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 HLS generated circuits 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

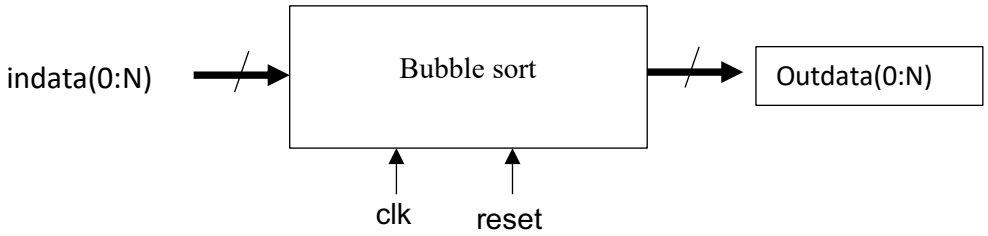
- NEC CyberWorkbench
- gcc (included in CyberWorkBench)

4. Sobel Overview

Bubble Sort is a simple comparison-based sorting algorithm that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order.

It works by "bubbling up" the largest (or smallest) element to its correct position in each pass through the list. After each pass, the next largest element is in place, so fewer comparisons are needed. The process repeats until no swaps are needed, meaning the list is sorted.

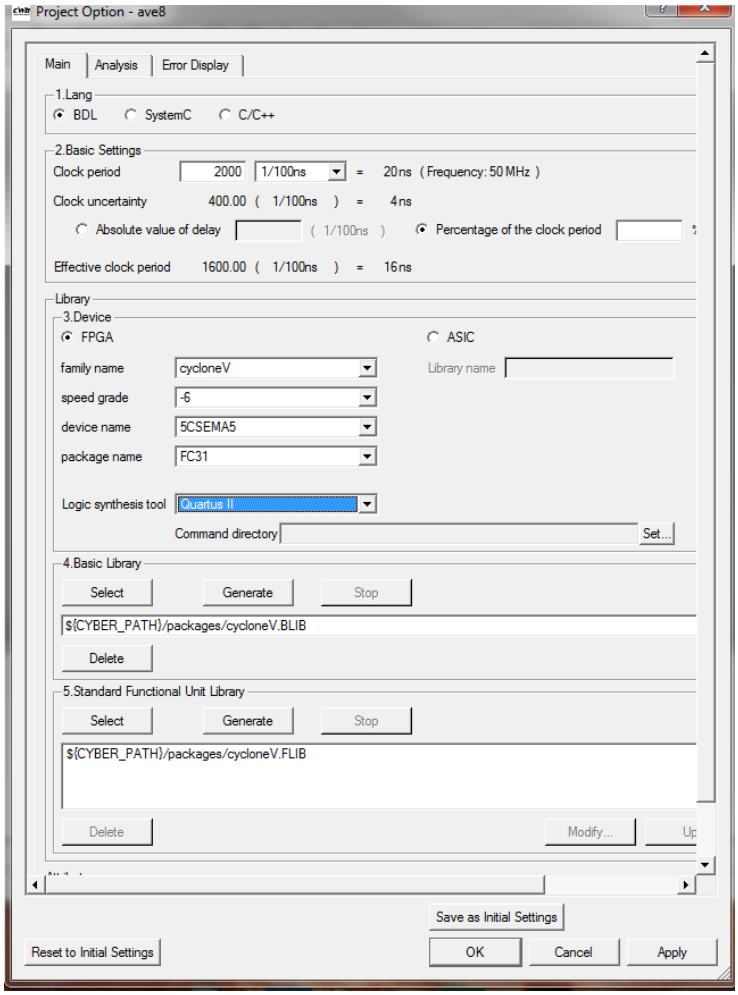
The figure below shows the top level view of the bubble sort including their inputs and outputs.



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 bubblesort (File→New)
3. Set the project parameters as shown in the figure:



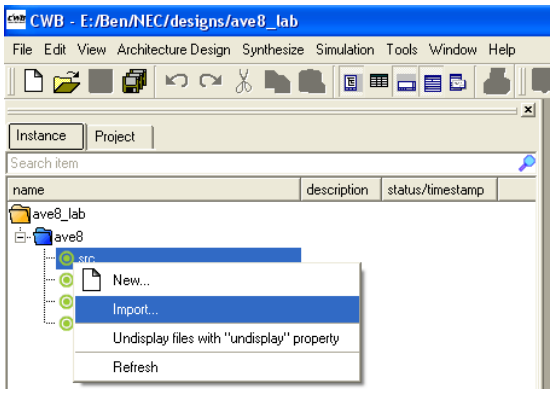
The main items that need to be specified are:

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

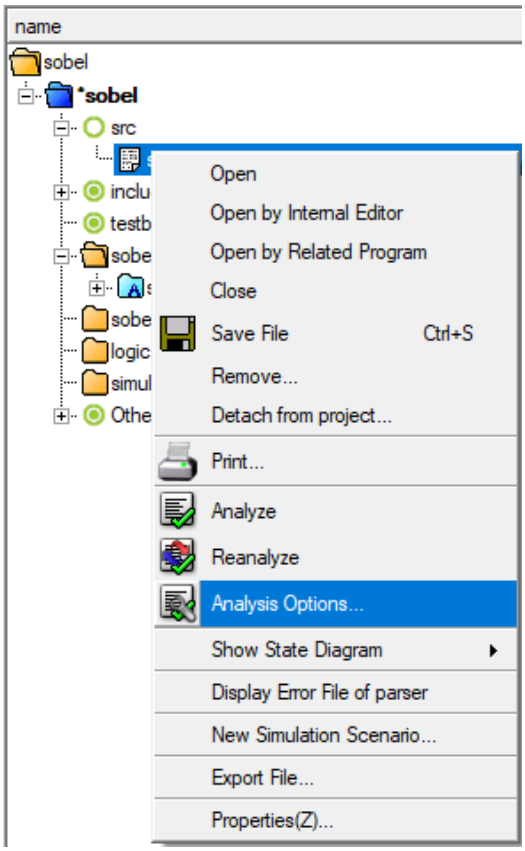
4. Click on Apply to generate the empty project

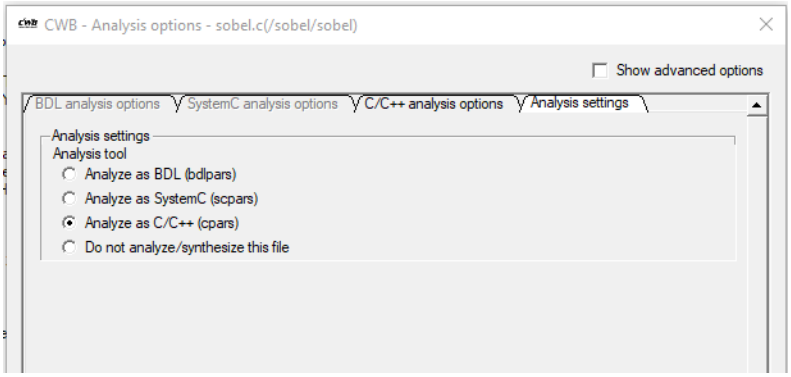
Editing or Inserting an ANSI-C description for synthesis

1. Right click on source → Import → bubblesortc

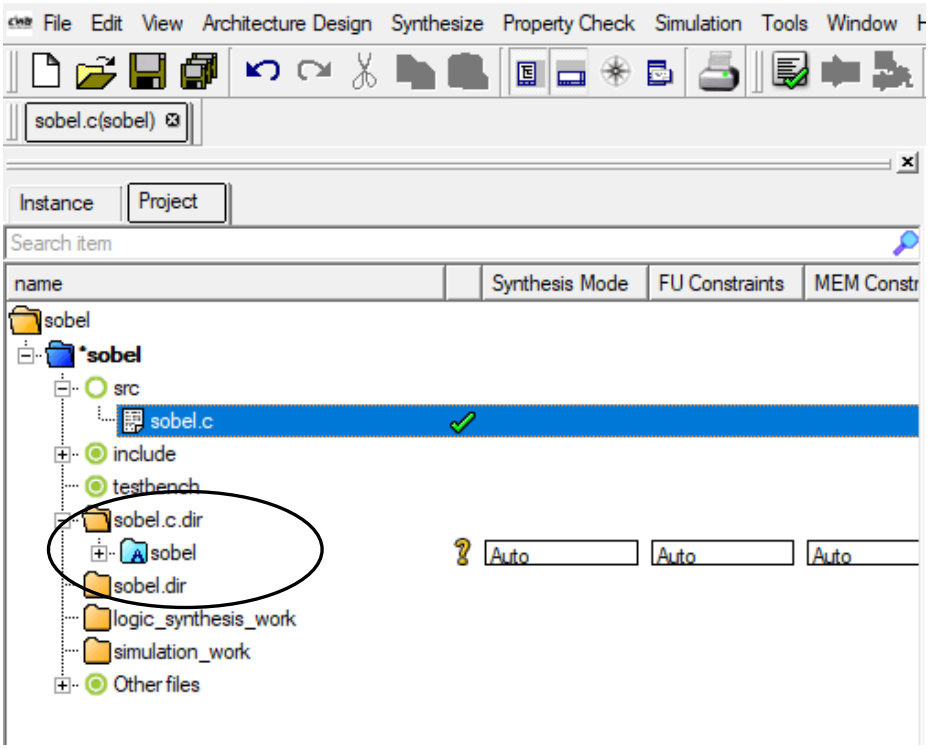


2. Re-write the bubblesort.c file to make it synthesizble. Include #ifdefs where needed and includein the main function /* Cyber func=process */. The file should still be compilable using gcc (review lecture notes -Lecture 10- if needed)
3. Specify the parser to use (just in case to ANSI-C parser). Right click on bubblesort.c→analysis options→Analysis settings →Analyze as C/C++ (cpars)



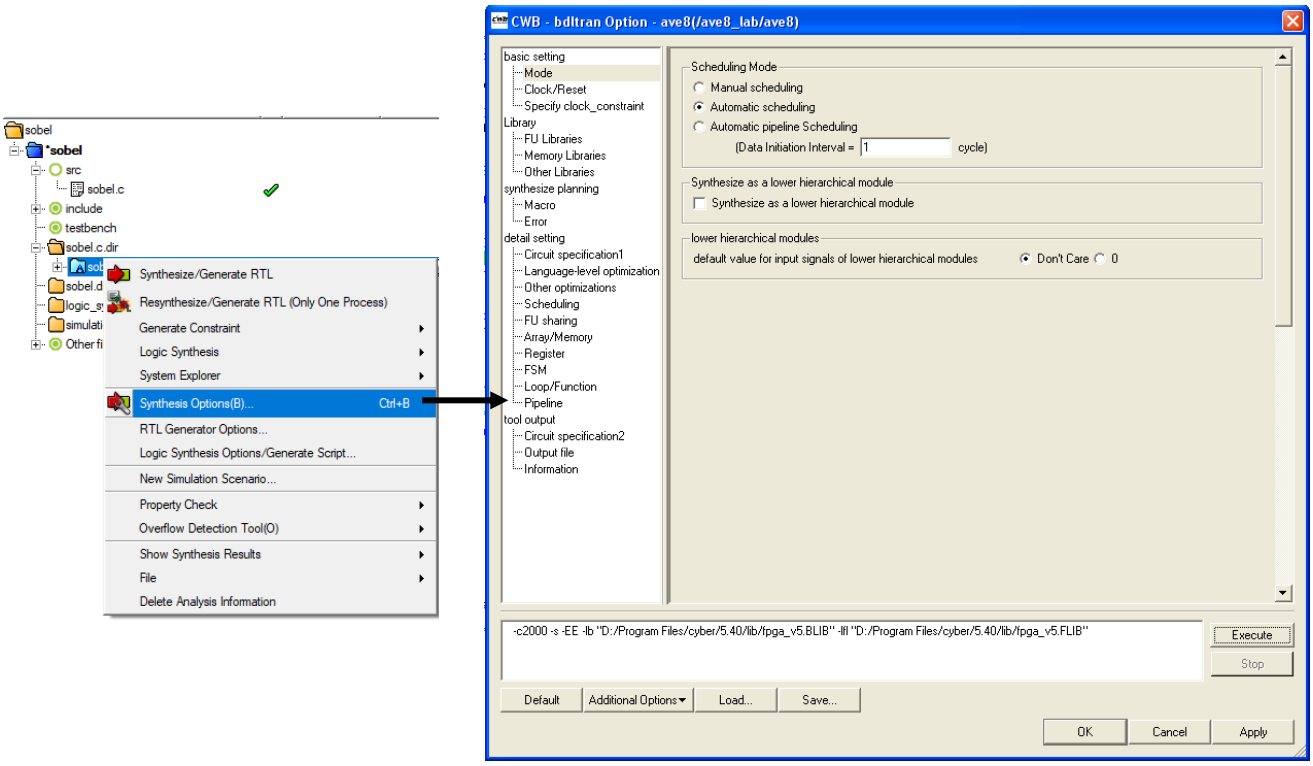


4. Parse source code to check if there are any syntax errors. The console window will indicate if any errors have occurred and where. A 'light blue' folder with an .IFF file will be generated if the parsing was successful



Setting up Synthesis mode and other constraints

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

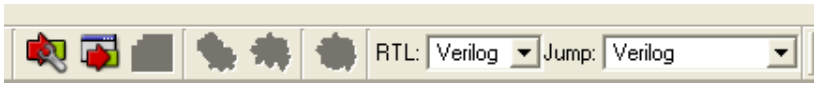


2. Set the scheduling mode to 'Automatic scheduling'. This implies that CWB will automatically time the C description at the scheduling phase. Manual scheduling implies that the user will manually time the description using the '$' sign as a clock boundary.
3. Other synthesis options that control the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

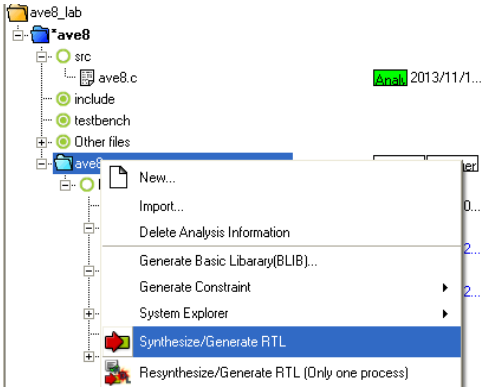
High-Level Synthesis

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

1. Select Verilog or VHDL at the toolbar.

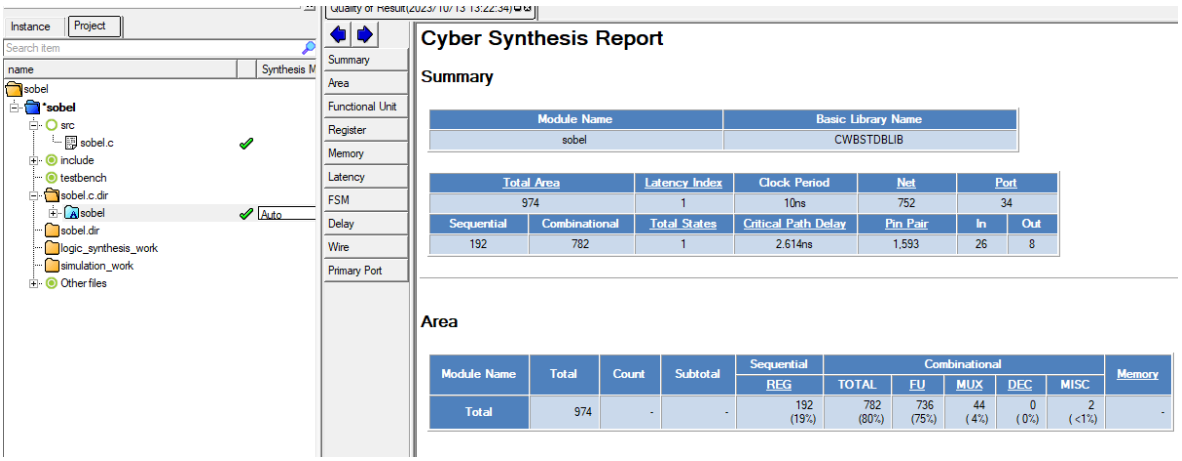


2. Right-click on 'light blue folder → Synthesize/Generate RTL (or toolbar)



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.…

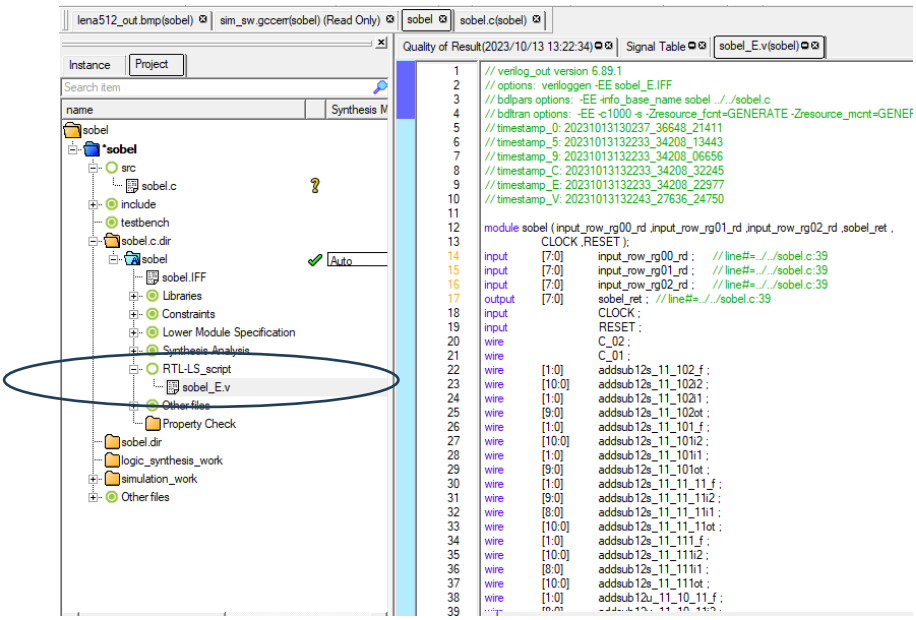
NOTE: The information reported is based on the FLIB and BLIB file provided. It is therefore important to provide the correct library files or to re-generate these if a different device is targeted.
Also, the reported data should be confirmed performing a logic synthesis and a simulation.



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.

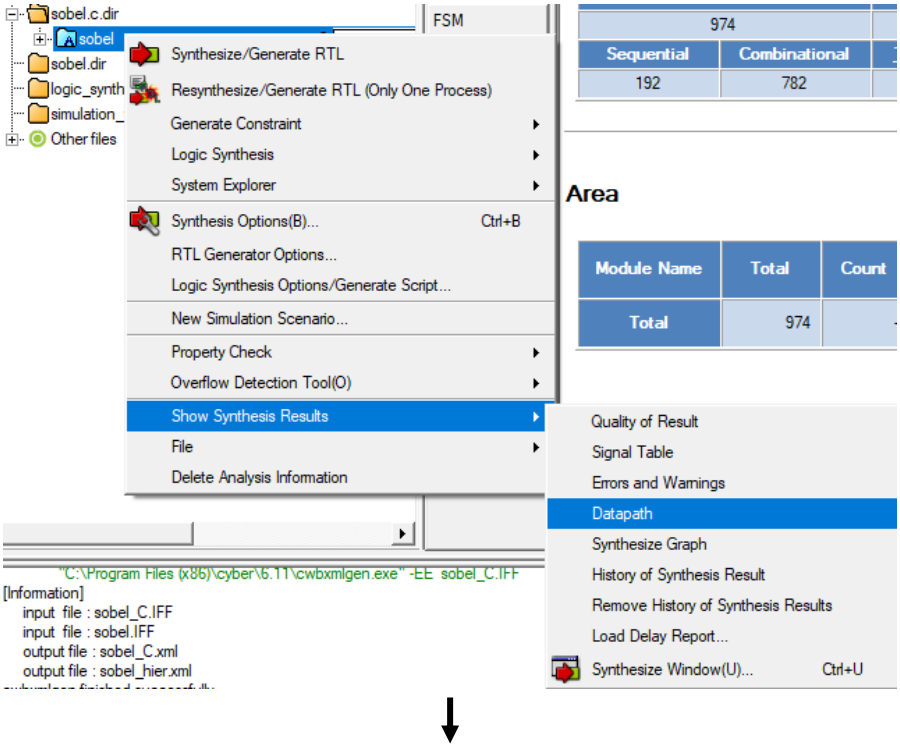
- Right click onl ight blue sobel folder →RTL-LS script → main_E.v

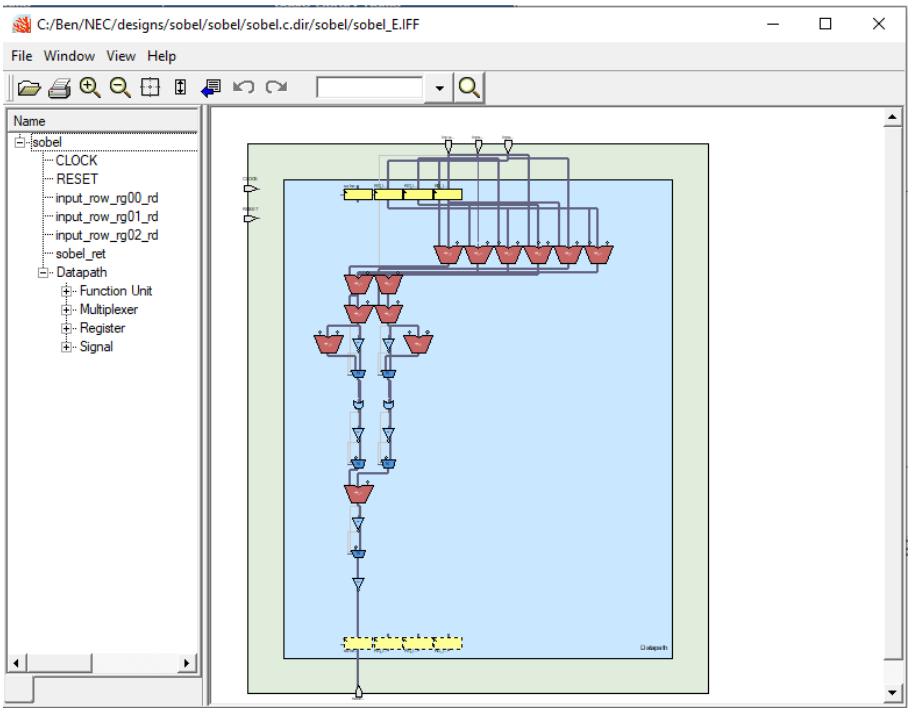


Review of Synthesis Result

The synthesis result can be reviewed in different ways.

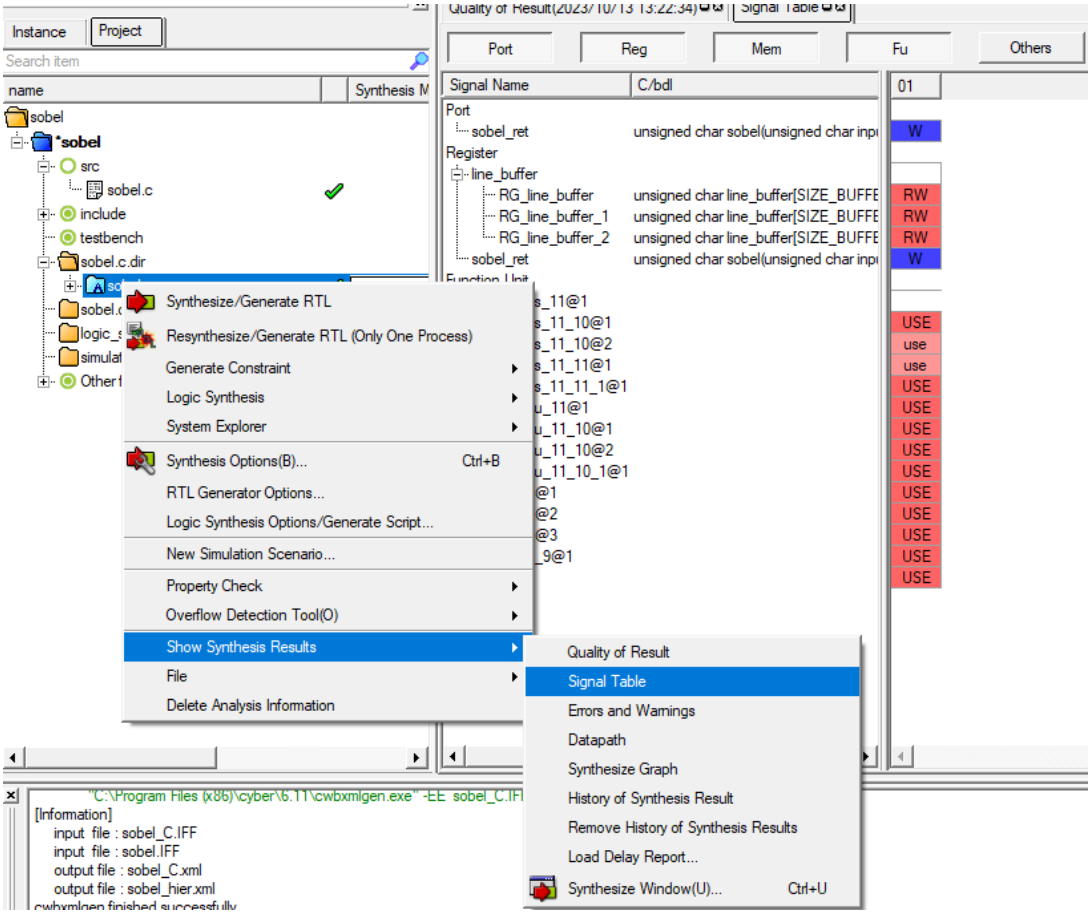
1. Schematic View: Right-click ‘light blue’ folder→ Show Datapath





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



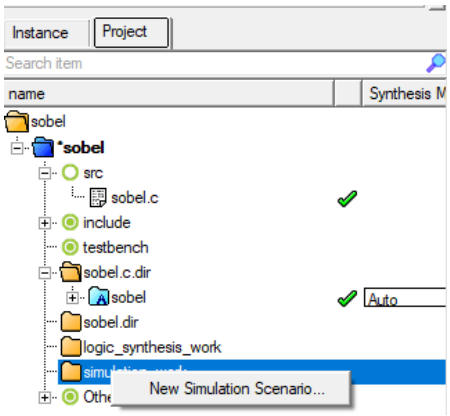
The Signal table shows the timing diagram of the synthesis. When inputs and outputs are being read or written and when registers accessed. State 00 = reset state and state 01 = stating doing the computation.

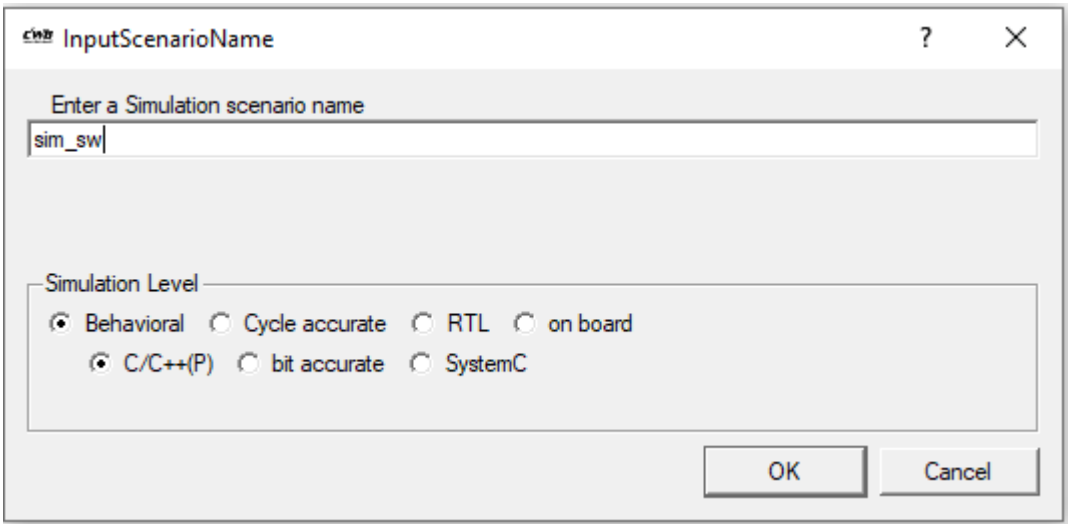
Design Verification: Software Simulation

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

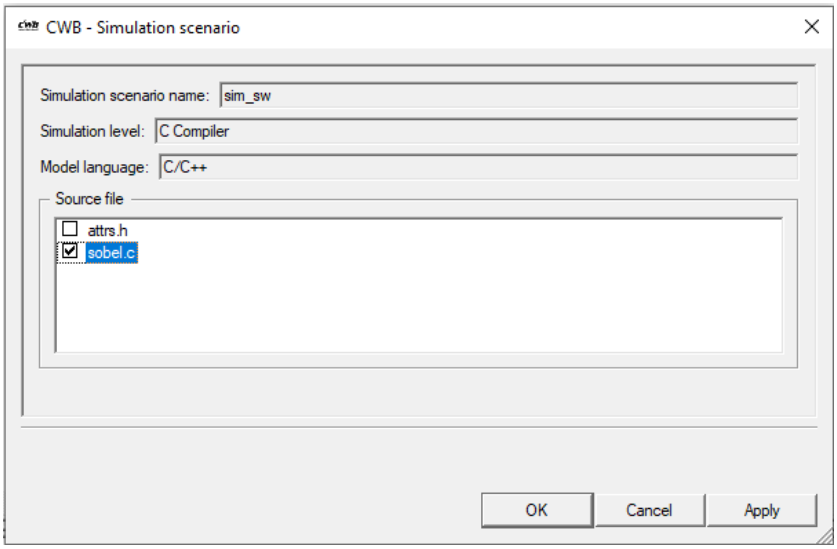
- Pure SW simulation: normal SW simulation
- Behavioral simulation: untimed simulation (like SW), but considering the HW data types (ter, var, reg)
- Cycle-accurate simulation : timed simulation of the scheduled synthesis results
- RTL simulation

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

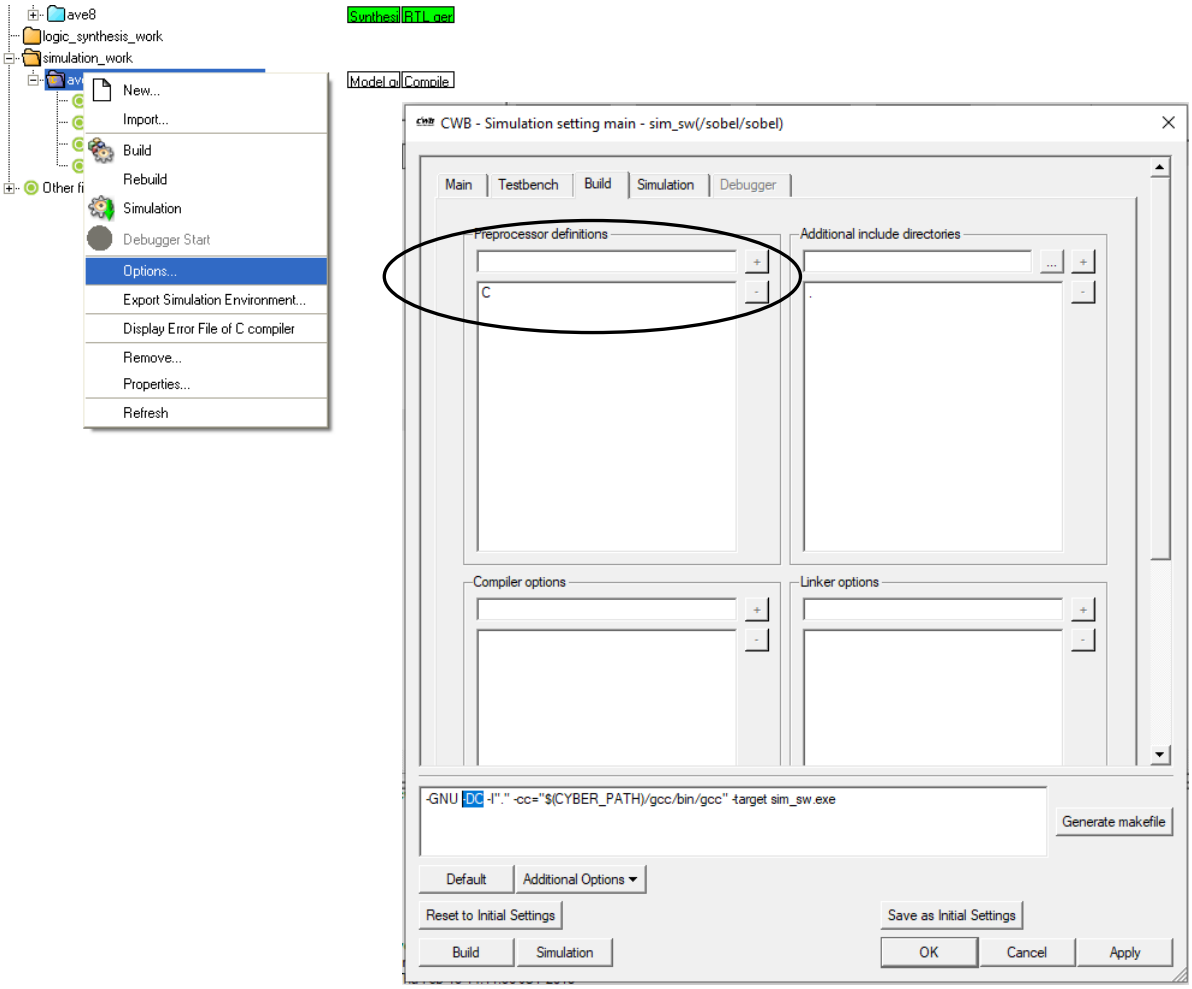




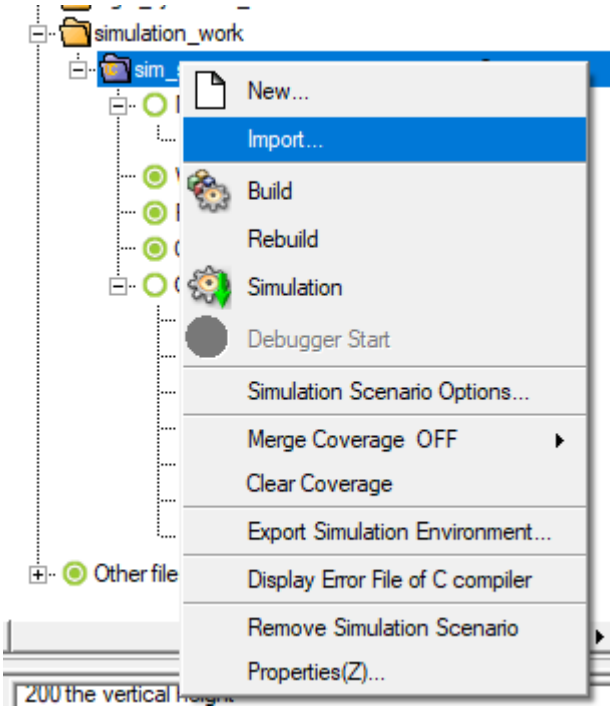
2. Select "Behavioral" and C/C++ and call the scenario name 'bubble_sw'
3. Select the top process (check the check box)



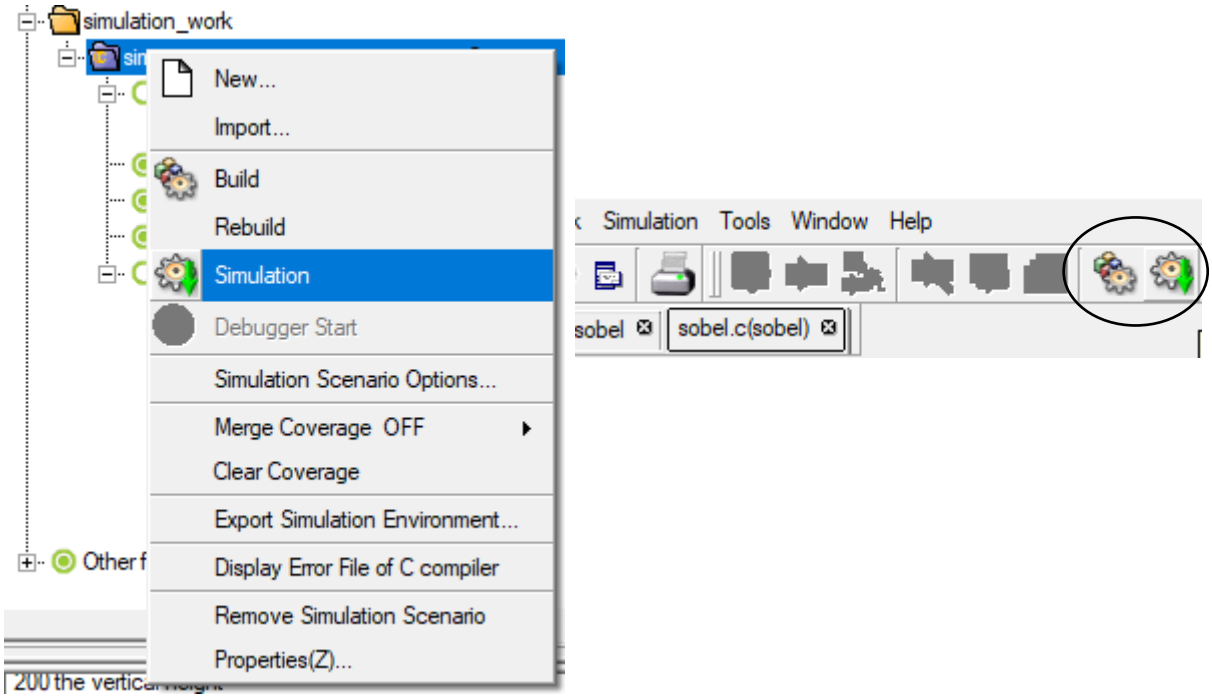
4. Right-click on ‘sobel_sw” folder → options→ Build tab → Add C to the preprocessor definitions.



5. Add the indata.txt file to the sim_sw simulation work folder



6. Compile the program and execute it.

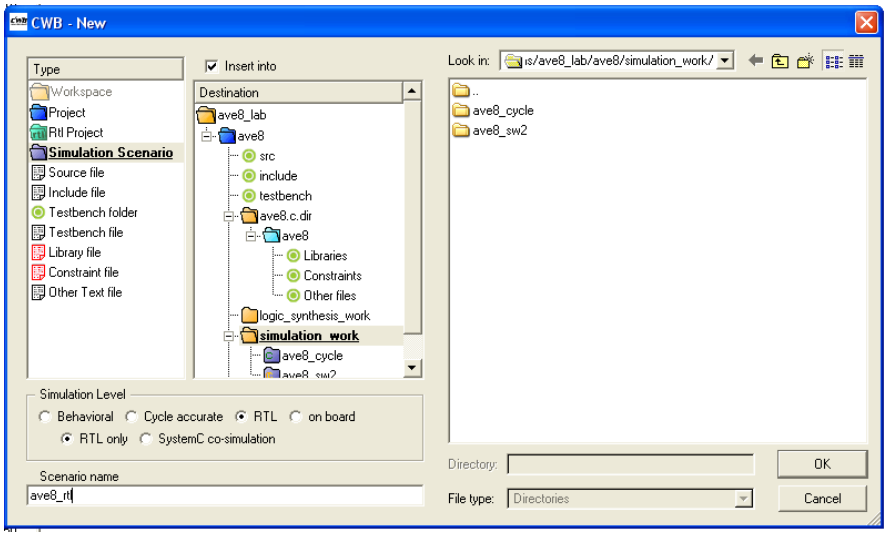


A otdata.txt file should have been generated. This will be our 'golden' output against which the simulation results throughout the different synthesis stages will be compared against.

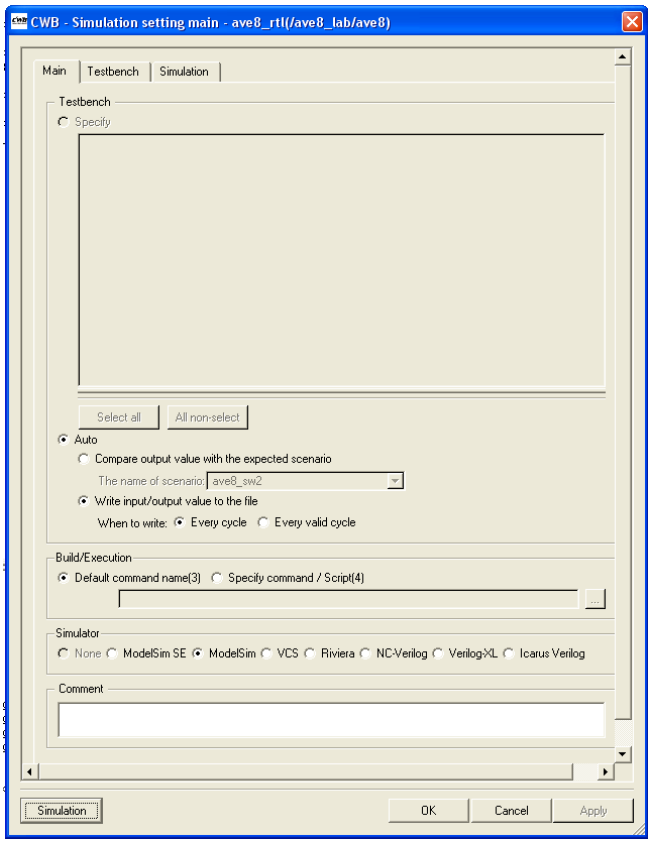
RTL simulation

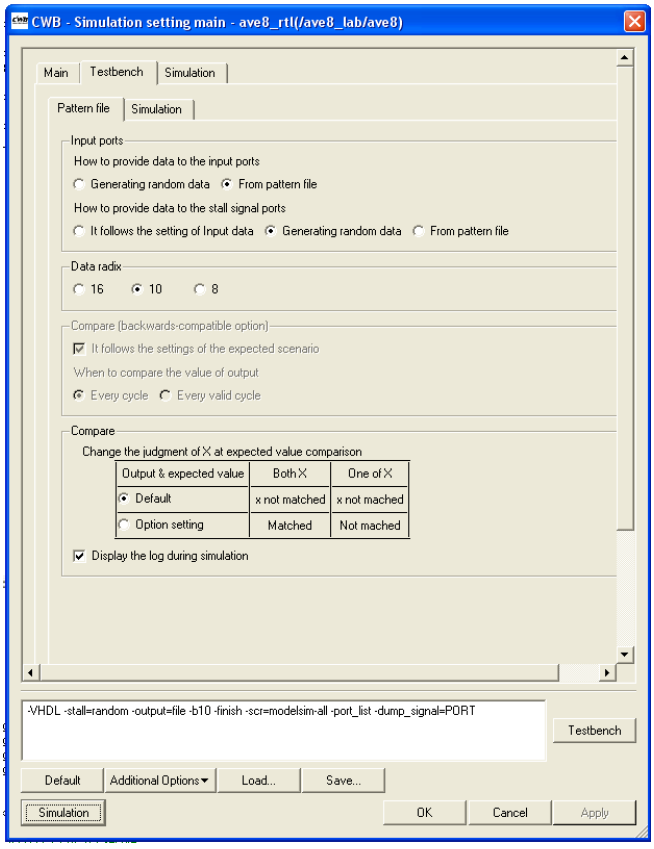
The next verification step is to simulate if the RTL generated also matches the ‘golden’ output.

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



2. Rename the indata.txt and otdata.txt files with the same names as the input and output ports of the design. You can find the names in the RTL Code generated. Rename the extension as .clv and 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.



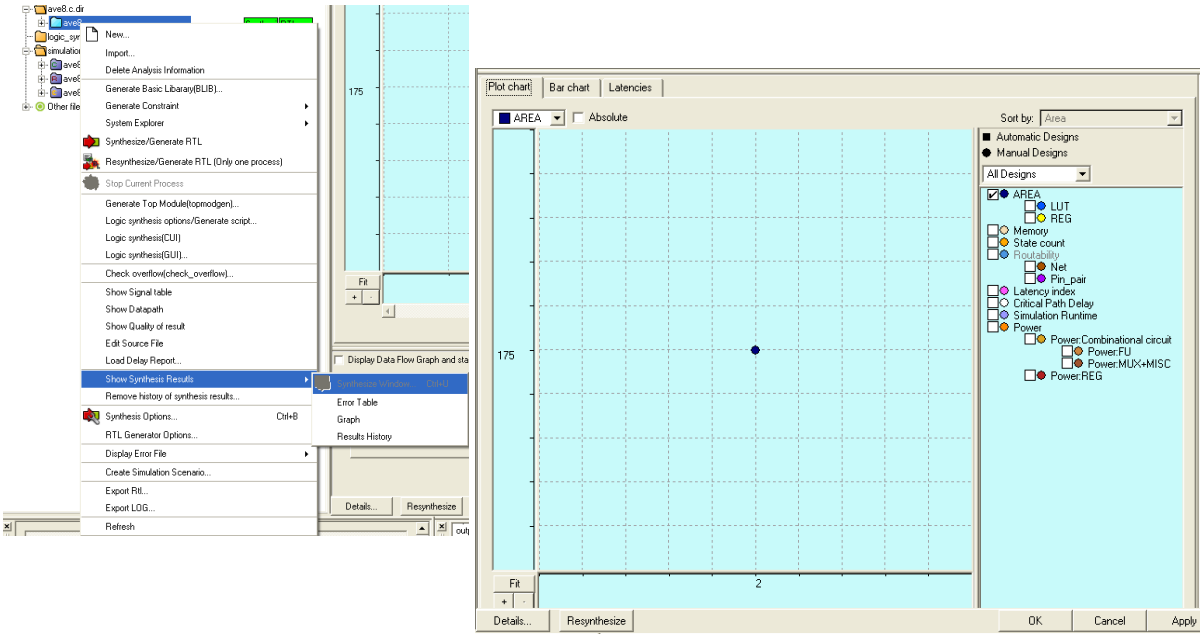


- CWB generates the RTL testbench, but also the scripts to run the simulation based on the selected RTL simulator.

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:

```
int main(void){

    int b_ary[DATANUM] /* Cyber array=REG*/;
    int cnti ,cntj ,tmp ;

    :                    :

  for(cnti = 0 ; cnti < DATANUM ; cnti++){
            if (fscanf(fp1, "%d", &b_ary[cnti]) == EOF) exit(0);
   }
  /* Cyber unroll_times=all */
  for(cnti = 0 ; cnti < DATANUM ; cnti++){
            /* Cyber unroll_times=all */
            for(cntj = DATANUM-1 ; cntj > cnti ; cntj--){
                    if(b_ary[cntj] < b_ary[cntj-1]){
                            tmp          = b_ary[cntj] ;
                            b_ary[cntj]  = b_ary[cntj-1] ;
                            b_ary[cntj-1] = tmp ;
```

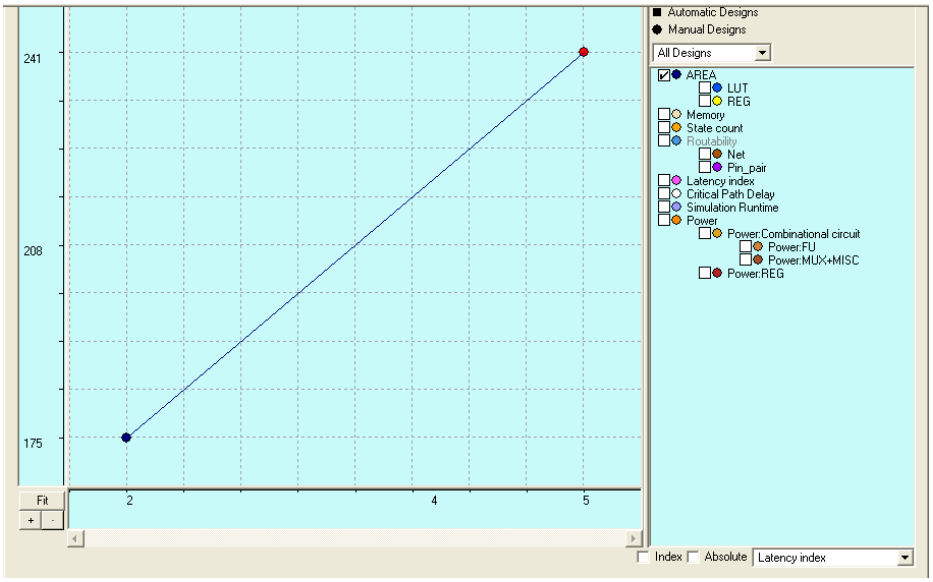
```
                    }
                }
            }
```

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

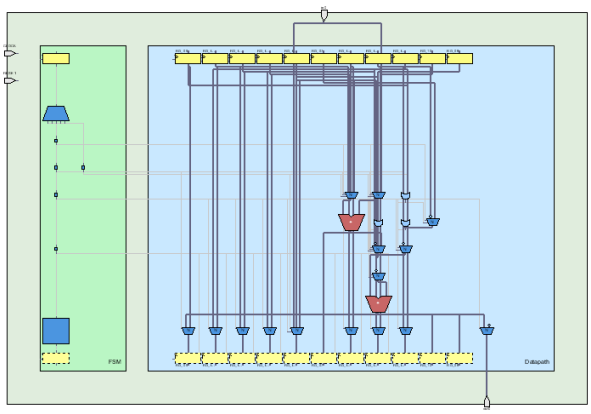
	Pragmas 1	Pragma 2	Pragma 3
Design 1	Array=RAM	Unroll_times=all	Unroll_times=all
Design 2	Array=REG	Unroll_times=all	Unroll_times=0
Design 3	Array=REG	Unroll_times=all	Unroll_times=all

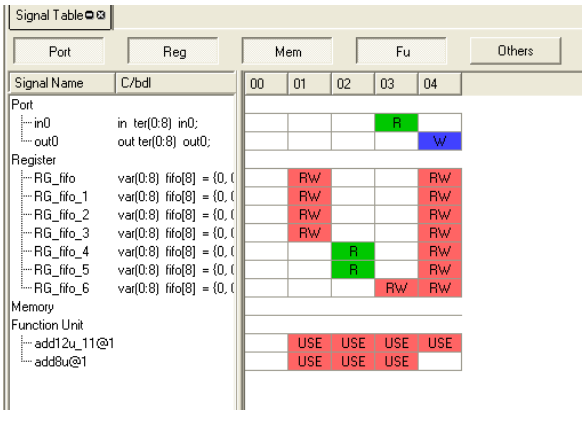
- Review the report sheet for details about which pragmas to change.

3. New design with different Area and latency characteristics is generated and plotted in the synthesis window



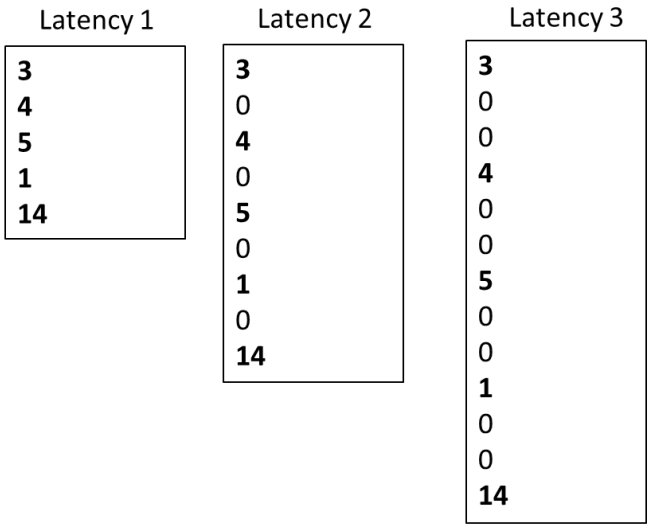
- Opening the schematic viewer and signal table to confirms that a design of different structure and performance has been generated.





- 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.



[END]