



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 a simulation using the same input vectors as for the SW verification for the RTL sim

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
```



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

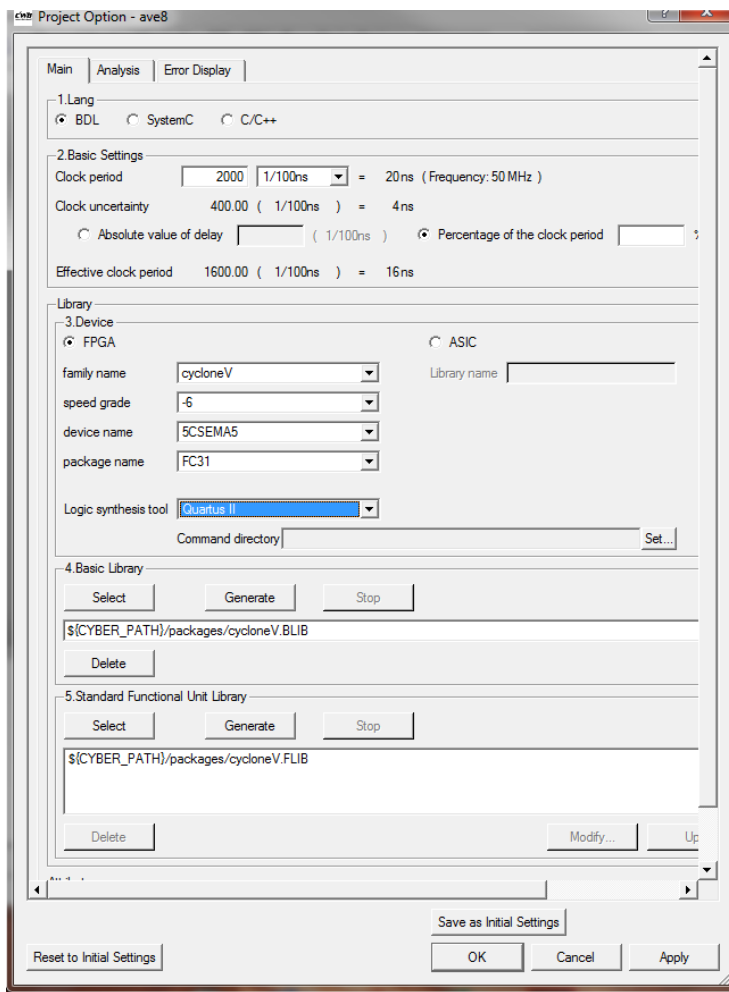
```
% cwb &
```

This should launch CyberWorkBench (CWB).

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:



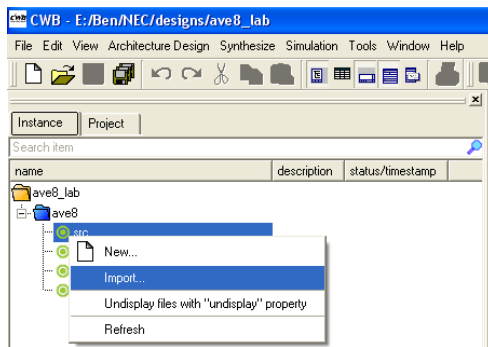
The main items that need to be specified are:

- Clock period ($2000 \times 1/100\text{ns} = 20\text{ns} \Leftrightarrow 50\text{ MHz}$)
- 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)

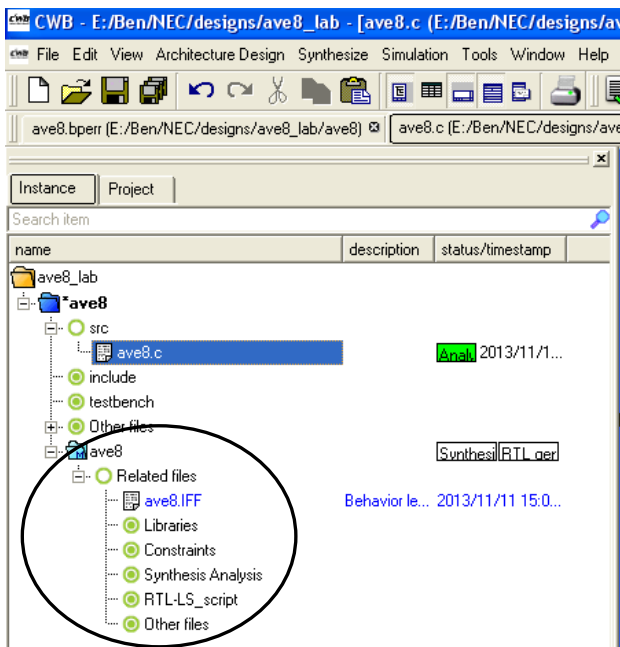
4. Click on Apply to generate the empty project

Editing or Inserting a ANSI-C description for synthesis

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

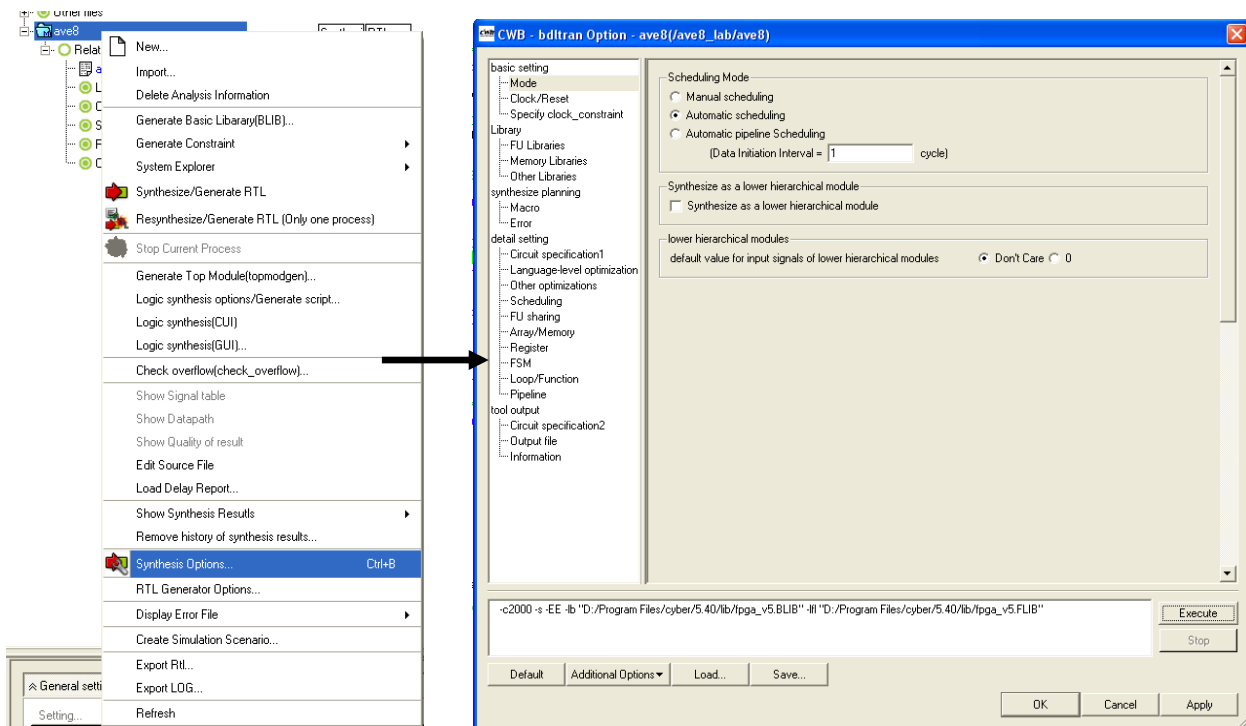


2. Parse source code to check if there are any syntax errors. The console window will indicate if any errors have occurred and where. A 'light-blue' folder with an .IFF file will be generated if the parsing was successful



Setting up Synthesis mode and other constraints

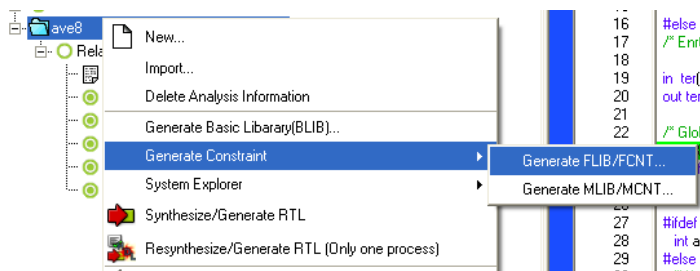
1. Right click on 'light-blue' folder → Synthesis options



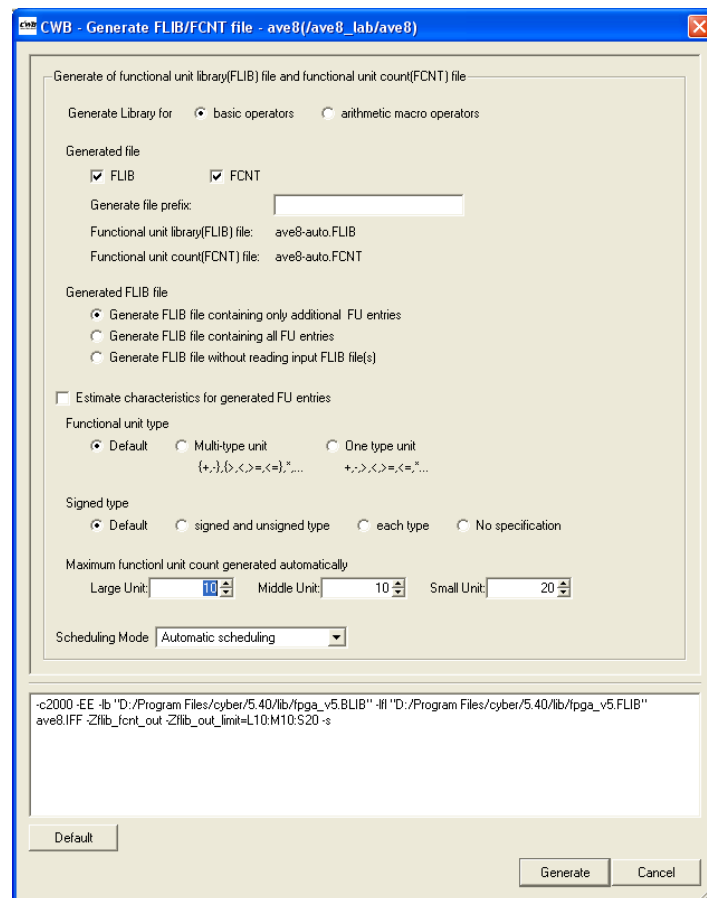
2. Set the scheduling mode to 'Automatic scheduling'. This implies that CWB will automatically time the C description at the scheduling phase. Manual scheduling implies that the user will manually time the description using the '\$' sign as a clock boundary.
3. Other synthesis options that control the 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. Right-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.



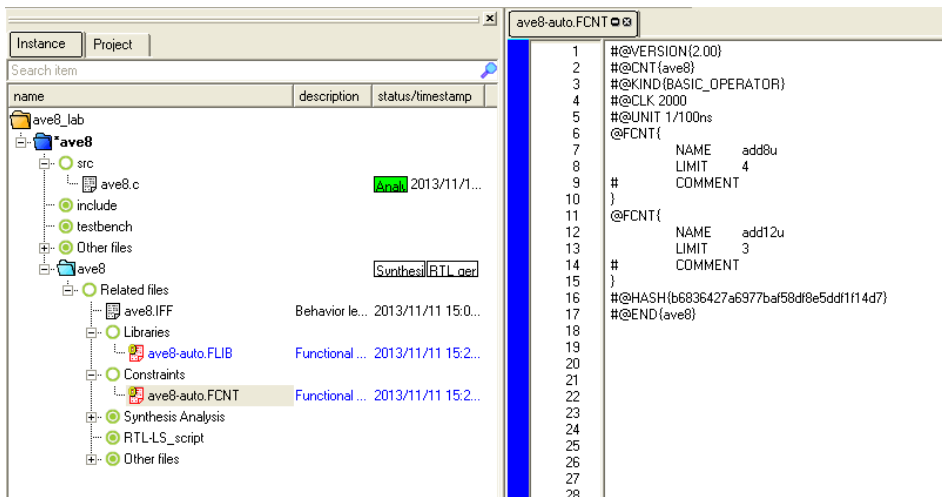
CWB considers FUs of different bws as Large (≤ 8 bits), middle ($8\text{bits} \leq 4$ bits) and small (> 4 bits). By default it limits the number of FUs allocated as indicated in the dialog window to 5:10:20

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.

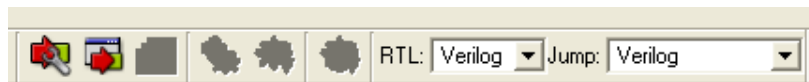
Review the contents of the files by ‘right-click → open by internal editor



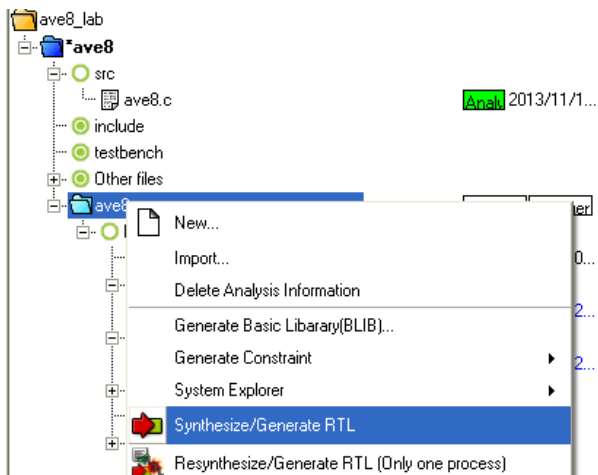
High-Level Synthesis

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

1. Select Verilog at the toolbar



2. Right-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-generate these if a different device is targeted. Also, the reported data should be confirmed performing a logic synthesis and a simulation.

Cyber RTL Synthesis Report (module name : ave8)

Summary

Basic Library Name: C:\BSTD8LIB (BLIB Version 2.00)
Clock period: 20ns

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

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

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

Important Information

False path	Multi-cycle path	False loop/Combinational loop	Latch (bit)
Unchecked	Unchecked	0	8

Latency

Loop	Type	Latency	Latency index	State No.	Folding loop	Folding stages	Total folding states	Hazard	User operator	Sub loops	Line
LL	S	Reset = L3	2	0.1	-	-	-	-	-	-	-
LL	S	1 * N1 (N1 >= 1)	1	1	-	-	-	-	-	-	../ave8.c:31

• Loop loop name (L,B)

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.

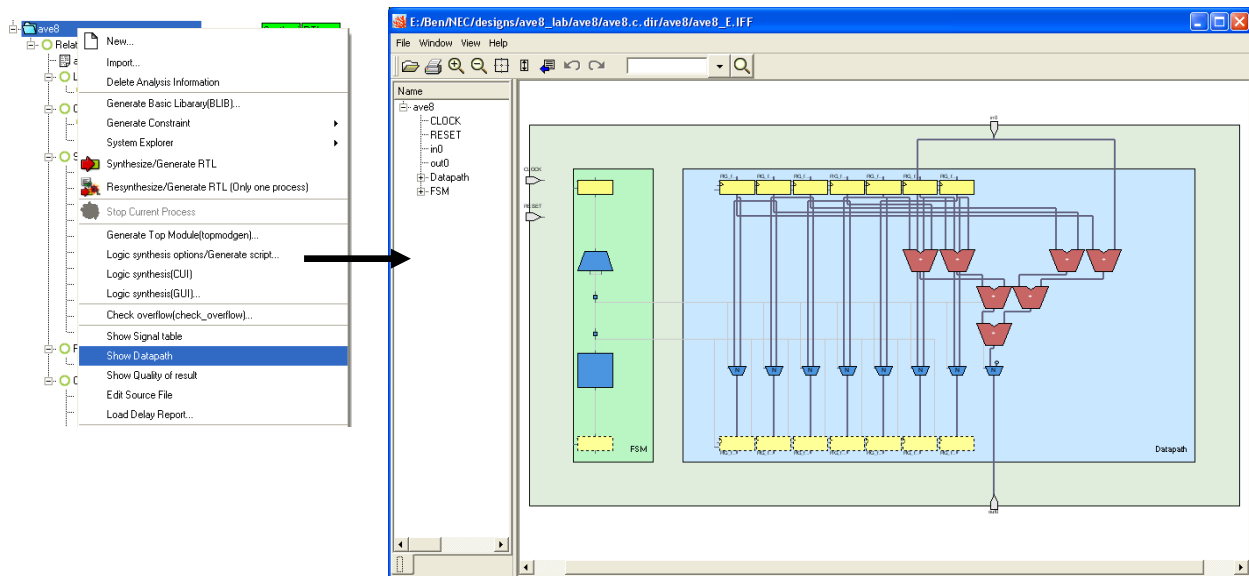
3. Open ave8.err file and review the warning, information and tips messages

Quality of Result(2013/11/11 15:35:08) ave8-auto.FCNT bdltran Error Viewer		
<div> <div>Error</div> <div>Warning</div> <div>Information</div> <div>Tips</div> </div>		
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)

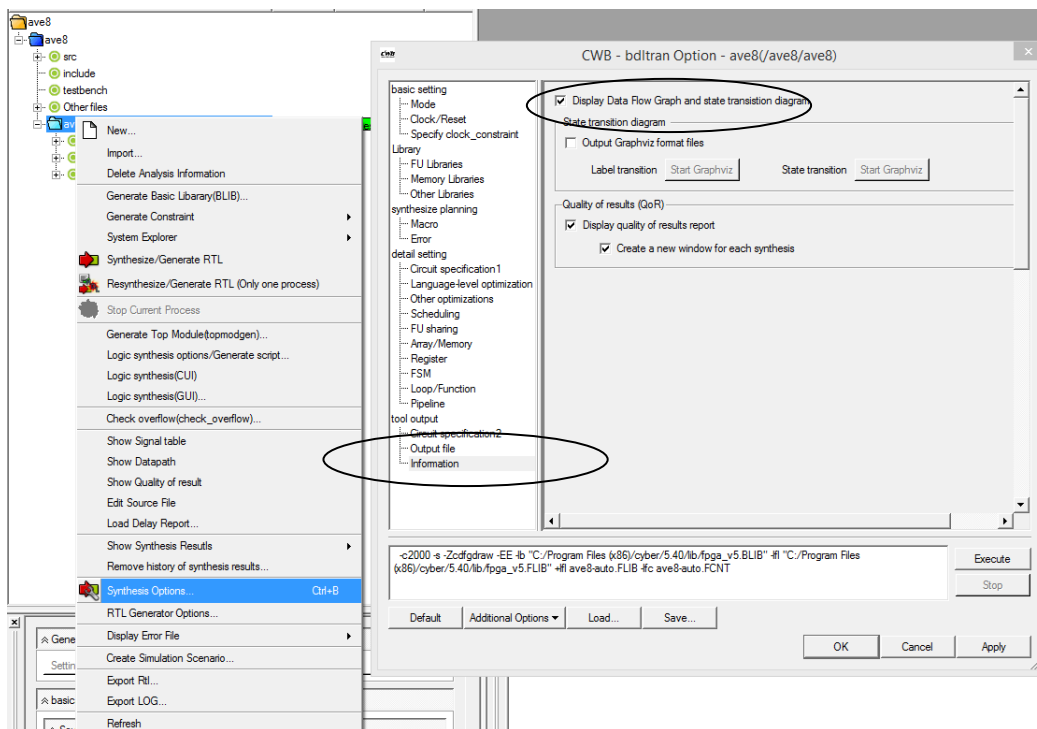
2. Signal Table: Right-click 'light-blue' folder→ Show Signal Table

Signal Name	C/bdl	00	01
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

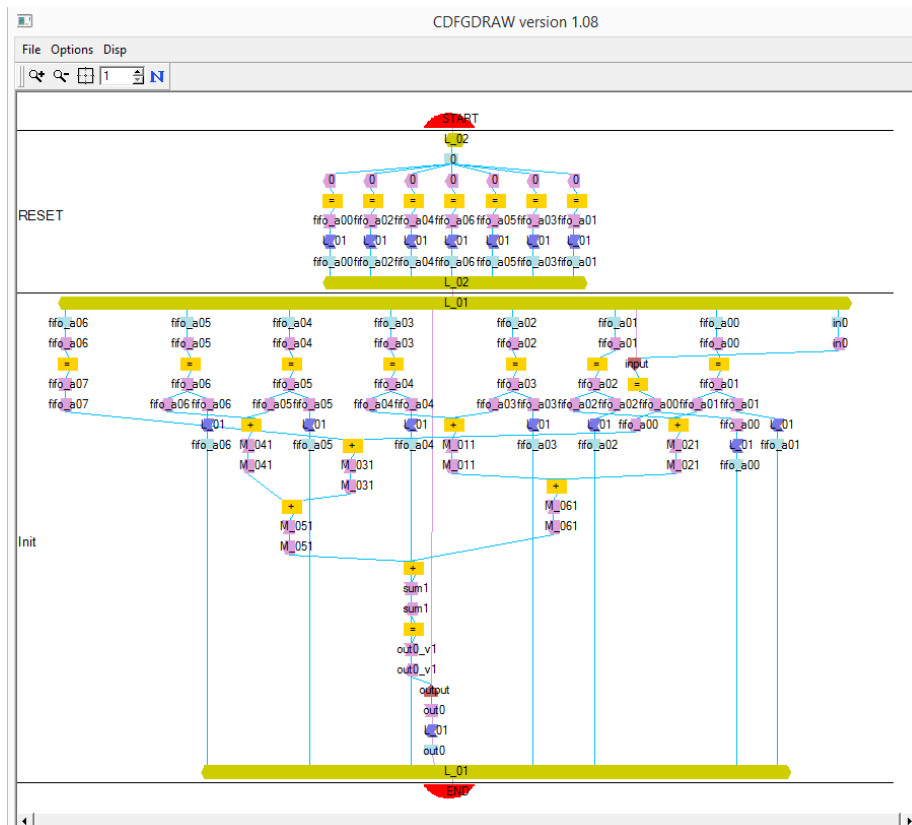
The Signal table shows the timing diagram of the synthesis. When inputs and outputs are being read or written and when registers accessed. State 00 = reset state and state 01 = stating doing the computation.

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



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



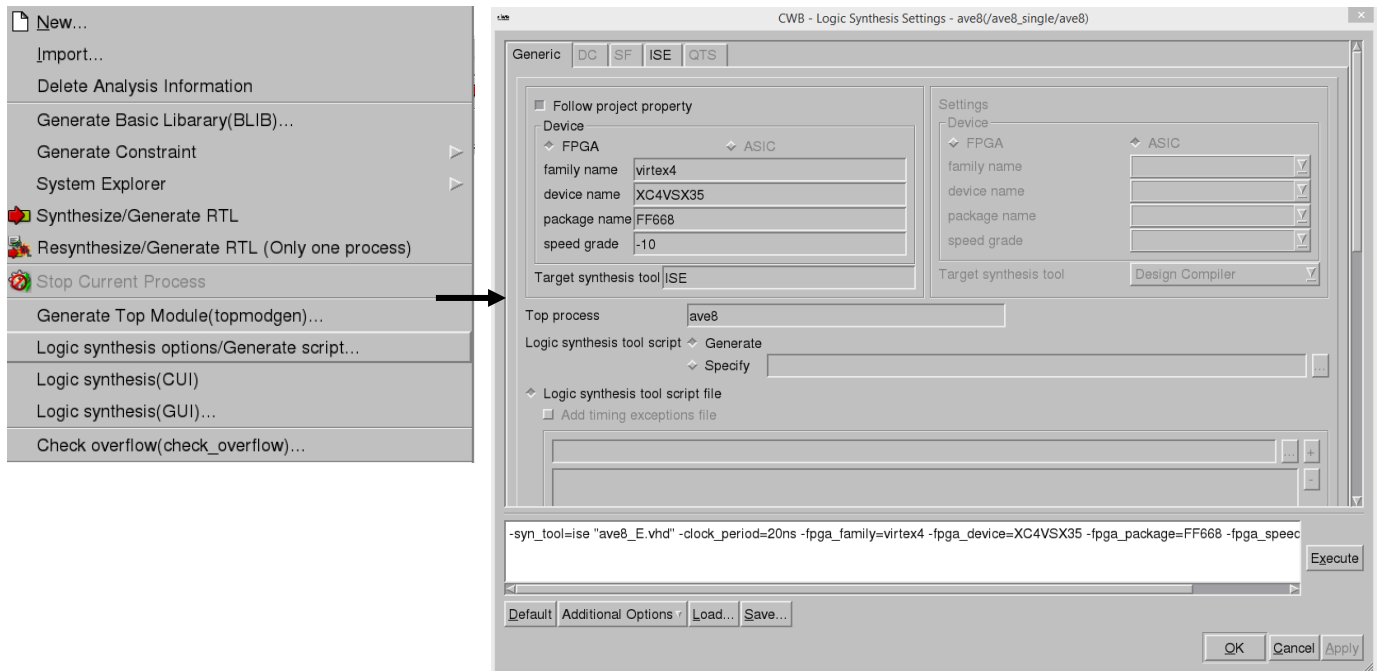
Logic Synthesis

It is now possible to call Altera's or 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).

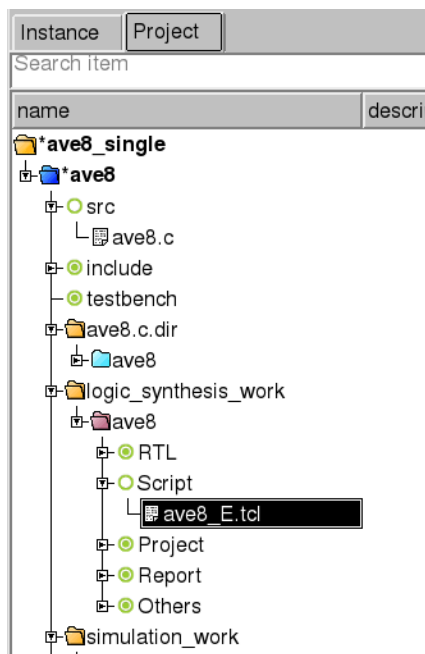
NOTE: Make sure that CWB has the path to Quartus setup in CWB's GUI toolbar: Tools→Options→Tool Path

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

This creates

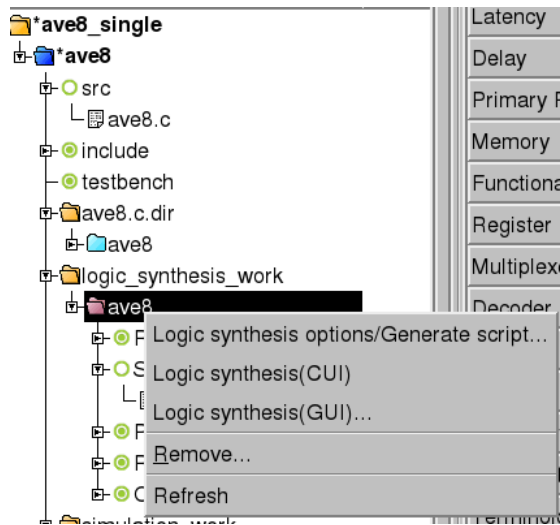


This will create a .tcl script file for Quartus II. The newly generated red folder includes all the information regarding the logic synthesis



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



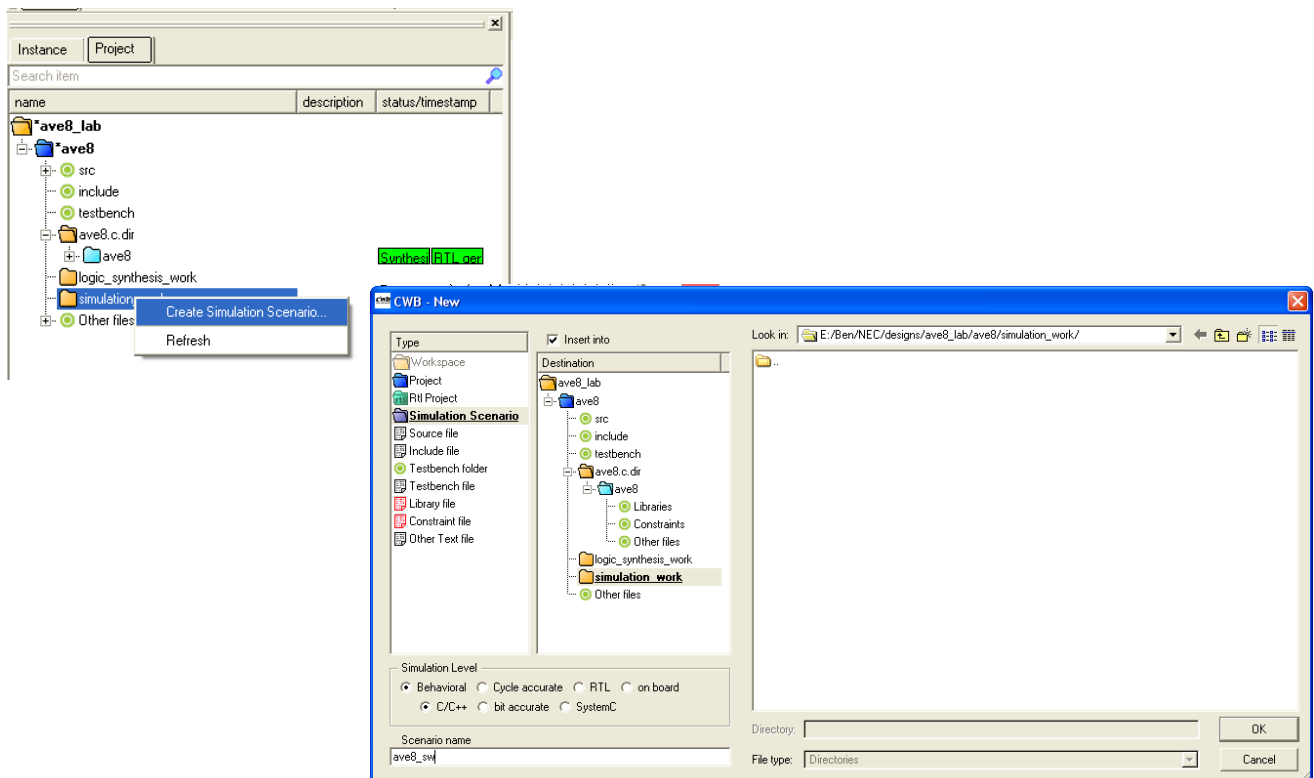
Once Quartus finishes, the logic synthesis results can be observed at the report folder.

Design Verification: Software Simulation

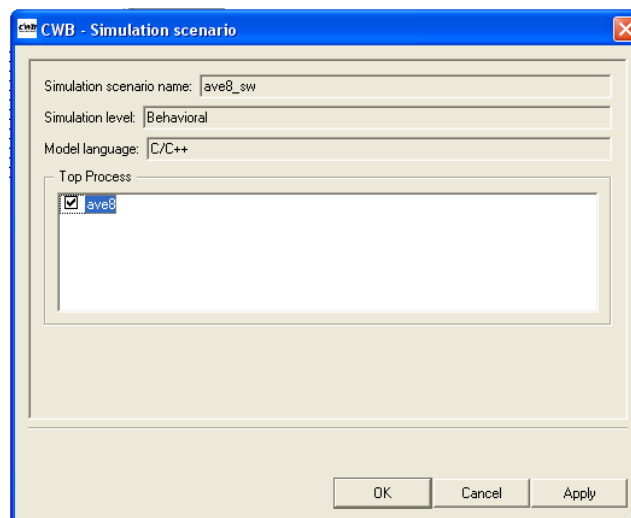
There are different levels of simulations that can be used to verify the design:

- 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

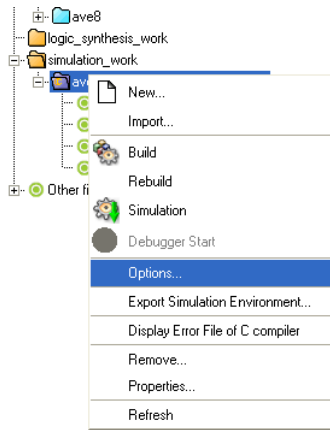
1. The first step is to do a normal SW simulation from within CWB. Change to the 'Project' tab → Create Simulation Scenario.



2. Select "Behavioral" and C/C++ and call the scenario name 'ave8_sw'
3. Select the top process (check the check box)

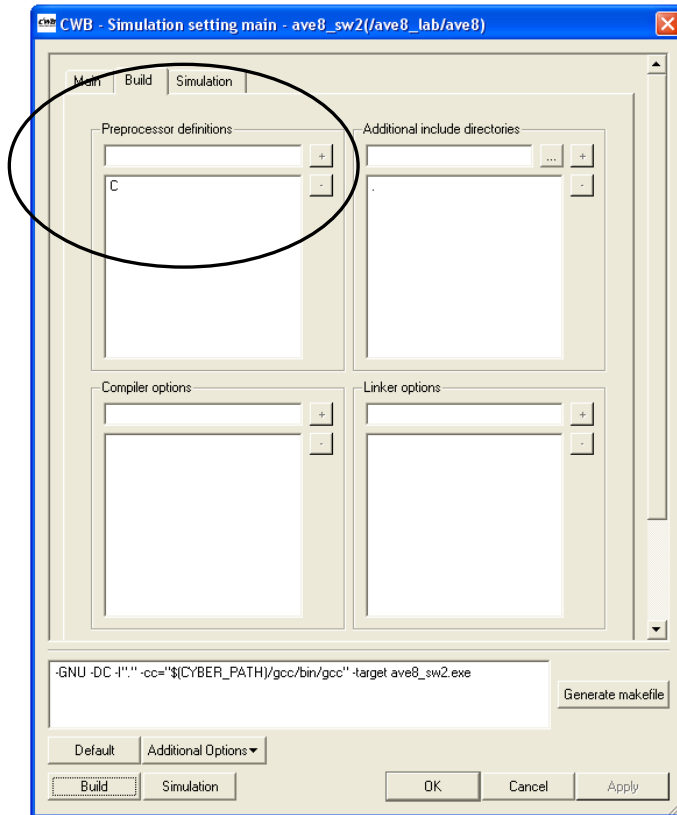


4. Right-click on 'ave8_sw' folder → options → Build tab → Add C to the preprocessor definitions

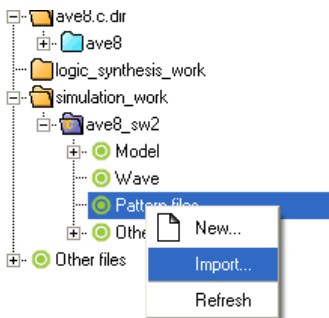


Synthesi RTL gen

Model Compile



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



Synthesi RTL gen

Compile

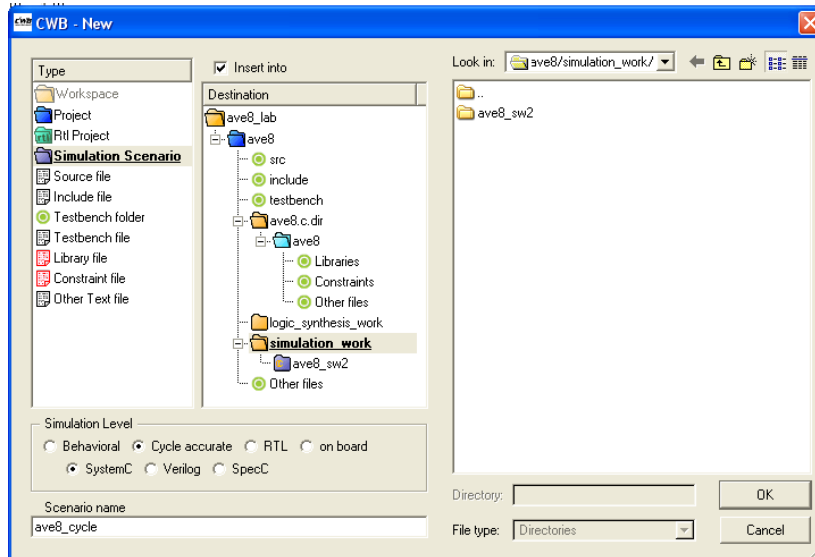
6. Build the binary (.exe file) and simulate it

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.

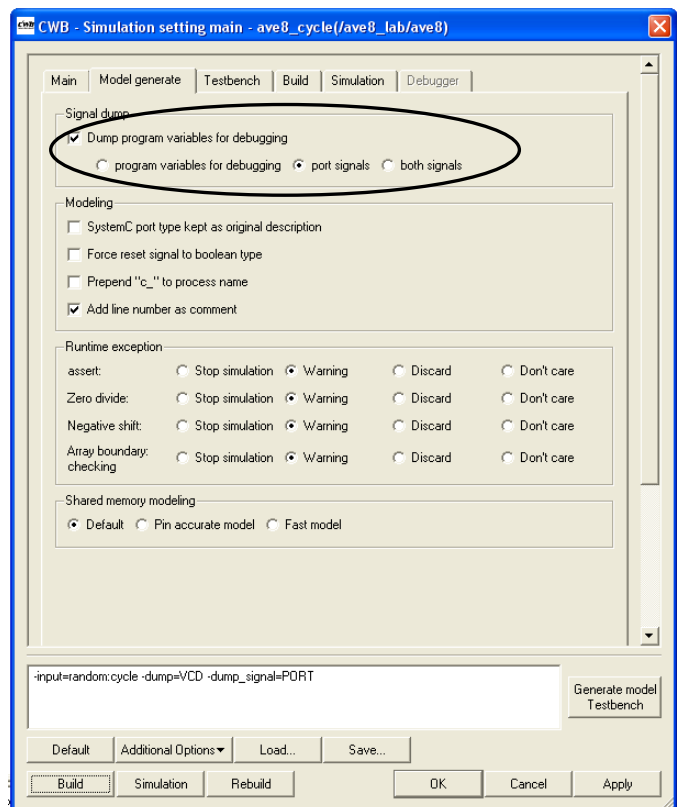
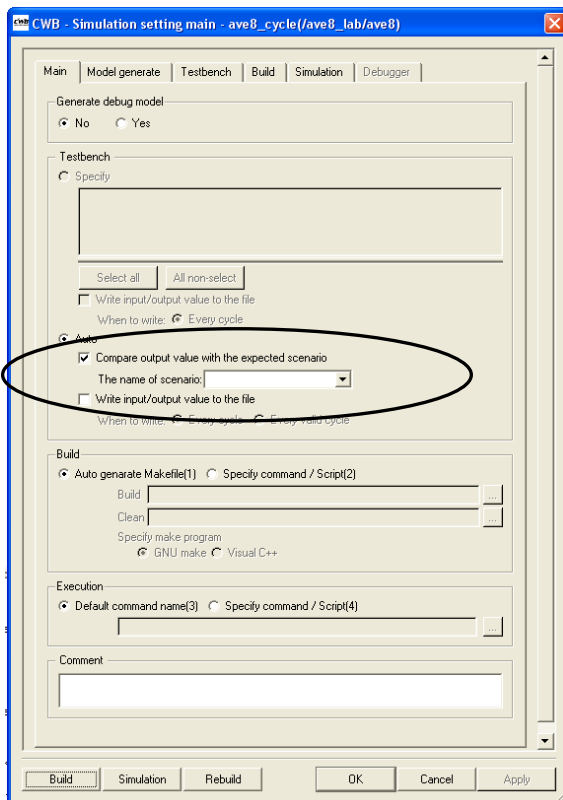
Cycle-Accurate simulation

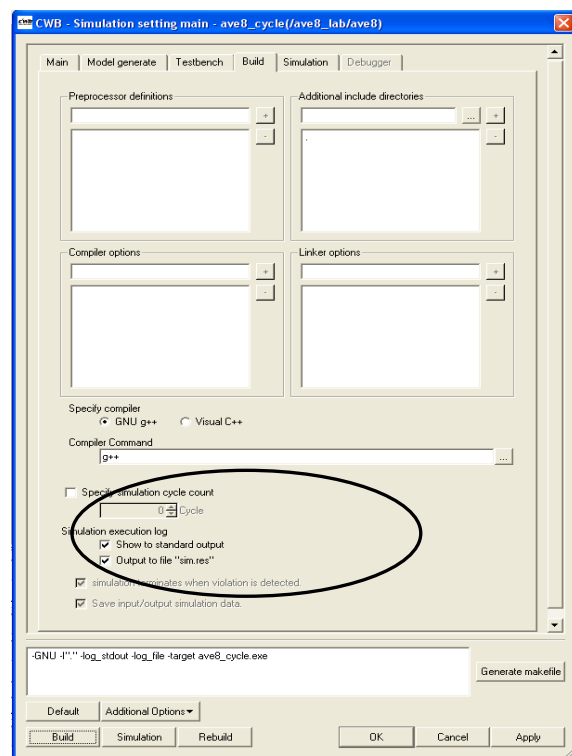
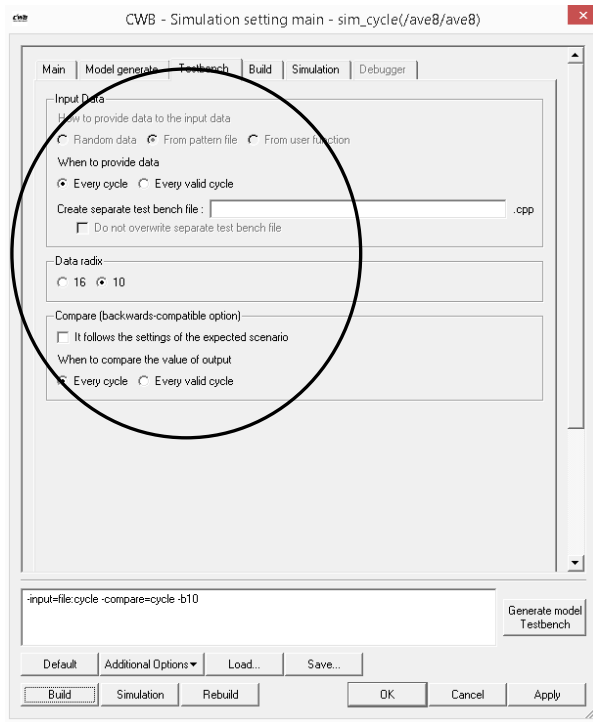
In order to verify the correctness of the timing and also the accurate performance of the synthesized designs two options are available: (1) simulate the generated RTL code or (2) create a cycle-accurate model. This second option is normally preferred as the simulation is faster than RTL.

1. Create a new simulation scenario (right-click on the 'simulation work' folder. Select 'Cycle accurate' options and enter scenario name (ave8_cycle)

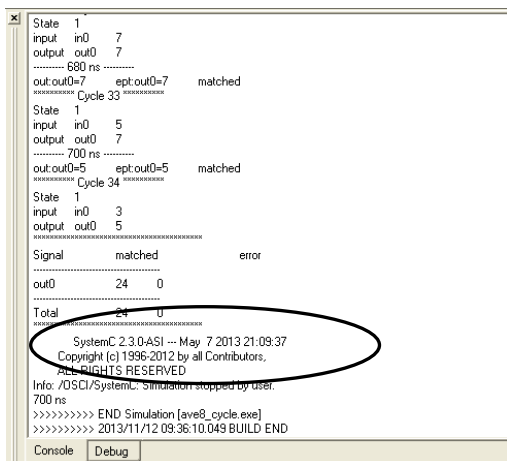


2. Specify testbench options (right click → options)

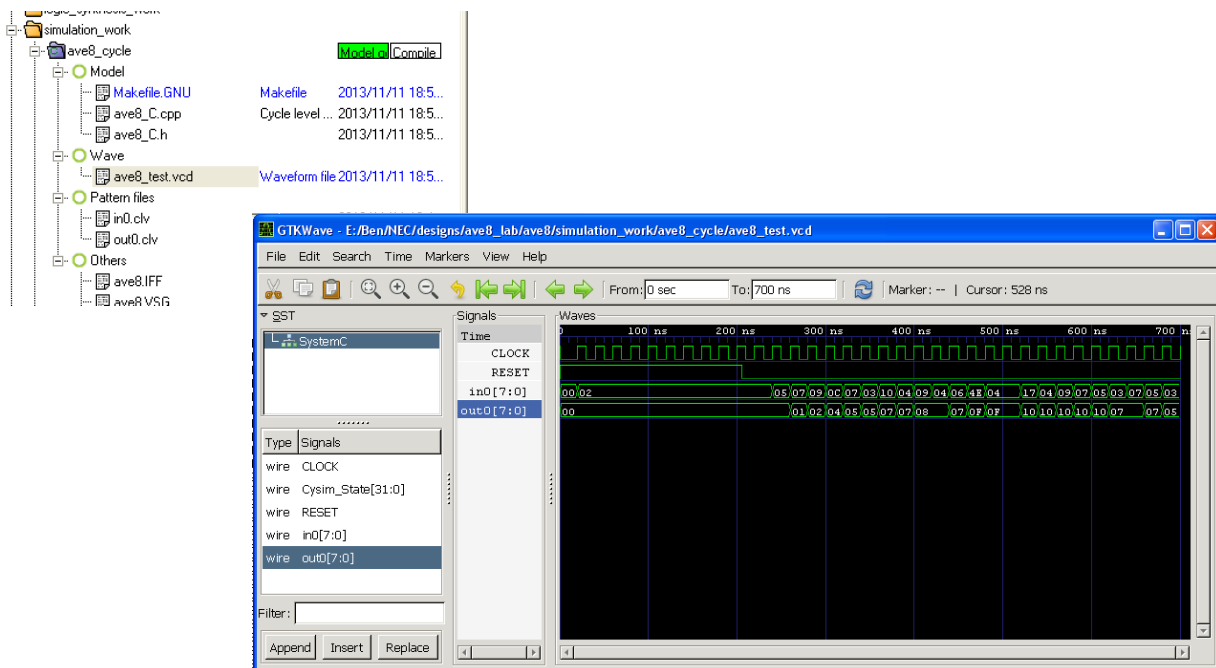




- Copy the indata.txt and outdata.txt file into the new simulation folder and rename as in0.clv and out0.clv. CWB needs an input and output file with the same name as the ports as input stimuli.
- Build and simulate the model. The console window will display if the outputs from the cycle-accurate simulation match the outputs from the SW simulation



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

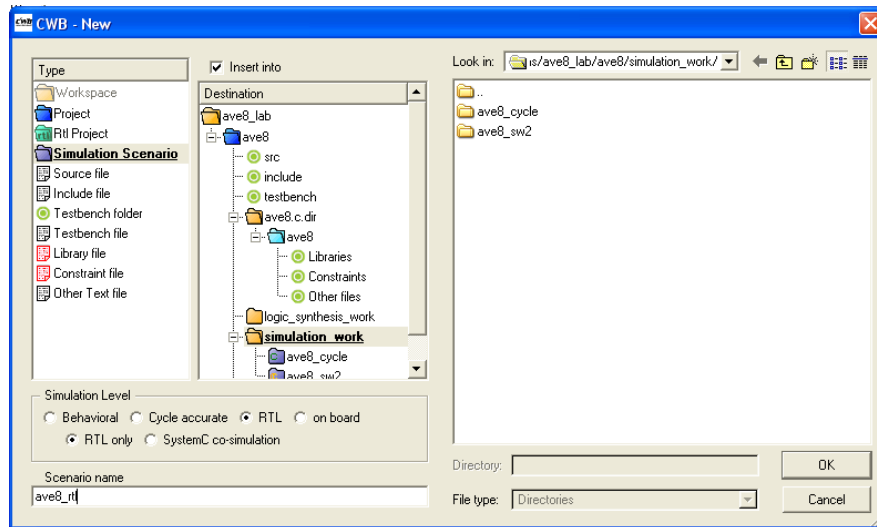


The simulation output should match the result of the SW simulation.

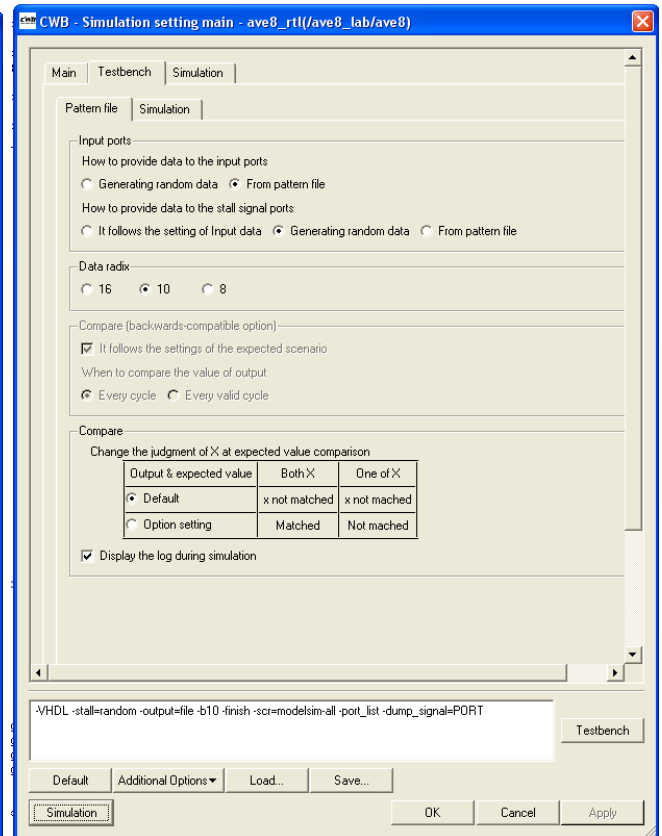
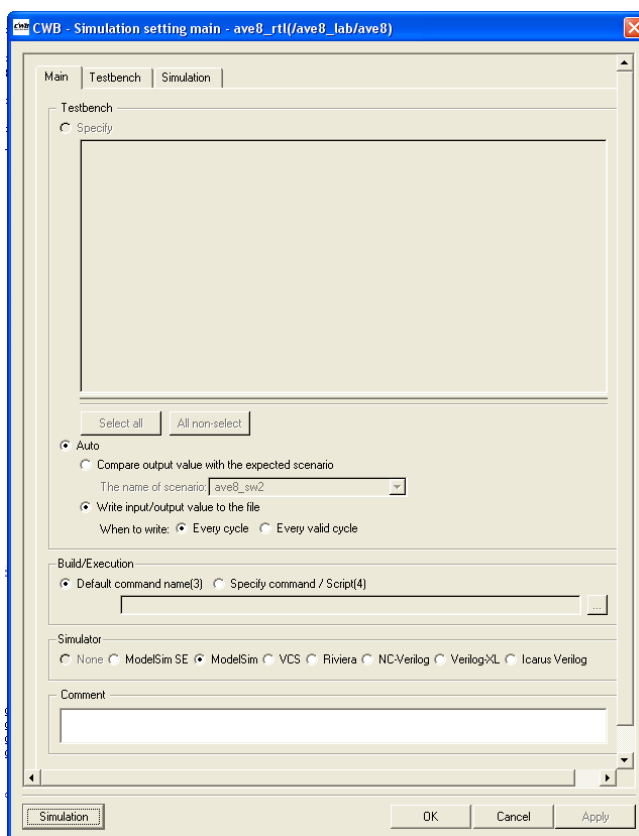
RTL simulation

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



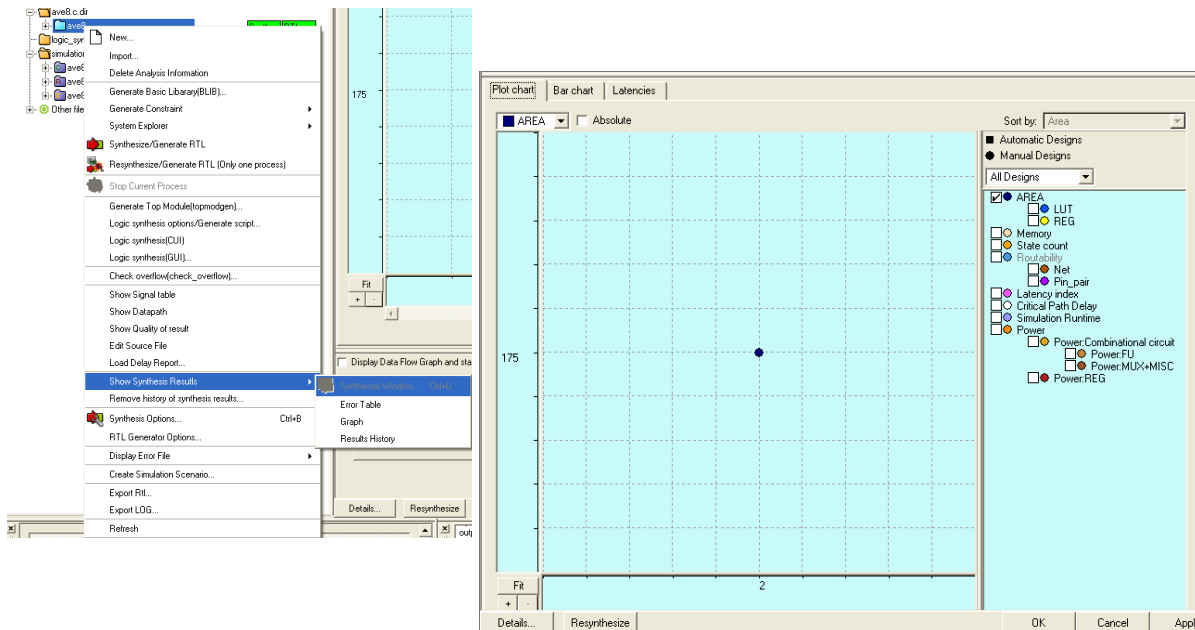
2. Generate the testbench : Select RTL simulator and input stimuli every cycle, Input data from file and radix as decimal. Click OK and run the simulation



Design Space Exploration

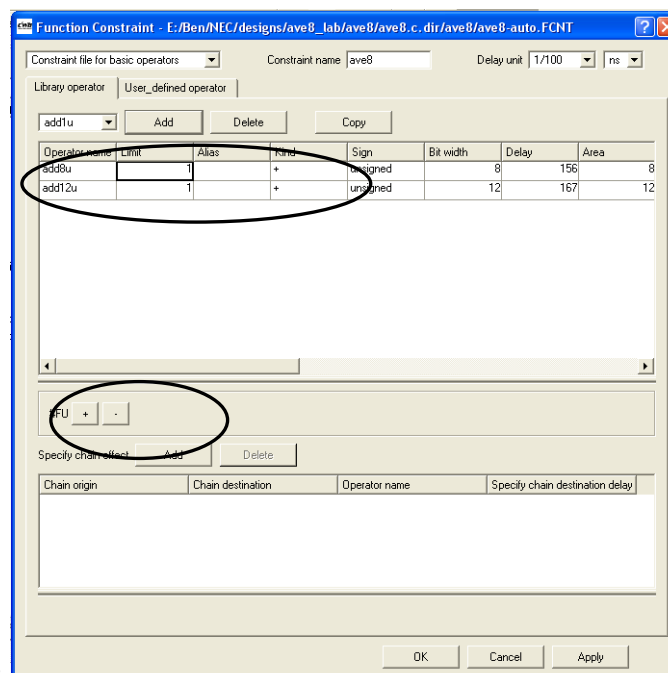
C-Based design allows the generation of different architectures with different area vs. performance constraints without having to modify the actual C code. This is mainly done by modifying the synthesis constraints. E.g. FCNT constraint file or synthesis options.

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



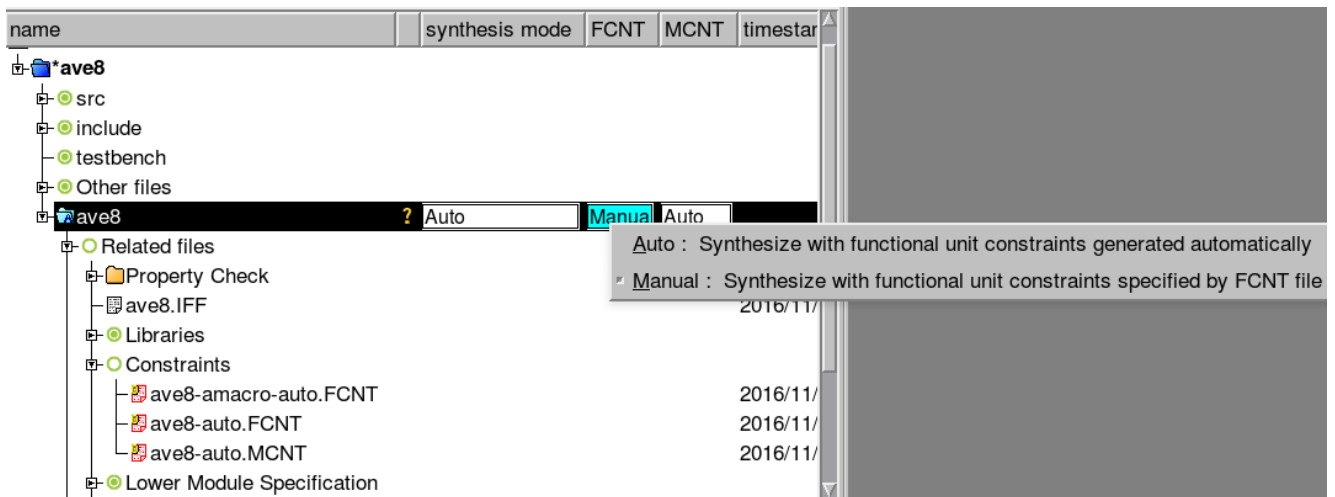
The Y-axis represents the area of the design, while the X-axis can be modified to represent different things. Modify to 'Latency index'

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

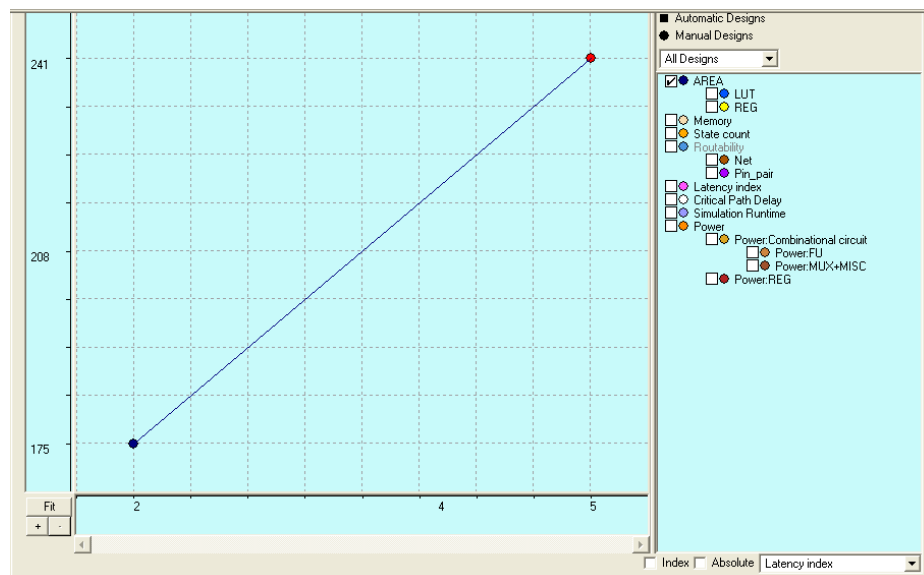


3. Apply the changes

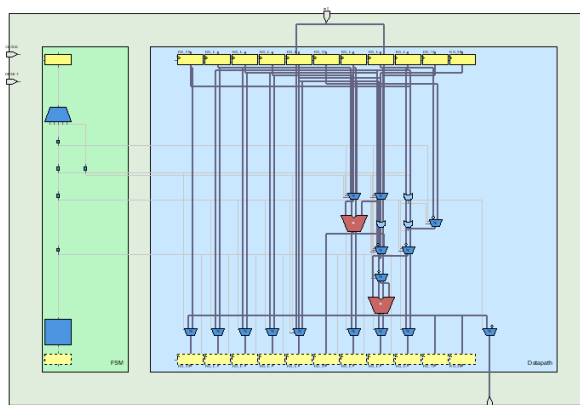
- Set manual FCNT generation so that CWB does not generate a new FCNT file when re-synthesizing



- 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 Table						
		Port	Reg	Mem	Fu	Others
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, 0};	RW				RW
RG_fifo_1	var(0:8) fifo[8] = {0, 0};	RW				RW
RG_fifo_2	var(0:8) fifo[8] = {0, 0};	RW				RW
RG_fifo_3	var(0:8) fifo[8] = {0, 0};	RW				RW
RG_fifo_4	var(0:8) fifo[8] = {0, 0};			R		RW
RG_fifo_5	var(0:8) fifo[8] = {0, 0};			R		RW
RG_fifo_6	var(0:8) fifo[8] = {0, 0};				RW	RW
Memory						
Function Unit						
add12u_11@1		USE	USE	USE	USE	
add8u@1		USE	USE	USE		

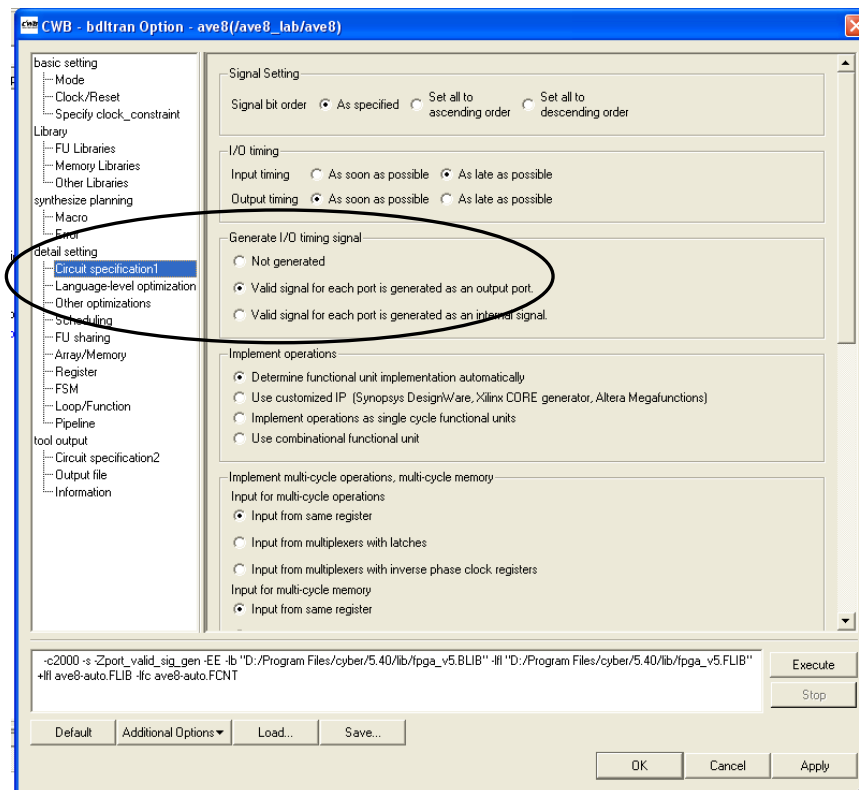
Re-verify the new design. No new simulation scenarios are needed. CWB will automatically re-generate the RTL, testbenches and cycle-accurate simulation models.

Transactional Level Simulation

The input and output files has 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 be 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 testvector 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.



3. Re-synthesize the design and re-run a cycle-accurate simulation.

[END]