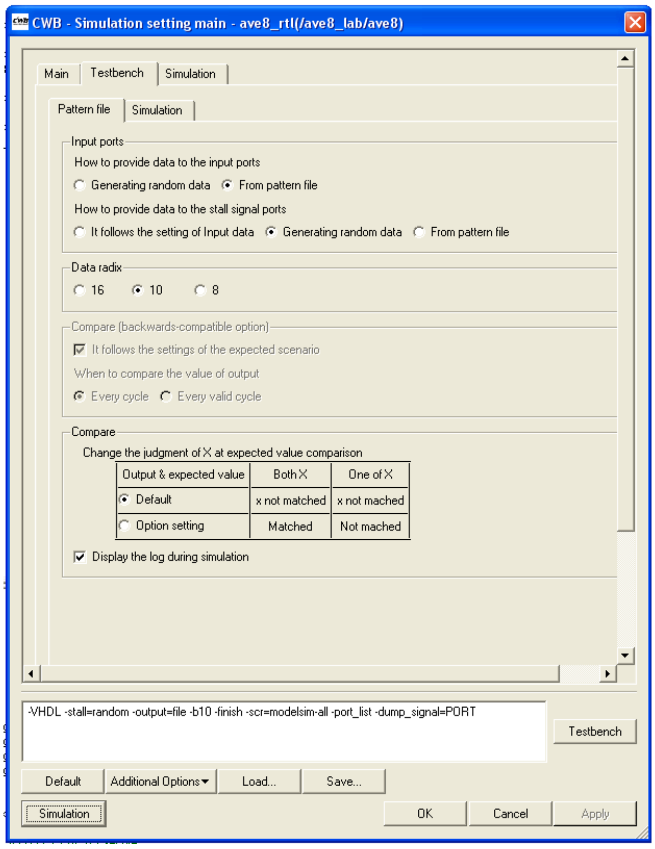
Testbench
- ( ) Specify
- Select all | All non-select
- (•) Auto
  - ( ) Compare output value with the expected scenario
    - The name of scenario: ave8_sw2
  - (•) Write input/output value to the file
    - When to write: (•) Every cycle ( ) Every valid cycle

Build/Execution
- (•) Default command name(3) ( ) Specify command / Script(4)

Simulator
- ( ) None ( ) ModelSim SE (•) ModelSim ( ) VCS ( ) Riviera ( ) NC-Verilog ( ) Verilog-XL ( ) Icarus Verilog

Comment

Simulation | OK | Cancel | Apply



CWB - Simulation setting main - ave8_rtl(/ave8_lab/ave8)

Main | Testbench | Simulation

Pattern file | Simulation

Input ports
- How to provide data to the input ports
  - ( ) Generating random data (•) From pattern file
- How to provide data to the stall signal ports
  - ( ) It follows the setting of Input data (•) Generating random data ( ) From pattern file

Data radix
- ( ) 16 (•) 10 ( ) 8

Compare (backwards-compatible option)
- [✓] It follows the settings of the expected scenario
- When to compare the value of output
  - (•) Every cycle ( ) Every valid cycle

Compare
- Change the judgment of X at expected value comparison

Output & expected value	Both X	One of X
(•) Default	x not matched	x not mached
( ) Option setting	Matched	Not mached

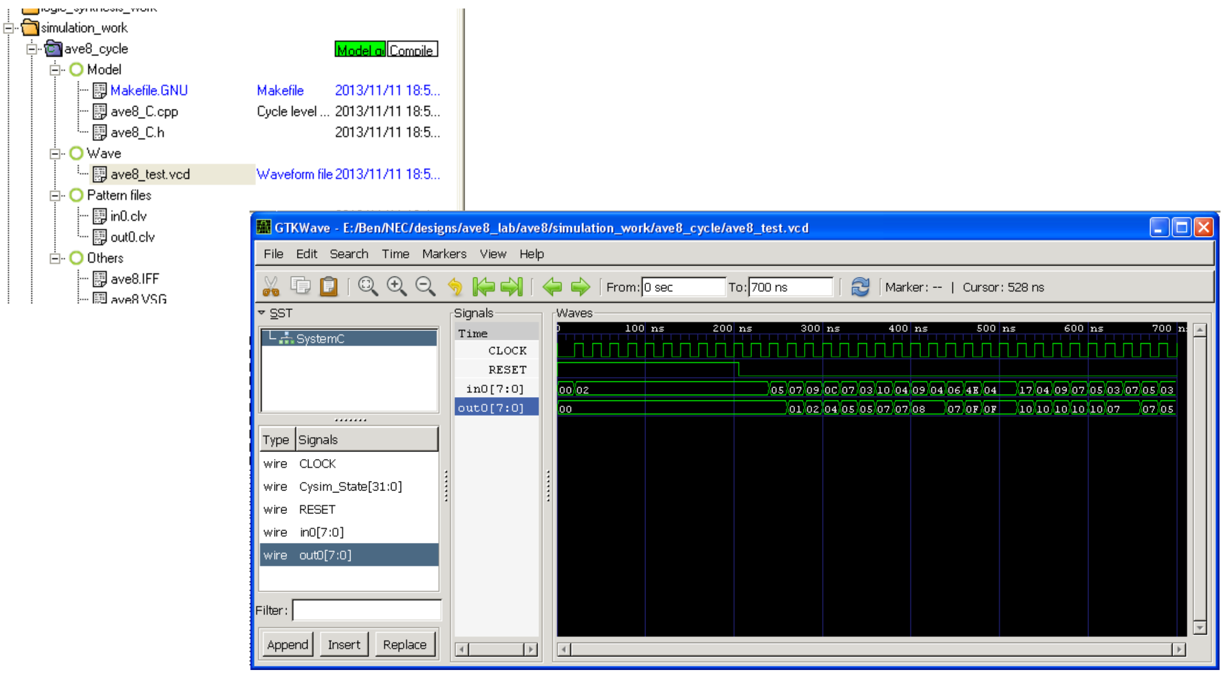
- [✓] Display the log during simulation

```
-VHDL -stall=random -output=file -b10 -finish -scr=modelsim-all -port_list -dump_signal=PORT
```

Testbench | Default | Additional Options | Load... | Save... | Simulation | OK | Cancel | Apply

Note: Make sure you have added Modelsim to you path by adding this to your ~/.bashrc file or calling it from the terminal: source /proj/cad/startup/profile.mentor_2022

A .vcd file which can be opened by a waveform viewer (e.g. GTKwave) is generated. Double click on it to open it.



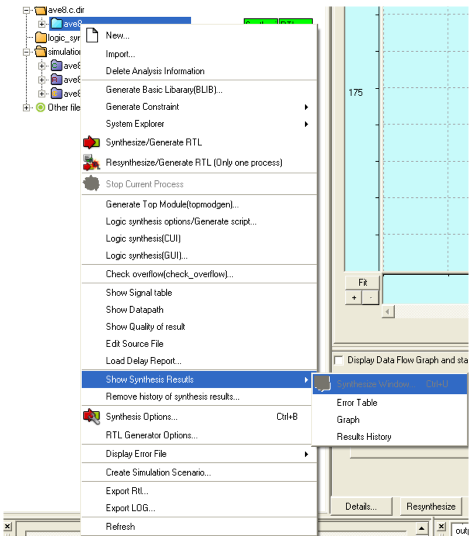
simulation_work
ave8_cycle
Model
Makefile.GNU Makefile 2013/11/11 18:5...
ave8_C.cpp Cycle level ... 2013/11/11 18:5...
ave8_C.h 2013/11/11 18:5...
Wave
ave8_test.vcd Waveform file 2013/11/11 18:5...
Pattern files
in0.clv
out0.clv
Others
ave8.IFF
ave8.VSG
GTKWave - E:/Ben/NEC/designs/ave8_lab/ave8/simulation_work/ave8_cycle/ave8_test.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 700 ns Marker: -- | Cursor: 528 ns
SST
SystemC
Type Signals
wire CLOCK
wire Cysim_State[31:0]
wire RESET
wire in0[7:0]
wire out0[7:0]
Filter:
Append Insert Replace
Signals
Time
CLOCK
RESET
in0[7:0]
out0[7:0]
Waves

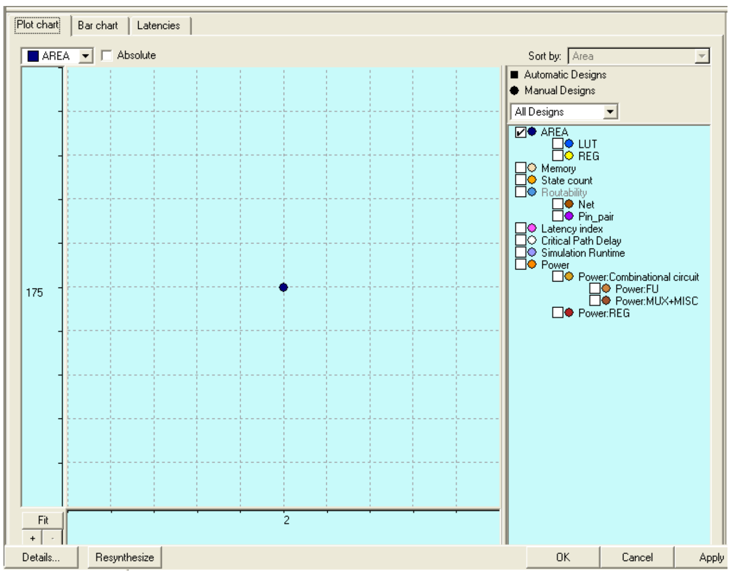
The simulation output should match the result of the SW simulation.

Design Space Exploration

C-Based design allows the generation of different architectures with different area vs. performance constraints without having to modify the actual C code. This is mainly done by modifying the synthesis constraints. E.g. FCNT constraint file or synthesis options.

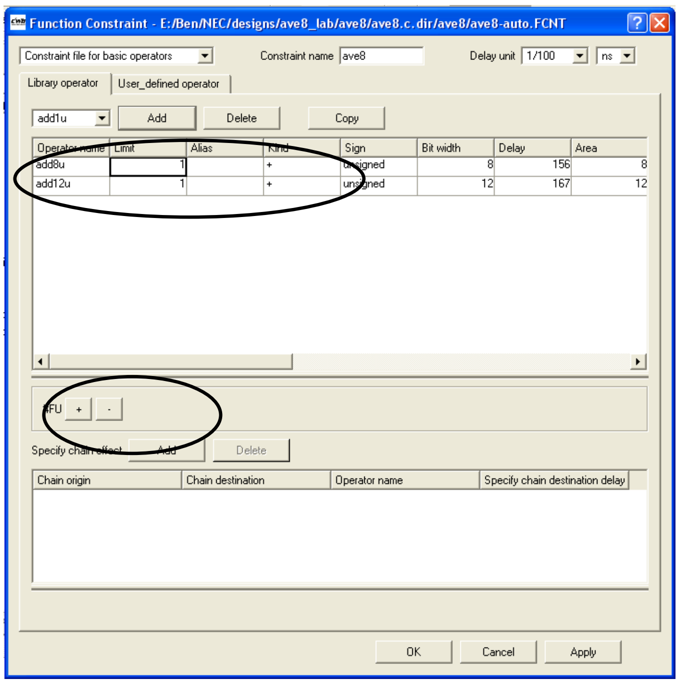
1. Open the synthesis window to observe the synthesis result of the current design in the design space exploration window. Right-click on 'light blue' folder→ Show synthesis results→ Synthesis window





The Y-axis represents the area of the design, while the y-axis can be modified to represent different things. Modify to 'Latency index'

2. Open the ave8-auto.FCNT resource constraint file and reduce the number of FUs to 1


Function Constraint - E:/Ben/NEC/designs/ave8_lab/ave8/ave8.c.dir/ave8/ave8-auto.FCNT

Constraint file for basic operators | Constraint name: ave8 | Delay unit: 1/100 ns

Library operator | User_defined operator

add1u

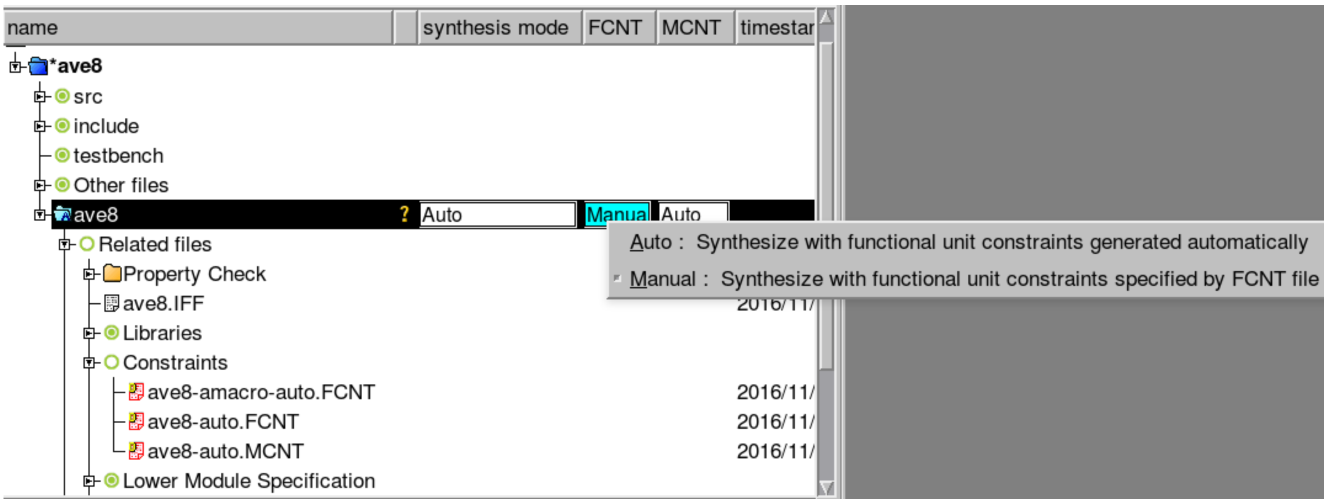
Operator name	Limit	Alias	Kind	Sign	Bit width	Delay	Area
add8u	1		+	unsigned	8	156	8
add12u	1		+	unsigned	12	167	12

FU + -

Specify chain offset

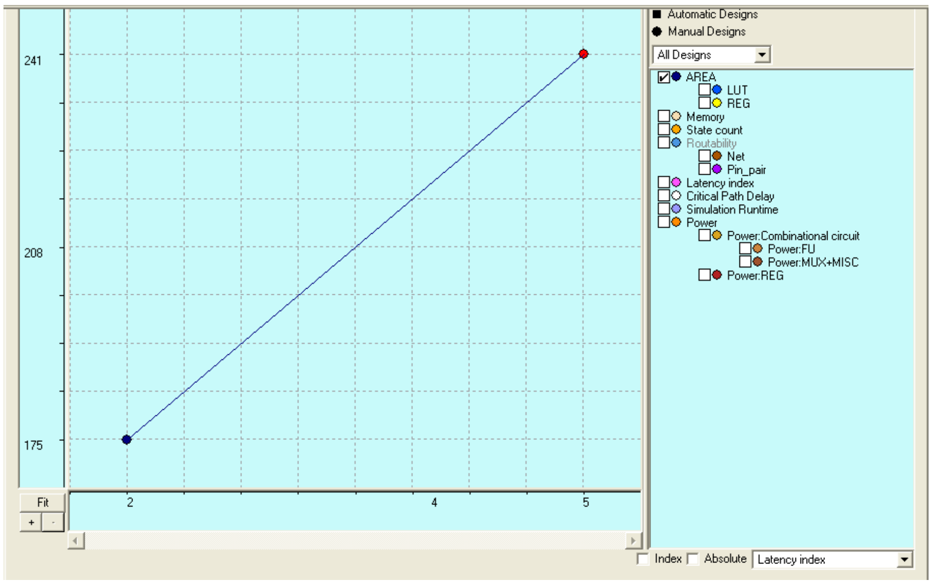
Chain origin	Chain destination	Operator name	Specify chain destination delay

3. Apply the changes
4. Set manual FCNT generation so that CWB does not generate a new FCNT file when re-synthesizing

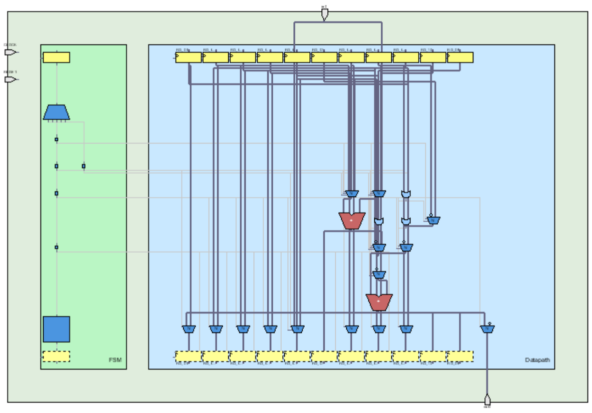


```
name                              synthesis mode   FCNT    MCNT    timestar
*ave8
  src
  include
  testbench
  Other files
  ave8                          ?  Auto            Manual  Auto
    Related files                  Auto :  Synthesize with functional unit constraints generated automatically
      Property Check               Manual :  Synthesize with functional unit constraints specified by FCNT file
      ave8.IFF                                                 2016/11/
      Libraries
      Constraints
        ave8-amacro-auto.FCNT                                  2016/11/
        ave8-auto.FCNT                                         2016/11/
        ave8-auto.MCNT                                         2016/11/
      Lower Module Specification
```

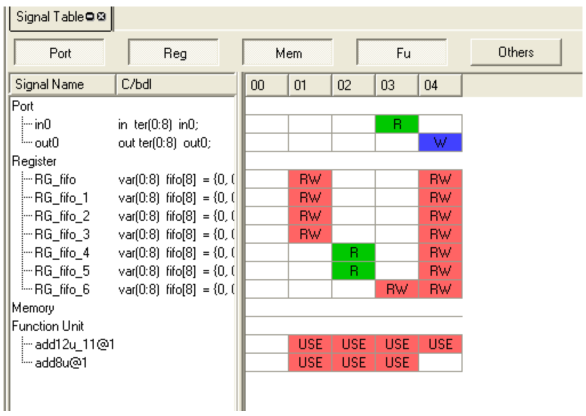
5. Re-synthesize the design. A new design with different Area and latency characteristics is generated and plotted in the synthesis window



Opening the schematic viewer confirms that only 2 FUs have been instantiated. The synthesized results maximizes resource sharing.



Opening the signal table shows that the latency has now increased from 1 cycle to 5



Signal Name	C/bdl	00	01	02	03	04
Port						
in0	in ter(0:8) in0;				R	
out0	out ter(0:8) out0;					W
Register						
RG_fifo	var(0:8) fifo[8] = {0, (		RW			RW
RG_fifo_1	var(0:8) fifo[8] = {0, (		RW			RW
RG_fifo_2	var(0:8) fifo[8] = {0, (		RW			RW
RG_fifo_3	var(0:8) fifo[8] = {0, (		RW			RW
RG_fifo_4	var(0:8) fifo[8] = {0, (			R		RW
RG_fifo_5	var(0:8) fifo[8] = {0, (			R		RW
RG_fifo_6	var(0:8) fifo[8] = {0, (				RW	RW
Memory						
Function Unit						
add12u_11@1			USE	USE	USE	USE
add8u@1			USE	USE	USE	

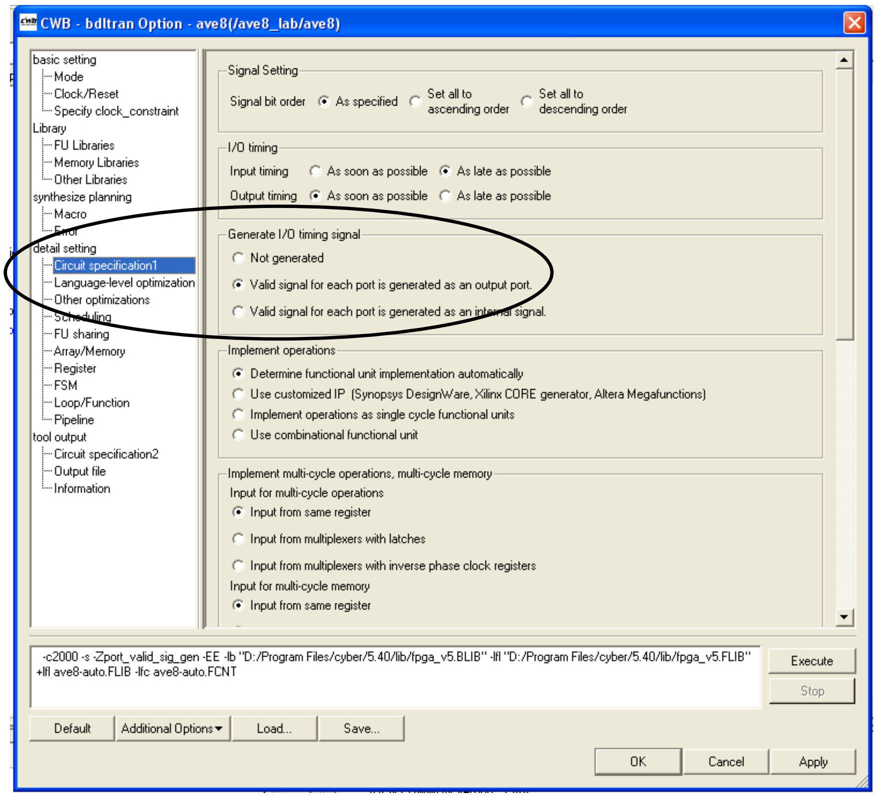
Re-verify the new design. No new simulation scenarios are needed. CWB will automatically re-generate the RTL, testbenches and cycle-accurate simulation models.

Transactional Level Simulation

The input and output files have the extension .clv because each data in them represent an input/output for that specific port each cycle. In this new design an input should only be read every 4 clock cycles. Two options are available:

- Modify the in0.clv file inserting 0 0 0 every data
- Using transactional level simulation. In this case the synthesizer will generate a valid signal for each port and set the signal when a new input is needed. The automatically generated testbench will monitor the values of these valid signals and only apply a new test vector when the valid signal for that particular port is set.

1. Copy the .clv files and rename them as .tlv (e.g in0.tlv and out0.tlv)
2. Open the synthesis options dialog window→ detail settings→ Circuit specifications1→ set 'Valid signal for each port is generated as an output port.



Signal Setting

Signal bit order [✓ As specified] Set all to ascending order Set all to descending order

I/O timing

Input timing As soon as possible [✓ As late as possible]

Output timing [✓ As soon as possible] As late as possible

Generate I/O timing signal

Not generated

[✓ Valid signal for each port is generated as an output port.]

Valid signal for each port is generated as an internal signal.

Implement operations

[✓ Determine functional unit implementation automatically]

Use customized IP (Synopsys DesignWare, Xilinx CORE generator, Altera Megafunctions)

Implement operations as single cycle functional units

Use combinational functional unit

Implement multi-cycle operations, multi-cycle memory

Input for multi-cycle operations

[✓ Input from same register]

Input from multiplexers with latches

Input from multiplexers with inverse phase clock registers

Input for multi-cycle memory

[✓ Input from same register]

```
 -c2000 -s -Zport_valid_sig_gen -EE -lb "D:/Program Files/cyber/5.40/lib/fpga_v5.BLIB" -lfl "D:/Program Files/cyber/5.40/lib/fpga_v5.FLIB"
+lfl ave8-auto.FLIB -lfc ave8-auto.FCNT
```

3. Re-synthesize the design and re-run an RTL simulation.

Appendix I – Calling CWB from command line

1.) Copy the .c file and the .FLIB and .BLIB file to the same directory. You should basically have:

```
ave8.C
cycloneV-7.FLIB
cyclone-7.BLIB
```

2.) Parse the behavioral description. If no errors are found a ave8.IFF is generated

```
% cpars ave8.c
```

3.) Synthesize the parsed file. If there are no issues, then an ave8_E.IFF file will be generated and all of the report files:

```
%bdltran -c2000 -s ave8.IFF -lfl cycloneV-7.FLIB -lb cycloneV-7.BLIB
```

- -c2000 is the clock period in 1/100ns units, hence, 2000=20ns = 50Mhz
- -s means automatic scheduling mode. HLS takes care of everything

If you want to synthesize the design using a manually edited Functional Unit Constraint file (FCNT), then add these options:

```
%bdltran -c2000 -s ave8.IFF -lfl cycloneV-7.FLIB -lb cycloneV-7.BLIB -Zresource_fcnt=USE -lfc ave8-auto.FCNT
```

4.) Generate Verilog or VHDL:

```
%veriloggen ave8_E.IFF
%vhdlgen ave8_E.IFF
```

The output should be either ave8_E.v or ave8_E.vhd which is the final HLS output.

[END]