ECEN 468   Advanced Logic Design
Department of Electrical and Computer Engineering
Texas A&M University

(Lab exercise created by Jaeyeon Won and Jiang Hu)

Lab 7

Design of SRAM and Synthesis

Purpose:

VCS is a simulation tool for hardware design. We will use this software to simulate functional modules and netlists. Design Analyzer is the graphic interface to the Synopsys family of logic synthesis tools. This lab is to familiarize you with the basics of synthesis using Design Analyzer through design of a simple SRAM cell. Also, we will implement 256K x 8bits SRAM which will be used later for the entire labs.

Preparation:

1. General Design Flow

The design will begin in this lab with the description of a SRAM cell using Verilog. In the following labs we will design other parts for further goal. For every labs, we will design a separate module of our whole system, and simulate it with testbench. Finally, we synthesize module which is designed using Verilog to generate the gate level description of the desired circuit. Finally, we will simulation again with netlist and standard delay format.

2. Brief introduction to SRAM as a memory element.

We will implement a Static Random Access Memory (SRAM) which can be implemented with a storage cell structure that does not require refresh. Therefore, it operates faster than Dynamic Random Access Memory, and used as fast-cache memory in a computer. As a starting point of our implementation, we will implement a simple SRAM cell. The module will be done (1)functional simulation, (2)synthesis and (3)gate simulation with netlist. Also, we will implement 256K SRAM which will be used for other labs. For 256K SRAM, we will do only functional simulation.

The basic SRAM cell represented by the block diagram symbol in Figure 1 has active-low inputs for cell select (CS_b) and write enable (WE_b). The cell select signal is generated by a decoder that selects among multiple cells in the same system. Note the absence of a clock singal. Storage registers and register files are implemented by flip-flops, but the storage devices of RAMs including SRAM and DRAM are implemented as transparent latches, which support asynchronous storage and retrieval of data and minimize the time that a RAM requires service from a shared bus.

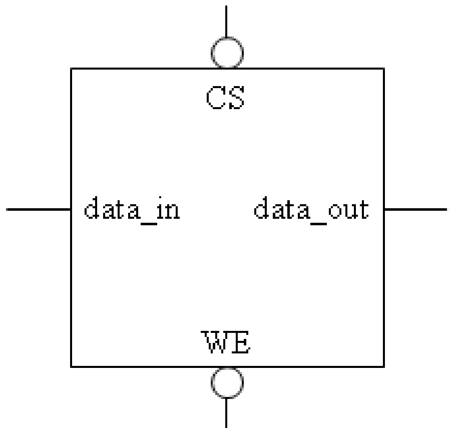


Figure 1 SRAM cell: block diagram symbol

Figure 2 is a functional schematic of a SRAM cell. For more details about SRAM structure, please refer to Lab1. In this lab, the signals WE_b and CS_b are active-low signals which means the signal is active when it is low, logically zero. For example, WE_b and CS_b signals must be ‘0’s for writing operation.

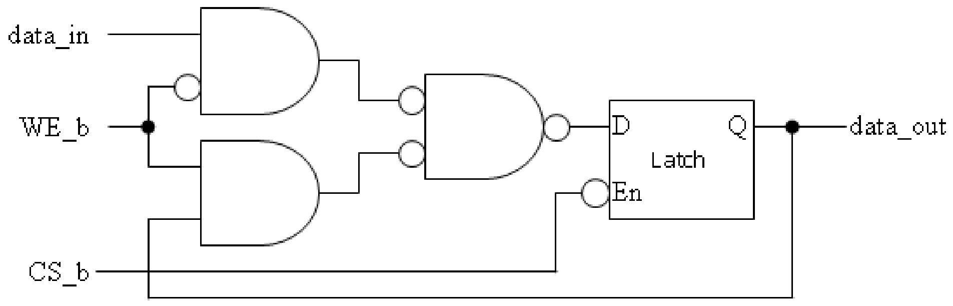


Figure 2 A function schematic of SRAM cell

3. Functional Simulation of a SRAM cell

Once the design is completed it must be tested. The functionality of the design module can be tested by applying a testbench and checking the results. The testbench module can instantiate the design module and directly drive the signals in the design module. The testbench can be complied along with the design module. At the end of compilation the simulation results will be displayed.

Download 'Lab7_codeA.tar.gz' file from the lecture website. The file contains SRAMCELL.v, SRAMCELL_tb.v, generic.sdb, osu018_stdcells.db and osu018_stdcells.v. You will design modules in these files after you decompress the file.

If the script *"source /softwares/setup/synopsys/setup.synopsys.tcsh"* is already in your .cshrc file, go to step a.

Open *".cshrc"* file(it is hidden by default) in your home directory. You can use any TXT editor such as vi, emacs, pico and etc.
Add *"source /softwares/setup/synopsys/setup.synopsys.tcsh"* to *".cshrc"* and save it.
Execute *"source .cshrc"*.
This file is executed automatically whenever you login to Linux server.

a. Make working folder for this lab.

```
HOME]$ mkdir ECEN468/Lab7/SRC1                    (make work folder)
HOME]$ cd ECEN468/Lab7/SRC1                       (move to work folder)
Move the file 'Lab7_codeA.tar.gz' into the working folder
ECEN468/Lab7/SRC1]$ tar xvfz Lab7_codeA.tar.gz    (decompress the file)
```

Please check the files depressed. It should contain the files below.
SRAMCELL.v, SRAMCELL_tb.v, generic.sdb, osu018_stdcells.db, osu018_stdcells.v

b. Insert your code into module files (SRAMCELL.v). You may refer to an example in this manual.
c. Your should use the testbench given (SRAMCELL_tb.v).

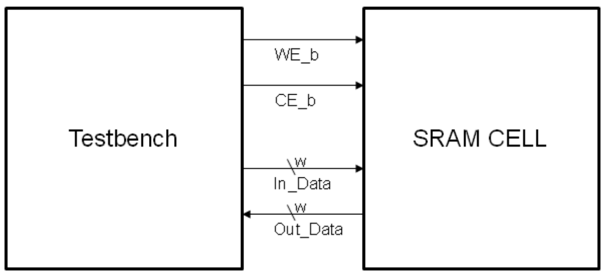


Figure 3 Simulation Pin mapping

d. Start Functional Simulation

```
ECEN468/Lab7/SRC1]$ vcs SRAMCELL_tb.v
```

If it shows compile errors, please check your codes again (go to step b or c). If compilation is complete, go to next step to get the results.

```
ECEN468/Lab7/SRC1]$ simv
```

If it is not same to your estimated results, please check your codes again (go to step b or c). Please refer to step e for your debug process. **If the result is correct, copy the output results and save them into a new file. You will need to submit the file.**

e. How to debug

To view graphical waveform of signals through WaveView, use this code in your testbench. Also you can use $monitor or $display commands to print messages in your codes.

```
initial
begin
        $dumpfile (“wave.dump”);
        $dumpvars(0, stimulus);
 end
```

4. Starting design analyzer

To start design analyzer, please follow steps. For your convenience, please start *design_analyzer* from separate directory with verilog files(*.v) because it generates a lot of intermediate files while it goes further.

Make directory for design analyzer. Now, you should be in ECEN468/Lab7/SRC1.

```
ECEN468/Lab7/SRC1]$ mkdir DV_WORK                  (make work folder)
ECEN468/Lab7/SRC1]$ cd DV_WORK                     (move to the work folder)
ECEN468/Lab7/SRC1/DV_WORK]$ design_vision &     (execute Design Analyzer)
```

5. Procedures of synthesis process:

5.1. Procedure to link the libraries given with the synthesis process:

a. After you begin *Design_Analyzer* , choose "*Setup-->Defaults*". A window pops up.

b. Fill in *osu018_stdcells.db* in "*Link Library*". (Delete all other library files)

c. Fill in *osu018_stdcells.db* in "*Target Library*". (Delete all other library files)

d. Fill in *generic.sdb* in the "*Symbol Library*" (Delete all other library files)

e. Click *Ok*.

5.2. Load a design

Analyze, *Elaborate* are commands provided by Design Compiler. They are available through the *File* menu in *Design Analyzer*.

The *Analyze* command reads a VHDL or Verilog file, checks for proper syntax and synthesizable logic, and stores the design in an intermediate format.

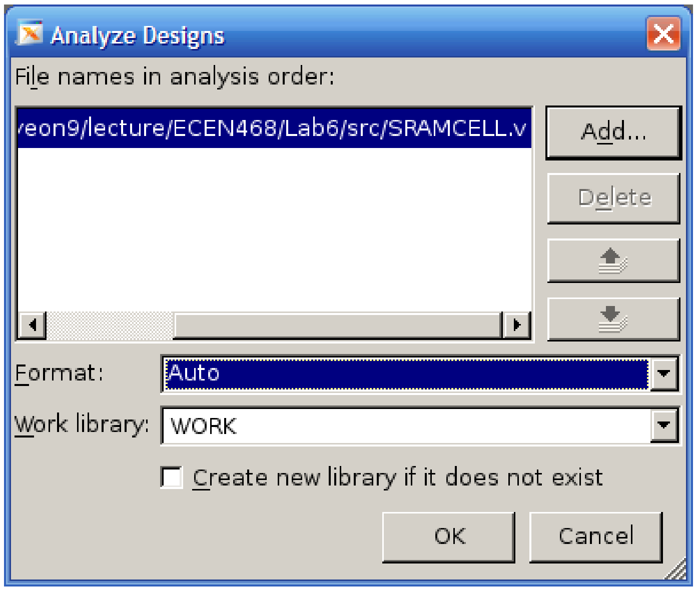
The *Elaborate* command creates a design from the intermediate format produced by *Analyze*. It replaces the HDL operators with synthetic operators and determines the correct bus sizes.

In this lab, *Analyze* and *Elaborate* are used to load the files.

5.3. Detailed procedures:

a) Analyze File

Select *"File->Analyze"*.
The Analyze File window appears as shown in Figure 4. Change the Format to **Auto**.
Click Add button, and select *SRAMCELL.v* (Top module).
Work library to **WORK**.
Click *Ok*.



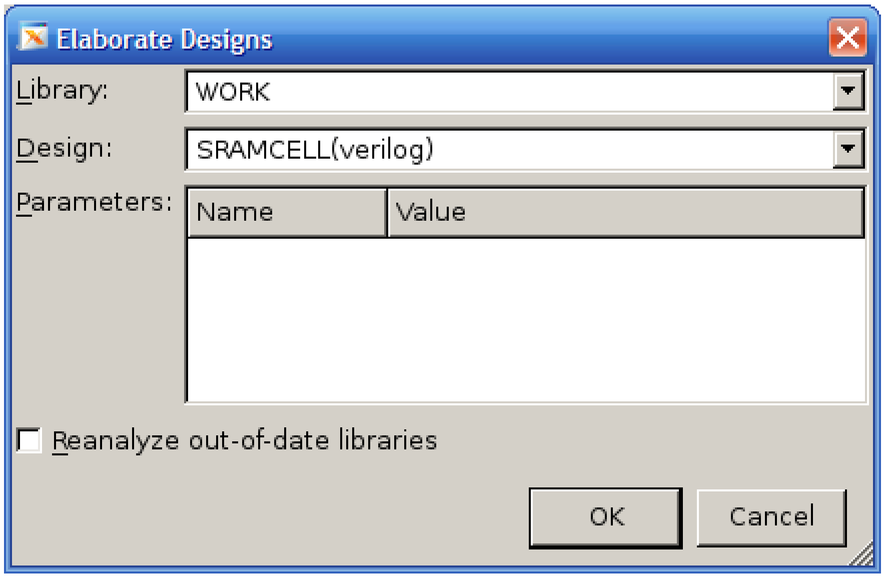
File names in analysis order:
veon9/lecture/ECEN468/Lab6/src/SRAMCELL.v
Format: Auto
Work library: WORK
Create new library if it does not exist

Figure 4 Analyze File window

b) Elaborate Design

Select *"File->Elaborate"*. The Elaborate Design window appears.

Choose the *work* library in the Library list.
Select the *SRAMCELL* (Top module) in the Design list.
Click *Ok*.



Elaborate Designs

Library: WORK

Design: SRAMCELL(verilog)

Parameters:

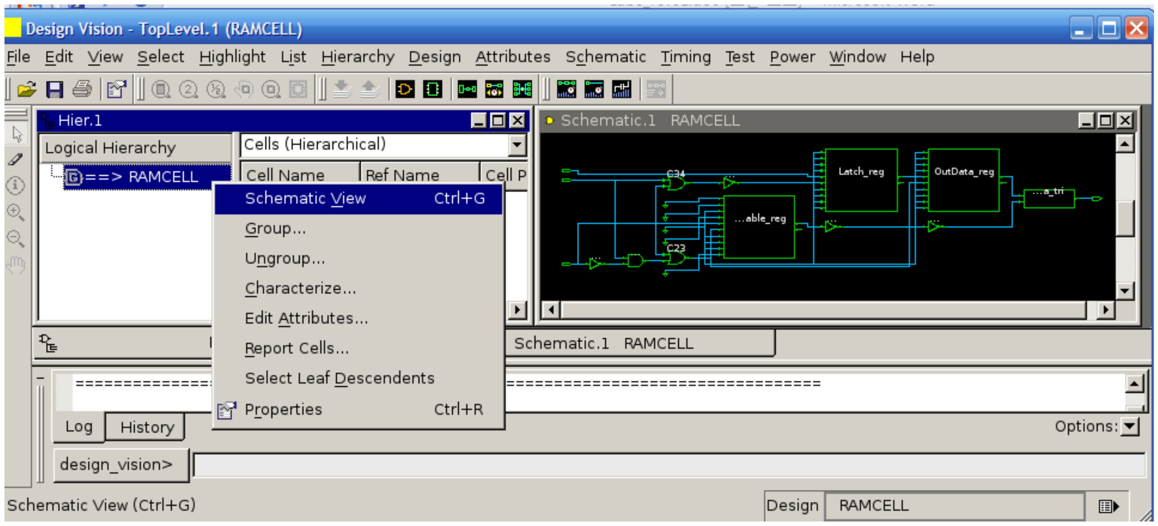
Name	Value

☐ Reanalyze out-of-date libraries

Figure 5 Elaborate Design window

c) Print Schematic

a) Select *SRAMCELL* (Top Module) in the Design View.
b) Click right button of mouse. And, generate the Schematic View.
c) Select File-Print.
d) Choose 'Print to File (PDF)' as printer name. And remember the directory in the output file.
e) Click Print. Then, you will see *.pdf in the directory.

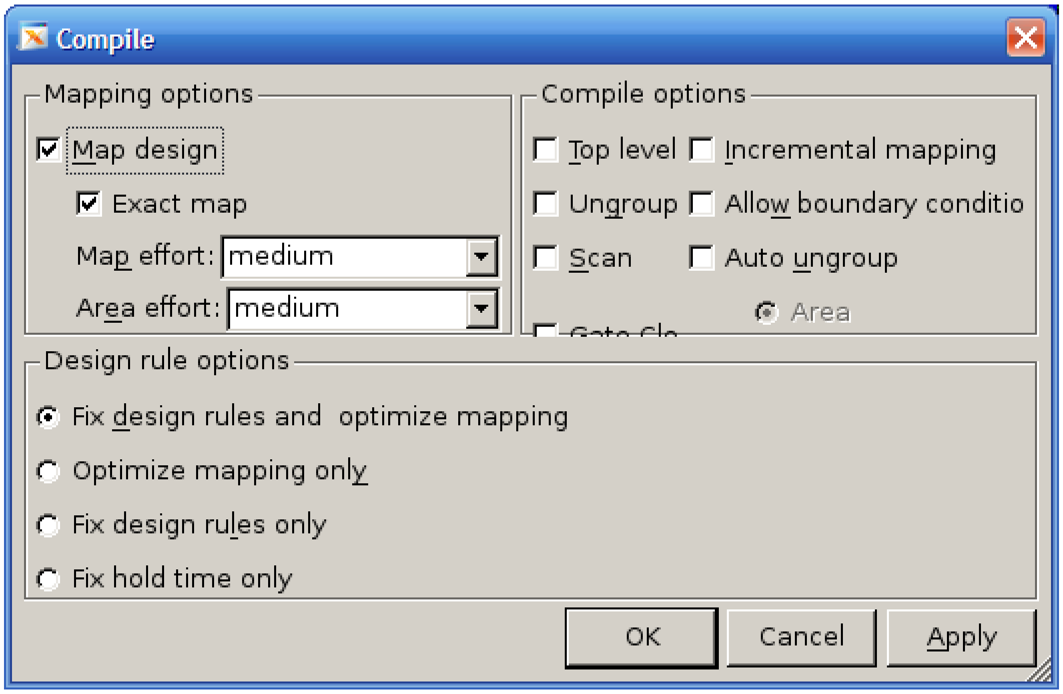


Design Vision - TopLevel.1 (RAMCELL)

Figure 6 Schematic View

d) Synthesize the modules

Use ***Design-Compile Design***. And Click *OK*.


Compile

Mapping options
- [✓] Map design
- [✓] Exact map
- Map effort: medium
- Area effort: medium

Compile options
- [ ] Top level
- [ ] Incremental mapping
- [ ] Ungroup
- [ ] Allow boundary conditio
- [ ] Scan
- [ ] Auto ungroup
- ● Area
- [ ] Gate Clo

Design rule options
- ● Fix design rules and optimize mapping
- ○ Optimize mapping only
- ○ Fix design rules only
- ○ Fix hold time only

Figure 7 Compile Window

e) Save the optimized verilog netlist.

Select *TOP* design (SRAMCELL)

Select *"File->Save As"*.

Enter ***SRAMCELL_gate.v*** in the File Name field to save the file.

Choose the **VERILOG (v)** option in Format list.

Check the *Save All Designs in Hierarchy* option.

Make sure you save the synthesized netlist of the design in verilog format.

f) Save the Standard Delay Format (SDF).

Save the design in the form a Standard Delay Format.

Use command "write_sdf SRAMCELL.sdf" in command window of Design Analyzer.

6. Gate Simulation

a) Copy Verilog netlist and sdf file to work folder.

```
ECEN468/Lab7/SRC1/DV_WORK]$ cp SRAMCELL_gate.v ../.  (copy to work folder)
ECEN468/Lab7/SRC1/DV_WORK]$ cp SRAMCELL.sdf ../.     (copy to work folder)
ECEN468/Lab7/SRC1/DV_WORK]$ cd ..                    (move to work folder)
```

b) Follow steps below to make your testbench for gate simulation

Now we will make testbench for gate simulation. For gate simulation, we need SRAMCELL_gate.v (netlist), SRAMCELL.sdf (SDF file), osu018_stdcells.v and testbench.

Please make sure those files should be in the current working folder. If not, please let TA know.

```
ECEN468/Lab7/SRC1]$ cp SRAMCELL_tb.v SRAMCELL_gate_tb.v
```

Please check that the below lines are in SRAMCELL_gate_tb.v

On the top of the file,

```
`timescale 1ns/10ps
`include “SRAMCELL_gate.v”
`include “osu018_stdcells.v”
```

On the bottom of the file, **(This line should be inside module)**

```
Initial
        $sdf_annotate(“SRAMCELL.sdf”, SRAMCELL_01);
```

Delete the line `include “SRAMCELL.v” in your testbench
Change the name of dump file to “wave_gate.dump” in your testbench

c) Start gate simulation

```
ECEN468/Lab7/SRC1]$ vcs SRAMCELL_gate_tb.v
```

If there is no syntax error, then do next step.

```
ECEN468/Lab7/SRC1]$ simv
```

Then, you will see the text message and get wave_gate.dump file. We sometimes want to see the waveform to debug. Please compare the result with the functional simulation result. If it is not same to your estimation, start simulation again from the functional simulation after you modify design modules.

Now, please show your result to TA within your regular lab hour to get credit for lab evaluation.

7. Functional Simulation of a 256K x 8bits SRAM

a. Make working folder for this lab.

```
HOME]$ mkdir ECEN468/Lab7/SRC2                   (make work folder)
HOME]$ cd ECEN468/Lab7/SRC2                      (move to work folder)
Move the file ‘Lab7_codeB.tar.gz’ into the working folder
ECEN468/Lab7/SRC2]$ tar xvfz Lab7_codeB.tar.gz   (decompress the file)
```

Please check the files decompressed. It should contain the files below.
SRAM.v, SRAM_tb.v

b. Insert your code into module files (SRAM.v).
c. Your should use the testbench given (SRAM_tb.v).

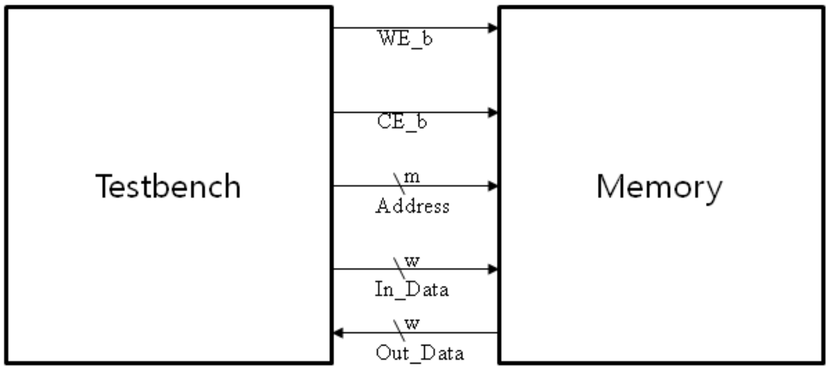


Figure 8 Simulation Pin mapping

d. Start Functional Simulation

```
ECEN468/Lab7/SRC2]$ vcs SRAM_tb.v
```

If it shows compile errors, please check your codes again (go to step b or c). If compilation is complete, go to next step to get the results.

```
ECEN468/Lab7/SRC2]$ simv
```

If it is not same to your estimated results, please check your codes again (go to step b or c). Please refer to step e for your debug process. **If the result is correct, copy the output results and save them into new file. You will need to submit the file.**

Now, please show your result to TA within your regular lab hour to get credit for lab evaluation.

Requirements:

1. General requirements.
   a. It should control SRAM CELL correctly. **(SRAM CELL part)**
      - bCE, bWE, InData, OutData
      - Low Active signals for bCE and bWE

   b. It should control 256K SRAM correctly. **(256K SRAM part)**
      - 256K words (1 Word : 8 bits)
      - Ports : bCE, bWE, Addr, InData, OutData
      - Low Active signals for bCE and bWE

   c. Detailed comments.

   d. Please do not change the names of the signals given and the other functions not mentioned in the report. (You will lose some points if you change any names of the input or output signals.)

   e. You may modify the test-bench given to get the results only if it is resaonable. Please write detailed comments if you have modified.

   f. Late penalty : 20% of total score will be deducted on each subsequent weekday after due date.

2. Submit hardcopy of your report to TA. The report should include contents below
   (SRAM CELL part)
   a. SRAMCELL.v with detailed comments (10 points)
   b. Text result and waveform result(captured) of the functional simulation with analysis(3 points)
   c. Text result and waveform result(captured) of the gate simulation with analysis (3 points)
   d. Schematic file (3 points)
   e. Only the parts modified in SRAMCELL_tb.v and SRAMCELL_gate_tb.v, if you have modified. (1 points)

   (256K SRAM part)
   f. SRAM.v with detailed comments (10 points)
   g. Text result and waveform result(captured) of the functional simulation with analysis(3 points)
   h. Only the parts modified in SRAM_tb.v, if you have modified. (2 points)

Note : Please send these files to TA's email address. (5 points)

SRAMCELL.v, SRAMCELL_tb.v, SRAMCELL_gate_tb.v, wave.dump, wave_gate.dump
SRAMCELL_gate.v, schematic file(*.pdf), SRAMCELL.sdf
SRAM.v, SRAM_tb.v

```
// ==================================
// 4-bits RIPPLE CARRY ADDER
// ==================================
module RCA4(INA, INB, CIN, SUM, COUT);                // MODULE DECLARATION

    output [3:0] SUM;                                 // OUTPUT DECLARATION
    output COUT;                                      // OUTPUT CARRY DECLARATION

    input [3:0] INA, INB;                             // INPUT DECLARATION
    input CIN;                                        // INPUT CARRY DECLARATION

    wire [2:0] TEMPC;                                 // TEMPORARY WIRES

    // Connect modules
    FA FA1(INA[0], INB[0], CIN,      TEMPC[0], SUM[0]);             // FULL ADDER1
    FA FA2(INA[1], INB[1], TEMPC[0], TEMPC[1], SUM[1]);             // FULL ADDER2
    FA FA3(INA[2], INB[2], TEMPC[1], TEMPC[2], SUM[2]);             // FULL ADDER3
    FA FA4(INA[3], INB[3], TEMPC[2], COUT, SUM[3]);                 // FULL ADDER4

endmodule

module FA(INA, INB, CIN, COUT, SUM);                                // 1BIT- FULL ADDER

    input INA, INB, CIN;                                            // INPUT DECLARATION
    output COUT, SUM;                                               // OUTPUT DECLARATION

    // DATA FLOW MODELING
    assign SUM = (INA^INB) ^ CIN;                                   // output Sum
    assign COUT = (INA&INB) | ((INA^INB)&CIN);                      // output Carry Out

endmodule
```

```
// ================================
// 4-bits RIPPLE CARRY ADDER - testbench
// ================================
// TESTBENCH
`timescale 1ns/10ps
`include "RCA.v"

module stimulus;
    reg [3:0] INA, INB;                                         // INPUT VALUES
    reg CIN;                                                    // INPUT CARRY

    wire [3:0] SUM;                                             // OUTPUT
    wire COUT;                                                  // OUTPUT CARRY

    RCA4 RCA4_01(INA, INB, CIN, SUM, COUT);                     // Connect to TOP MODULE

    initial                                                     // Initial Conditions
    begin
        CIN = 1'b0;
        INA = 4'd0;
        INB = 4'd0;
    end

    initial
    begin
        #90   INA = 4'dx;   INB = 4'dx;     // A=x, B=x
        #10   INA = 4'd1;   INB = 4'd6;     // A=1, B=6
        #90   INA = 4'dx;   INB = 4'dx;     // A=x, B=x
        #10   INA = 4'd1;   INB = 4'd7;     // A=1, B=7
        #90   INA = 4'dx;   INB = 4'dx;     // A=x, B=x
        #10   INA = 4'd1;   INB = 4'd8;     // A=1, B=8
        #300 $stop;                                             // stop
        end

    // Dump signals to view waveform
    initial
    begin
        $dumpfile ("wave.dump");
        $dumpvars (0, stimulus);
    end

    initial                                                     // output to text
        $monitor($time, " INA=(%d)+INB=(%d) => SUM=%d, COUT:%b", INA, INB, SUM, COUT);
endmodule

// ================================
// Simulation result
// ================================

                     0 INA=( 0)+INB=( 0) => SUM= 0, COUT:0
                    90 INA=( x)+INB=( x) => SUM= x, COUT:x
                   100 INA=( 1)+INB=( 6) => SUM= 7, COUT:0
                   190 INA=( x)+INB=( x) => SUM= x, COUT:x
                   200 INA=( 1)+INB=( 7) => SUM= 8, COUT:0
                   290 INA=( x)+INB=( x) => SUM= x, COUT:x
                   300 INA=( 1)+INB=( 8) => SUM= 9, COUT:0
```