AEDVICES Consulting
166B Rue du Rocher de Lorzier
38430 Moirans– France
trainings@aedvices.com
http://www.aedvices.com

Application Engineering, Design & Verification in ICs and Embedded Systems

Training on
Verification Methodologies for IP and SoC Designs

LAB
SystemVerilog Threads

Objectives

This lab goes through the main programming concepts of SystemVerilog thread controls. It shows how threads can be used to control different parts of the testbenches.

Global Explanation

This lab will use the previous *driver* and *monitor* created classes. These classes are instantiated in the main program that call their methods in threads fork off by the fork structure. All seen before the monitor will be responsible to monitor the signal from the DUT interface and the driver to drive the DUT signals. These two "components" will run in concurrent threads.

Instructions

Follow instructions given in "aedv_training_labs_intructions_for_questa.pdf".
Open the file <SANDBOX>/labs-Xdays/labNN-systemverilog_threads/lab_v2.sv"
Select

● System Verilog

Commenté [AM1]: We must keep the original name?

Step 1: Analysing the fork.

- Open the file: <SANDBOX>/labs-Xdays/labNN-systemverilog_threads /lab_prog.sv
- Search for ***LAB-TODO-STEP1-a***
- Analyse and run
  - What does the fork do?
  - Does the simulation finish correctly or do you get a *TIMEOUT* (710ns)? Why?
- Search for ***LAB-TODO-STEP1-b***
- To end the simulation, replace the *join* keyword with the *join_none* one and run
  - Does the driver drive any transaction? Why?
- Search for ***LAB-TODO-STEP1-c***
- To end the simulation and drive the transactions, replace the *join_none* keyword with the *join_any* one and run
  - Does the driver drive any transaction now? Why?
  - Does the simulation end now? Why?

Step 2: Thread B.

- Search for ***LAB-TODO-STEP2-a***

- Create a third thread called *thread_B* to perform **5** transactions. (Tip: Copy the thread A and modify)
- Run
    - Both threads are executed? Why?

Step 3: Semaphore.

- Let us use the semaphore to workaround the race condition.
- Search for ***LAB-TODO-STEP3-a***
- Declare and instantiate a mutex (a semaphore with only one item)
- Search for ***LAB-TODO-STEP3-b***
- Below the first label, call the *get()* method to take the mutex in the beginning of the transaction configuration and driving
- Below the second label, call the *put()* method to put back the mutex qhen the transaction is completely driven
- Do the same
- Run
    - Both threads are executed now? Why?
    - Is the Thread A executed completely? Why?

Step 4: Process.

- Let us use the process class features to manipulate the threads and have our expected behavior (Execute all the transaction in each thread and finish the monitor thread when done).
- Search for ***LAB-TODO-STEP4-a***
- Declare a vector of process of size 3
- Search for ***LAB-TODO-STEP4-b***
- Below each label call the *sel()* method from process to get process handle.
- Do the same thing for the Thread B
- Search for ***LAB-TODO-STEP4-c***
- Out of the *fork* block use a *foreach* loop to confirm that all the threads are done (except the process 0, Monitor thread), if not, wait until they are done.
- Search for ***LAB-TODO-STEP4-d***
- When all the processes are done, kill the first one (Monito thread) using the *kill()* method from the process class.

Commenté [AM2]: There are many differences between process and thread when the subject is OS, for example, these differences are valid for the System Verilog features?