CS 207 Digital Logic - Spring 2020
Lab 4

James Yu

Mar. 11, 2020

Objective

1. Observe waveform using GTKwave.
2. Try behavioral modelling.

Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension ".v".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`
4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

1 GTKwave

In all previous exercises, we use log messages to observe the value changes of outputs with respect to inputs. In this lab session, we will learn a new way of observing these changes — waveform.

Recall the Verilog code in page 30 of the lecture notes. We add several new lines (4–9) to make a new file:

gtkwave.v

```
1 module initial_begin_end();
```

```
reg clk,reset,enable,data;

initial begin
//Output vcd file name, can be specified at will
$dumpfile("wave.vcd");
//The parameter needs to set as module name
$dumpvars(0, initial_begin_end);
end

initial begin
  $monitor("%3t clk=%b reset=%b enable=%b data=%b", $time, clk, reset, enable, data);
  #1   clk = 0;
  #10  reset = 0;
  #5   enable = 0;
  #3   data = 0;
  #1  $finish;
end
endmodule
```

The `$dumpfile` is used to dump the changes in the values of nets and registers in a file that is named as its argument. For example, `$dumpfile("test.vcd");` will dump the changes in a file named `test.vcd`. The changes are recorded in a file called VCD file that stands for value change dump. A VCD (value change dump) stores all the information about value changes. We can not have more than one `$dumpfile` statements in Verilog simulation.

The `$dumpvars` is used to specify which variables are to be dumped (in the file mentioned by `$dumpfile`). The simplest way to use it is without any argument: `$dumpvars;`. In this case, it dumps ALL variables in the current testbench module and in all other modules instantiated by it. The general syntax of the `$dumpvars` include two arguments as

```
$dumpvars(<levels> <, <module_or_variable>>*);
```

We basically can specify which modules, and which variables in modules will be dumped. The simplest way to use this is to set the level to `0` and module name as the top module (typically the top testbench module) as in

```
$dumpvars(0, toptestbench_module);
```

When level is set to `0`, and only the module name is specified, it dumps ALL the variables of that module and all the variables in ALL lower level modules instantiated by this top module. If any module in not instantiated by this top module, then its variable will not be covered.

If you wish to also cover variables of a module that is not instantiated by the top module , you could write

```
$dumpvars(0, toptestbench_module, not_instantiated_module);
```

If we wish to dump the variables only in the top module but not the modules instantiated below it, we could have `1` as its first argument as in

```
$dumpvars(1, toptestbench_module);
```

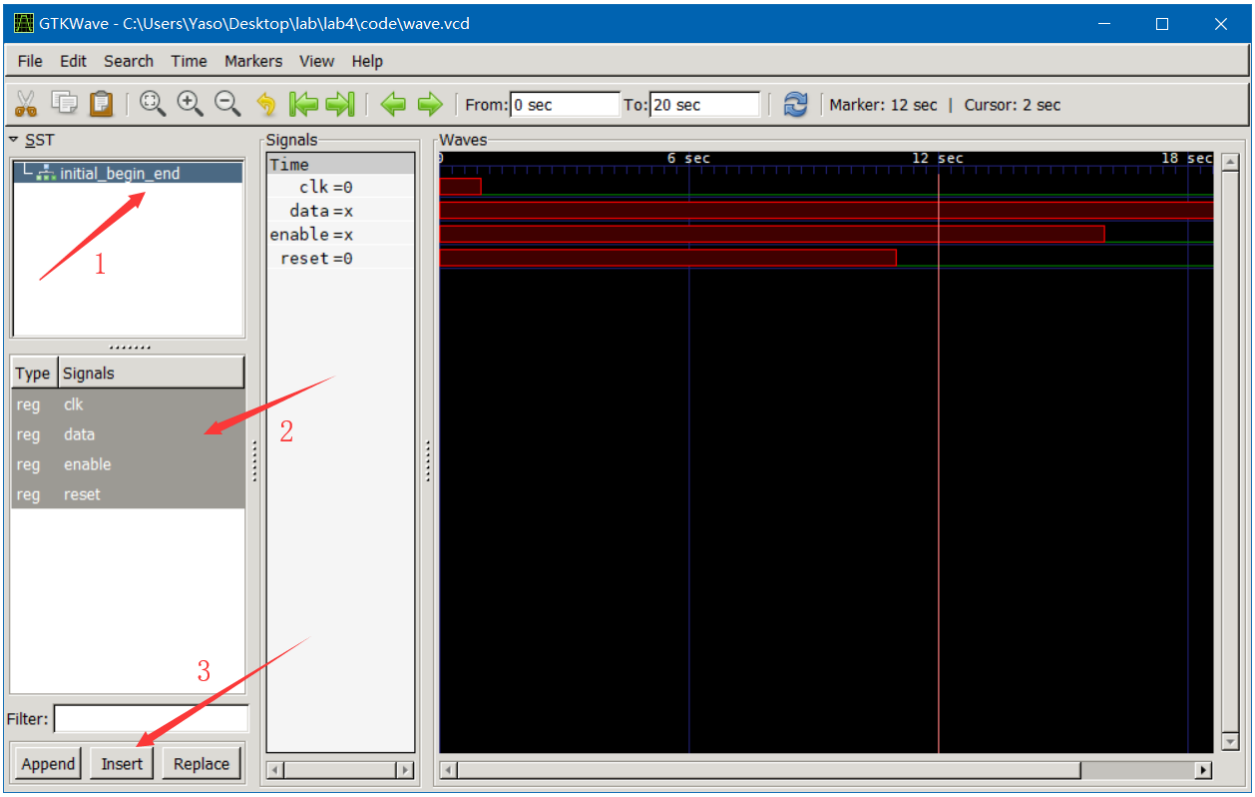
This is helpful when you do not wish to be bothered about the registers and the variables inside the instantiated module. But, during the debug and to find root cause, it may be helpful to use `0` as its first argument and dump all variables.

The following example will dump all variables in the in the module named `toptestbench_module` and the modules one level below it.

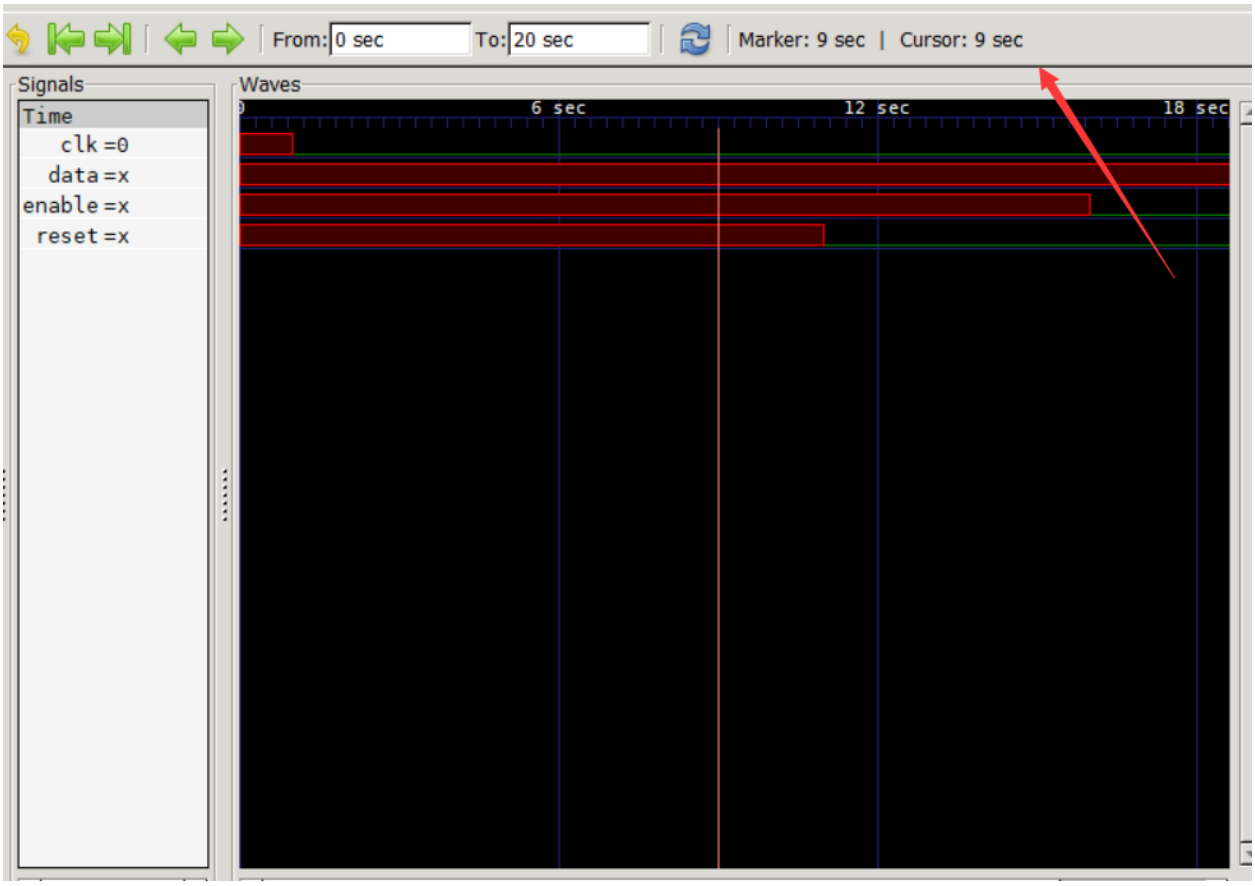
```
$dumpvars(2, toptestbench_module);
```

Value of level more than `2` are rarely used.

After compiling and executing the Verilog code, you will notice a new file `wave.vcd` is created. Open GTKwave and drag this new file into GTKwave. Choose the module, then select the signals to



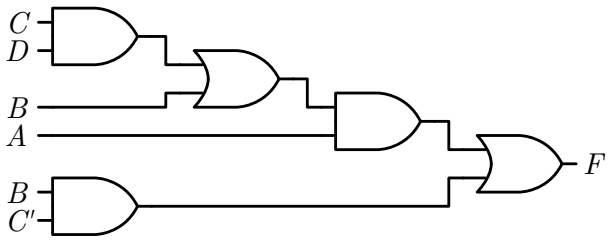
be observed and press `Insert` to show the waveform. You may also drag the vertical line to change the simulation time.



2 Exercise

2.1 Exercise 1

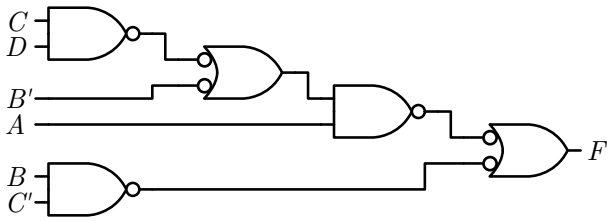
Write two **gate-level** Verilog modules to verify the equivalency of the following two circuits:



The first circuit should follow the design below:

nand1.v

```
module nand1(F, A, B, C, D);
// Module Code
endmodule
```



and save in file `nand1.v`.

The second circuit should follow the design below and only use NAND gate:

nand2.v

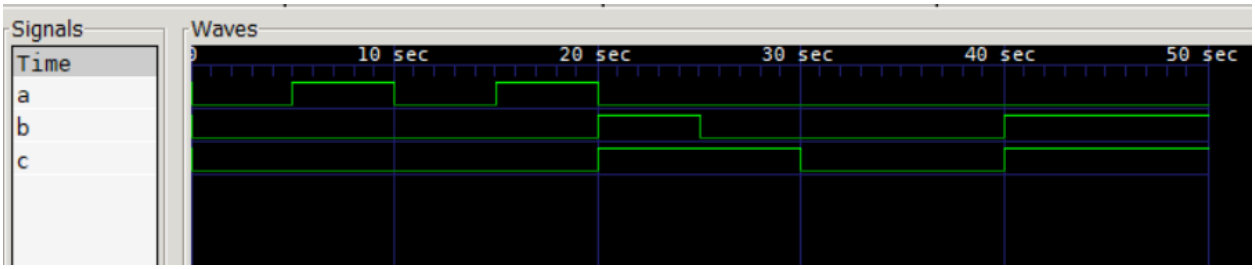
```
module nand2(F, A, B, C, D);
// Module Code
endmodule
```

and save in file `nand2.v`.

> **Assignment**
> Save the source code in **`nand1.v`** and **`nand2.v`**. Upload this file to Sakai under **Assignments → Lab Exercise 4**.

2.2 Exercise 2

Implement a Verilog module to output the following waveform, which stops at time `50`:



The module head should be defined as follows:

waveform.v

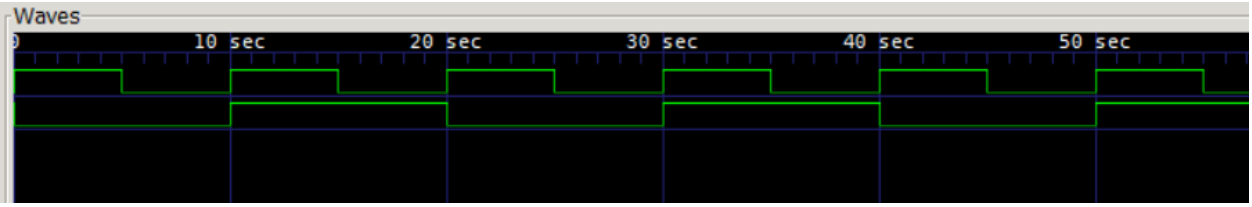
```
module waveform(a, b, c);
output a, b, c;
// Module Code
endmodule
```

Assignment

Save the source code in `waveform.v`. Upload this file to Sakai under **Assignments → Lab Exercise 4**.

2.3 Exercise 3

Implement a Verilog module to develop a clock signal at 0.1Hz. Then use this signal to produce a 0.05Hz with `always` and `posedge` keywords. The waveform looks like the following:



The module head should be defined as follows:

timer.v

```
module timer(a, b);
output a, b;
// Module Code
endmodule
```

Assignment

Save the source code in `timer.v`. Upload this file to Sakai under **Assignments → Lab Exercise 4**.

Assignment

When you finished Lab Exercise 4, there should be four `.v` files in the Sakai system: `nand1.v`, `nand2.v`, `waveform.v`, and `timer.v`.