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 9

Design of System Bus

Purpose:

In this lab, we will design System Bus to connect modules with Verilog, and will use Design Analyzer as a tool for synthesis as well as Lab6. Also, we will use this module to connect SRAM part and UART which are done in previous labs.

Preparation:

1. Brief introduction to System Bus.

In computer architecture, a bus is a sub-system that transfers data between components inside a computer, or between computers. Early computer buses were literally parallel electrical wires with multiple connections, but the term is now used for any physical arrangement that provides the same logical functionality as a parallel electrical bus. By using system bus, the number of data pins used to connect all devices can be decreased sharply. It is easy to see in case of the system includes multiple CPUs and memory devices.

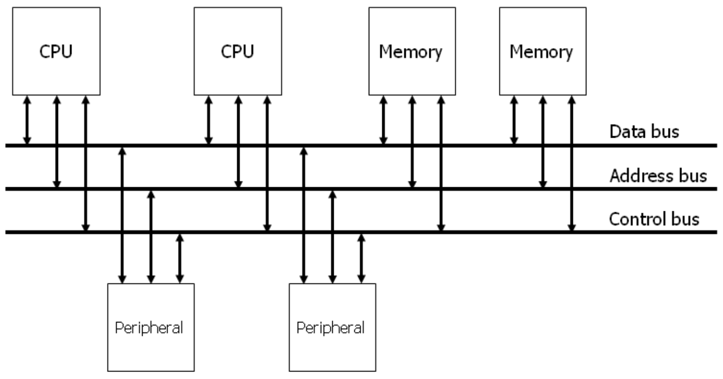


Figure 1 System Bus

An address bus is a computer bus (a series of lines connecting two or more devices) that is used to specify a physical address. When a processor or DMA-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus). The width of the address bus determines the amount of memory a system can address. A control bus is part of a computer bus, used by CPUs for communicating with other devices within the computer. While the address bus carries the information on which device the CPU is communicating with and the data bus carries the actual data being processed, the control bus carries commands from the CPU and returns status signals from the devices, for example if the data is being read or written to the device the appropriated line (read or write) will be active.

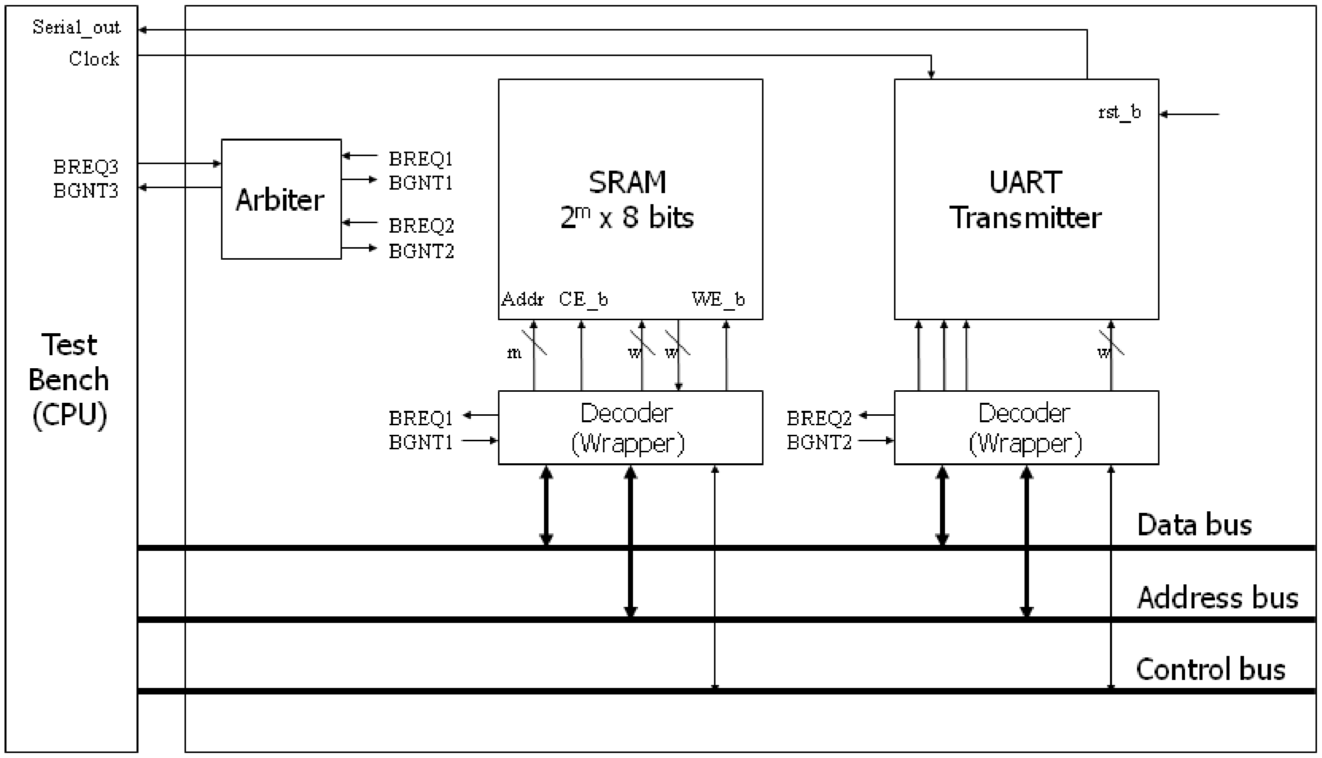


Figure 2 Connections among memory, peripheral and system bus

Whenever a device needs to communicate with another device connected to the mainboard, it must to so over the bus. Because the bus is shared amongst all the devices, a method for deciding which device gets to use the bus must be used. The method used to determine who gets access to the bus and when is referred to as bus arbitration. The bus arbitration mechanism is designed so that high priority devices like the processor and RAM get first access to the bus, while other device (disks, video cards, sound cards etc.) get lower priority, and often have to wait to access the bus. There are several modes can used for the arbitration. Among them, we will use centralized fixed-priority arbitration which has simple circuit to implement and shown in Figure 3.

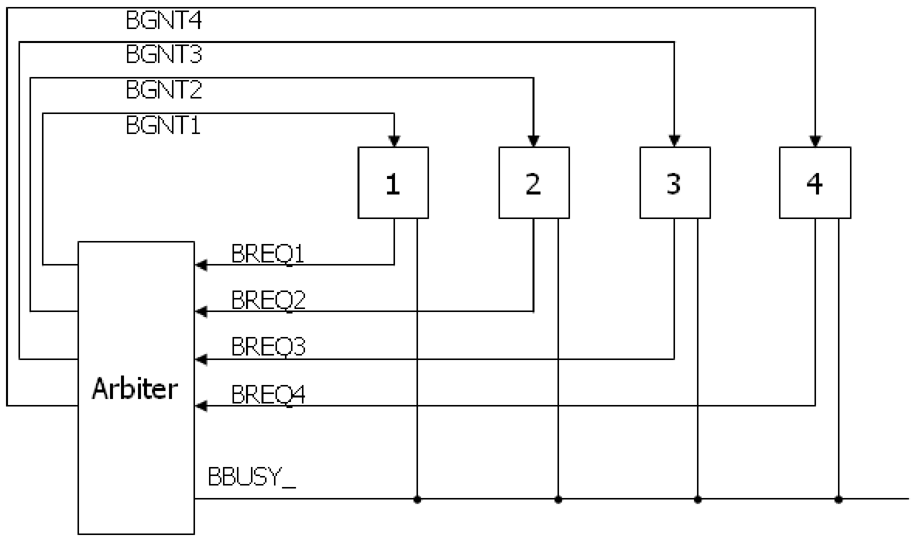


Figure 3 Circuit of Centralized fixed-priority arbitration

In Figure 3, there are three types of signals used. BREQ signals are generated by CPU, memory and peripherals. If they want to use bus to read or write, they must be allocated from the arbiter. To get the right to use the bus, it sends request signal which is BREQ to the arbiter. The arbiter sends grant signals to the device which sent request signals if the bus is free and can be used. If two or more request signals are generated, the arbiter computes their priorities and send only one grant signal to the device has higher priority. If the bus is free, the status of BBUSY_ signal is high (logic 1). If any device gets the right to use the bus, they read data from the bus or write data to the bus while it makes BBUSY_ signal being low state (logic 0). After it completes its operation related to bus, release the BBUSY_ signal which will go high state back. For example, (1) if master 3 sends the request signal (BREQ3) to the arbiter while master 1 is using the bus, (2) the arbiter sends the grant signal (BGNT3). (3) When master 1 completes its operation and make BBUSY_ signal inactive (goes high), (4) master 3 makes BBUSY_ signal active (goes low) and starts to read or write to the bus.

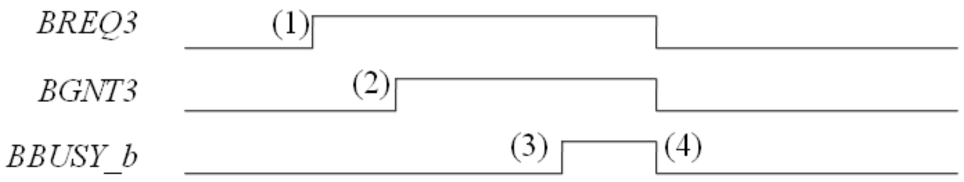


Figure 4 An example of the process for arbitration

We assume that the test-bench has the highest priority and the UART has the lowest priority. **We can also see the module attached between SRAM/UART and system bus. It is called decoder or wrapper. Small size system has a few number of pins to control other devices, but large size system has a lot of pins to control all devices, so the number of control pins must be connected with devices will be huge. Thus, control signals can be also included in the system bus. The width**

of address bus depends on the application we design. We assume that the width of the address bus is 32 bits. The 4 most significant bits are used for identification number of devices. The ID of CPU which is a test-bench in our design is 0011, the ID of SRAM is 0001 and 0010 for UART transmitter. The remaining is used for control signals and address signals for memory. Figure 5 and Figure 6 shows the circuit for decoder and address map.

ID[31:28]	reserved[27:20]	CE_b[19]	WE_b[18]	Addr[17:0]

Figure 5 address map to control memory

ID[31:28]	reserved[27:4]	T_byte[2]	Byte_ready[1]	Load_XMT_datareg[0]

Figure 6 address map to control UART

3. Functional Simulation

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 'Lab9_code.tar.gz' file from the lecture website. The file contains Arbiter.v, Mainsystem.v, WRAP_SRAM.v, WRAP_UART.v, tb.v, generic.sdb, osu018_stdcells.v and osu018_stdcells.db. You will design modules in these files after you decompress the file.

a. Make working folder for this lab.

```
HOME]$ mkdir ECEN468/Lab9/SRC                  (make work folder)
HOME]$ cd ECEN468/Lab9/SRC                     (move to work folder)
Move the file 'Lab9_code.tar.gz' into the working folder
ECEN468/Lab9/SRC]$ tar xvfz Lab9_code.tar.gz   (decompress the file)
```

Please check the files decompressed. It should contain the files below.
Arbiter.v, Mainsystem.v, WRAP_SRAM.v, WRAP_UART.v, tb.v, VirtSRAM.v
generic.sdb , osu018_stdcells.v, osu018_stdcells.db

b. Copy SRAM.v that you designed from Lab1 to this lab folder.
c. Copy Control_Unit.v, Datapath_Unit.v and UART_XMTR.v that you designed from Lab8 to this lab folder.

d. Insert your code into module files. Your module should satisfy the requirements (WRAP_SRAM.v, WRAP_UART.v). Regarding the inout ports, you may refer to Mainsystem.v file.
e. You should use the test-bench given.

f. Start Functional Simulation

```
ECEN468/Lab9/SRC]$ vcs tb.v
```

If it shows compile errors, please check your codes again (go to step d) or make sure that your previous modules are correct. You may sometimes need to modify your previous modules to acquire correct results. If compilation is complete, go to next step to get the results.

```
ECEN468/Lab9/SRC]$ simv
```

If it is not same to your estimated results, please check your codes again (go to step d).

Please show your result to TA within your regular lab hour to get credit for lab evaluation.

g. Preparation for synthesis

In this lab, we will synthesize whole modules except SRAM and generated its netlist to do gate-simulation. The 256K SRAM is used for this system, and too large to synthesize. In general, memory modules are replaced in a layout process with standard memory modules are provided by company. Thus, we will synthesize all modules except SRAM, and connect the module to the netlist after synthesis. You will see virtSRAM.v. The module should have exactly same in and out ports to your SRAM module. There is one difference in its behavior which does not have large size of internal memory and simple behavior. We will use this module instead of SRAM.v for synthesis.

Open Mainsystem.v file. And modify `include “SRAM.v” to `include “virtSRAM.v”.

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/Lab9/SRC.

```
ECEN468/Lab9/SRC]$ mkdir DV_WORK                    (make work folder)
ECEN468/Lab9/SRC]$ cd DV_WORK                       (move to the work folder)
ECEN468/Lab9/SRC/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. Choose *osu018_stdcells.db* in "*Link Library*". (Delete all other library files)
c. Choose *osu018_stdcells.db* in "*Target Library*". (Delete all other library files)
d. Choose *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.

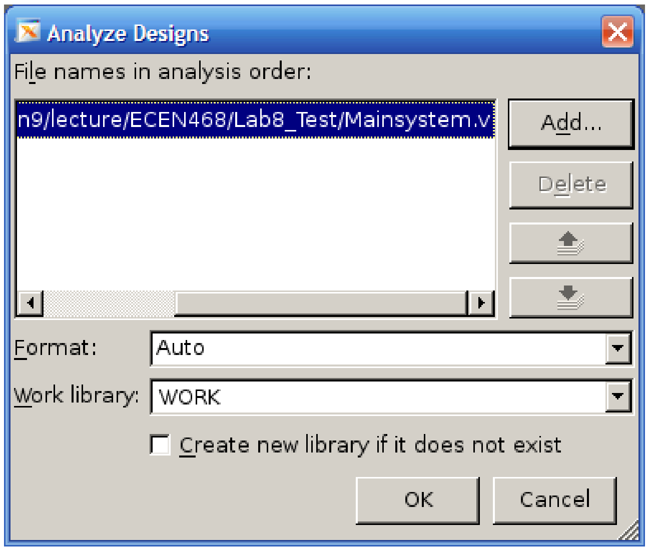
The *Elaborate* command creates a design from the intermediate format produced by *Analyze*. It replaces the HDL operators with synthetic operators and determines the correct bus sizes.

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 9. Change the Format to **Auto.**
Click Add button, and select *Mainsystem.v* (Top module).
Work library to **WORK**. Click *Ok*.



File names in analysis order:
n9/lecture/ECEN468/Lab8_Test/Mainsystem.v
Format: Auto
Work library: WORK
☐ Create new library if it does not exist

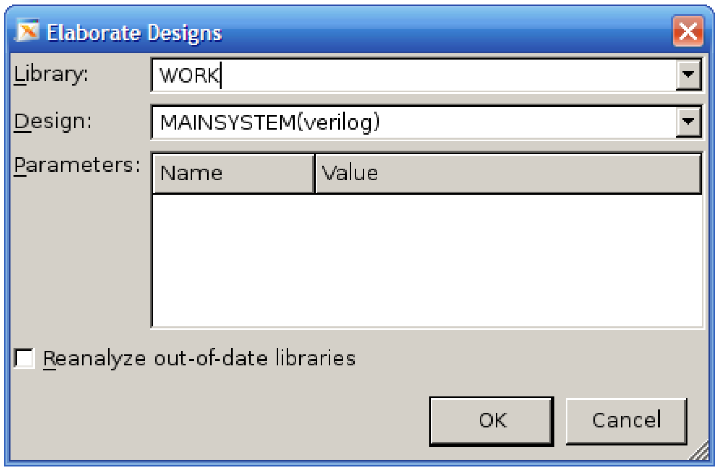
Figure 9 Analyze File window

b) Elaborate Design

Select *“File->Elaborate”*. The Elaborate Design window appears.

Choose the *work* library in the Library list.

Select the *MAINSYSTEM* (Top module) in the Design list. Click *Ok*.



Library: WORK

Design: MAINSYSTEM(verilog)

Parameters:

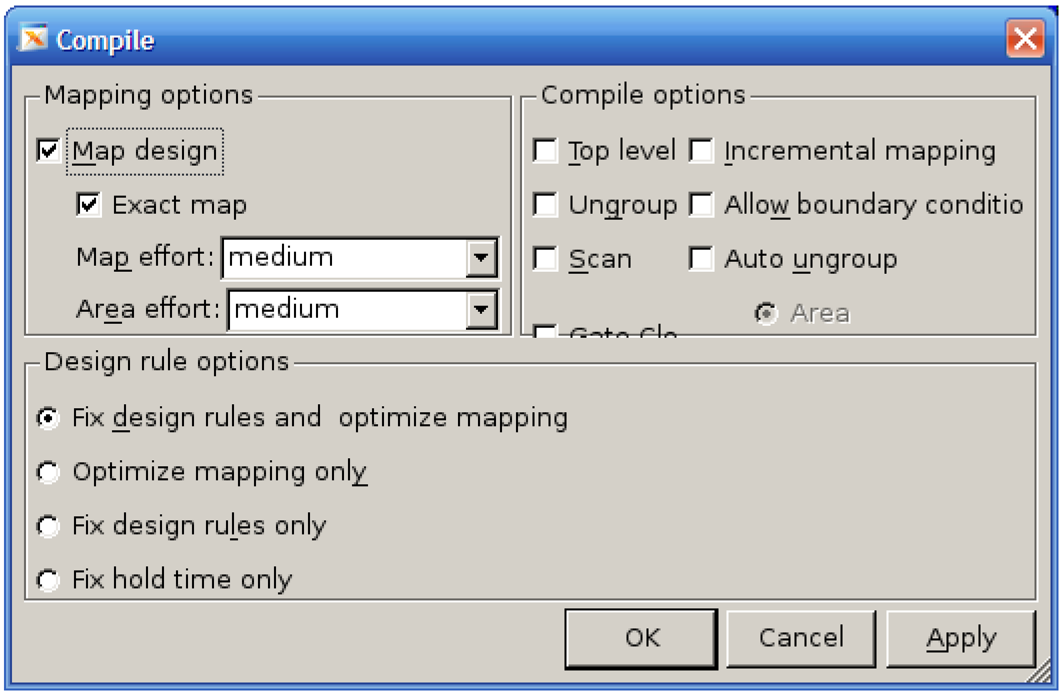
Name	Value

☐ Reanalyze out-of-date libraries

Figure 10 Elaborate Design window

c) Synthesize the modules

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



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 12 Compile Window

d) Save the optimized verilog netlist.

Select *TOP* design (MAINSYSTEM)

Select *"File->Save As"*.

Enter ***Mainsystem_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.

e) Save the Standard Delay Format (SDF).

Save the design in the form a Standard Delay Format.

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

6. Gate Simulation

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

```
ECEN468/Lab9/SRC/DV_WORK]$ cp Mainsystem_gate.v ../.     (copy it to work folder)
ECEN468/Lab9/SRC/DV_WORK]$ cp Mainsystem.sdf ../.        (copy it to work folder)
ECEN468/Lab9/SRC/DV_WORK]$ cd ..                          (move to work folder)
```

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

Now we will make testbench for gate simulation. For gate simulation, we need Mainsystem_gate.v (netlist), Mainsystem.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.

c) We used virtSRAM.v instead of SRAM.v. Open Mainsystem_gate.v file, then you will see the SRAM module which has been synthesized with "virtSRAM.v". We will use original SRAM module in SRAM.v file for gate simulation. Delete the SRAM module in Mainsystem_gate.v because we will not use this SRAM module synthesized.

```
ECEN468/Lab9/SRC]$ cp tb.v gate_tb.v (copy new testbench)
```

Please check that the below lines are in gate_tb.v

On top of the file,

```
`timescale 1ns/10ps
`include "Mainsystem_gate.v"
`include "osu018_stdcells.v"
`include "SRAM.v"                 // We will use original SRAM module.
```

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

```
Initial
        $sdf_annotate("Mainsystem.sdf", MAINSYSTEM_01);
```

Delete the line `include "Mainsystem.v" in your testbench
Change the name of dump file to "wave_gate.dump" in your testbench

d) Start gate simulation

```
ECEN468/Lab9/SRC]$ vcs gate_tb.v
```

If you see the error below, please make changes at module WRAP_SRAM and WRAP_UART in Mainsystem_gate.v file.

Error : The 32-bit expression "Addressbus" is connected to 20-bit port "AddressBus".
    tri [19:0] AddressBus → tri [31:0] AddressBus;

Error : The 32-bit expression "Addressbus" is connected to 4-bit port "AddressBus".
    tri [31:28] AddressBus → tri [31:0] AddressBus;

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

```
ECEN468/Lab9/SRC]$ 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.

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 and UART correctly.
   b. Detailed comments.
   c. 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.)
   d. You may modify the test-bench given to get the results only if it is resaonable. Please write detailed comments if you have modified.
   e. 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
   a. Text result of the functional simulation (5 points)
   b. Waveform result(screen captured) of the function simulation with analysis. (5 points)
   c. Text result of the gate simulation (5 points)
   d. Waveform result(screen captured) of the gate simulation with analysis. (5 points)
   e. WRAP_SRAM.v and WRAP_UART.v with detailed comments (8 points)
   f. Only the parts modified in tb.v and gate_tb.v, if you have modified. (2 points)
   g. Calculate the maximum clock speed of the test-bench you used if you have changed clock period of the testbench. Otherwise, calculate the frequency given in the testbench. (5 points) (You will get scores based on the maximum clock speed.)

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

Control_Unit.v, Datapath_Unit.v, UART_XMTR.v, SRAM.v, WRAP_SRAM.v, WRAP_UART.v
Synthesis results : Mainsystem_gate.v(final netlist for synthesis), Mainsystem.sdf
Simulation results : wave.dump, wave_gate.dump
