



**EEDG/CE 6370**  
**Design and Analysis of Reconfigurable Systems**  
**Homework 7**  
**High-Level Synthesis Optimizations**

### 1. Laboratory Objectives

- Learn how to control the synthesis result of a commercial HLS tool through different synthesis options like synthesis directives to control how to synthesize:
  - Arrays, loops and functions

### 2. Summary

- In this lab you will learn to specify HLS synthesis directives to generate hardware circuits with different area vs. performance trade offs.

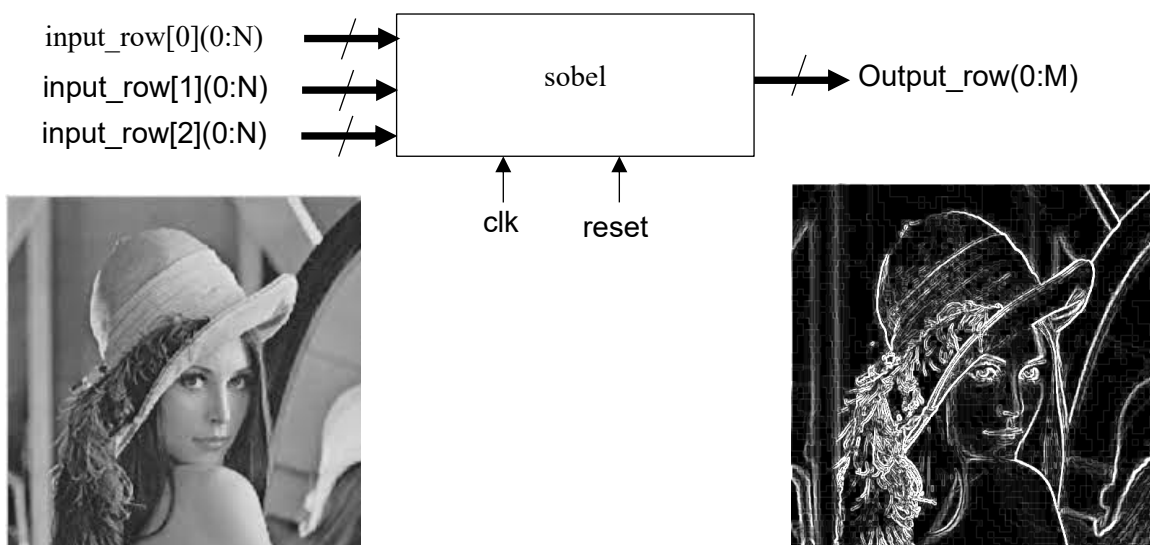
### 3. Tool Requirement

- NEC CyberWorkbench
- gcc (included in CyberWorkBench)

### 4. Sobel Overview

A sobel filter is used in image processing mainly to detect edges in images. The filter convolves the image with a small, separable, and integer-valued filter (typically 3x3) in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

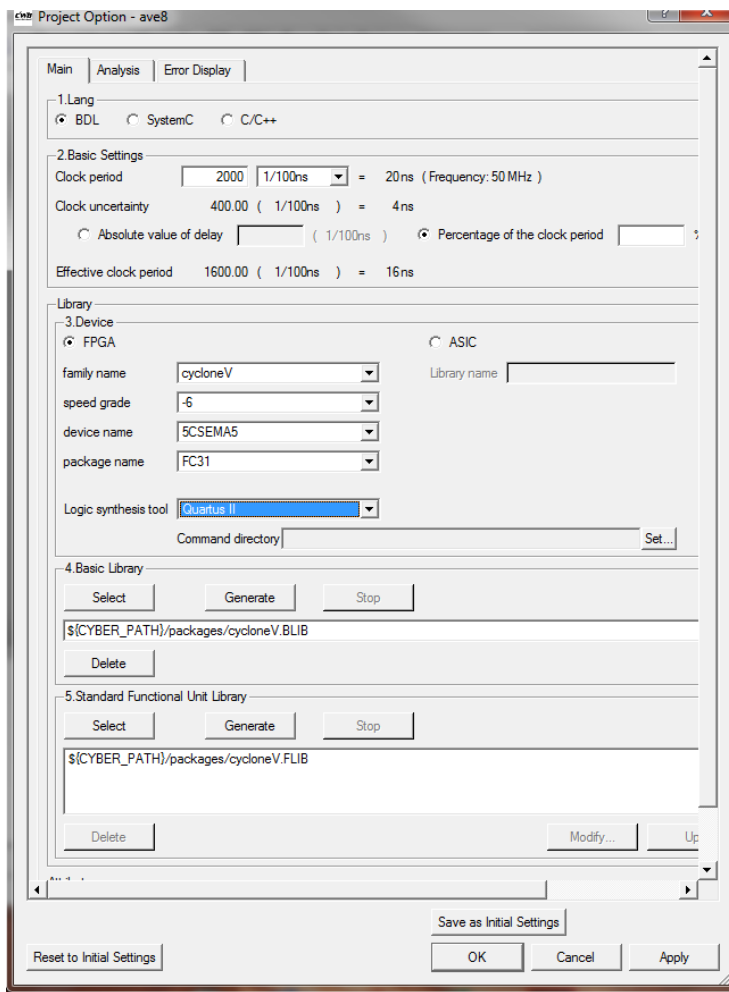
The figure below shows the structure of the sobel filter including their inputs and outputs.



### 5. CWB 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 sobel (File→New)
3. Set the project parameters as shown in the figure:



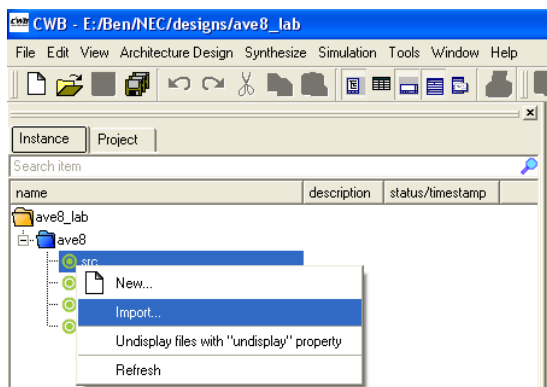
The main items that need to be specified are:

- 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) cycloneV.BLIB
  - FLIB (Functional Unit Libirary) cycloneV.FLIB

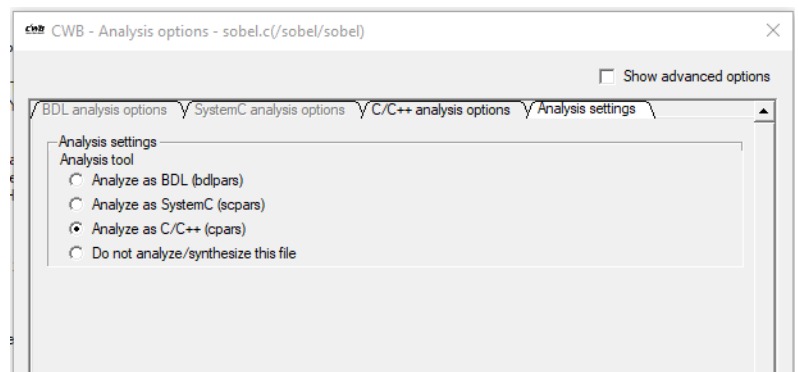
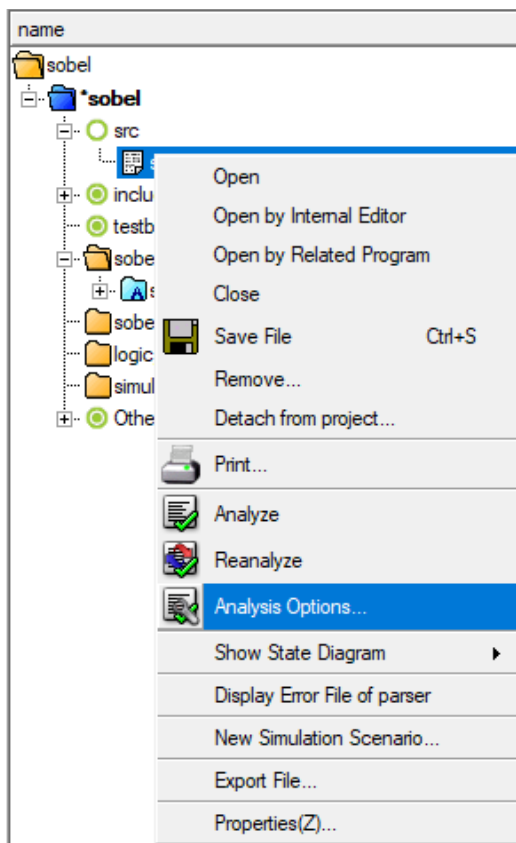
4. Click on Apply to generate the empty project

## Editing or Inserting an ANSI-C description for synthesis

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



2. Specify the parser to use (just in case to ANSI-C parser). Right click on sobel.c → analysis options → Analysis settings → Analyze as C/C++ (cpars)



3. 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



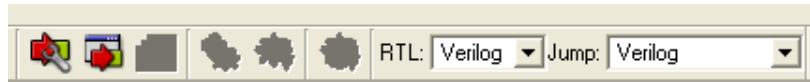
frequency.

- Click on 'Apply' and OK to accept the changes.

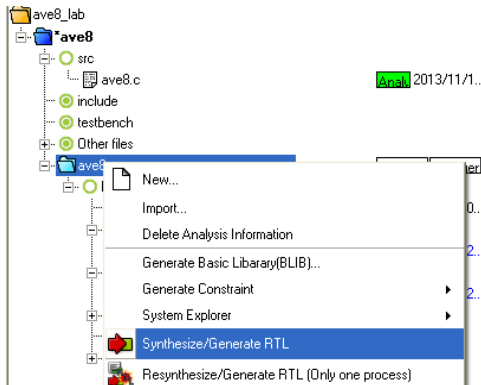
## High-Level Synthesis

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

- Select Verilog or VHDL at the toolbar.

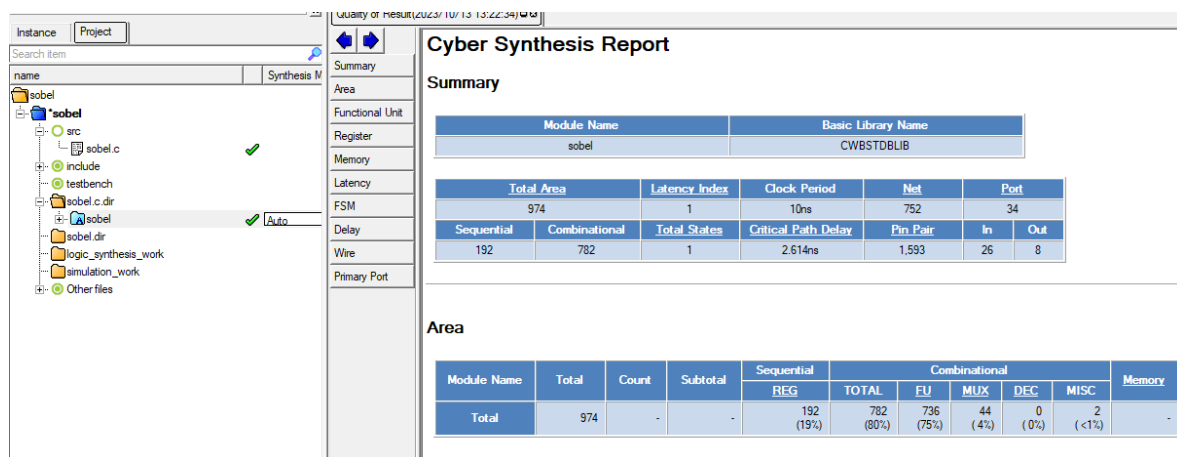


- Right-click on 'light blue folder → Synthesize/Generate RTL (or toolbar)



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 Synthesis Report**

**Summary**

Module Name		Basic Library Name	
sobel		CWBSTD8LIB	

Total Area		Latency Index	Clock Period	Net	Port	
974		1	10ns	752	34	
Sequential	Combinational	Total States	Critical Path Delay	Pin Pair	In	Out
192	782	1	2.614ns	1.593	26	8

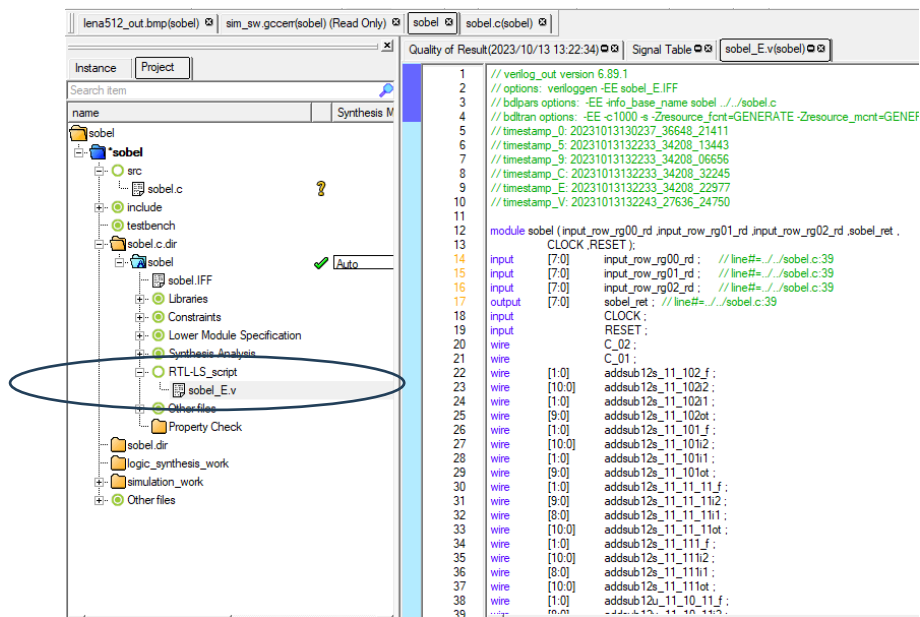
**Area**

Module Name	Total	Count	Subtotal	Sequential	Combinational					Memory
				REG	TOTAL	EU	MUX	DEC	MISC	
Total	974	-	-	192 (19%)	782 (80%)	736 (75%)	44 (4%)	0 (0%)	2 (<1%)	-

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.

Open the generated RTL code to check that the hardware module has 3 inputs and one output.

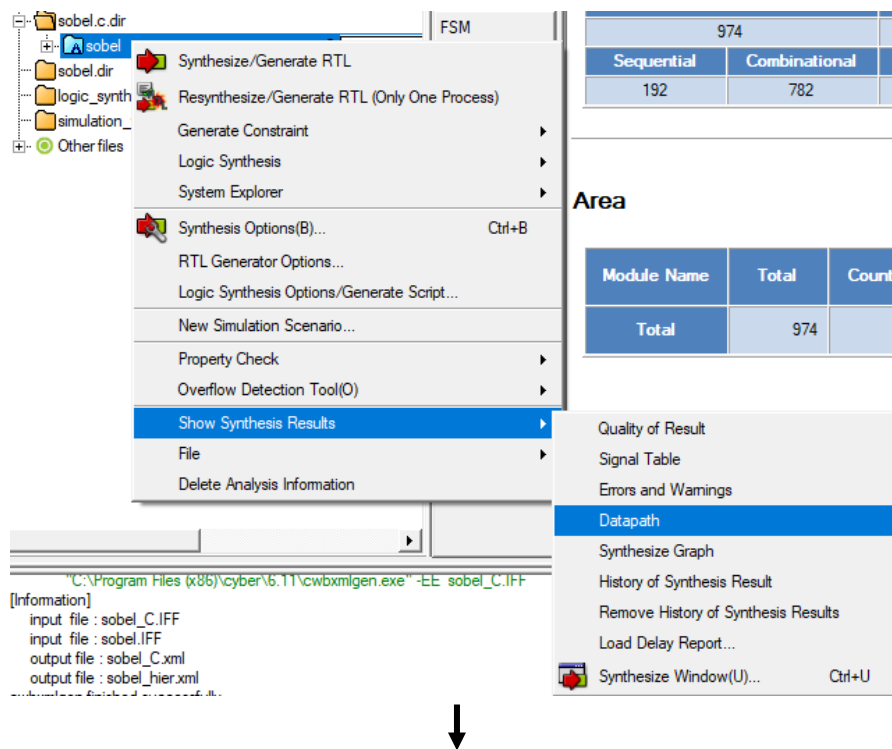
- Right click on light blue sobel folder → RTL-LS script → sobel\_E.v

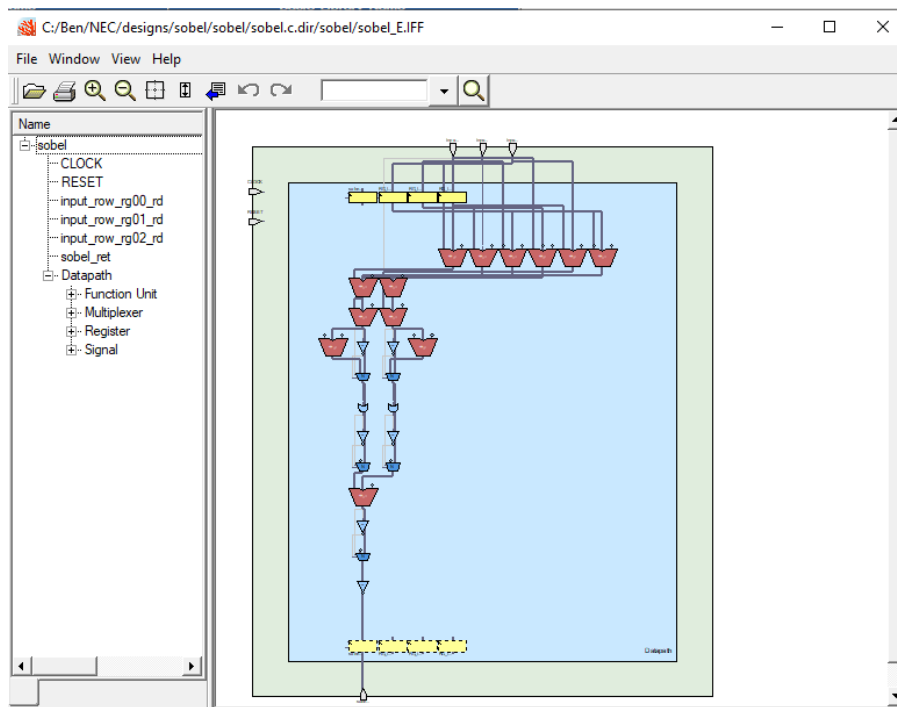


## Review of Synthesis Result

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

Port	Reg	Mem	Fu	Others
sobel_ret	unsigned char sobel(unsigned char inpi			W
RG_line_buffer	unsigned char line_buffer[SIZE_BUF			RW
RG_line_buffer_1	unsigned char line_buffer[SIZE_BUF			RW
RG_line_buffer_2	unsigned char line_buffer[SIZE_BUF			RW
sobel_ret	unsigned char sobel(unsigned char inpi			W
s_11@1				USE
s_11_10@1				USE
s_11_10@2				USE
s_11_11@1				USE
s_11_11_1@1				USE
u_11@1				USE
u_11_10@1				USE
u_11_10@2				USE
u_11_10_1@1				USE
@1				USE
@2				USE
@3				USE
_9@1				USE

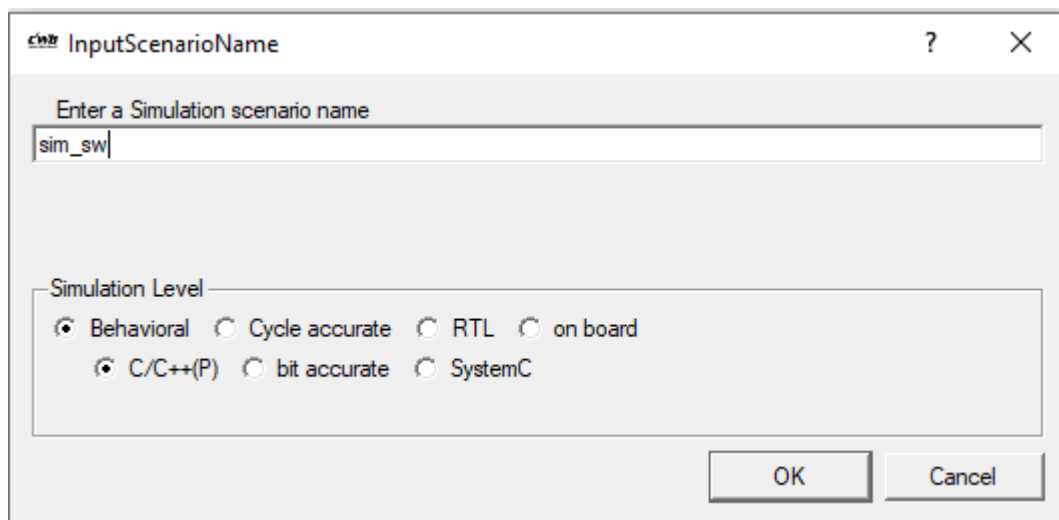
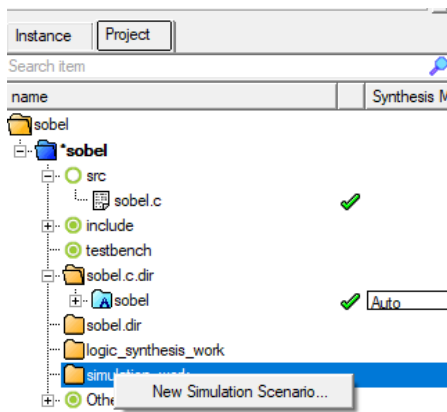
The Signal table shows the timing diagram of the synthesis. When inputs and outputs are being read or written and when registers accessed. State 00 = reset state and state 01 = stating doing the computation.

## Design Verification: Software Simulation

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

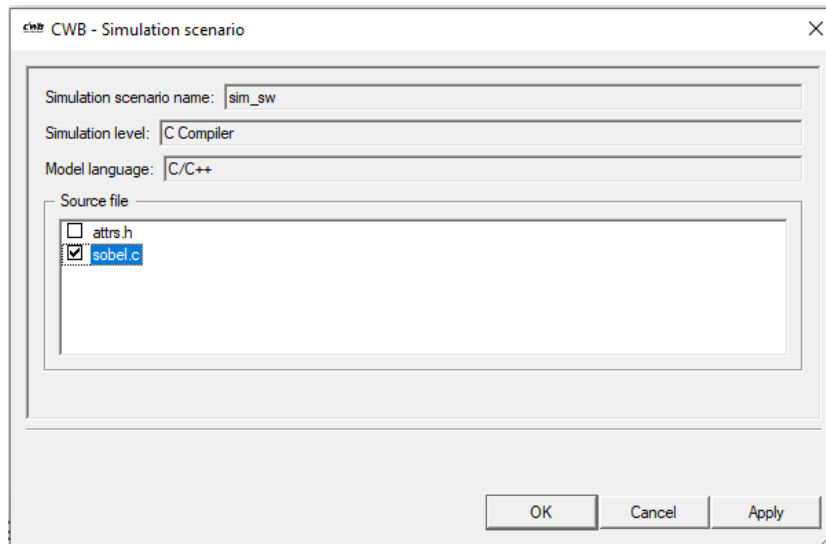
- Pure SW simulation: normal SW simulation
- Behavioral simulation: untimed simulation (like SW), but considering the HW data types (ter, var, reg)
- Cycle-accurate simulation : timed simulation of the scheduled synthesis results
- RTL simulation

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

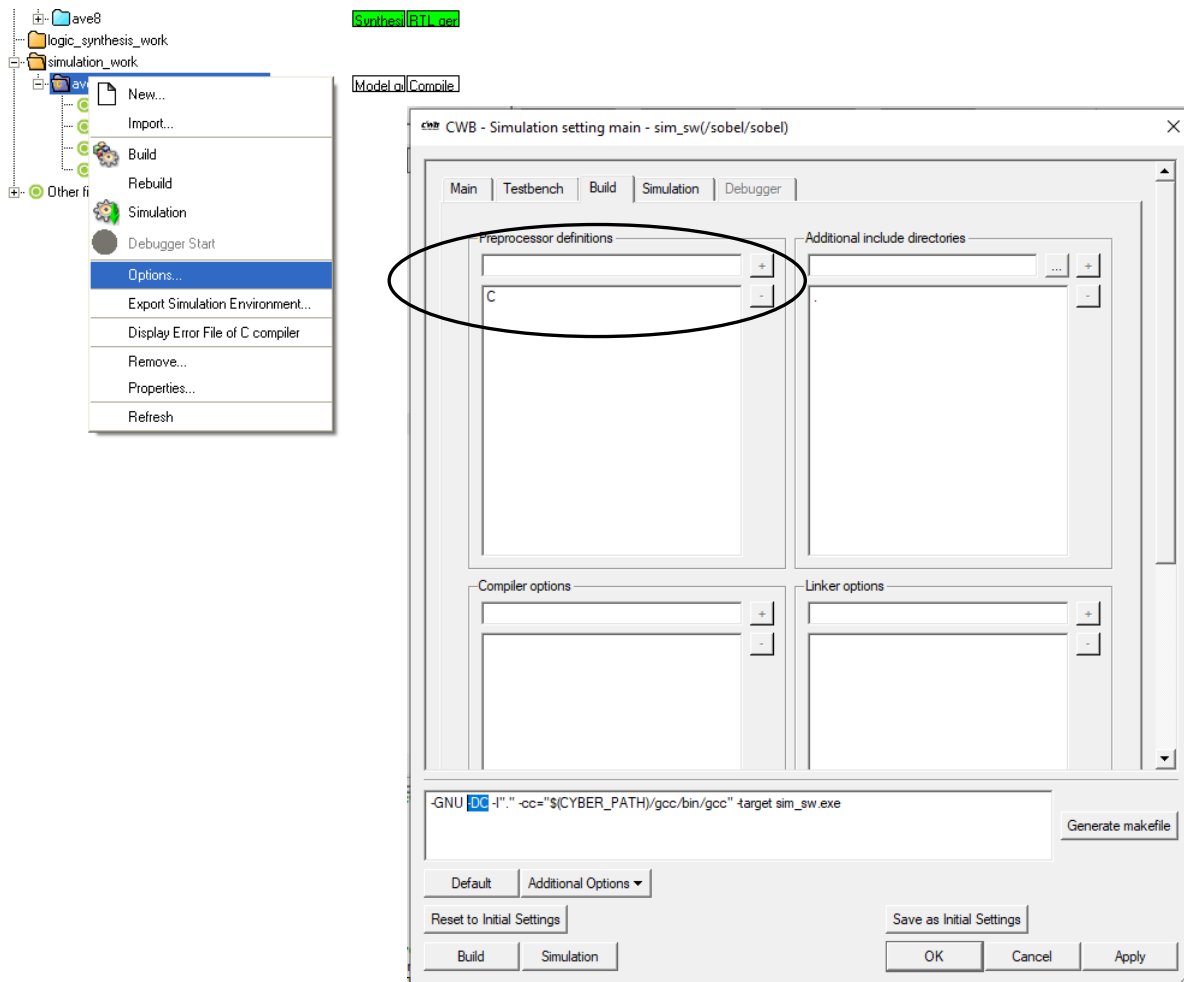


2. Select "Behavioral" and C/C++ and call the scenario name 'sobel\_sw'
3. Select the top process (check the check box)

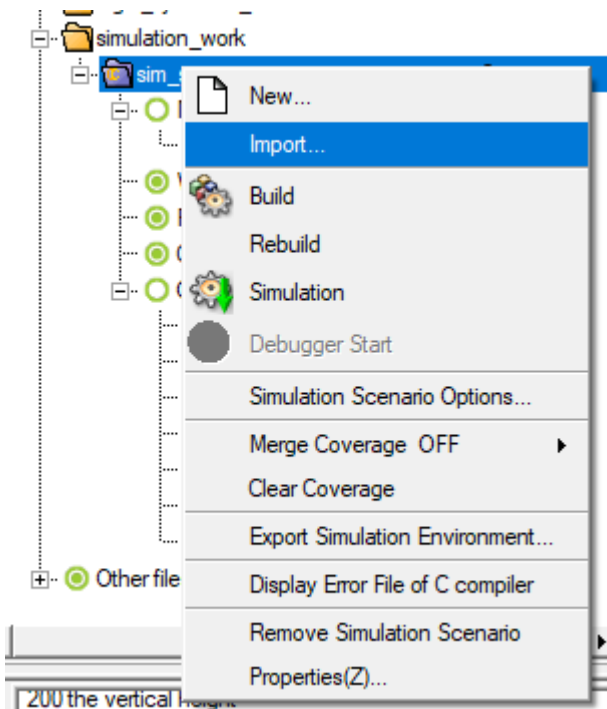




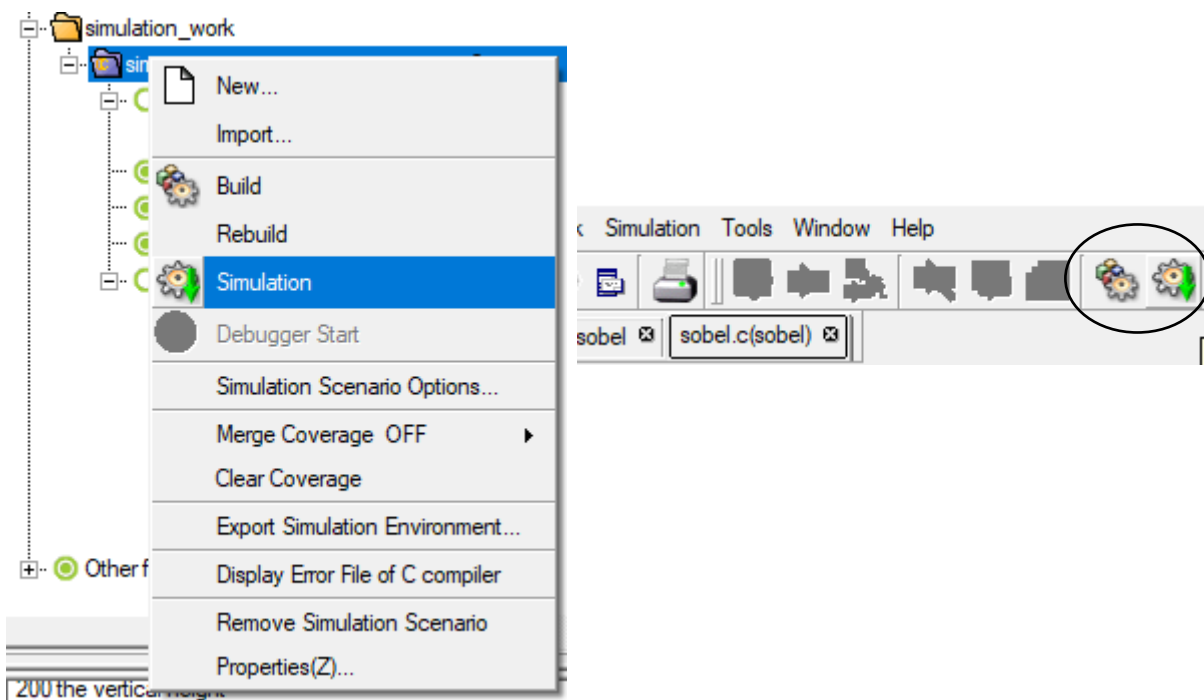
4. Right-click on 'sobel\_sw' folder → options → Build tab → Add C to the preprocessor definitions.



5. Add the lena512.bmp file to the sim\_sw simulation work folder



6. Compile the program and execute it.



A lena512\_out.bmp should have been generated. This will be our ‘golden’ output against which the simulation results throughout the different synthesis stages will be compared against.

## Cycle-Accurate simulation

To verify the correctness of the timing and also the accurate performance of the synthesized designs two options are available:

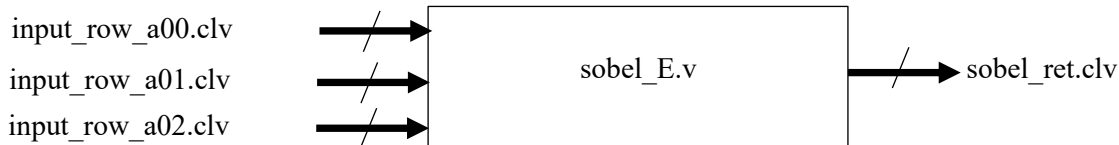
1. simulate the generated RTL code
2. created a cycle-accurate model.

This second option is normally preferred as the simulation is faster than RTL.

The problem now is that CWB does not take the .bmp file as input test vector. The convention used is that it requires a separate file for each of the IOs where the naming convention is :

<input/output name>.clv

Thus, we need to generate the following files:



The names of the files required can be extracted from the RTL file generated by CWB (sobel\_E.v). Thus, to generate these files we need to modify the sobel testbench (main) and dump the inputs and outputs to separate files. E.g.:

```
int main(){
#ifdef TB
FILE *fp_input1, *fp_input2, *fp_input3, *fp_output;
// File names created should be the same as Inputs and outputs of RTL
description see sobel_E.v/.vhd1
fp_input1 = fopen(" input_row_rg00_rd.clv", "wt");
fp_input2 = fopen(" input_row_rg01_rd.clv", "wt");
fp_input3 = fopen(" input_row_rg02_rd.clv", "wt");
fp_output = fopen(" sobel_ret.clv", "wt");
#endif

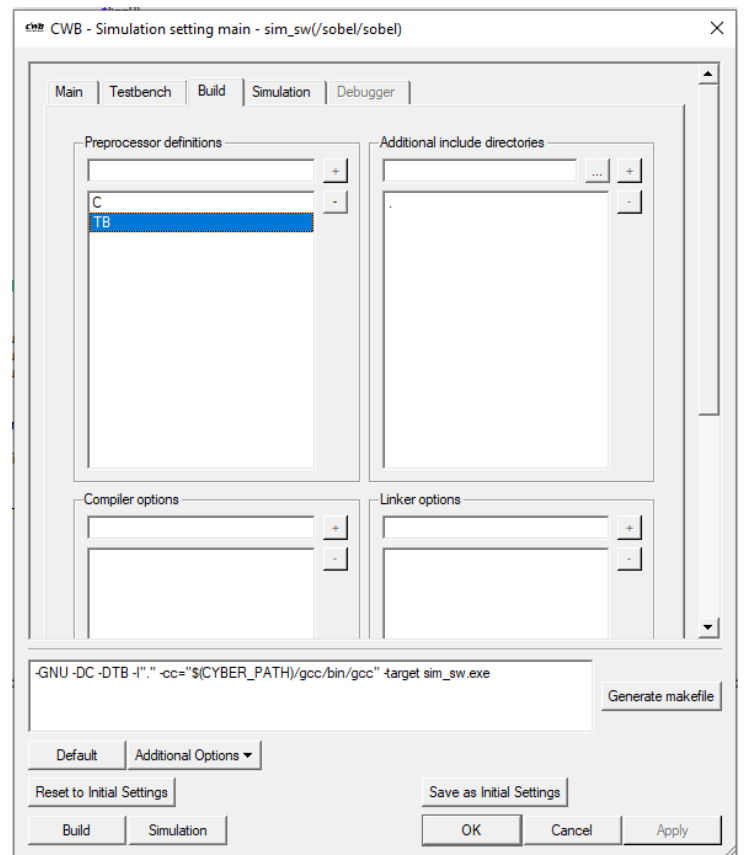
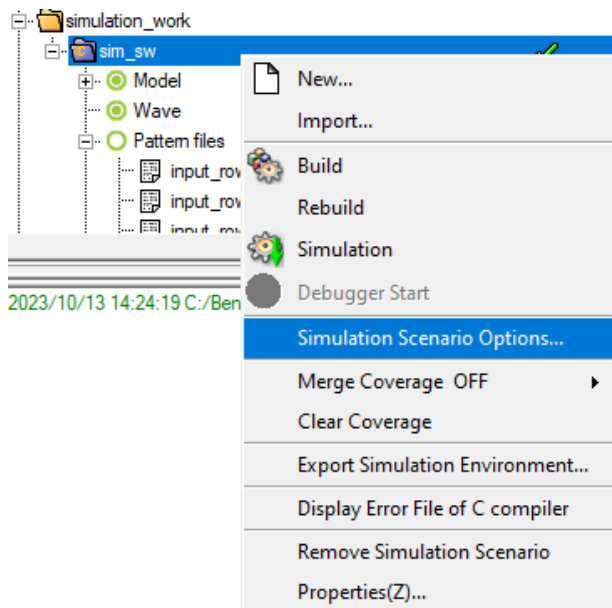
        :
        :
        /* Sobel function call */
#ifdef TB
fprintf(fp_input1, "%u\n", input_row[0]);
fprintf(fp_input2, "%u\n", input_row[1]);
fprintf(fp_input3, "%u\n", input_row[2]);
#endif

        output_row = sobel(input_row);
        bitmapFinalImage[(i*COLS)+j+k] = output_row;

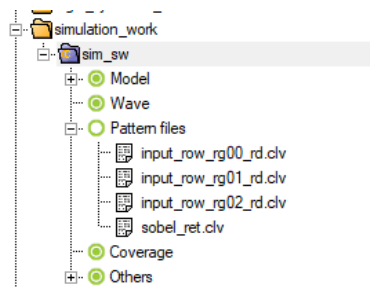
#ifdef TB
fprintf(fp_output, "%u\n", output_row);
#endif
}
```

Adding this code will create 4 .clv files that will contain all of the test vectors for CWB. Because this new code is only compiled defining TB as pre-compiler directive, this needs to be enabled and the new sobel file compiled and executed.

1. sim\_sw folder right click → Simulation Scenario Options

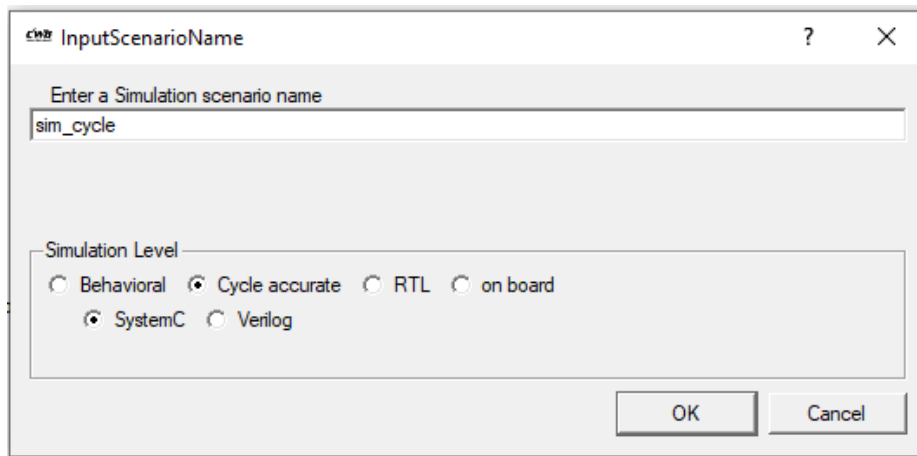


2. Re-run the simulation. Four .clv files should have been generated in the Patterns files folder in sim\_sw.

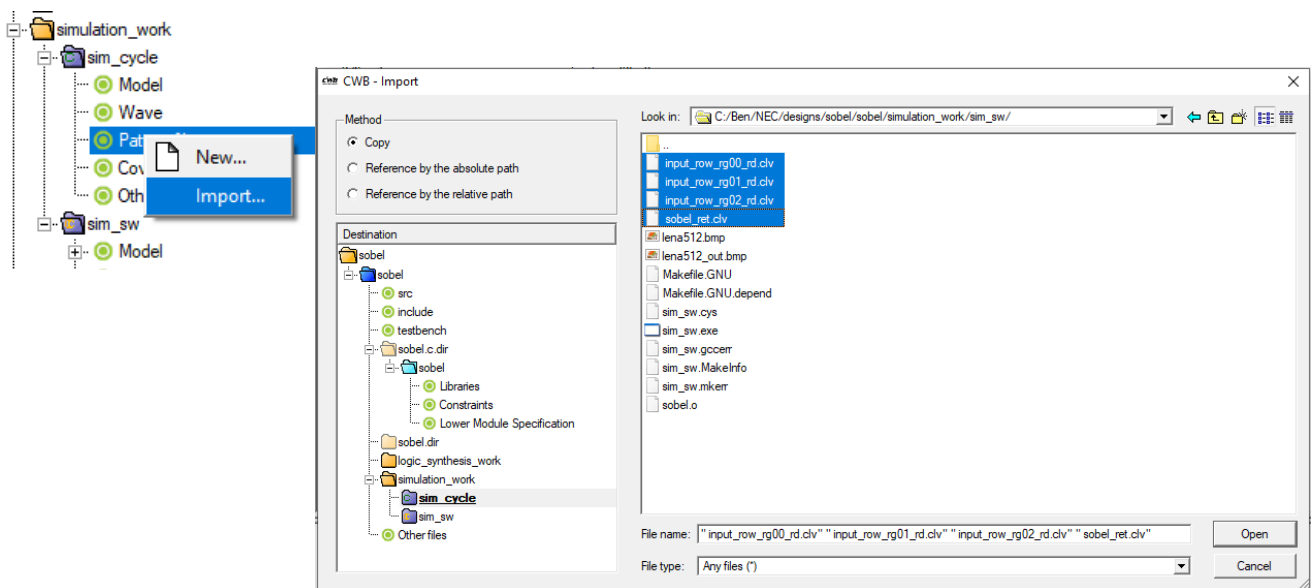


These files will be used as test vectors and golden outputs for the cycle-accurate and RTL simulations. For this do the following:

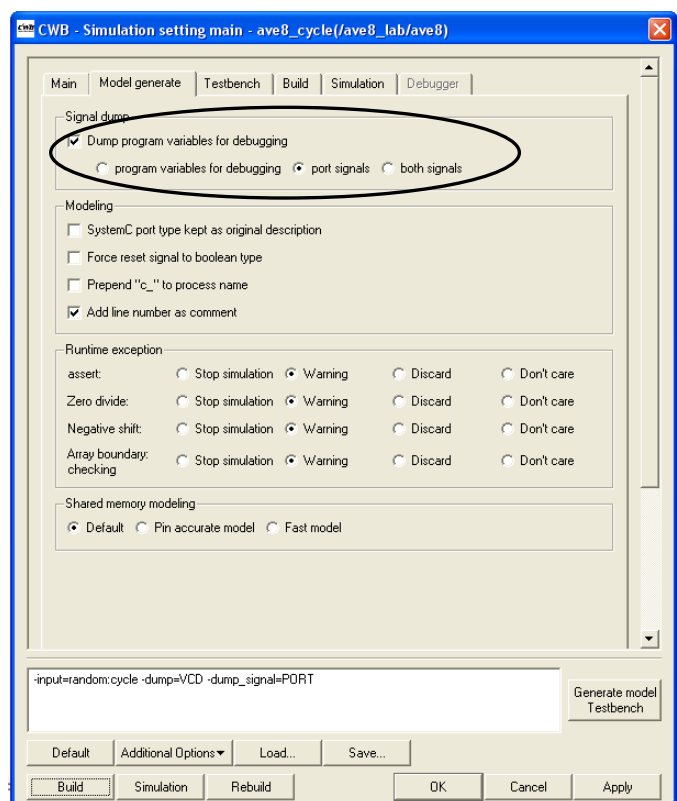
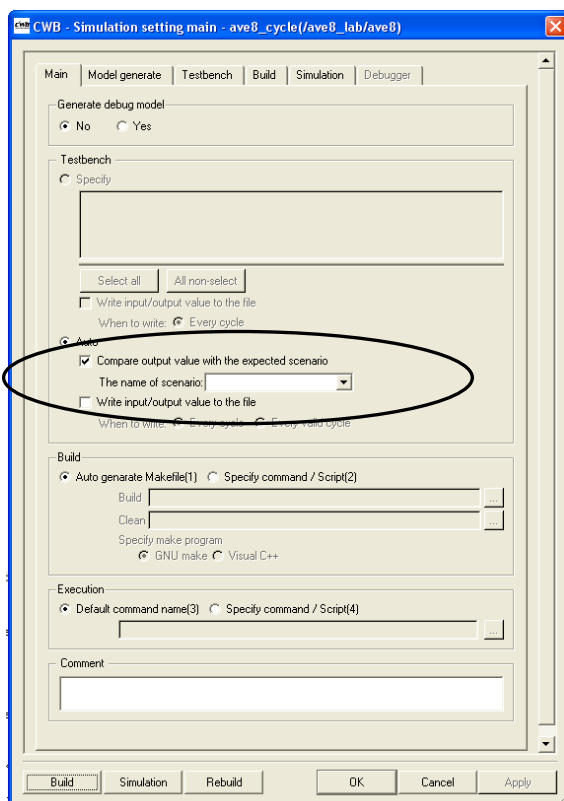
- Create a new simulation scenario (right-click on the 'simulation work' folder. Select 'Cycle accurate' options and enter scenario name (sim\_cycle) and SystemC



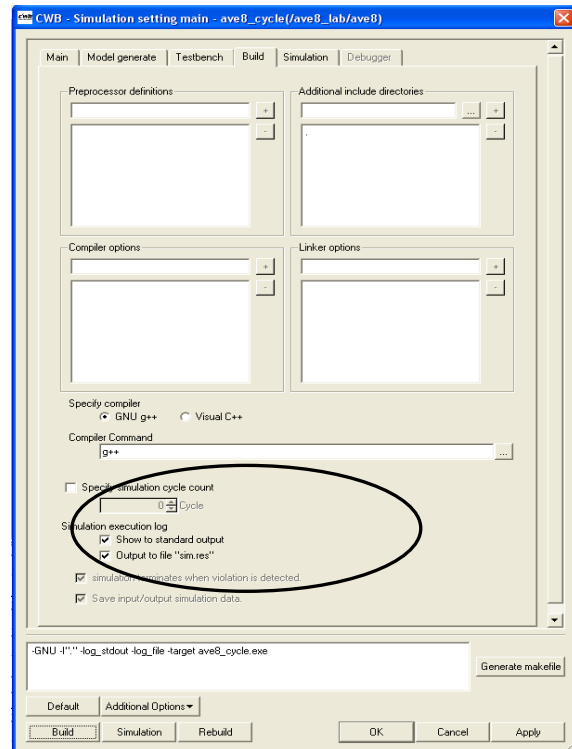
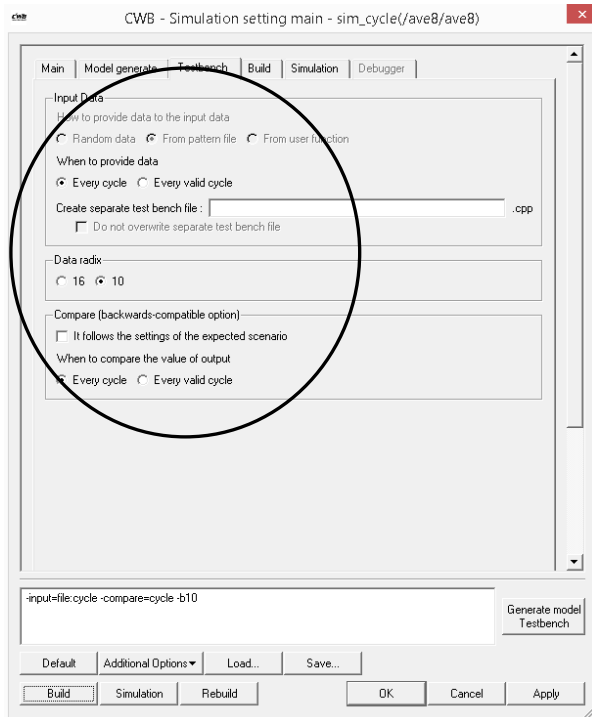
3. Import the .clv files generated in sim\_sw to sim\_cycle folder as follows:



- Specify testbench options (right click sim\_cycle → options)

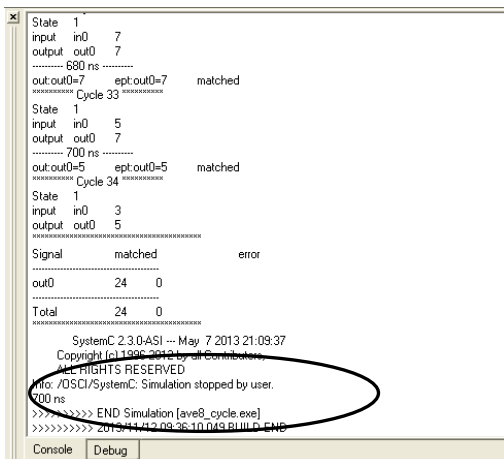


- Change the data radix from hex to decimal
- Select the option to print out the simulation on the console window and dumping this to sim.res

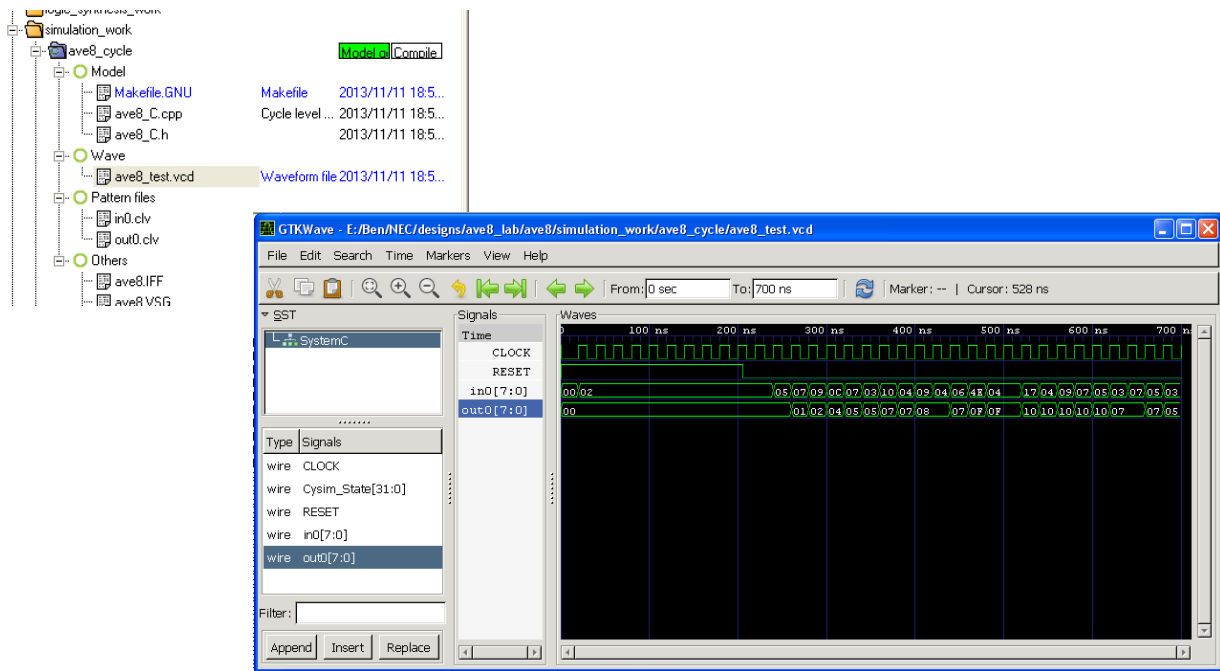


- Build and simulate the model. The console window will display if the outputs from the cycle-accurate simulation match the outputs from the SW simulation.

**NOTE: Because this is a cycle-accurate simulation it will take much longer then the SW simulation**



- The simulation will also generate a .vcd file which can be opened by a waveform viewer (e.g. GTKwave) is generated. Double click on it to open it.

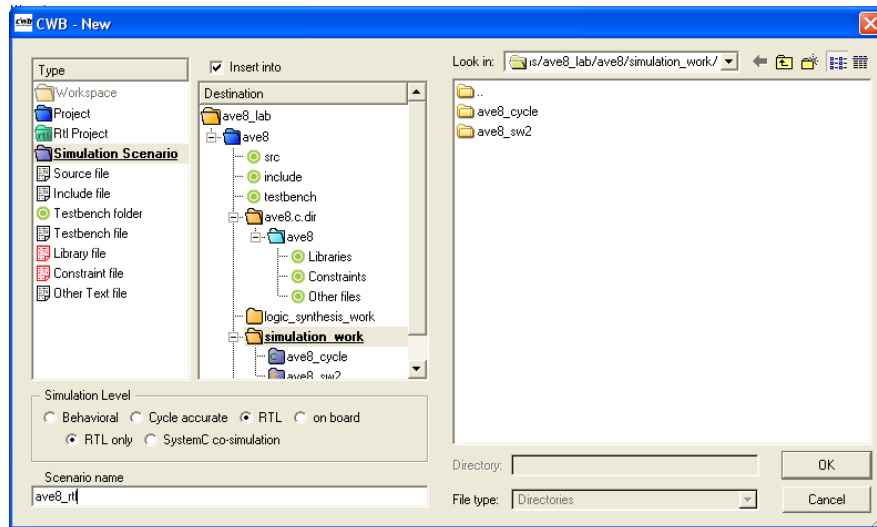


The simulation output should match the result of the SW simulation.

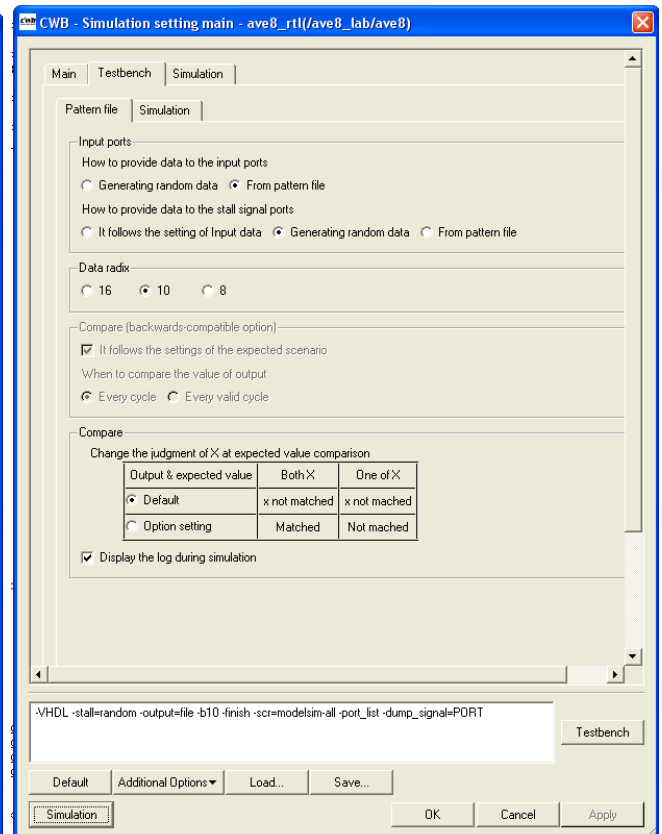
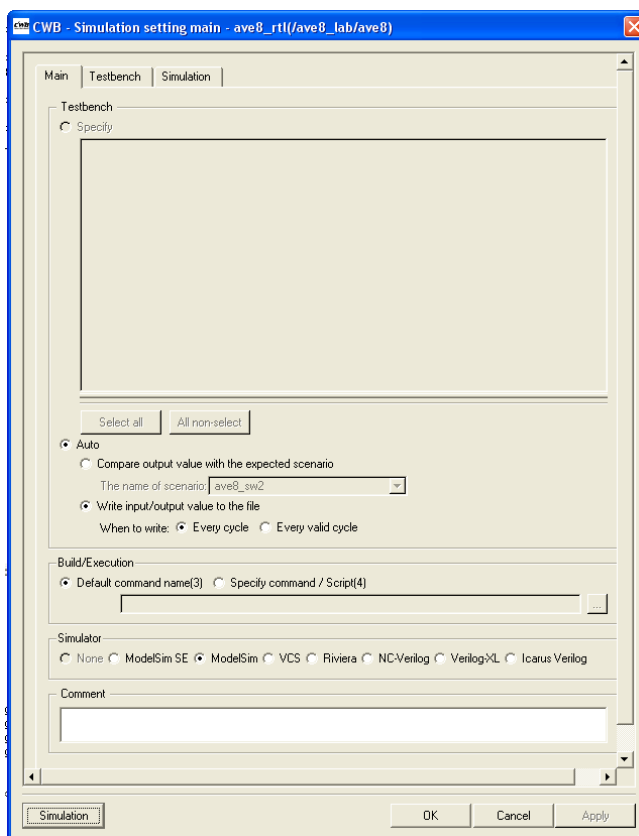
## RTL simulation

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 sobel\_rtl



2. Copy the .clv test vectors to this new simulation scenario.
3. 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.



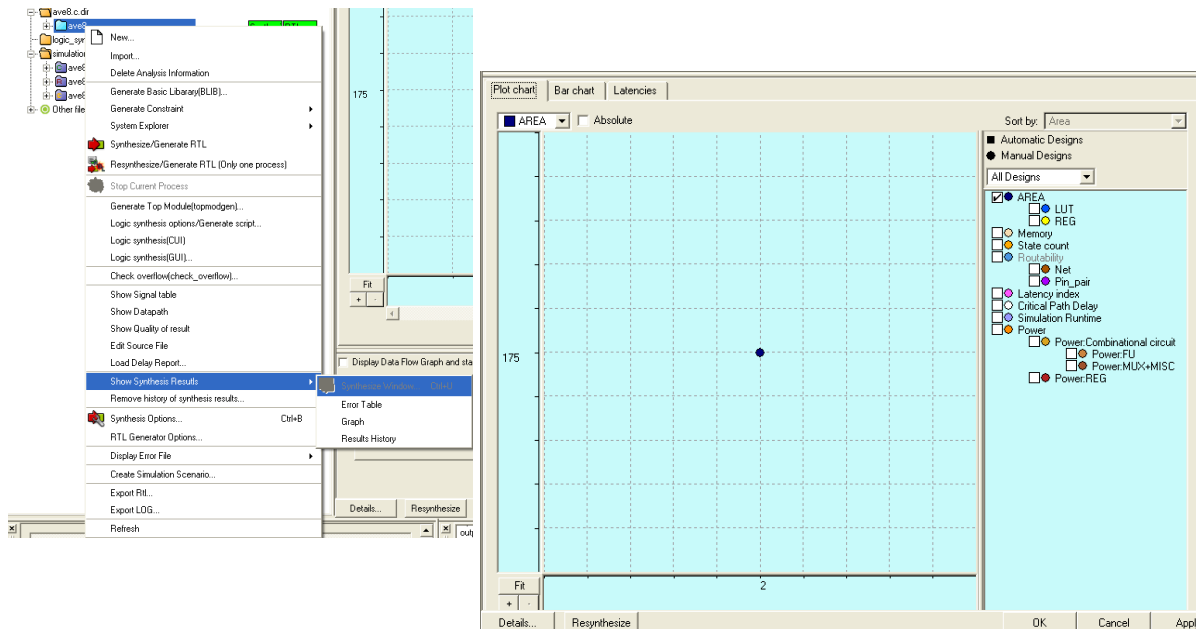
- CWB generates the RTL testbench, but also the scripts to run the simulation based on the selected RTL simulator.



## 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. Set different pragma combinations on the loops and arrays as follows and re-synthesize the design:

```
char Gx[3][3] /* Cyber array=ROM */ ={{1 ,0 ,-1},
    { 2, 0, -2},
    { 1, 0,-1}};

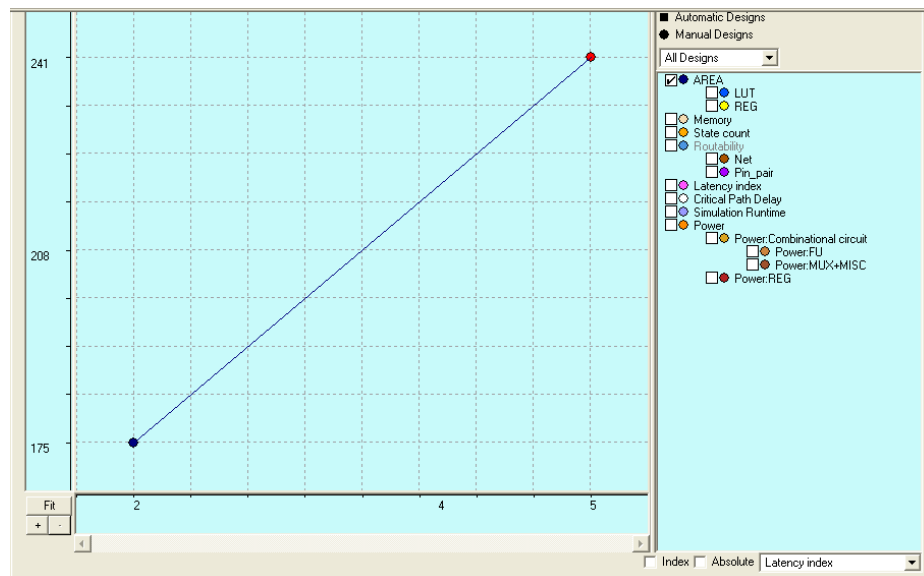
char Gy[3][3] /* Cyber array=ROM */ ={{1, 2, 1},
    {0, 0, 0},
    {-1, -2, -1}};

:
:
// Convolution starts here
//-----X GRADIENT APPROXIMATION-----
//-----Y GRADIENT APPROXIMATION-----
/* Cyber unroll_times=all */
for(rowOffset = -1; rowOffset <= 1; rowOffset++){
/* Cyber unroll_times=all */
for(colOffset = -1; colOffset <=1; colOffset++){
    sumX = sumX + line_buffer[1 +rowOffset][1-colOffset] * Gx[1+rowOffset][1+colOffset];
    sumY = sumY + line_buffer[1 +rowOffset][1-colOffset] * Gy[1+rowOffset][1+colOffset];
}
}
```

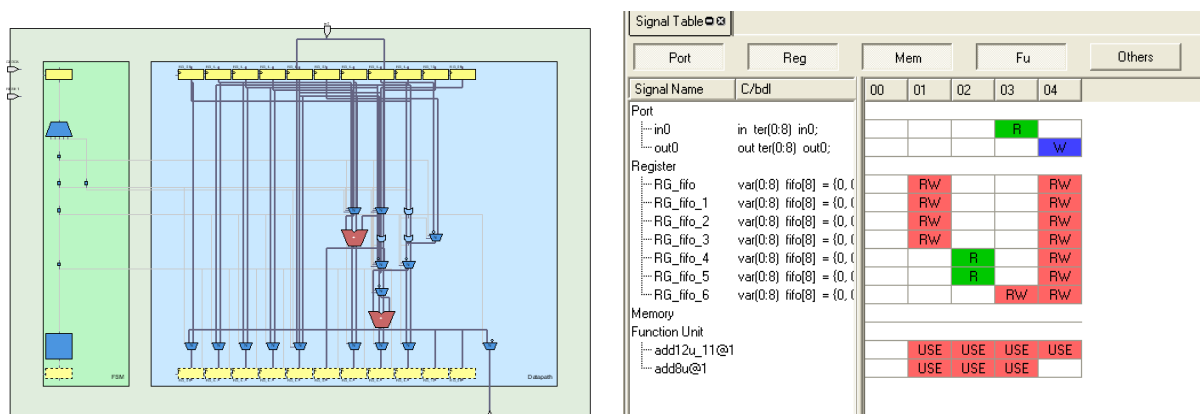
- Change the pragmas to the following configurations and re-synthesize each new version. Some examples include:

	Pragmas 1	Pragma 2	Pragma 3	Pragma 4
Design 1	Array=logic	Array=logic	Unroll_times=all	Unroll_times=all
Design 2	Array=ROM	Array=ROM	Unroll_times=0	Unroll_times=0
Design 3	Array=logic	Array=logic	Unroll_times=all	Unroll_times=0

- Review the report sheet for details about which pragmas to change.
3. New design with different Area and latency characteristics is generated and plotted in the synthesis window



- Opening the schematic viewer and signal table to confirms that a design of different structure and performance has been generated.



- You can re-verify the functionality of the new design. No new simulation scenarios are needed. CWB will automatically re-generate the RTL, testbenches and cycle-accurate simulation models when re-running it.

**Note: If the Latency is not 1, then you need to generate a new .clv files that insert 0 between the test vectors as shown below.**

Latency 1	Latency 2	Latency 3
3	3	3
4	0	0
5	4	0
1	0	4
14	0	0
	5	0
	0	0
	1	5
	0	0
	14	0
		1
		0
		0
		14

[END]