Dynamic Timing Analysis

Dynamic Timing Analysis

- Dynamic vs. Static Timing Analysis
- Dynamic Timing Analysis
- Simulation vs. Evaluation
- Testbench – Build The Test Cases
- Testbench - Catch The Outputs

Dynamic Timing Analysis

- ❖ **Dynamic vs. Static Timing Analysis**
- ❖ Dynamic Timing Analysis
- ❖ Simulation vs. Evaluation
- ❖ Testbench – Build The Test Cases
- ❖ Testbench - Catch The Outputs

Dynamic vs. Static Timing Analysis

❖Timing analysis is integral part of ASIC/VLSI design flow. Anything else can be compromised but not timing! Timing analysis can be **static** or **dynamic**.

❖**Dynamic timing analysis verifies functionality** of the design by applying input vectors and checking for correct output vectors whereas **Static Timing Analysis** checks **static delay requirements** of the circuit without any input or output vectors.

❖**Dynamic timing analysis(DTA)** has to be accomplished and functionality of the design must be cleared **before** the design is subjected to **Static Timing Analysis (STA).**

❖**Dynamic Timing Analysis (DTA)** and **Static Timing Analysis (STA)** are **not alternatives** to each other.

❖**Quality of the Dynamic Timing Analysis (DTA) increases** with the **increase of input test vectors**. Increased test vectors increase simulation time. Dynamic timing analysis can be **used for synchronous as well as asynchronous designs.** Dynamic Timing Analysis (DTA) is also **best suitable for designs having clocks crossing multiple domains**

❖**Static Timing Analysis (STA) can't run on asynchronous deigns** and hence Dynamic Timing Analysis (DTA) is the best way to analyze asynchronous designs. .

Dynamic vs. Static Timing Analysis

❖In **Static Timing Analysis (STA)** static delays such as **gate delay and net delays** are considered in each path and these delays are compared against their required maximum and minimum values.

❖ Circuit to be analyzed is broken into different timing paths constituting of gates, flip flops and their interconnections. Each timing path has to process the data within a clock period which is determined by the maximum frequency of operation.

❖**Cell delays are available in the corresponding technology libraries.** Cell delay values are tabulated based on input transition and fanout load which are characterized by simulation tools.

❖ **Net delays are calculated based on the Wire Load Models(WLM)** or extracted resistance R and capacitance C. Wire Load Models(WLM) are available in the Technology File. These values are Table Look Up(TLU) values calculated based on the net fanout length.

Static Timing Analysis

Advantages of STA:

❖All timing paths are considered for the timing analysis. This is not the case in simulation.

❖Analysis times are relatively short when compared with event and circuit simulation.

❖Timing can be analyzed for worst case, best case simultaneously. This type of analysis is not possible in dynamic timing analysis.

❖Static Timing Analysis (STA) works with timing models. STA has more pessimism and thus gives maximum delay of the design. DTA performs full timing simulation. The problem associated with DTA is the computational complexity involved in finding the input patterns (vectors) that produce maximum delay at the output and hence it is slow.

Static Timing Analysis

Disadvantages of STA:

❖All paths in the design may not run always in worst case delay. Hence the analysis is pessimistic.

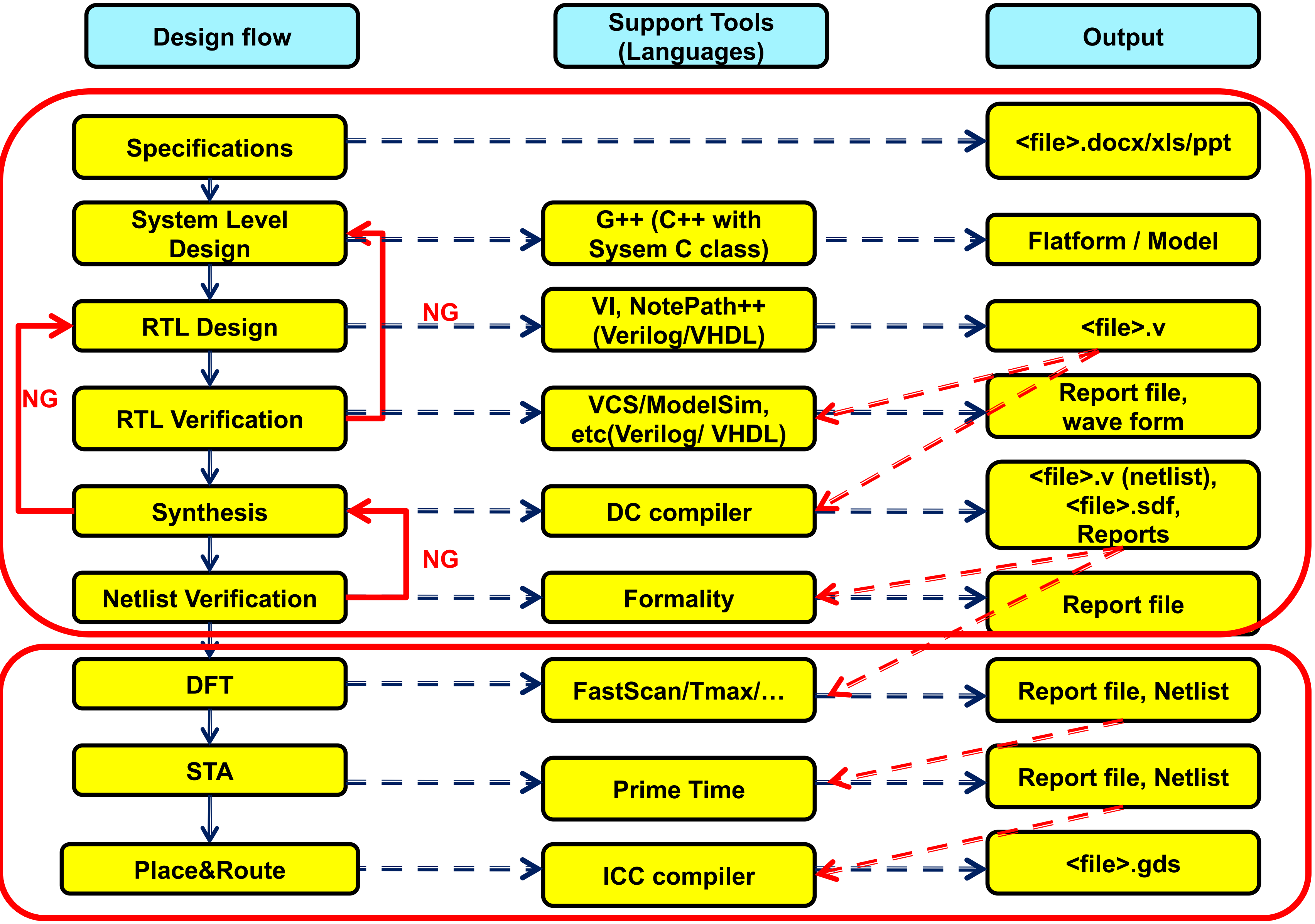
❖Clock related all information has to be fed to the design in the form of constraints.

❖Inconsistency or incorrectness or under constraining of these constraints may lead to disastrous timing analysis.

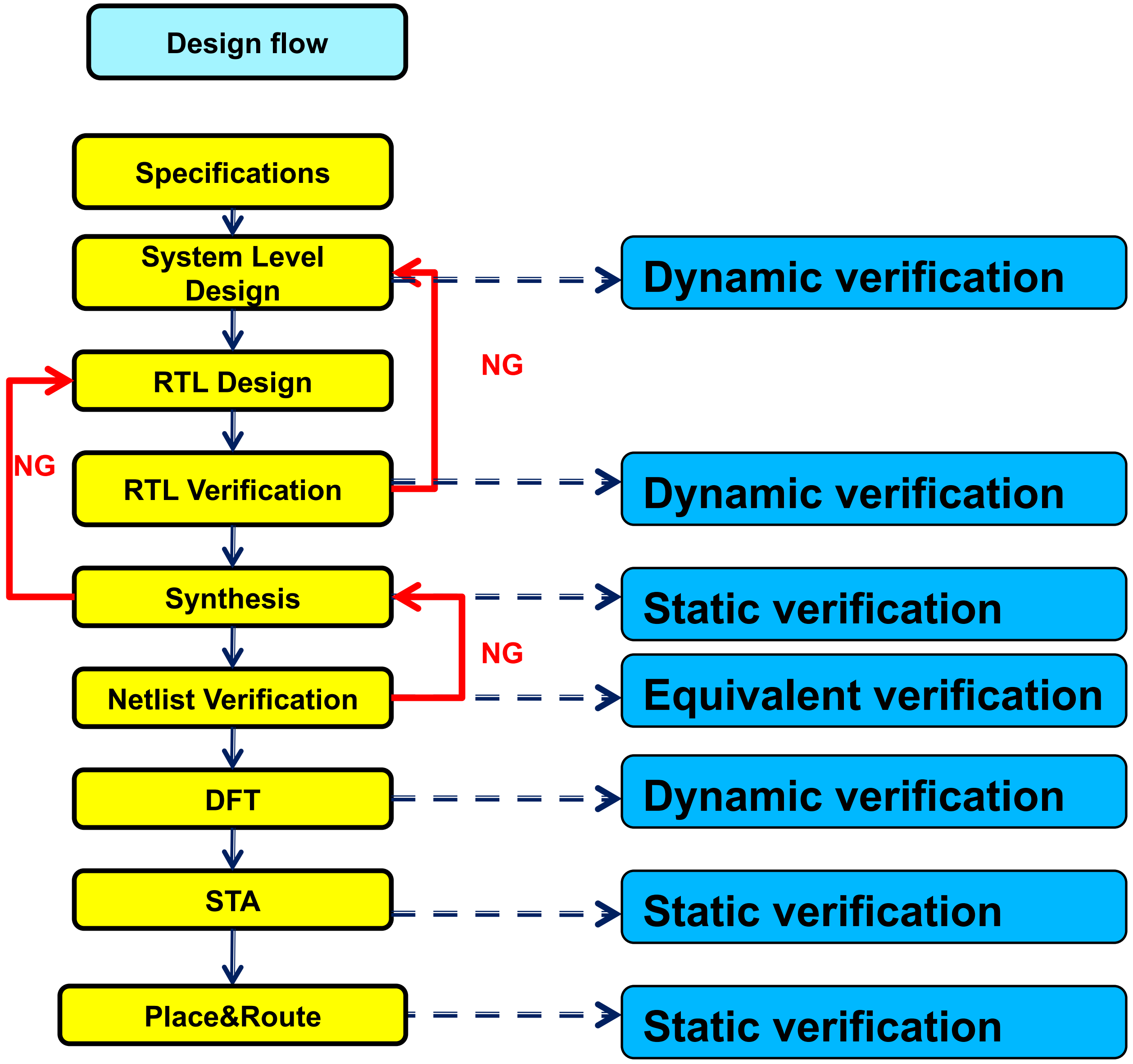
❖STA does not check for logical correctness of the design.

❖STA is not suitable for asynchronous circuits.

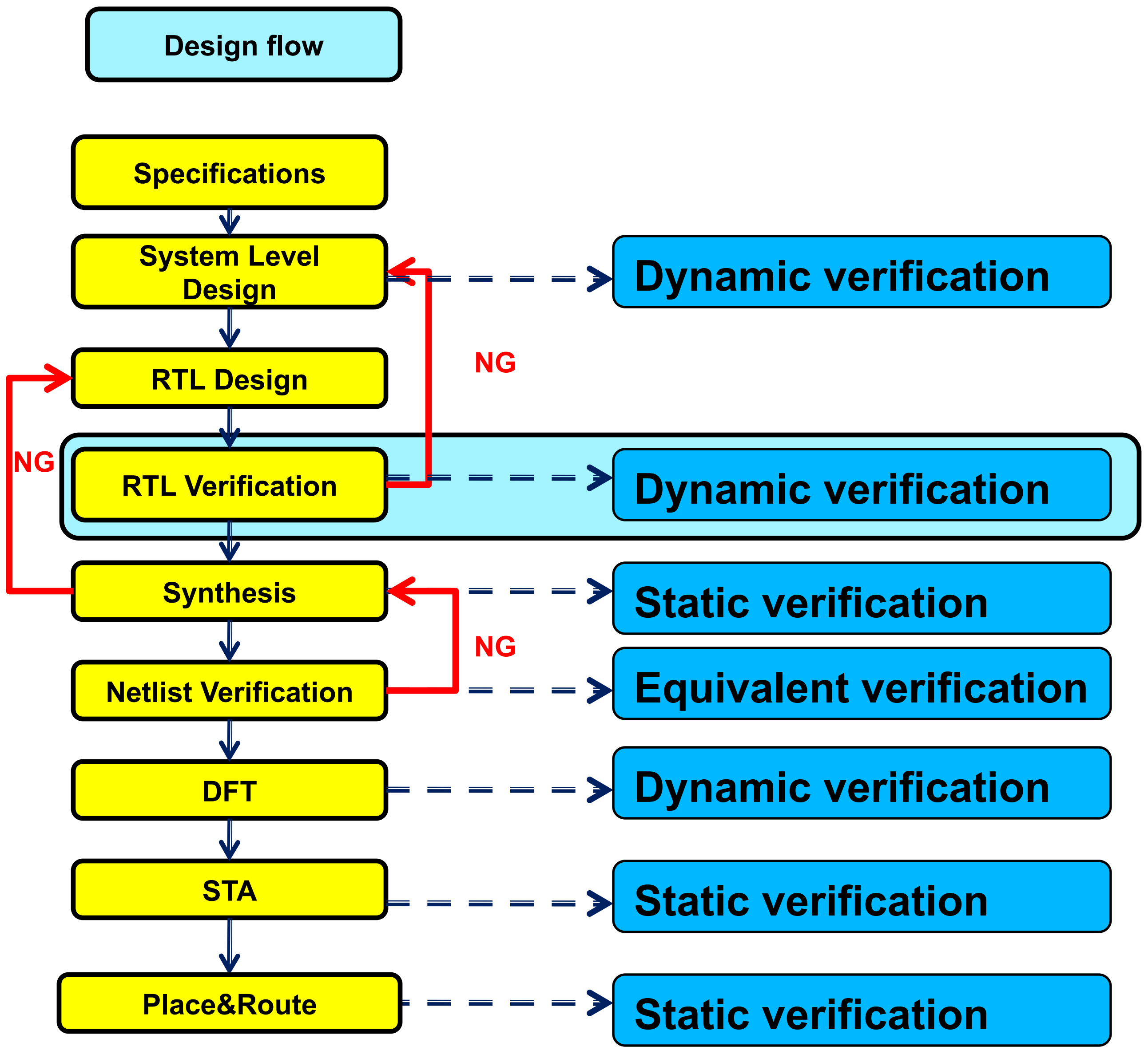
Dynamic & Static Verification In Flow



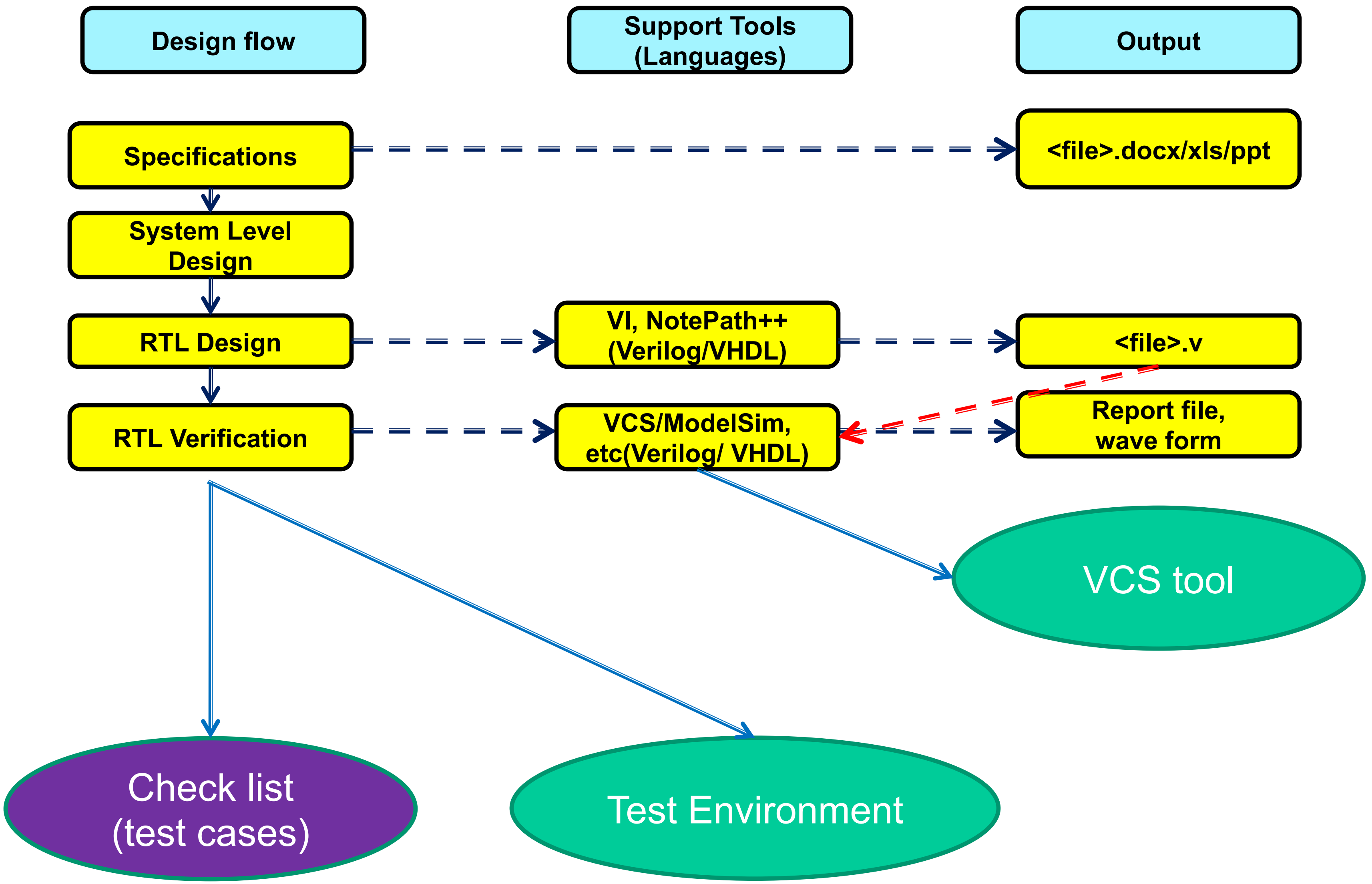
Dynamic & Static Verification In Flow



Dynamic & Static Verification In Flow



Dynamic & Static Verification In Flow



Dynamic Timing Analysis

- Dynamic vs. Static Timing Analysis
- **Dynamic Timing Analysis**
- Simulation vs. Evaluation
- Testbench – Build The Test Cases
- Testbench - Catch The Outputs

How to make the test cases?

Q1. What is a Test Case?

Q2. Which techniques are used to develop Test Cases?

Q3. Which simple example of fields are used in test cases ?

How to make the test cases?

Q1. What is a Test Case?

How to make the test cases?

Q1. What is a Test Case?

❖ A test case has an input, an action and an expected result. In using test cases, the tester is trying to break the application. The whole point of using test cases is to find defects. Hopefully, **serious defects that crash the system are found before your application** is released to the customer. These show stoppers, or defects that may delay release of the application, must be found and fixed in order to save time and money and prevent customer dissatisfaction.

❖ Test cases must **exercise every feature** of the application to prevent defects from being released. Each test case needs to contain a set of test steps of a feature or function. At the end of the test the expected results are compared to actual results to determine if the application is working as it should.

How to make the test cases?

Q1. What is a Test Case?

❖ A test case has an input, an action and an expected result. In using test cases, the tester is trying to break the application. The whole point of using test cases is to find defects. Hopefully, **serious defects that crash the system are found before your application** is released to the customer. These show stoppers, or defects that may delay release of the application, must be found and fixed in order to save time and money and prevent customer dissatisfaction.

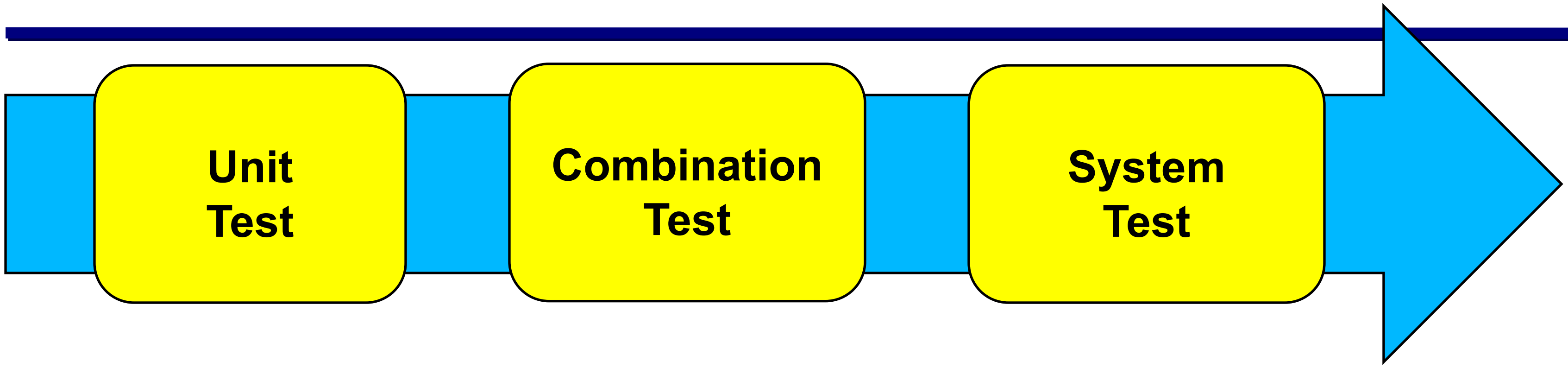
❖ Test cases must **exercise every feature** of the application to prevent defects from being released. Each test case needs to contain a set of test steps of a feature or function. At the end of the test the expected results are compared to actual results to determine if the application is working as it should.

How to make the test cases?

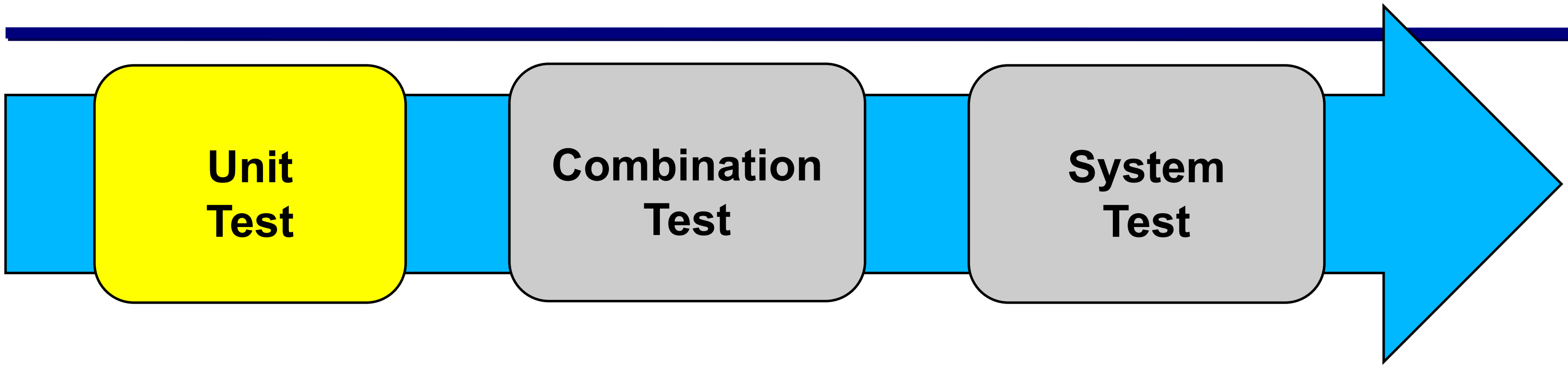
Q3. Which simple example of fields are used in test cases ?

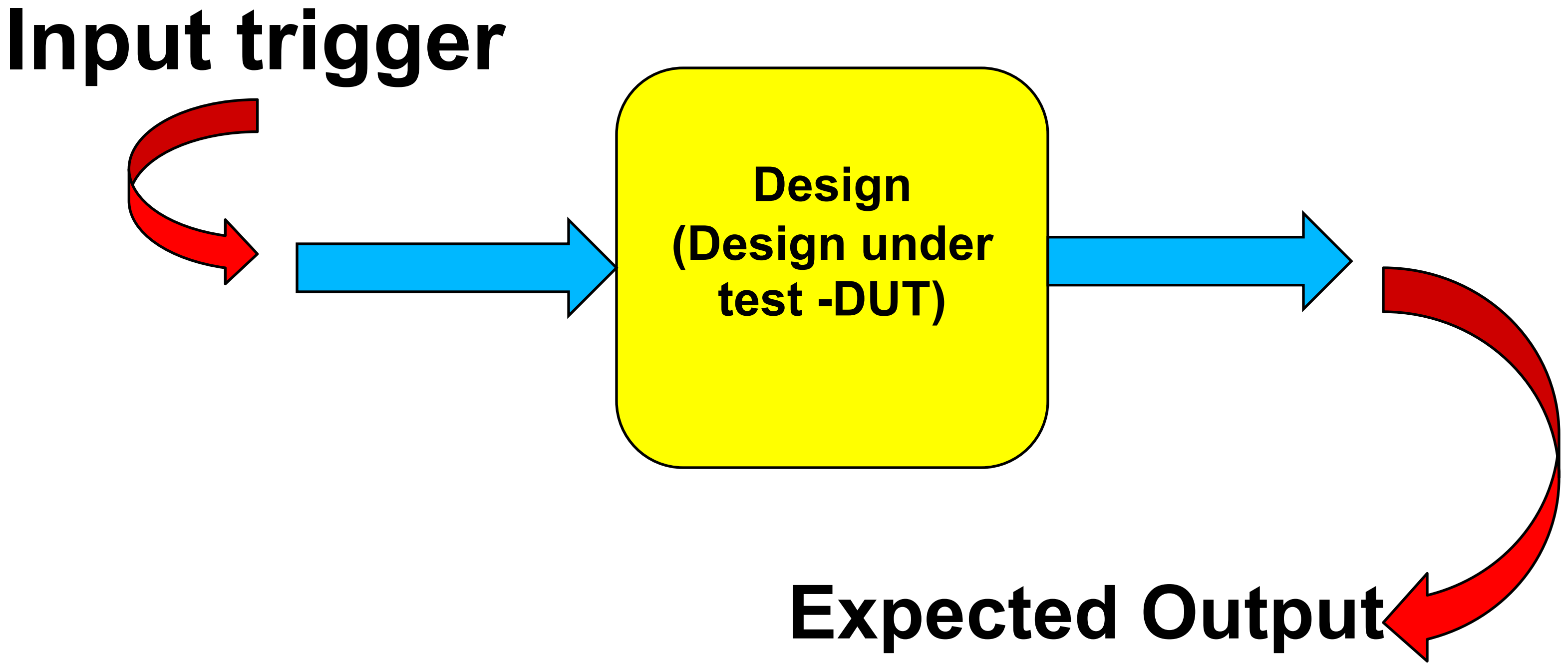
+ Test case ID
+ Unit to test
+ Assumptions
+ Test data
+ Steps to be executed
+ Expected result
+ Actual result
+ Pass/Fail
+ Comments

Test Environment

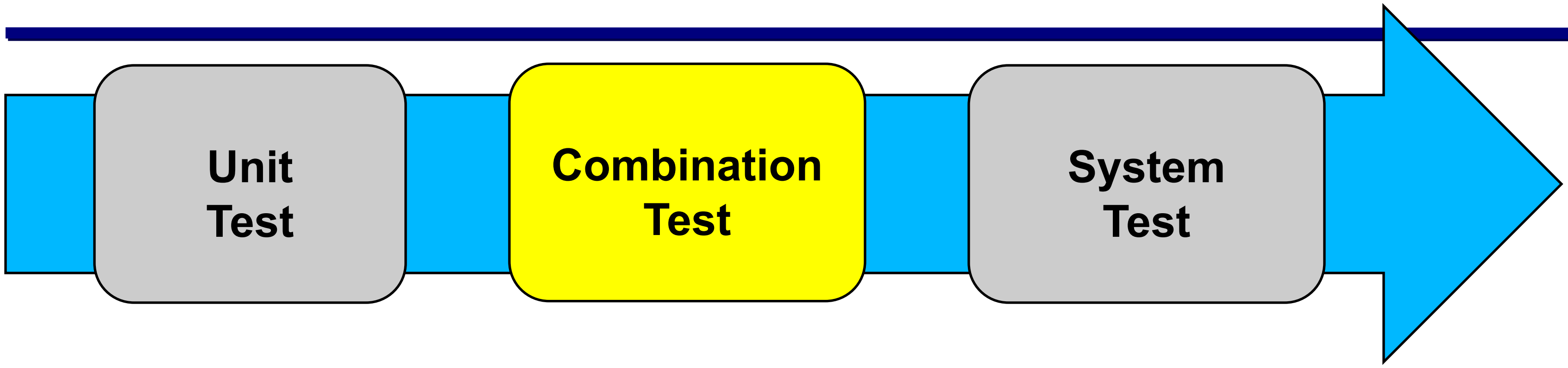


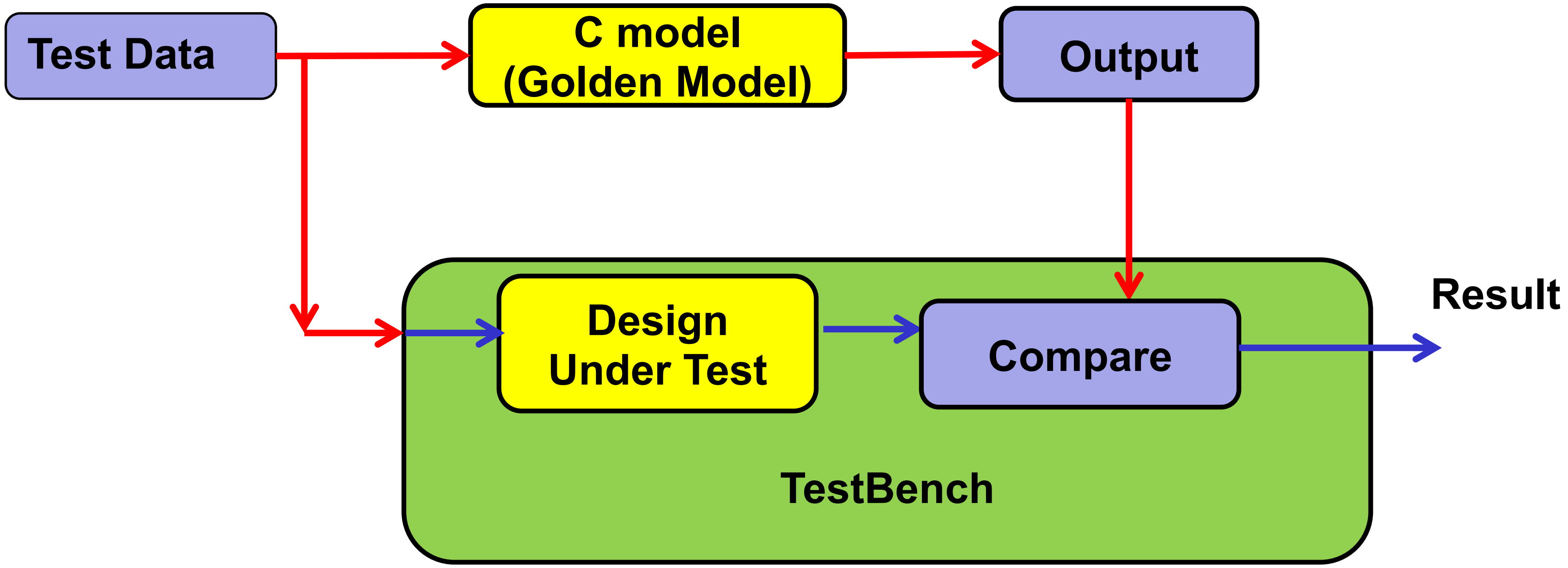
Test Environment



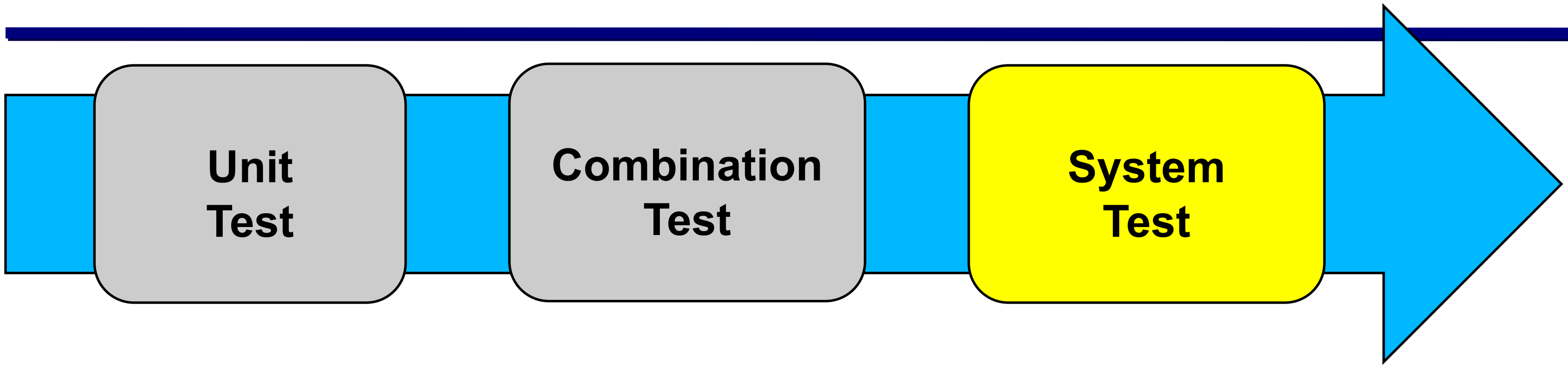


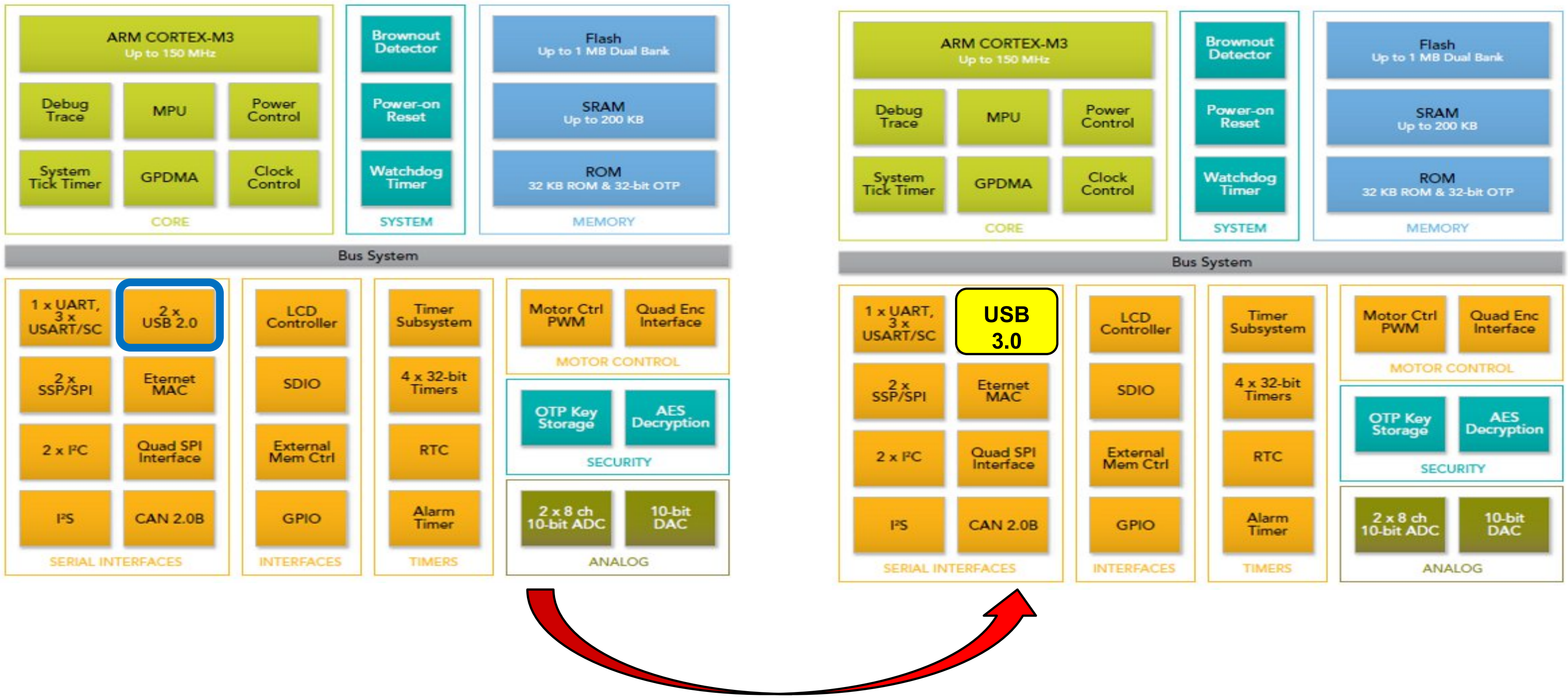
Test Environment





Test Environment



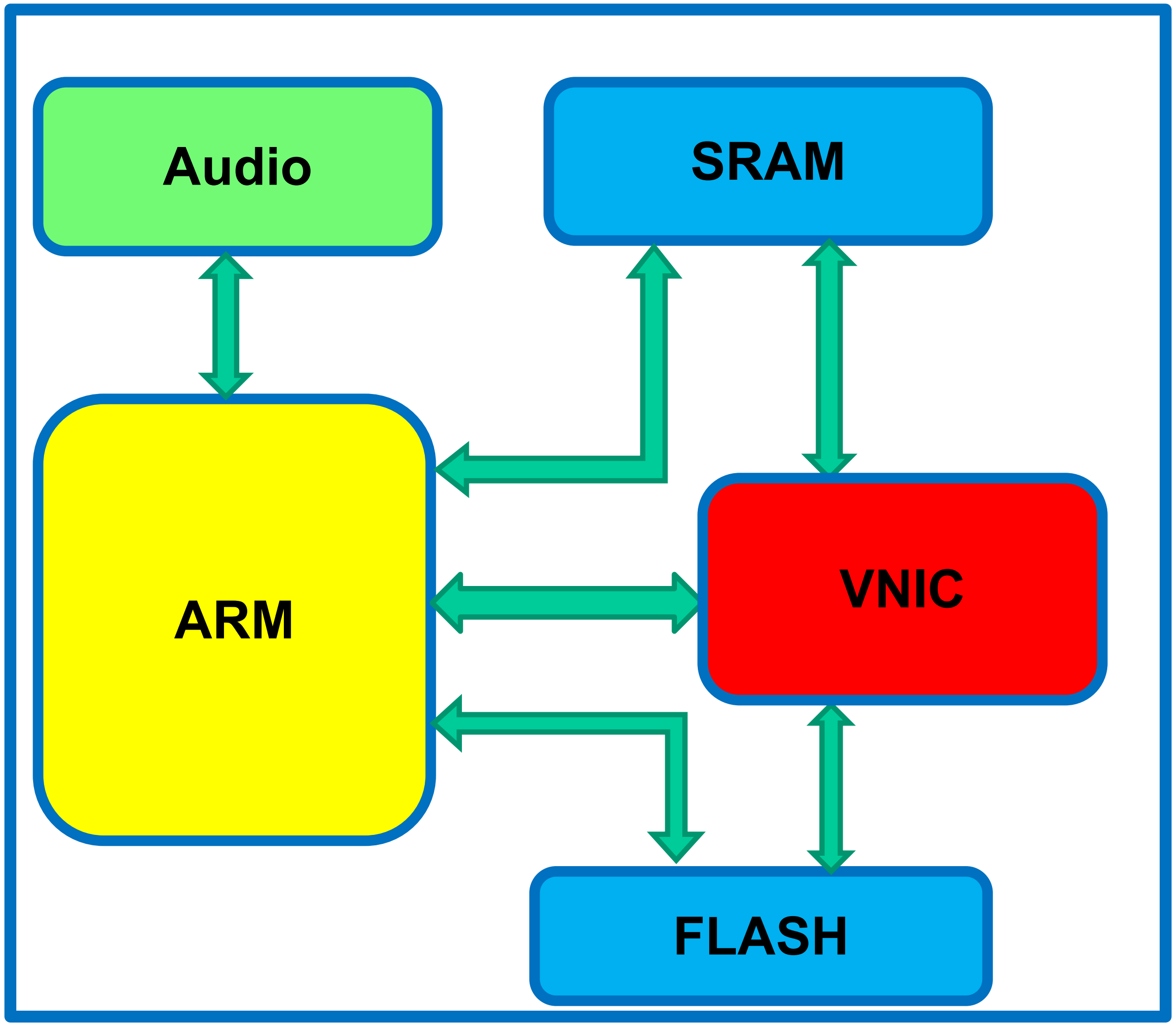


Dynamic Timing Analysis

- ❖ Dynamic vs. Static Timing Analysis
- ❖ Dynamic Timing Analysis
- ❖ **Simulation vs. Evaluation**
- ❖ **Testbench – Build The Test Cases**
- ❖ Testbench - Catch The Outputs

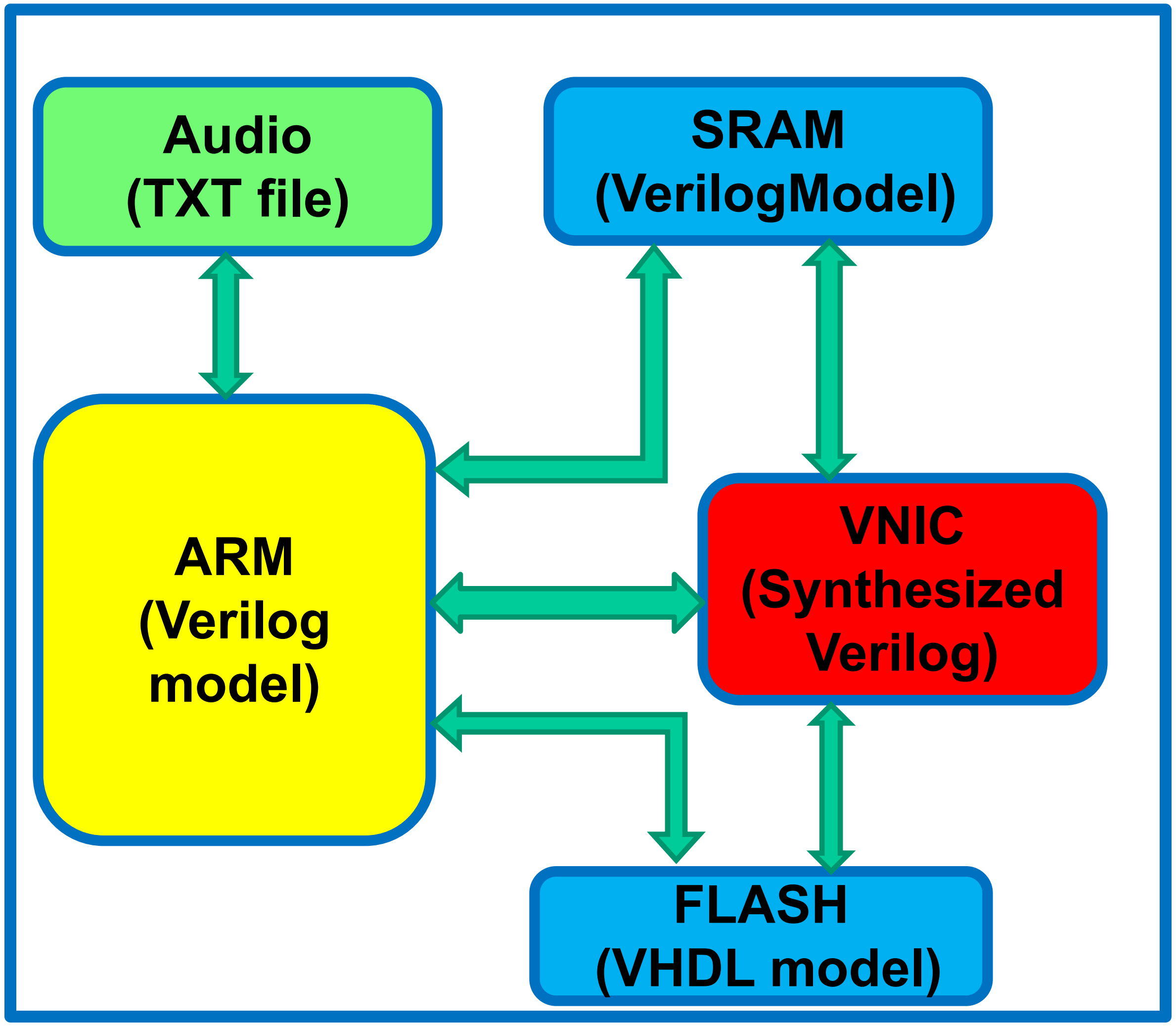
Simulation vs. Evaluation

Specification



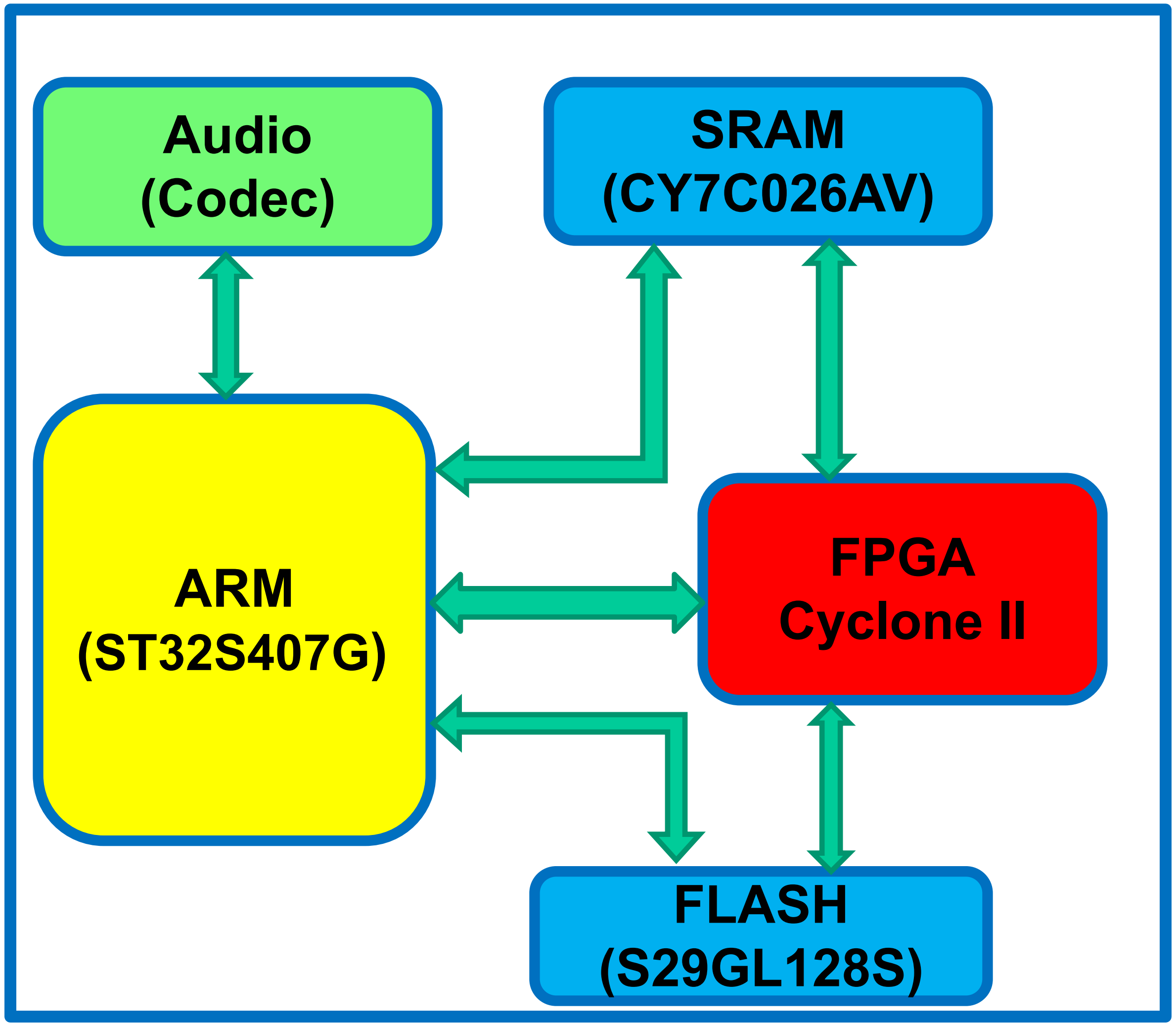
Simulation vs. Evaluation

Simulation Environment

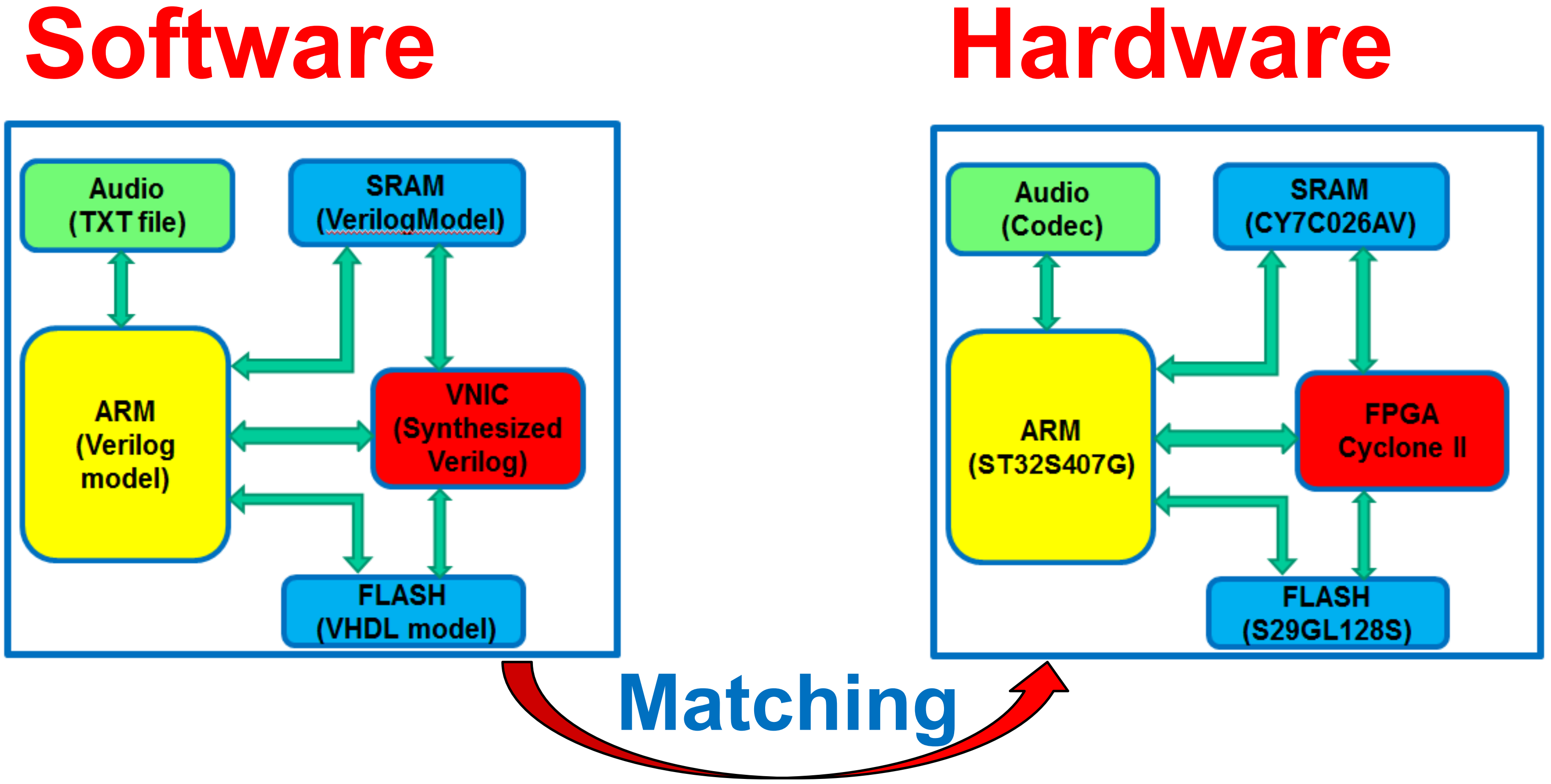


Simulation vs. Evaluation

Evaluation Environment



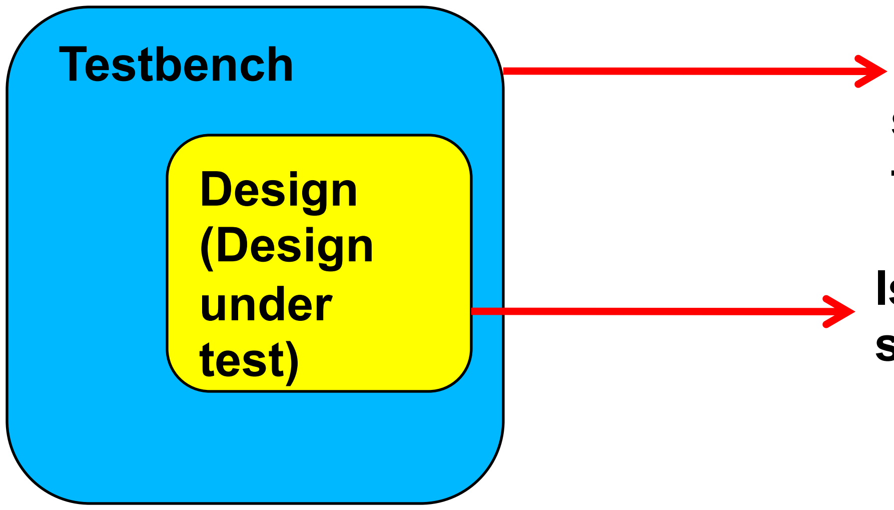
Simulation vs. Evaluation



Dynamic Timing Analysis

- Dynamic vs. Static Timing Analysis
- Dynamic Timing Analysis
- Simulation vs. Evaluation
- **Testbench – Build The Test Cases**
- Testbench - Catch The Outputs

Dynamic Timing Analysis - Testbench

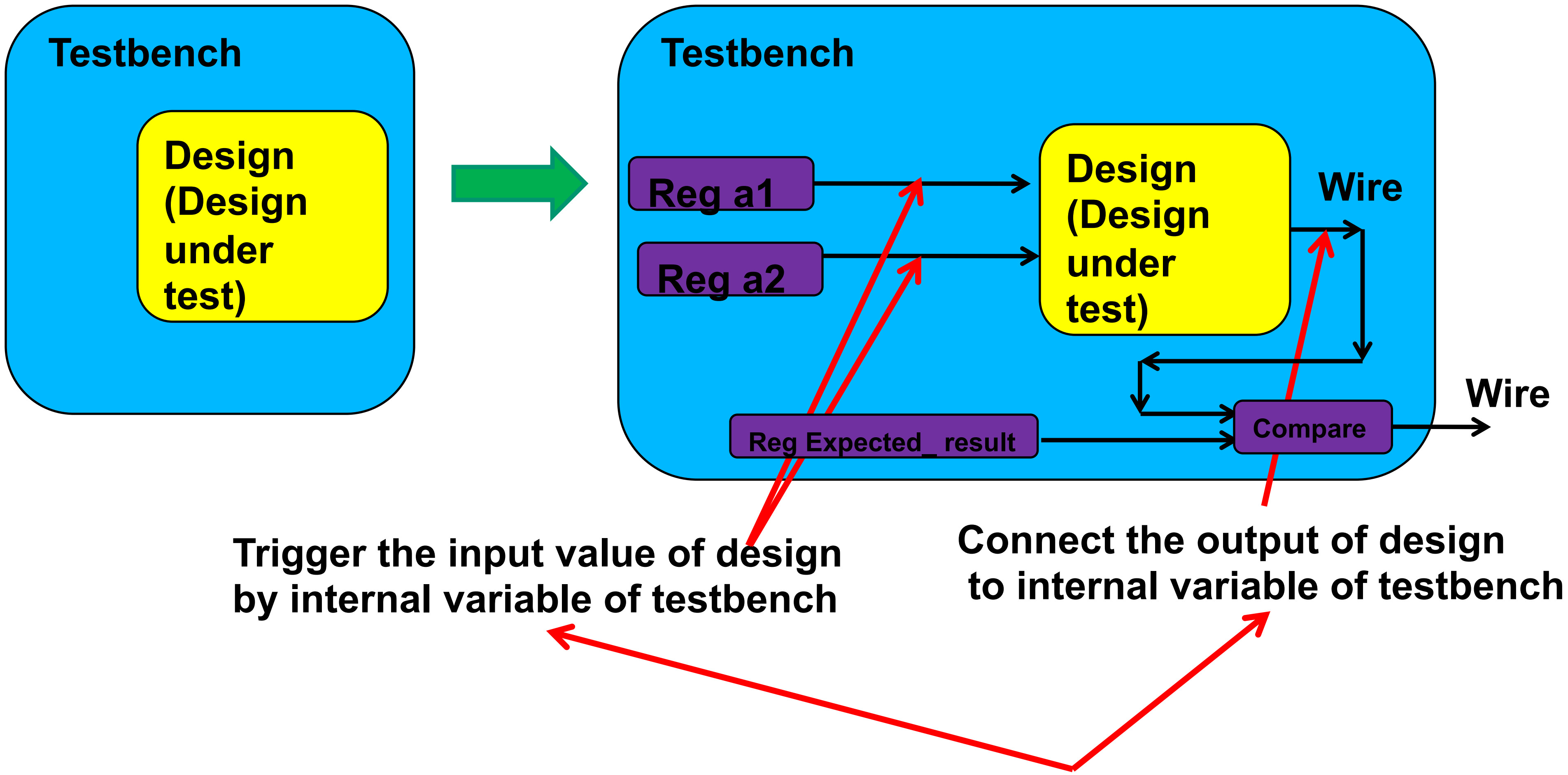


Is written in verilog/System Verilog/VHDL source code to verify the design in dynamic timing verification phase

Is written in verilog/VHDL source code to synthesize to netlist for product

❖After compiling a hardware description language like Verilog, it is required to apply inputs to the program in order to obtain out puts. Applying the inputs involves initial conditions. As the systems designed using Verilog are electronic, the initial conditions plays a vital role. Hence, all these conditions along with the information as to where the input is expected to change from 1 to 0 or 0 to 1 is provided to the Verilog program. **This is done in the form of a wave or another Verilog program. These are called Verilog test benches**. In other words, test benches are the means of applying inputs to Verilog program.

Dynamic Timing Analysis - Testbench

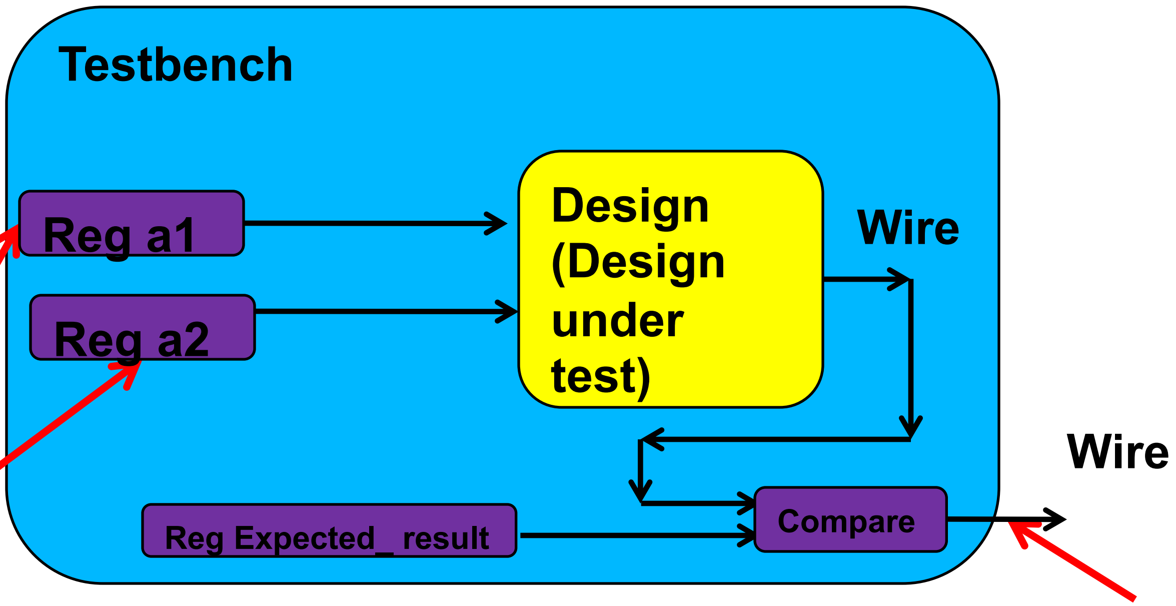


Trigger the input value of design by internal variable of testbench

Connect the output of design to internal variable of testbench

Instance the design into testbench

Dynamic Timing Analysis - Testbench

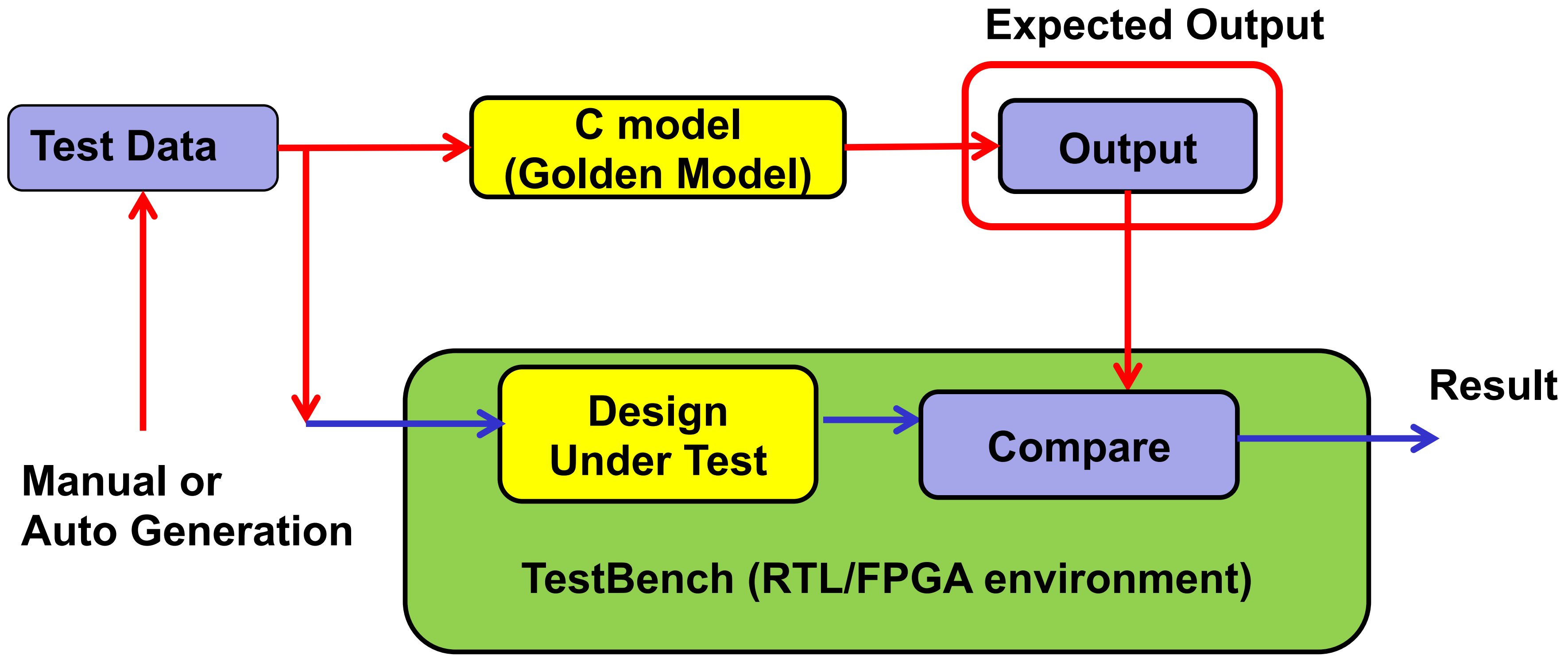


How to change the value of “a1” and “a2” to test many cases ?

How to catch the result value ?
+ Catch the report
+ Catch the wave form

How to write the testbench in verilog code ?
Which support tool to catch the wave form?

Dynamic Timing Analysis - Testbench



Dynamic Timing Analysis - Testbench

Testbench fearture

- **Module Declaration**
- **Parameter Declaration**
- **Internal variable Declaration**
- **Instance design**
- **Clock generation**
- **Test Case**
- **Catching the output**
- **Export the waveform**

```
module t_examp1_and_gate ;
```

```
parameter DATA_WIDTH = 8;
```

```
reg   t_system_clock;
reg   t_system_rst_n;
reg   [DATA_WIDTH-1:0] t_second_data_in;
Reg   [DATA_WIDTH-1:0] t_first_data_in;
wire  [DATA_WIDTH-1:0] t_data_out_and_gate;
```

```
examp1_and_gate examp1_and_gate_01(
                  .system_clock(t_system_clock)
                , .system_rst_n(t_system_rst_n)
                , .fist_data_in(t_first_data_in)
                , .second_data_in(t_second_data_in)
                , .data_out_and_gate(t_data_out_and_gate)
                );
```

Dynamic Timing Analysis - Testbench

Testbench fearture

- **Module Declaration**
- **Parameter Declaration**
- **Internal variable Declaration**
- **Instance design**
- **Clock generation**
- **Test Case**
- **Catching the output**
- **Export the waveform**

```
always begin
      t_system_clock = 1'b0;
    # 50;
      t_system_clock = 1'b1;
    # 50;
end
```

```
initial begin
  #1    t_system_rst_n = 0;
  #10  t_system_rst_n = 1;
  #10  t_first_data_in   = 8'b1101_0101;
         t_second_data_in = 8'b1010_1010;
  #100 $finish;
end
```

```
initial begin
    $monitor("time=%d, system_clock=%b, system_rst_n=%b,
                t_first_data_in:%b, t_second_data_in:%b,
                data_out_and_gate=%b \n", $stime,
                t_system_clock,  t_system_rst_n, t_first_data_in,
                t_second_data_in,     t_data_out_and_gate);
end
```

Dynamic Timing Analysis - Testbench

Testbench fearture

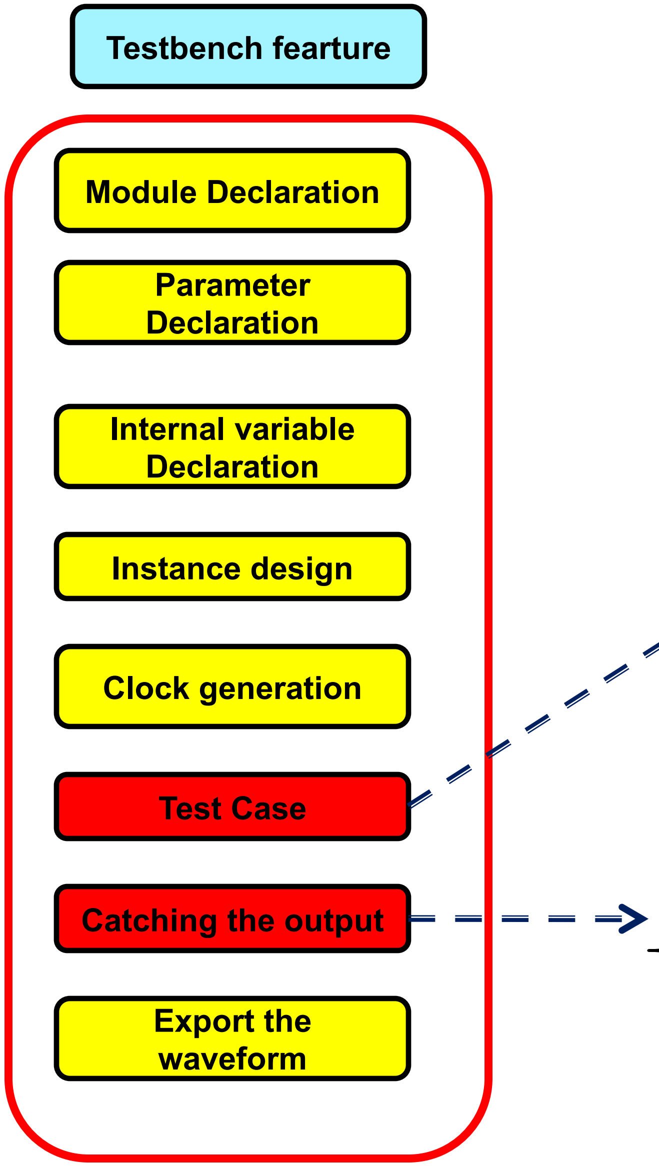
- Module Declaration
- Parameter Declaration
- Internal variable Declaration
- Instance design
- Clock generation
- Test Case
- Catching the output
- Export the waveform

```
initial begin
   $vcdplusfile ("examp1_and_gate.vpd");
   $vcdpluson();
end
```

Dynamic Timing Analysis - Testbench

```
//========= FIle : t_examp1_and_gate.v
//========= Version: 1.0
//========= Update Note: Fist version

module t_examp1_and_gate;

//parameter define
parameter DATA_WIDTH = 8;

//inputs define
reg t_system_clock;
reg t_system_rst_n;
reg [DATA_WIDTH-1:0] t_second_data_in;
reg [DATA_WIDTH-1:0] t_first_data_in;

//output define
wire [DATA_WIDTH-1:0] t_data_out_and_gate;

//clock define
always begin
      t_system_clock = 1'b0;
    # 50;
      t_system_clock = 1'b1;
    # 50;
end

//instance define
examp1_and_gate examp1_and_gate_01(
  .system_clock(t_system_clock)
, .system_rst_n(t_system_rst_n)
, .fist_data_in(t_first_data_in)
, .second_data_in(t_second_data_in)
, .data_out_and_gate(t_data_out_and_gate)
                                  );
//Catch the output
initial begin
   $monitor("time=%d, system_clock=%b,
    system_rst_n=%b, t_first_data_in:%b,
    t_second_data_in:%b, data_out_and_gate=%b \n",
    $stime,    t_system_clock,  t_system_rst_n,
    t_first_data_in,    t_second_data_in,
    t_data_out_and_gate);
end

//Declare Testcases
initial begin
  #1   t_system_rst_n = 0;
  #10  t_system_rst_n = 1;
  #10  t_first_data_in  = 8'b1101_0101;
       t_second_data_in = 8'b1010_1010;
  #100 $finish;
end

//Export the waveform
initial begin
  $vcdplusfile ("examp1_and_gate.vpd");
  $vcdpluson();
end

endmodule
```

Dynamic Timing Analysis - Testbench



```
initial begin
  #1   t_system_rst_n = 0;
  #10  t_system_rst_n = 1;
  #10  t_first_data_in  = 8'b1101_0101;
       t_second_data_in = 8'b1010_1010;
  #100 $finish;
end
```

```
initial begin
   $monitor("time=%d, system_clock=%b, system_rst_n=%b,
                  t_first_data_in:%b, t_second_data_in:%b,
                  data_out_and_gate=%b \n", $stime,
                  t_system_clock,  t_system_rst_n, t_first_data_in,
                  t_second_data_in,    t_data_out_and_gate);
end
```

Dynamic Timing Analysis - Testbench

Testbench fearture

- Module Declaration
- Parameter Declaration
- Internal variable Declaration
- Instance design
- Clock generation
- Test Case
- Catching the output
- Export the waveform

Q1. HOW TO INTEGRATE MANY TESTCASES ?
Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

```
initial begin
  #1   t_system_rst_n = 0;
  #10  t_system_rst_n = 1;
  #10  t_first_data_in  = 8'b1101_0101;
       t_second_data_in = 8'b1010_1010;
  #100 $finish;
end
```

```
initial begin
   $monitor("time=%d, system_clock=%b, system_rst_n=%b,
             t_first_data_in:%b, t_second_data_in:%b,
             data_out_and_gate=%b \n", $stime,
             t_system_clock,  t_system_rst_n, t_first_data_in,
             t_second_data_in,    t_data_out_and_gate);

end
```

Dynamic Timing Analysis

- Dynamic vs. Static Timing Analysis
- Dynamic Timing Verification-Testbench
- **Testbench – Build The Test Cases**
- Testbench - Catch The Outputs
- How To Make The Good Test Cases

Testbench – Build The Test Cases

Q1. HOW TO INTEGRATE MANY TESTCASES ?

```
initial begin
    clk = 0;
    read_data = 0;
    rd = 0;
    wr = 0;
    ce = 0;
    addr = 0;
    data_wr = 0;
    data_rd = 0;

   //Call the write and read tasks here
    #1 cpu_write(8'h11,8'hAA);
    #1 cpu_read(8'h11,read_data);
    #1 cpu_write(8'h12,8'hAB);
    #1 cpu_read(8'h12,read_data);
    #1 cpu_write(8'h13,8'h0A);
    #1 cpu_read(8'h13,read_data);
    #100 $finish;
 end
```

Build Test Case

Testbench – Build The Test Cases

Q1. HOW TO INTEGRATE MANY TESTCASES ?

task

```
initial begin
    clk = 0;
    read_data = 0;
    rd = 0;
    wr = 0;
    ce = 0;
    addr = 0;
    data_wr = 0;
    data_rd = 0;

   //Call the write and read tasks here
    #1 cpu_write(8'h11,8'hAA);              //TEST CASE 1
    #1 cpu_read(8'h11,read_data);          //TEST CASE 2
    #1 cpu_write(8'h12,8'hAB);              //TEST CASE 3
    #1 cpu_read(8'h12,read_data);          //TEST CASE 4
    #1 cpu_write(8'h13,8'h0A);              //TEST CASE 5
    #1 cpu_read(8'h13,read_data);          //TEST CASE 6
    #100 $finish;
 end
```

Testbench - task

Q1. HOW TO INTEGRATE MANY TESTCASES ?

task

```
initial begin
    clk = 0;
    read_data = 0;
    rd = 0;
    wr = 0;
    ce = 0;
    addr = 0;
    data_wr = 0;
    data_rd = 0;

   //Call the write and read tasks here
    #1 cpu_write(8'h11,8'hAA);          //TEST CASE 1
    #1 cpu_read(8'h11,read_data);       //TEST CASE 2
    #1 cpu_write(8'h12,8'hAB);          //TEST CASE 3
    #1 cpu_read(8'h12,read_data);       //TEST CASE 4
    #1 cpu_write(8'h13,8'h0A);          //TEST CASE 5
    #1 cpu_read(8'h13,read_data);       //TEST CASE 6
    #100 $finish;
 end
```

```
task cpu_read;

input [7:0] address;
output [7:0] data;
begin
   @ (posedge clk);
       addr = address;
       ce = 1;
       rd = 1;

   @ (negedge clk);
       #10 data = data_rd;

   @ (posedge clk);
       addr = 0;
       ce = 0;
       rd = 0;
end
endtask
```

Testbench - task

Q1. HOW TO INTEGRATE MANY TESTCASES ?

task

```
initial begin
    clk = 0;
    read_data = 0;
    rd = 0;
    wr = 0;
    ce = 0;
    addr = 0;
    data_wr = 0;
    data_rd = 0;

   //Call the write and read tasks here
    #1 cpu_write(8'h11,8'hAA);          //TEST CASE 1
    #1 cpu_read(8'h11,read_data);       //TEST CASE 2
    #1 cpu_write(8'h12,8'hAB);          //TEST CASE 3
    #1 cpu_read(8'h12,read_data);       //TEST CASE 4
    #1 cpu_write(8'h13,8'h0A);          //TEST CASE 5
    #1 cpu_read(8'h13,read_data);       //TEST CASE 6
    #100 $finish;
 end
```

```
task cpu_read;

input [7:0] address;
output [7:0] data;
begin
   @ (posedge clk);
      addr = address;
      ce = 1;
      rd = 1;

   @ (negedge clk);
      #10 data = data_rd;

   @ (posedge clk);
      addr = 0;
      ce = 0;
      rd = 0;
end
endtask
```

Testbench - task

Q1. HOW TO INTEGRATE MANY TESTCASES ?

```
initial begin
    clk = 0;
    read_data = 0;
    rd = 0;
    wr = 0;
    ce = 0;
    addr = 0;
    data_wr = 0;
    data_rd = 0;

   //Call the write and read tasks here
    #1 cpu_write(8'h11,8'hAA);          //TEST CASE 1
    #1 cpu_read(8'h11,read_data);       //TEST CASE 2
    #1 cpu_write(8'h12,8'hAB);          //TEST CASE 3
    #1 cpu_read(8'h12,read_data);       //TEST CASE 4
    #1 cpu_write(8'h13,8'h0A);          //TEST CASE 5
    #1 cpu_read(8'h13,read_data);       //TEST CASE 6
    #100 $finish;
 end
```

```
task cpu_read;

input [7:0] address;
output [7:0] data;
begin
   @ (posedge clk);
      addr = address;
      ce = 1;
      rd = 1;

   @ (negedge clk);
      #10 data = data_rd;

   @ (posedge clk);
      addr = 0;
      ce = 0;
      rd = 0;
end
endtask
```

Testbench - task

Q1. HOW TO INTEGRATE MANY TESTCASES ?

❖Tasks are defined in the module in which they are used. It is possible to define a task in a separate file and use the compile directive 'include to include the task in the file which instantiates the task.

❖Tasks can include timing delays, like posedge, negedge, # delay and wait.

❖Tasks can have any number of inputs and outputs.

❖The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task by the caller are used.

❖Tasks can take, drive and source global variables, when no local variables are used. When local variables are used, basically output is assigned only at the end of task execution.

❖Tasks can **call another task or function**.

❖Tasks can be used for modeling **both combinational and sequential logic.**

❖A task must be specifically called with a statement, it cannot be used within an expression as a function can.

Testbench - function

Q1. HOW TO INTEGRATE MANY TESTCASES ?

```
module function_calling(a, b, c, d, e, f);
input a, b, c, d, e ;
output  f;
wire f;
`include "myfunction.v“
assign f = (myfunction (a,b,c,d)) ? e :0;
endmodule
```

```
module simple_function();
   function myfunction;
   input a, b, c, d;
      begin
         myfunction = ((a+b) + (c-d));
      end
   endfunction
endmodule
```

Testbench - function

Q1. HOW TO INTEGRATE MANY TESTCASES ?

```
module function_calling(a, b, c, d, e, f);
input a, b, c, d, e ;
output  f;
wire f;
`include "myfunction.v“
assign f = (myfunction (a,b,c,d)) ? e :0;
endmodule
```

```
module simple_function();
   function myfunction;
   input a, b, c, d;
      begin
        myfunction = ((a+b) + (c-d));
      end
   endfunction
endmodule
```

Testbench - function

Q1. HOW TO INTEGRATE MANY TESTCASES ?

❖A Verilog HDL function is the same as a task, with very little differences, like function cannot drive more than one output, can not contain delays.

❖Functions are defined in the module in which they are used. It is possible to define functions in separate files and use compile directive 'include to include the function in the file which instantiates the task.

❖Functions **can not include timing delays**, **like posedge, negedge, # delay,** which means that functions should be executed in "zero" time delay.

❖Functions can have any number of inputs but only one output.

❖The variables declared within the function are local to that function. The order of declaration within the function defines how the variables passed to the function by the caller are used.

❖Functions can take, drive, and source global variables, when no local variables are used. When local variables are used, basically output is assigned only at the end of function execution.

❖Functions can be used for **modeling combinational logic**.

❖Functions can **call other functions, but can not call tasks**.

Dynamic Timing Analysis

- Dynamic vs. Static Timing Analysis
- Dynamic Timing Analysis
- Simulation vs. Evaluation
- Testbench – Build The Test Cases
- **Testbench - Catch The Outputs**

Testbench – Catch the outputs

Testbench fearture

- Module Declaration
- Parameter Declaration
- Internal variable Declaration
- Instance design
- Clock generation
- Test Case
- Catching the output
- Export the waveform

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

```
initial begin
    $monitor("time=%d, system_clock=%b, system_rst_n=%b,
                t_first_data_in:%b, t_second_data_in:%b,
                data_out_and_gate=%b \n", $stime,
                t_system_clock,  t_system_rst_n, t_first_data_in,
                t_second_data_in,    t_data_out_and_gate);
end
```

Testbench – Catch the outputs

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

$display	$fdisplay	$write	$fwrite
$displayh	$fdisplayh	$writeh	$fwriteh
$displayb	$fdisplayb	$writeb	$fwriteb
$displayo	$fdisplayo	$writeo	$fwriteo
$strobe	$fstrobe	$monitor	$fmonitor
$strobeh	$fstrobeh	$monitorh	$fmonitorh
$strobeb	$fstrobeb	$monitorb	$fmonitorb
$strobeo	$fstrobeo	$monitoro	$fmonitoro

} **System Functions**

Testbench – Catch the outputs

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

$display	$fdisplay	**$write**	$fwrite
$displayh	$fdisplayh	$writeh	$fwriteh
$displayb	$fdisplayb	$writeb	$fwriteb
$displayo	$fdisplayo	$writeo	$fwriteo
$strobe	$fstrobe	**$monitor**	$fmonitor
$strobeh	$fstrobeh	$monitorh	$fmonitorh
$strobeb	$fstrobeb	$monitorb	$fmonitorb
$strobeo	$fstrobeo	$monitoro	$fmonitoro

System Functions

Catch the outputs - $display & $write

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖The **$display** and **$write** tasks are the main system task routines for displaying information. The two tasks are identical except that $display automatically adds a new line character to the end of its output, whereas $write does not. Thus, if you want to print several messages on a single line, use $write.

❖The **$display** and **$write** tasks display their parameters in the same order that they appear in the parameter list. Each parameter can be a quoted string, an expression that returns a value, or a null parameter. The syntax for these tasks is as follows:

$display(“P1=%b, P2=%d, ... , Pn=%o \n**”, P1, P2, …, Pn);**
$write(P1, P2, ... , Pn);

❖ **Notes:**

\n	is the new line character
\t	is the tab character
\\	is the \ character
\"	is the " character
\o	is a character specified in 1-3 octal digits
%%	is the percent character

Catch the outputs - $display & $write

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖ **Notes:**

%h or %H →	Display in hexadecimal format
%d or %D →	Display in decimal format
%o or %O →	Display in octal format
%b or %B →	Display in binary format
%c or %C →	Display in ASCII character format
%v or %V →	Display net signal strength
%m or %M →	Display hierarchical name
%s or %S →	Display as a string
%t or %T →	Display in current time format
%e or %E →	Display real number in an exponential format.
%f or %F →	Display real number in a decimal format.
%g or %G →	Display real number in exponential or decimal format, whichever format results in the shorter printed output.

Catch the outputs - $display & $write

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖ **Notes:**

%h or %H →	Display in hexadecimal format
%d or %D →	Display in decimal format
%o or %O →	Display in octal format
%b or %B →	Display in binary format
%c or %C →	Display in ASCII character format
%v or %V →	Display net signal strength
%m or %M →	Display hierarchical name
%s or %S →	Display as a string
%t or %T →	Display in current time format
%e or %E →	Display real number in an exponential format.
%f or %F →	Display real number in a decimal format.
%g or %G →	Display real number in exponential or decimal format, whichever format results in the shorter printed output.

```
$display("%d", 1'bx);
$display("%h", 14'bx01010);
$display("%h %o", 12'b001xxx101x01,12'b001xxx101x01);
```

Which is dumped ?

Catch the outputs - $display & $write

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖ **Notes:**

%h or %H →	Display in hexadecimal format
%d or %D →	Display in decimal format
%o or %O →	Display in octal format
%b or %B →	Display in binary format
%c or %C →	Display in ASCII character format
%v or %V →	Display net signal strength
%m or %M →	Display hierarchical name
%s or %S →	Display as a string
%t or %T →	Display in current time format
%e or %E →	Display real number in an exponential format.
%f or %F →	Display real number in a decimal format.
%g or %G →	Display real number in exponential or decimal format, whichever format results in the shorter printed output.

$**display**("%d", 1'bx); **X**

$**display**("%h", 14'bx01010); **XXXA**

$**display**("%h %o", 12'b001xxx101x01,12'b001xxx101x01); **XXX 1X5X**

Catch the outputs - $monitor

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖When you invoke a **$monitor** task with one or more parameters, the simulator sets up a mechanism whereby each time a variable or an expression in the parameter list changes value.

❖The **$monitoron** and **$monitoroff** tasks control a monitor flag that enables and disables the monitoring, so that you can easily control when monitoring should occur. Use **$monitoroff** to turn off the flag and disable the monitoring. Use **$monitoron** to turn on the flag so that monitoring is enabled and the most recent call to $monitor can resume its display.

❖A call to **$monitoron** always produces a display immediately after it is invoked, regardless of whether a value change has taken place; this is used to establish the initial values at the beginning of a monitoring session.

❖ **Example: $monitor**(**$time:** "rxd=%b txd=%b“, **rxd**, **txd**);

Catch the outputs - $strobe

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖When Verilog-XL encounters the **$strobe** system task, it displays the specified information at the end of the time unit. The parameters for this task are specified in exactly the same manner as for the $display system task—including the use of escape sequences for special characters and format specifications (see “Display and Write Tasks” on page 333). The syntax is as follows:

```
$strobe(P1, P2, ..., Pn);
```

❖The following example shows how the $strobe system task is used:

```
forever @(negedge clock)
$strobe ("At time %d, data is %h“, $time,data);
```

❖In this example, **$strobe** writes the time and data information to the standard output and to the log file at each negative edge of the clock. The action occurs just before simulation time is advanced, after all other events at that time have occurred, so that the data written is sure to be the correct data for that simulation time.

❖The **strobe** tasks produce output when they are executed, and there is no on/off control necessary.

Catch the outputs - Summary

Q2. WHICH METHOD TO CATCH THE OUTPUT SUITABLY ?

❖ **$display** and **$strobe** display once every time they are executed, where as **$monitor** displays every time one of its parameters changes.

❖The difference between **$display** and **$strobe** is that $strobe displays the parameters at the very end of the current simulation time unit rather than exactly where it is executed.

❖**$strobe** and **$monitor** show the updated values of all requested variables at the end of a simulation time step, after all other assignments for that simulation time step are complete.

Q & A