Lab 6.1

Lab 6.1

- In this lab, you will:
  - Learn the difference between the $*write*, $*monitor*, and $*strobe* statements
  - Learn how strings can be passed around
  - Learn how to initialize a design from a memory image file

Lab 6.1

- In directory ~/lab6.1, you will find a file named "monitor.v". It contains an *always* block monitoring the changes on a register named "R" using the $*display* statement.
- Add a $*monitor* statement that emulates the behavior of the *always* block, in the same *initial* block that defined the global time format. Simulate.
- Explain the differences in the reported changes.
- Replace the $*display* statement with a $*strobe* statement, then simulate again to verify that both monitors are now equivalent.

Lab 6.1

- In the same directory, you will find the following files:
  - rom.v Model of a 16x8 Read-Only memory
  - dumprom.v Dump the content of ROM
- Simulate both files: the entire content of the ROM is unknown.
- Add a parameter, set by default to "rom.dat", to the ROM model.
- Add the necessary statements to initialize the ROM from the file specified by the parameter.
- Create a file named "rom.dat" and provide some values.

Lab 6.1

- Simulate once more and verify that the ROM was properly initialized.
- In that same directory, you will find a file named "dump2rom.v". This module instantiates two ROMs.
- Simulate this new module along with your ROM model: Why is the content of both ROMs identical?

Lab 6.1

- Override the ROM initialization file parameter of each ROM instance with different filenames.
- Create these new files, then simulate again. Verify that each ROM has the proper content.

Lab 6.1: Optional

- Modify the ROM model to use parameters to define its data width, starting address, and ending address.
- Verify your modifications with various widths, starting, and ending addresses.
  - Try width > 32
  - Try starting address > 0
  - Try starting address > ending address

Lab 6.1: Optional

- *Note to Verilog-XL users: At the time these notes were written, XL 2.3.2 seemed to load memories backward if the memory was declared with start address > end address.*

Lab 6.2

Lab 6.2

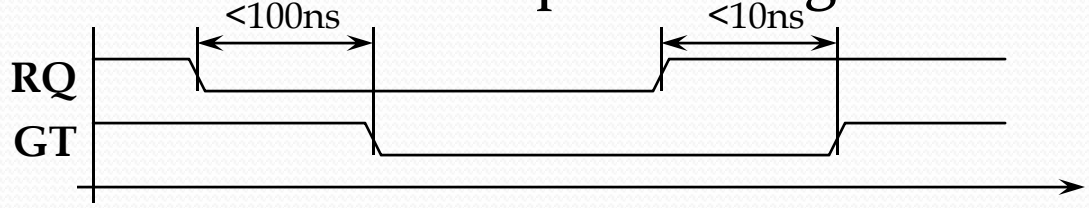
- In this lab, you will:
  - Become familiar with disabled block
  - Learn how to use the fork/join statement

Lab 6.2

- In directory ~/lab6.2, you will find a file named "forever.v". It contains an *initial* block with a forever loop that waits for a register "R" to become '1' at the rising edge of a CLK. Once this is detected, it should print the message "Done", then terminate the simulation.
- Run the simulation to verify if the model performs as expected. Fix any problems you may find.

Lab 6.2

- In the same directory, you will find the following files:
  - arbiter1.v Arbiter #1
  - arbiter2.v Arbiter #2
  - arbiter3.v Arbiter #3
  - testarb.v Tester for arbiter
- Do not look at the source of the arbiters.
- The arbiter is supposed to provide a grant signal within 100ns of a request, and remove that same grant signal within 10ns of the request being removed.



Lab 6.2

- The tester module instantiates a single arbiter connected to a single request/grant signal pair named "RQ" and "GT respectively. It also contains a partially completed *initial* block to test the arbiter operations.
- Complete the *initial* block to verify that the arbiter meets the specification for the grant assertion and deassertion. Use a *fork/join* statement.
- Have your tester produce relevant messages to help diagnose what is going wrong with the arbiter.

Lab 6.2

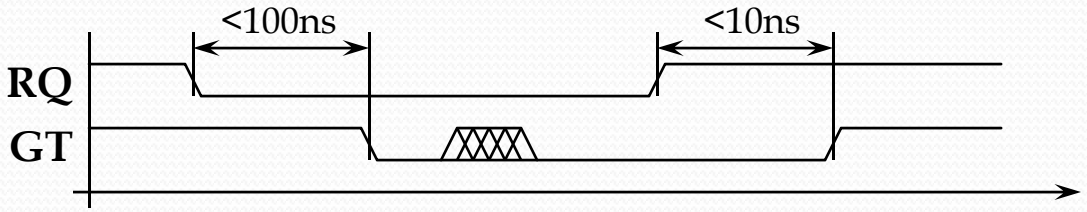
- Run your test procedure against each arbiter individually using the commands:

```
% ... testarb.v arbiter1.v
% ... testarb.v arbiter2.v
% ... testarb.v arbiter3.v
```

- Which arbiter works?
- Which one does not?
  - What is wrong with the arbiters that do not work?
  - Looking at the source of each arbiter to answer the last three questions is considered cheating…

Lab 6.2: Optional

- One of the arbiter exhibits the following behavior:



- Modify the arbiter tester to catch and diagnose this problem.
- Which arbiter is it?

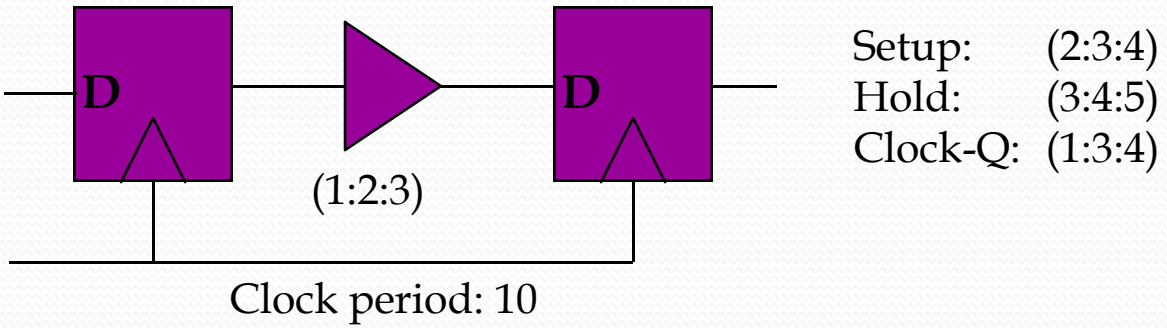
Lab 6.3

Lab 6.3

- In this lab, you will:
  - Learn how to invoke a simulation with minimum, typical, or maximum delays
  - Learn that simulation results can change by using different delay modes

Lab 6.3

- In directory ~/lab6.3, you will find the following files:
  - dff.v Model of a D flip-flop
  - shift.v Model of a 2-bit shift-register
  - testshft.v Tester for the 2-bit shift register
- The shift register has the following structure, with each device having the specified min:typ:max delay:



Lab 6.3

- If you use a compiled simulator with separate compilation and simulation commands, the +*mindelays*, +*typdelays*, and +*maxdelays* options should be specified on the simulation command.
- Run the simulation using typical delays.
- Run the simulation using minimum delays.
- Run the simulation using maximum delays.
- Explain any difference in the result.

Lab 6.3

- Can a timing problem that would occur if a device operated at "max" while another device operates at "min" be detected?
  - "Epilog" is a simulator from NextWave Design Automation that performs a min/max simulation concurrently, and detects timing problems caused by timing "spread".

Lab 6.3: Optional

- Can you fix the design so it will work under all possible operating conditions?
  - Changing the delay values is considered cheating…