2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Heartbeat based early detection of Hang issues

Vinaykumar Kori, Cadence Design Systems, Noida, India (vkori@cadence.com)

Tejbal Prasad, Cadence Design Systems, Noida, India (tejbal@cadence.com)

***Abstract*— Test hangs are a significant roadblock in verification process. For the Verification team, most of the time is spent on debugging test hang/timeout issues and hindering efficient validation and causing delays in project timelines. It consumes unnecessary compute resources.**

This paper aims to provide a comprehensive and effective framework. It enables test writers to code run time expectation in the test at logical level. Thereby it eases out detection of hang/timeout issues.

Keywords*— *Verification, Error Handling, Debugging, Hang Issues

I. INTRODUCTION

Debugging test hang scenarios is a critical aspect of Verification, particularly in complex and distributed computing environments. Test hangs, where the execution of test cases becomes unresponsive or stalls indefinitely, can significantly impact the efficiency and reliability of testing processes, leading to delays in release and potential quality issues.

Hang scenarios are very frequent in any verification environment. These scenarios are very expensive with respect to compute resource usage. A significant amount of Verification team's effort and time are spent on debugging these scenarios. Identifying and resolving test hangs require sophisticated debugging techniques and tools to pinpoint the root causes and underlying issues effectively.

Some of the conventional ways are listed below.

1. *Test timeout (UVM Timeout)*
   Test timeout (UVM Timeout) is a safety switch to prevent run-away simulation. In verification environment, every test cases have a test timeout duration and after timeout it will terminate the simulation. But it suffers from below limitations.
   - It does not pinpoint which process is hogging the test.
   - Setting proper timeout for each test is cumbersome and often leads to common UVM Timeout for all tests.

2. *Feature Specific Timeout Checkers:*
   This timing checkers does a decent job on flagging timeout failure.
   - These checkers are written only for specific features.
   - Orthogonal sequence, which takes the test in a different mode makes these checkers ineffective. E.g., While running traffic test went through a low power mode transition.

Conventional verification environment does not provide a mechanism to add the timing expectation of different processes. A robust verification environment with clear and best practices can help minimize the effort spent on debugging the hang/timeout issues.

II. SOLUTION

To take care of this challenge, we have introduced a specialized component which will maintain the run time expectation of each main process running in a test.

Figure 1 illustrates a simple block diagram having multiple components with multiple thread running on and an instance of a Heartbeat component. The heartbeat component instance is going to keep track of all the current running process. Test writer will have to register their process when it starts with its nature and timeout duration.

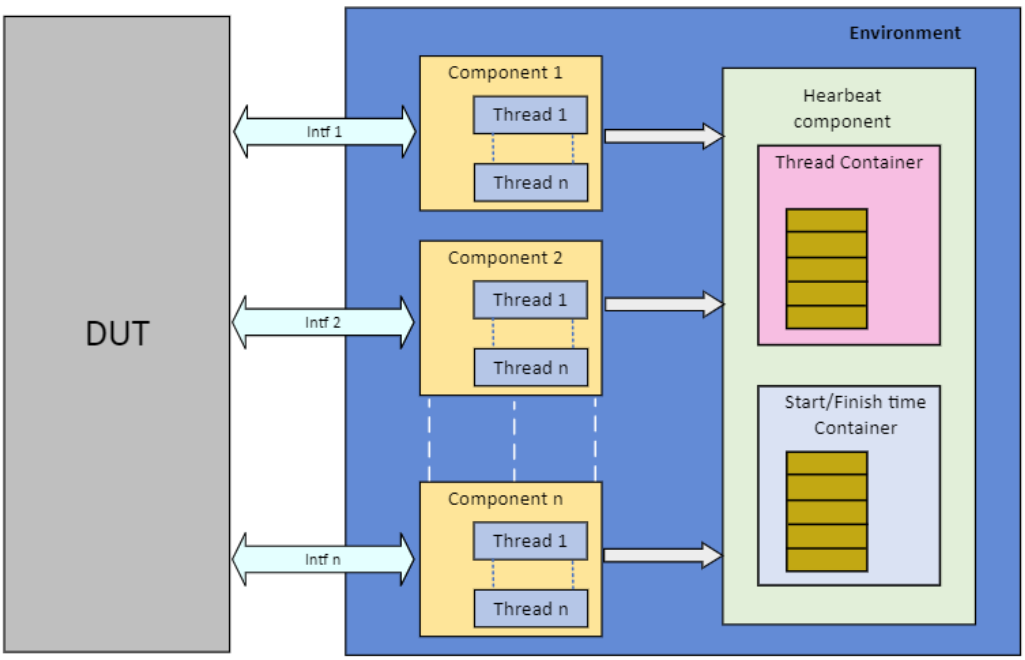

Figure 1: Block diagram of heartbeat-based component with verification environment

Figure 2 illustrates a simple use scenario, where test writer has to register time consuming processes with Heartbeat component on various stages of execution.

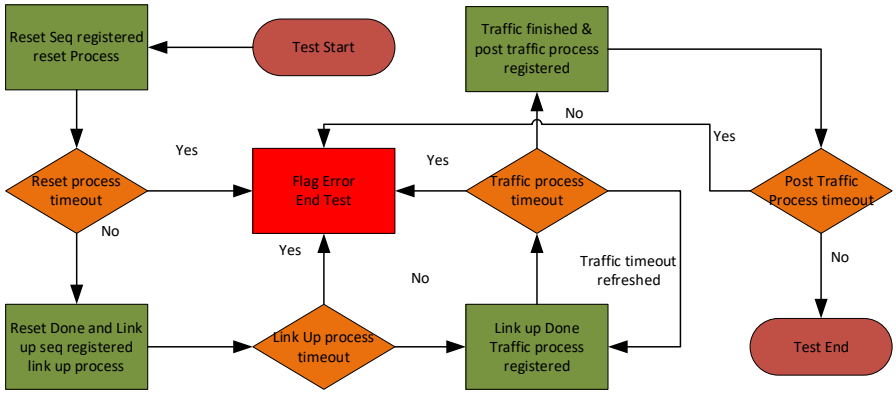

Figure 2: Flow chart with heartbeat flow

Figure 3 illustrates a heartbeat flow with orthogonal sequence, each sequence will register its process inside the heartbeat component. Heartbeat component will look at the timeout duration and nature of each process. Based on the nature of process Heartbeat could pause/terminate the other process timeout.
E.g., if a reset sequence, all the process timeout needs to be terminated, whereas if low power sequence has started the other process's timeout needs to be paused.

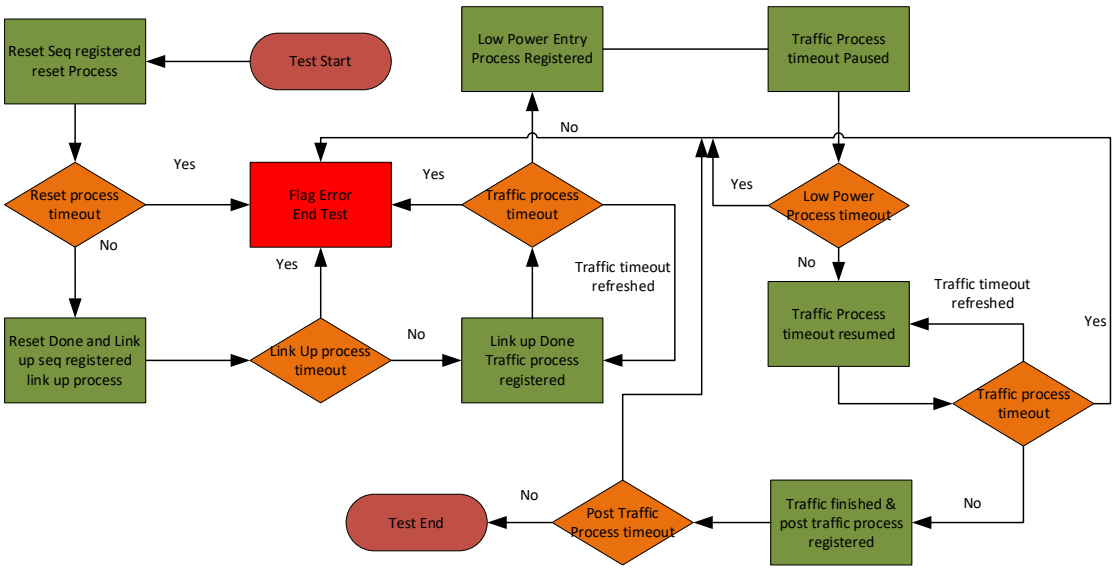

Figure 3: Heartbeat flow with orthogonal sequence

III. WORKING MODEL

This chapter demonstrates the use/integration of Heartbeat component in user's testbench.

A. *Creating Instance*

The user is expected to create a single instance of this component and pass this handle to all the sub env's from Top TB env. Below example demonstrates the creation of heartbeat component.

```
// Instance of Heartbeat component
cdn_heartbeat_component heartbeat_env;

virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);

  // Creating Heartbeat component
  heartbeat_env = cdn_heartbeat_component::type_id::create("heartbeat_env", this);

endfunction : build_phase
```

Figure 4: Example of Creating heartbeat component.

B. *Triggering Event*

It is essential to check the current running threads on certain time intervals to detect if the current running process is not exceeding the expected time duration. We have used an event-based algorithm to check the current running threads, which user have to trigger in the testbench on certain time interval. This trigger event has nothing to do with timing values. Heartbeat component will just only perform timing checks when this event is triggered. *`HB_TRIGGER_EVENT* macro is used to trigger the event and it takes heartbeat component instance as input argument.

Below is the example where we are triggering heartbeat component on each clock edge.

```
//------------------------------------------------------------------------
// Task : set_trigger_event_to_heartbeat
// This method is used to triggering the event of heartbeat component
//------------------------------------------------------------------------
virtual task set_trigger_event_to_heartbeat();
  forever begin
    @(posedge misc_if.core_clk);

    // Triggering heartbeat event
    `HB_TRIGGER_EVENT(heartbeat_env)
  end
endtask : set_trigger_event_to_heartbeat
```

Figure 5: Example of triggering event

C. Registering/Completion of Thread

1) Registering Thread

To add a specific thread in heartbeat component, *`HB_ADD_THREAD* macro is used. Adding thread will take the current time as Thread start time. It takes below four arguments.

- Heartbeat component Instance
- Thread Name: A unique name given to each thread.
- Thread Class: It will specify the type of thread to be added in heartbeat component.
  - NORMAL: It is used to add normal thread or process which does not require special handling and can run concurrently. Like Normal Traffic, Link up sequence, Config Sequence etc.
  - PRIORITY: It is used for special thread/process which is going to affect the existing thread execution. E.g., Interrupt/Power Management related sequence.
  - RESET: This thread class is used for special thread which is going to terminate all the existing thread execution. E.g., Reset/Link down sequence.
- Expected finish time duration.

2) Completion of Thread

After completion of thread, it is required to remove thread from heartbeat component and to remove a specific thread from heartbeat component, *`HB_DEL_THREAD* macro is used, and it takes below two arguments.

- Heartbeat component instance.
- Thread Name

It is mandatory to give the same thread name as given while registering the thread.

Below figure shows the example of Registering/Completion of thread from a given heartbeat component. Here, we have started a sequence and expecting that sequence will be completed within 30us time duration from given start time.

```
// Adding Thread in Heartbeat container
`HB_ADD_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,          // Heartbeat Instance
      $sformatf("%0s.top_level_tlp_traffic",cxl_dev_m.name()),    // Thread name
      NORMAL,                                                     // Thread class
      20us)                                                       // Exp. Finish time

`uvm_send(v_cxl_tlp_vseq)

// Delete thread from Heartbeat container
`HB_DEL_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,          // Heartbeat Instance
      $sformatf("%0s.top_level_tlp_traffic",cxl_dev_m.name()))    // Thread name
```

Figure 6: Example of Adding and Deleting Thread in heartbeat component.

Once the thread is registered, the heartbeat component will take care about how long the thread is expected to run and would report error if it does not finish in the expected time.

D. Pausing and Resuming Thread

There are scenarios where we need to pause timeouts running for existing threads. This is to cater for ISR (Interrupt Service Routine) or Low Power scenario.

While registering these special processes the user needs to specify the thread class as "*PRIORITY*". When a thread is registered with this thread class our Heartbeat component puts all the processes running with "NORMAL" thread class into pause state.

On completion of PRIORITY thread process, paused timeouts resume. Completion of thread is assumed when a thread is deleted using the *`HB_DEL_THREAD* macro.

Below example shows pause/resume scenario because of a *PRIORTY* thread registration. So, the moment a *PRIORTY* thread is registers all running threads with thread class *NORMAL* will be paused, and their timeout counters will be frozen. After this thread is over this is deleted from heartbeat component.

```
// Adding Thread in Heartbeat container to pause concurrent threads
`HB_ADD_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,     // Heartbeat Instance
      $sformatf("cxl_power_management_seq"),                 // Thread name
      PRIORITY,                                              // Thread class
      30us)                                                  // Exp. Finish time

`uvm_send(v_cxl_low_power_seq)

// Delete thread from Heartbeat container to resume concurrent threads
`HB_DEL_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,     // Heartbeat Instance
      $sformatf("cxl_power_management_seq"))                 // Thread name
```

Figure 7: Example of Pausing and Resuming Thread

E. Reset Thread Class handeling

To support multiple resets in the same test and their implications on the existing running process we have defined this thread class. It is to mimic the normal expectation of what happens when a reset is asserted while normal traffic is ongoing.

When a thread is registered with thread class reset, we terminate all the registered thread on completion of this thread. This is to ensure that post reset sequence needs to register new thread.

1) Terminate Thread

To terminate the concurrent threads, we must use *`HB_ADD_THREAD* macro with thread class as *RESET*. It will remove all the concurrent thread from heartbeat container, and it will not allow to add any new concurrent thread until it is not being removed.

2) Activate Thread

To activate addition of concurrent threads, we have to use *`HB_DELETE_THREAD* macro. It will remove thread with *RESET* thread class and allow new concurrent sequence in heartbeat component.

Below example shows the pausing and resuming traffic running thread when certain event has occurred.

```
// Adding Thread in Heartbeat container to remove all concurrent threads
`HB_ADD_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,      // Heartbeat Instance
      $sformatf("cxl_reset_seq"),                             // Thread name
      RESET,                                                  // Thread class
      30us)                                                   // Exp. Finish time

`uvm_send(v_cxl_reset_seq)

// Delete thread from Heartbeat container to allow adding concurrent threads
`HB_DEL_THREAD(cxl_dev_seqr.m_cxl_dev_env.heartbeat_env,      // Heartbeat Instance
      $sformatf("cxl_reset_seq"))                             // Thread name
```

Figure 8: Example of terminating and Activating Thread

F. Result

The Heartbeat component accumulates all the data given by user or calculated within component and provides interactive way of reporting of all the running thread. In addition, UVM_ERROR is reported whenever the thread is exceeding the expected completion time duration.

The reporting mechanism contains following information.

- Thread Name
- Current Status of each running threads.
- Start time and finish time (Expected completion time of thread)
- Pause time and Resume time of each thread.
- Remaining time of each thread.

Below figure shows an example of current active running thread.

```
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|     HEARBEAT COMPONENT : heartbeat_env (current_time =         165.400 us)                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    THREAD_NAME                    |   TYPE   |  STATUS  |      START_TIME      |     FINISH_TIME      |      PAUSE_TIME      |      RESUME_TIME     |    REMAINING_TIME    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| CXL_RP.top_level_tlp_traffic                      |   NORMAL |  RUNNING |          73.183 us |          30.000 us |          99.000 us |         135.633 us |           0.004 us |
| top_level_cxl_tlp_traffic                         |   NORMAL |  RUNNING |           0.622 us |         500.000 us |          99.000 us |         135.633 us |         397.443 us |
| top_level_pcie_init_seq                           |   NORMAL |  RUNNING |         113.719 us |          70.000 us |           0.000 us |         135.633 us |         153.952 us |
| top_level_pcie_tlp_traffic                        |   NORMAL |  RUNNING |           0.622 us |         500.000 us |          99.000 us |         135.633 us |         397.443 us |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
```

Figure 9: Example of Running thread.

Below figure shows an example of paused thread. For paused threads, timing check will be skipped, and remaining time will not be calculated.

```
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|     HEARBEAT COMPONENT : heartbeat_env (current_time =          73.400 us)                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    THREAD_NAME                    |   TYPE   |  STATUS  |      START_TIME      |     FINISH_TIME      |      PAUSE_TIME      |      RESUME_TIME     |    REMAINING_TIME    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| CXL_EP.top_level_pm_seq                           | PRIORITY |  RUNNING |          73.207 us |           0.000 us |           0.000 us |           0.000 us |                   -- |
| CXL_EP.top_level_tlp_traffic                      |   NORMAL |   PAUSED |          73.183 us |          30.000 us |          73.207 us |           0.000 us |                   -- |
| CXL_RP.top_level_pm_seq                           | PRIORITY |  RUNNING |          73.368 us |           0.000 us |           0.000 us |           0.000 us |                   -- |
| CXL_RP.top_level_tlp_traffic                      |   NORMAL |   PAUSED |          73.183 us |          30.000 us |          73.207 us |           0.000 us |                   -- |
| top_level_cxl_tlp_traffic                         |   NORMAL |   PAUSED |           0.622 us |         500.000 us |          73.207 us |           0.000 us |                   -- |
| top_level_pcie_tlp_traffic                        |   NORMAL |   PAUSED |           0.622 us |         500.000 us |          73.207 us |           0.000 us |                   -- |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
```

Figure 10: Example of paused thread

Below figure shows an example of an error occurred due to thread has exceeded the expected execution time.

```
UVM_ERROR @ 155.244 us: uvm_test_top.sve.link_env[0].dev_env0.heartbeat_env
[heartbeat_env] Thread with CXL_RP.top_level_tlp_traffic name took more than
expected 20.000 us time
```

Figure 11: Example of error when thread exceeding time.

User can control frequency of reporting prints using *+HB_THREAD_PRINT_DELAY* command line arg. Below error print can be received when thread is exceeding expected completion time duration.

IV. CONCLUSION

We have used the mentioned Heartbeat component in our existing project and the seamless integration of heartbeat component in testbench paves the way of tracking all the long running threads. It also provides the interactive reporting mechanism of all the running threads.

By the use of heartbeat-based test hang detection, we were able to detect timeout issues on early stage and it has significantly reduced the effort on debugging timeout/hang issue. It enables test writers to add timing expectation for each process. Heartbeat component had pinpointed the process that has timed out and saved compute resource usage by eliminating long running test. Heartbeat component has also exposed the design bugs which are hidden behind long running test. This is a generic and easy solution for timeout detection, which can be reused with any environment.

Furthermore, the approach outlined in this paper is scalable and portable to different testbenches. We are already tracking all threads through proposed approach.

V. REFERENCE

[1] "IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017), vol., no., pp.1-1354, 2024.

[2] Universal Verification Methodology (UVM) 1.2 Class Reference