

# Distributed simulation by separating test bench and DUT

ECE 667 (Synthesis & Verification of Digital Systems.)  
Dusung, Kim

## Introduction

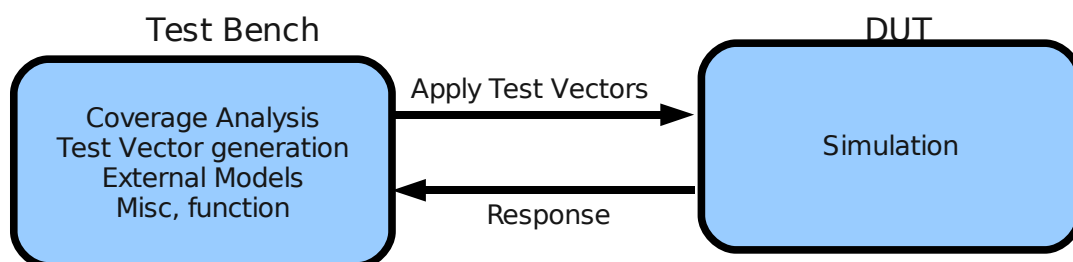
As the semiconductor technology has improved rapidly, the complexity of the latest design is getting higher. As a result of that, the verification is becoming one of the cost-consuming processes in the entire process. Thus, there have been many efforts to find out more efficient verification methods resulting in several remarkable advanced techniques which have played an important role recently.

However, most of verification engineers still use the traditional methods based on simulation, not only because it is easy to construct verification environment, but also because it has powerful debugging features. But, the point is that in spite of its merits, the slow speed of it remains as a big problem to solve.

I implemented a simple simulation system by separating test bench and DUT to speed up the simulation. An entire design is divided into two blocks. That is test bench and DUT, respectively. Each of them is simulated by separate processors. If the test bench and DUT is complex enough to overcome the communication overhead, we are able to get performance improvement without loss of any merits of the standalone simulator. If not, we are not able to get any benefit from this method. This separation makes sense because recent test bench is so complex that it is more than 30% of entire simulation overhead in some cases.

We will see the technical issues in next section and analyze the performance of the system.

## Problem outline



[Fig 1. Separation of test bench and DUT ]

Generally, most hardware design written in verilog HDL language can

easily be separated into two parts structurally, test bench and DUT. Test bench applies the test vectors to the DUT. Then it can be simulated and response to the test bench. [Fig.1] illustrates the situation. One thing we can see carefully is that the role of test bench. In recent design, the role of test bench is test vector generation, coverage analysis, execution of foreign models and assertion check. Therefore, the response from DUT is as important as test vector. We can also know that the size of test bench is not small enough to ignore the simulation overhead for test bench.

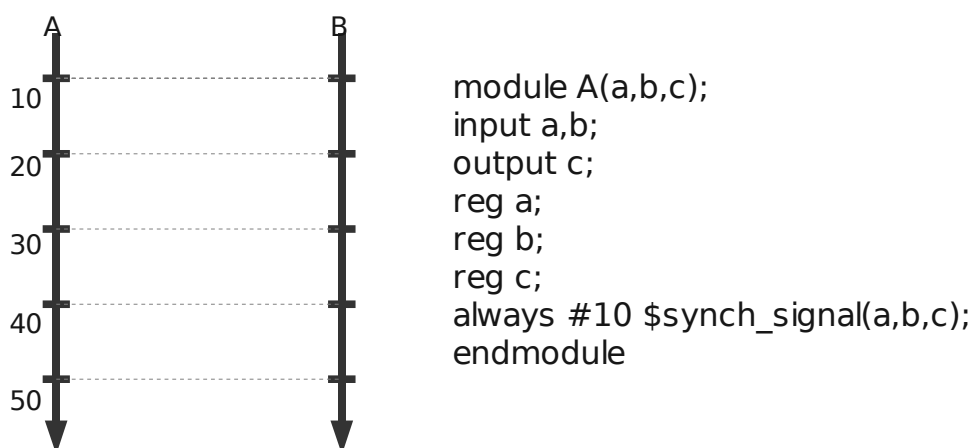
We need to simulate Test bench and DUT in different machines concurrently. Test vector and response can be applied to the DUT and test bench by ethernet. We expect that we will be able to speed up the simulation by dividing the entire design simulated in single simulator into two pieces simulated in two simulators.

## Technical issues

### 1. Synchronization

The crucial problem of this system is that when and how often we need to send/receive the signals. Since the event can occur arbitrarily in an event based simulation, the synchronization time is not deterministic. In this condition, we cannot predict the time the signal arrives to the one of the distributed nodes, so it must be blocked until the signal comes. Therefore, we cannot get the gain by separating the design.

In our system, the synchronization between DUT and TB is established at every rising edge. By doing this, we can do the simulation until next synchronization time, which means the DUT can accept the test vector at every rising edge and response to the test bench at every rising edge. The example is shown in [Fig. 2].



[Fig 2. The example of synchronization based on clock cycle.]

The module A and module B are simulated separately. Initially, A and B are simulated to the time 10 and exchange their signals and then

simulate again until time 20. Because the simulation of DUT is still based on event, we do not lose any merits of event based simulation inside of DUT.

## 2. Signal order between two nodes.

This is another problem of synchronization. We also need to think about that which part should send signal first. Even though the hardware is executed concurrently for all components, a simulator executes the components sequentially, since it is software. In a distributed simulation, the same problem exists between two separated nodes. This could make the simulation result different from that of original standalone simulation. As this is a typical race condition of software simulator, it is up to design engineers to prevent this problem.

## 3. Additional overheads

### (1) Interfacing overhead

We use the existing simulator as a local simulator of entire system. Therefore, we need an interface to access the internal signals. The PLI is a standard interface to access the inside of the design during the simulation. In the implementation using the PLI, we call the callback function at every rising edge. We also need to rearrange the scheduling queue to make the function call last event.

### (2) Communication overhead

This is a common problem of every kind of distributed systems. Since the frequency of communication is fixed in our system, we need to reduce the amount of data size for communication as much as possible.

## Analysis of the system

We can formulate the speed of the distributed simulation.

Let  $D$  is a design to be simulated.

Let  $O$  is a function which represents the simulation overhead of design. Thus,  $O(D)$  denotes the simulation overhead of design  $D$ . Since the simulation overhead is highly related to the number of events inside in the design  $N_{\text{evt}}(D)$ , we can write  $O(D) \approx N_{\text{evt}}(D)$ .

The simulation speed of design  $D$  can be written as  $S(D) = PO(D)$ , where  $P$  is processing speed for each event.

To compare the performance with distributed simulation, we need to write different way.

$$D = D_{\text{DUT}} + D_{\text{TB}}$$

$$N_{\text{evt}}(D) = N_{\text{evt}}(D_{\text{DUT}}) + N_{\text{evt}}(D_{\text{TB}}) + N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} )$$

$$S_s(D) = P / (N_{\text{evt}}(D_{\text{DUT}}) + N_{\text{evt}}(D_{\text{TB}}) + N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} ))$$

For distributed system, we can write that,

$$S_D(D_{\text{DUT}}) = P / \text{evt}(D_{\text{DUT}}) + P_{\text{DC}} / N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} )$$

$$S_D(D_{\text{TB}}) = P / \text{evt}(D_{\text{TB}}) + P_{\text{DC}} / N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} )$$

In the distributed system, the signals between DUT and test bench need to be passed through ethernet. Therefore  $P_{\text{DC}}$  is slower than  $P$ .

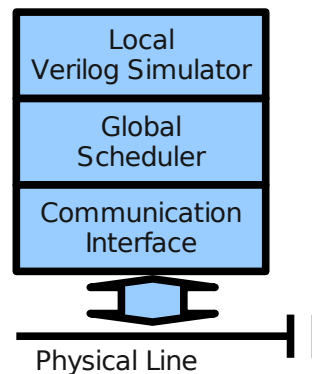
Simulation speed will bound to the  $S_D(D_{\text{DUT}})$ . In [Fig. 2], if the simulate A is faster than the other, A always has to wait for B at every rising edges. In our system, since DUT is bigger than test bench, the simulation speed is bounded to the simulation speed of DUT. Therefore, in order to get a gain from distributed simulation,

$$P_{\text{DC}} / N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} ) > P / (N_{\text{evt}}(D_{\text{TB}}) + N_{\text{evt}}( D_{\text{DUT}} \cap D_{\text{TB}} ))$$

needs to be satisfied. That is, the communication overhead should be smaller than testbench overhead.

## Implementation details

I implemented the system with 3 layers, Local Verilog Simulation, Global Scheduling and Communication layer, respectively. The Fig 3. show the architecture.



[Fig 3. The layer of distributed simulation]

### 1. Local simulator.

The role of local simulator is to simulate the separated design. The input is applied to the design by using PLI routine, and output is also extracted in the same way. I used the VCS simulator as a local simulator.

### 2. Global Scheduler.

The global scheduler blocks the local simulation at every rising clock, then waits for incoming signals. The incoming signals are put into the

buffer. After the buffering is done, the scheduler forces the signals in the buffer to the simulator and it returns the control to the simulator.

### 3. Communication Interface.

The communication interface was implemented as a server client model by using socket API. So, the distributed simulation system can be executed either in a single machine and in multiple machines. The protocol for the communication designed very simply to reduce the communication overhead.

The separation of design can be done manually. We need to create the stub for remote module in each node. The example of this separation is shown in Example 1-(a) and Example 1-(b).

```

/* DUT */
module Onebit_Addition(clk, a, b, out);
input clk,a,b;
output out;
reg out;
wire clk,a,b;

always @(posedge clk)
begin
out <= a+b;
end
endmodule

/* TB */
module TB_Onebit_Addition;
reg clk,a,b;
wire out;
initial begin
clk <= 1;
a <= 0;
b <= 0;
#100 $stop;
end
always #30 a = ~a;
always #20 b = ~b;
always begin
#10 clk = ~clk;
end
Onebit_Addition addition(clk,a,b,out);
endmodule

```

[Example 1-a. The verilog code before the separation]

DUT (Node A)	TB (Node B)
<pre> module Onebit_Addition(clk, a, b, out); input clk,a,b; output out; reg out; wire clk,a,b; always @(posedge clk) begin </pre>	<pre> /* TB */ module TB_Onebit_Addition; reg clk,a,b; wire out; initial begin clk &lt;= 1; a &lt;= 0; </pre>

DUT (Node A)	TB (Node B)
<pre> out &lt;= a+b; end endmodule /* STUB for TB */ module TB_Onebit_Addition; reg clk,a,b; wire out; initial begin \$dumpfile("dist.dump"); \$dumpvars(0,TB_Onebit_Addition .addition ) ; clk &lt;= 1; a &lt;=0; b &lt;=0; #100 \$stop; end initial begin forever #10 \$CommChannel(1,"192.168.2.2",9901,3,1,"T B_Onebit_Addition .clk","TB_Onebit_Addition .a","TB_Onebit_Addition .b","TB_Onebit_Addition .out"); end Onebit_Addition addition(clk,a,b,out); endmodule </pre>	<pre> b &lt;=0; #100 \$stop; end always #30 a = ~a; always #20 b = ~b; always begin #10 clk = ~clk; end Onebit_Addition addition(clk,a,b,out); endmodule  /* Stub for DUT */ module Onebit_Addition(clk, a, b, out); input clk,a,b; output out; reg out; wire clk,a,b; initial begin forever #10 \$CommChannel(2,"192.168.2.1",9901,1,3,"TB _Onebit_Addition .addition.out","TB_Onebit_ Addition .addition.clk","TB_Onebit_Addition .addition.a","TB_Onebit_Addition .addition.b"); end endmodule </pre>

[Example 1-b. The verilog code after the separation]

In Example 1-b. Node A has real DUT and virtual TB. It has a system task to communicate with remote module instead of test vector generation routine. On the contrary, node B has real TB and virtual DUT. It has only I/O ports and a system task to communicate with remote module.

The \$CommChannel is the system task to synchronize two separated designs. Since it is always scheduled as a last event at current simulation time, there are no race condition with other signals.

## Experimental Result

I used the design for AES(Advanced Encrypt Standard). It has 5 inputs and 2 outputs.

Input – clk, rst, ld, key[0:127], text\_in[0:127]

Output – done, text\_out[0,127]

It has 284 distinct test sets and applies the sets infinitely by looping. It generate 128 bit cipher text which is text\_out generated by the 128 bit key and 128 bit text\_in.

### 1. Correctness.

I don't prove if the design is correct. Since I used VCS simulator that I believe it is reliable, the only thing I need to concern is that the equivalence of the result between my simulation and original standalone

simulation. Thus I compared the dump file of two simulators. Unfortunately, the equivalence of the two dump file cannot guarantee the fact that the two simulation is equivalent, because the test bench cannot covers every case. However, we can think that if the test bench covers enough to verify the DUT works correctly, my system is correct enough in this design.

I used “vcdiff” from synopsys to check the equivalence of two dump file. The two dump files were equivalent for 10000 cycles that include entire test sets.

```
vcdiff - Version Y-2006.06
      Copyright (c) 1991-2003 by Synopsys Inc.
      ALL RIGHTS RESERVED

vcdiff aes_dist.dump ../aes_org/aes/aes_org.dump
< aes_dist.dump: scopes:523 signals:3247
> ../aes_org/aes/aes_org.dump: scopes:523 signals:3247

--- vcdiff summary ---
--- compares: 16208658
--- diffs: 0
```

## 2. Overhead investigation.

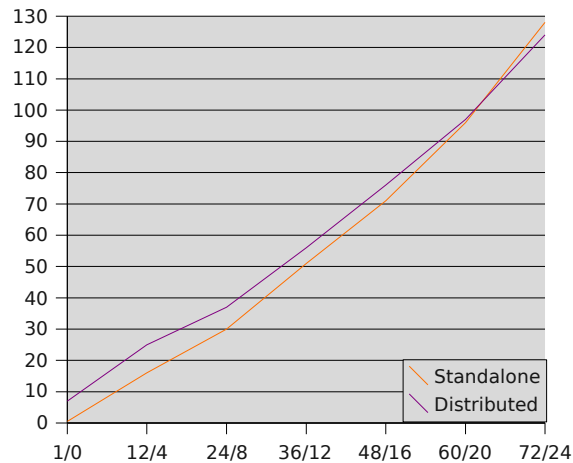
There are 2 additional overheads in the distributed simulation system. Those are interfacing overhead and communication overhead. Actually, because I used the VCS simulator as a local simulator, the PLI overhead must be checked. This test was done by deleting the communication module and by providing loop-back routine. Fortunately, The overhead of PLI interfacing was less than 5% of total additional cost. Therefore, we can ignore it in this experiment.

## 3. Performance.

Since the AES is too small, we cannot speed up as I described in analysis section. But, by increasing the number of instance of DUT (duplicating) , the speed of simulation can be faster than original standalone simulator, relatively. In other words, if we have big enough design and its test bench, we can make speed it up. One point is that we need to make sure the design size is reasonable.

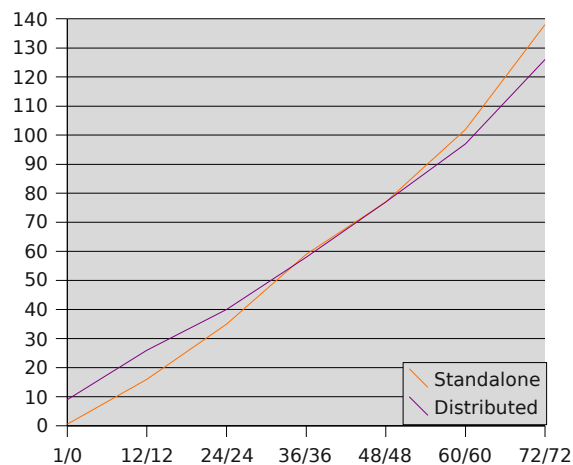
Initially, I used single instance of DUT for AES and its test bench and increased the number of instance of DUT continuously. I also gave the

dummy instances of DUT in the test bench because the test bench also need to be big enough. As the recent test bench includes complex vector generation routine, result analysis routine and models like memory, this extension is reasonable. But definitely, the DUT is bigger than test bench so I gave 1/3 instance of DUT as dummy in test bench. Plot 1. shows the result in this condition.



[Plot 1. The simulation speed when DUT/TB is 1/3]

When there are 60 copies of instance of DUT, the performance of distributed simulation is getting better than standalone simulation.



[Plot 2. The simulation speed when DUT/TB is 1/1]

If there are the same amount of instances of DUT in test bench, which means test bench has similar simulation overhead with DUT, the performance of distributed simulation is much better than previous condition. Plot 2. shows

the result in this condition. When there are about 48 copies of instance of DUT the performance of distributed simulation is getting better than standalone simulation.

## Conclusion & Future work

The distributed simulation system is one of the methods to improve the simulation speed. We have seen that if the design is big and complex enough, we can get a gain. The threshold point is quite realistic, so this approach can be applied to current real big design. But, in this system, there are still important restrictions such as cycle based synchronizations and blocking for DUT. More important thing we need to think in the future is to make general distributed system which has arbitrary number of simulation nodes without any significant loss of benefits of standalone simulation.

## Appendix

### Data

#### 1. Data of Plot 1.

- Number of cycles : 10000
- unit : sec

Number of Instance(DUT/TB)	1/0	12/4	24/8	36/12	48/16	60/20	72/24
Original VCS	< 0.5	16	30	51	71	96	128
Distributed simulation	7	25	37	56	76	97	124

#### 2. Data of Plot 2.

- Number of cycles : 10000
- unit : sec

Number of Instance(DUT/TB)	1/0	12/12	24/12	36/36	48/48	60/60	72/72
Original VCS	< 0.5	16	35	59	77	102	138
Distributed simulation	9	26	40	58	77	97	126