System Verilog

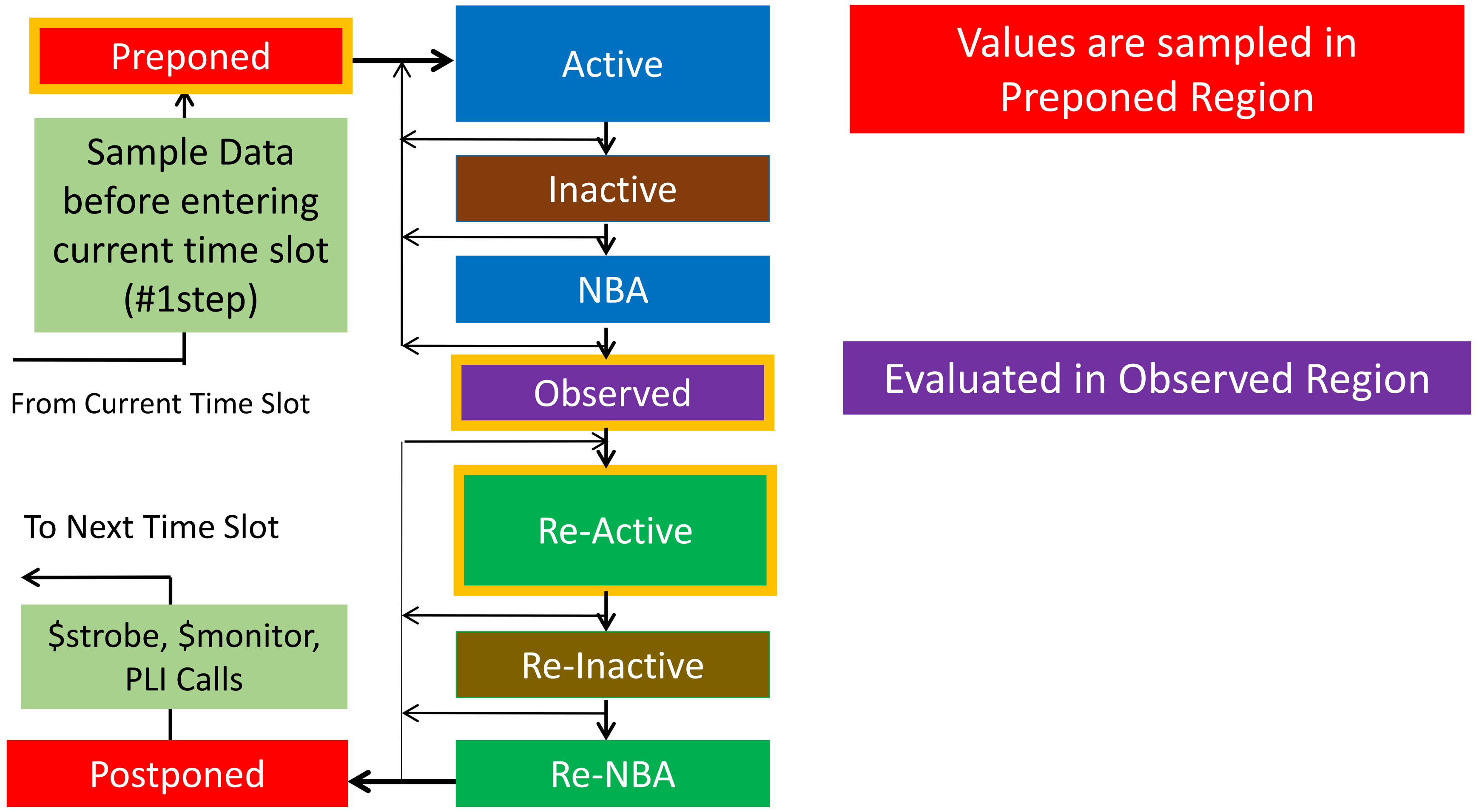
ASSERTIONS

Assertions and Coverage

- Assertions
  - These are checks which used to verify that your design meets the given requirements.
  - Example: grant should be high two clock cycles after request.

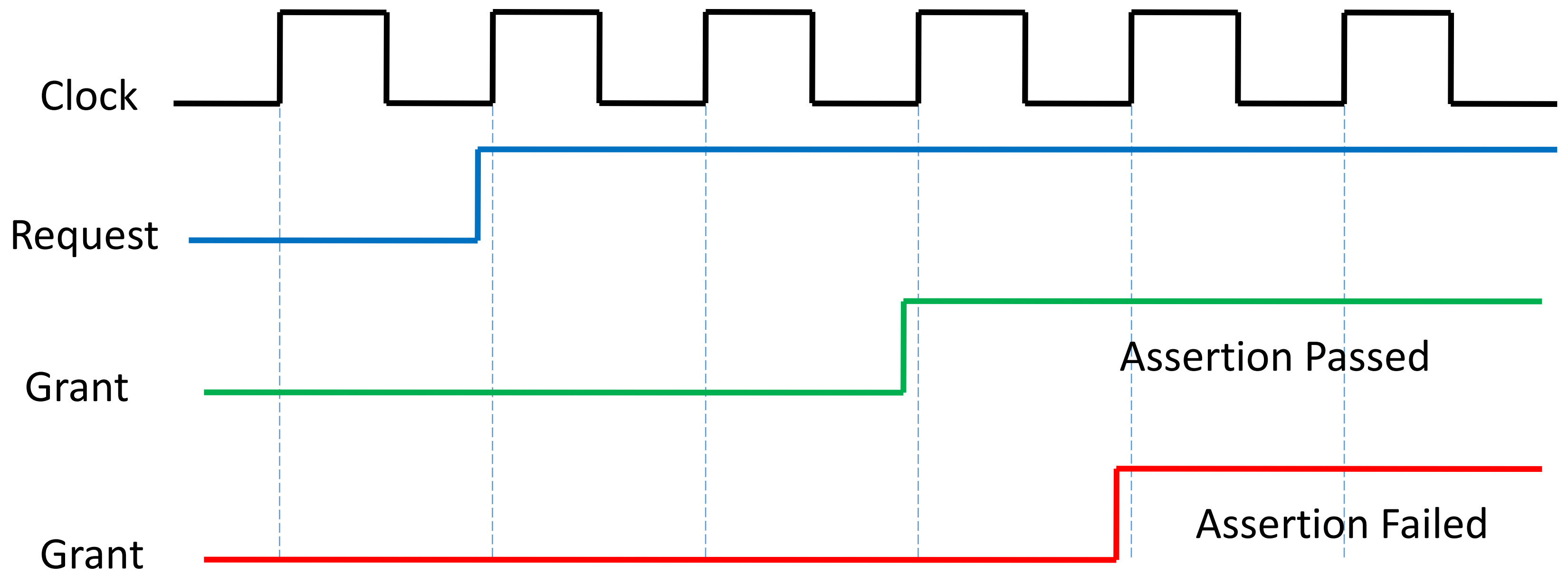
- Coverage
  - These are used to judge what percentage of your test plan or functionality has been verified.
  - They are used to judge quality of stimulus.
  - They help us in finding what part of code remains untested.

Assertion Region


Preponed
Active
Sample Data before entering current time slot (#1step)
Inactive
NBA
From Current Time Slot
Observed
Re-Active
To Next Time Slot
$strobe, $monitor, PLI Calls
Re-Inactive
Postponed
Re-NBA
Values are sampled in Preponed Region
Evaluated in Observed Region

Assertions

Design Rule : Grant should be asserted 2 clock cycles after request



Assertions

- Types of Assertions
  - ❖Immediate Assertions.
  - ❖Concurrent Assertions.

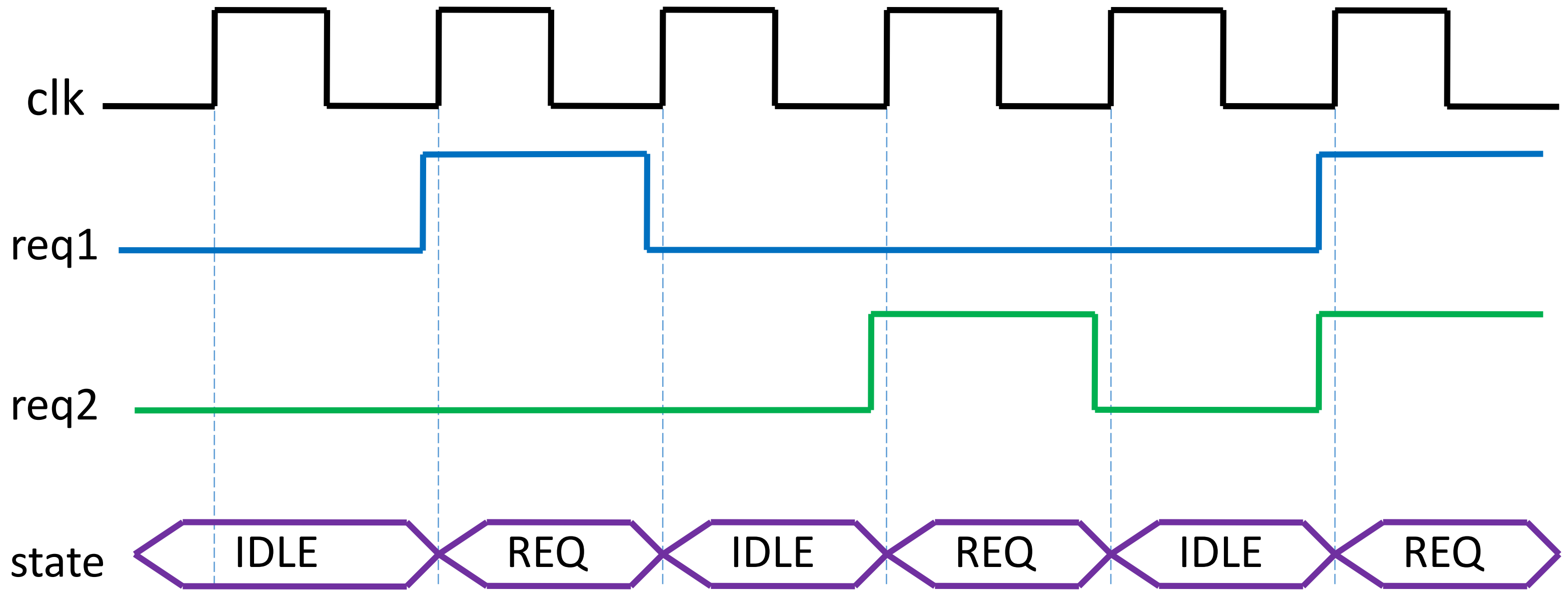
Immediate Assertions

- These are used to check condition at current time.
- These checks are Non Temporal i.e. checks are not performed across time or clock cycles.
- These are used inside procedural blocks (initial/always and tasks/functions).
- Assertion fails if expression evaluates to 0, X or Z.
- In case fail_statement is not provided and assertion fails, then in that case an error is reported during runtime.

```
[Label] : assert (expression) [pass _statement];
                              [else fail_statement;]
```

Example

Design Rule : State Machine should go to REQ state only if req1 or req2 is high.



Example

```
always @ (posedge clk)
if (state==REQ)                    //if current state is REQ
assert ( req1 || req2)             //Check whether req1 or
                                   //req2 is high
$info("Correct State");
else
$error("Incorrect State");
```

Assertions Severity

- $info indicates that the assertion failure carries no specific severity. Useful for printing some messages.

- $warning indicates runtime warning. Can be used to indicate non severe errors.

- $error indicates runtime error. Can be used to indicate protocol errors.

- $fatal indicates fatal error that would stop simulation.

Examples

```
always @ (posedge clk)
assert(func(a, b)) ->myevent; else error=error + 1;
//Trigger myevent if function returns 1 else increase error count.

always @ (negedge clk)
assert (y==0) error_flag=0; else error_flag=1;
//y should not be 1 at negedge of clk

always @ (state)
assert($onehot(state)) else $fatal(“state is not one hot”);
//In a one-hot encoded state machine all states should be one-hot
```

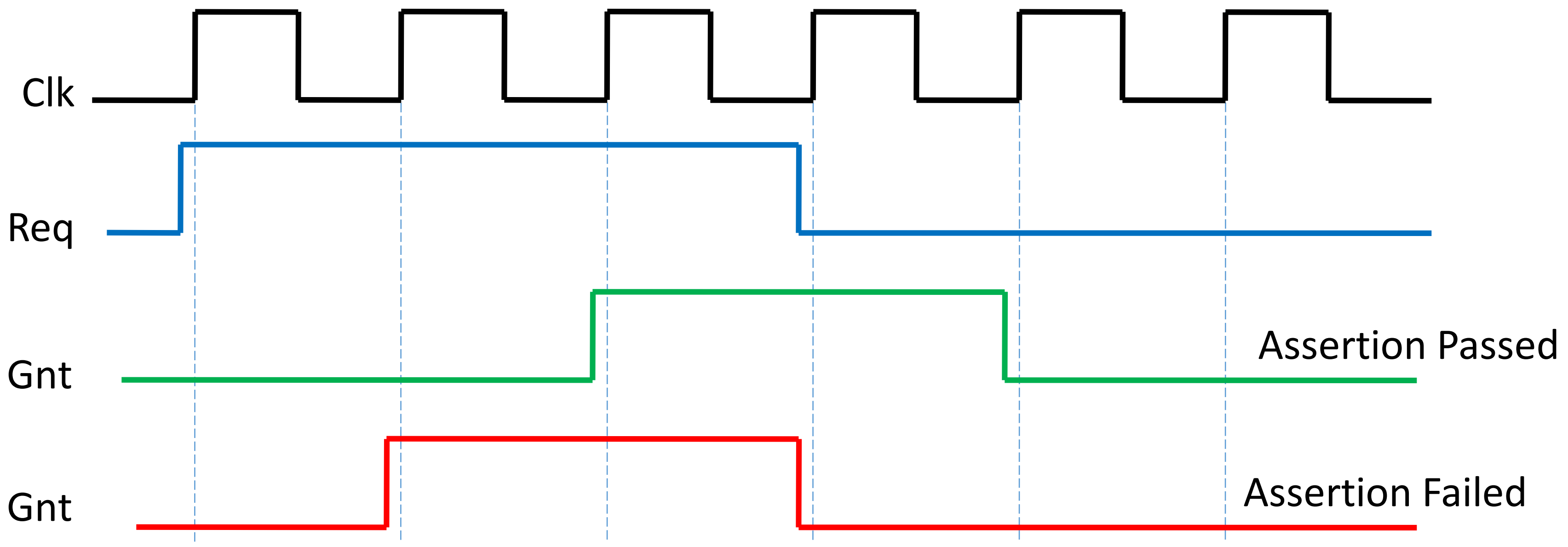
Concurrent Assertions

- These assertions test for a sequence of events spread over multiple clock cycles i.e. they are Temporal in nature.
- property keyword is used to define concurrent assertions.
- property is used to define a design specification that needs to be verified
- They are called concurrent because they occur in parallel with other design blocks.

```
[Label] : assert property (property_name) [pass_statement];
                                          [else fail_statement;]
```

Assertions

Design Rule : Grant should be high 2 clock cycles after request, followed by low request and then grant in consecutive cycles.



Example

```
property req_gnt;
@ (posedge clk) req ##2 gnt ##1 !req ##1 !gnt;
endproperty

assert property(req_gnt) else $error(“req_gnt property violated”);
```

- ## followed by a number is used to indicate no. of clock cycles.
- If gnt is not high 2 clock cycles after req goes high, violation will be reported.
- If req and gnt come at proper time but req is not low in next clock cycle, that will also lead to violations.

Properties and Sequences

- Assertions can directly include a property.

```
assert property (@ (posedge clk) a ##1 b);
```

- Assertions can be split into assertion and property declared separately

```
property myproperty;
@ (posedge clk) a ##1 b ##1 c;
endproperty

assert property (myproperty);
```

Properties and Sequences

- A property can be disabled conditionally

```
property disable_property;
@ (posedge clk) disable iff (reset)
a ##1 b ##1 c;
endproperty
```

- property block contains definition of sequence of events.
- Complex properties can be structured using multiple sequence blocks.

Properties and Sequences

```
sequence s1;
a ##1 b ##1 c;
endsequence
```

```
sequence s2;
a ##1 c;
endsequence
```

```
property p1;
@ (posedge clk) disable iff (reset)
s1 ##1 s2;
endsequence
```

```
assert property(p1);
```

Sequences

- Sequence is series of true/false expression spread over one or more clock cycles.
- It acts like basic building block for creating complex property specifications.
- Sampling edge can be specified inside a sequence. If not defined, it is inferred from property block or assert block.

```
sequence s1;
@(posedge clk) a ##1 !b ##1 c ##0 !d;
endsequence
```

Linear Sequences

- Linear sequence is finite list of System Verilog Boolean expression in a linear order of increasing time.

- A sequence is set to match if all these conditions are true:
  - The first boolean expression evaluates to true at the first sampling edge.
  - The second boolean expression evaluates to true after the delay from first expression.
  - and so forth, up to and including the last boolean expression evaluating to true at the last sampling edge.

- Sequence is evaluated on every sampling edge.

Example

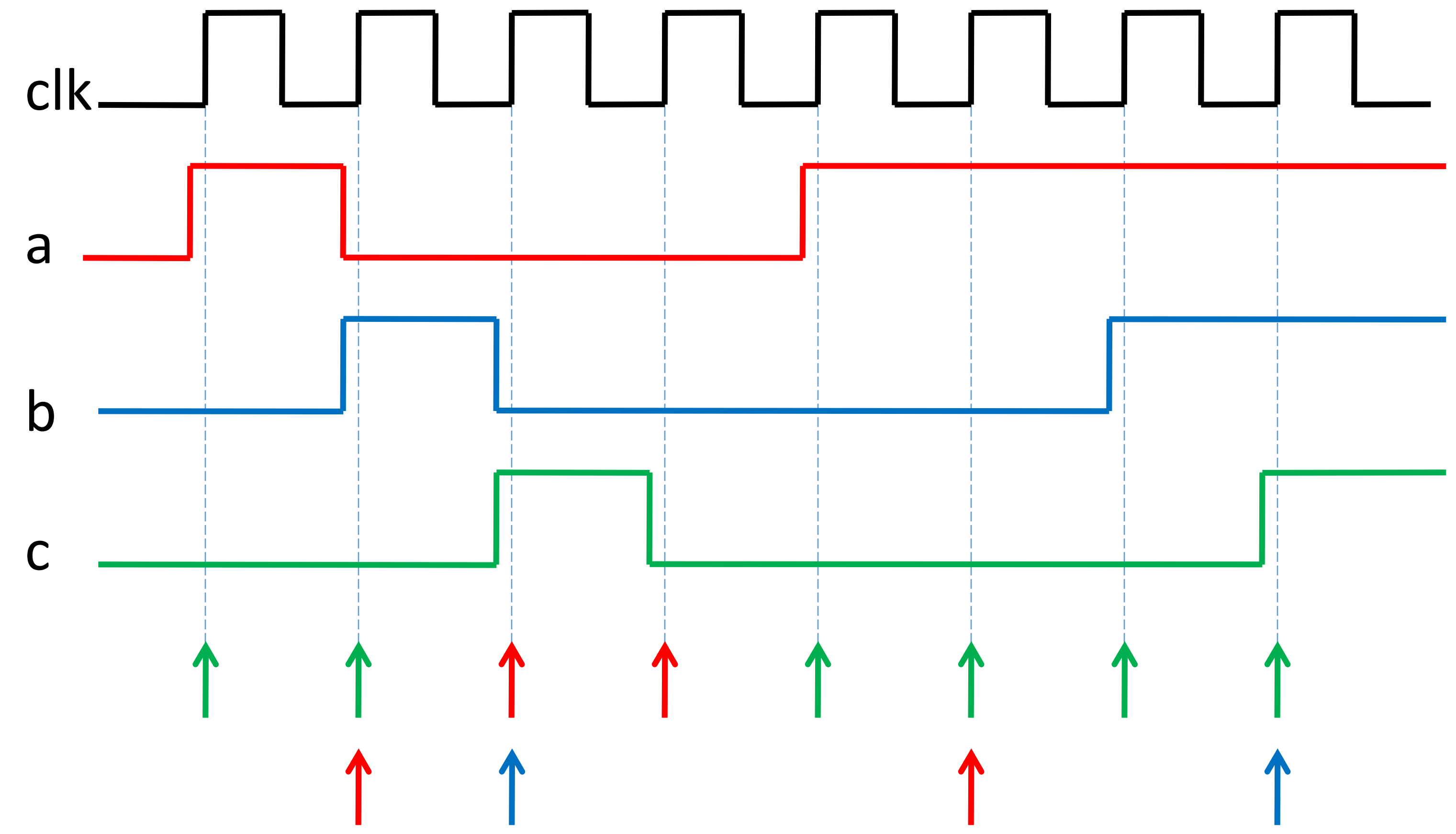
```
module test;
bit clk;
logic a=0, b=0, c=0;

always #5 clk=~clk;
property abc;
@ (posedge clk) a ##1 b ##1 c;
endproperty

assert property(abc)
$info("Sequence Occurred");
//program block
endmodule
```

```
program assert_test;
initial begin
#4 a=1;
#10 a=0; b=1;
#10 b=0; c=1;
#10 c=0;
#10 a=1;
#20 b=1;
#10 c=1;
#10;
end
endprogram
```

Example



Error Reported Evaluation in progress Sequence Occurred

Declaring Sequences

- A sequence can be declared inside:
  - Module
  - Interface
  - Program
  - Clocking block
  - Package
- Syntax:

```
sequence sequence_name [ (arguments) ];
boolean_expression;
endsequence [ : sequence_name]
```

Sequence Arguments

- Sequences can have optional Formal Arguments.
- Actual arguments can be passed during instantiation.

```
sequence s1 (data, en)
( !a && (data==2’b11)) ##1 (b==en)
endsequence
```

- Clock need not be specified in a sequence.
- In this case clock will be inferred from the property or assert statement where this sequence is instantiated.

Implication Operator

- Evaluation of a sequence can be pre-conditioned with an implication operator.
- Antecedent – LHS of implication operator
- Consequent – RHS of implication operator
- Consequent will be evaluated only if Antecedent is true.
- There are two types of implication operators:
  - Overlapping (Antecedent |-> Consequent)
  - Non-Overlapping (Antecedent |=> Consequent)

Overlapping Implication Operator

- If antecedent is true then Consequent evaluation starts immediately.
- If antecedent is false then consequent is not evaluated and sequence evaluation is considered as true this is called vacuous pass.
- $assertvacuousoff [(levels[, list])] can be used to disable vacuous pass.

```
property p1;
@ (posedge clk) en |-> (req ##2 ack);
endproperty
```

Example

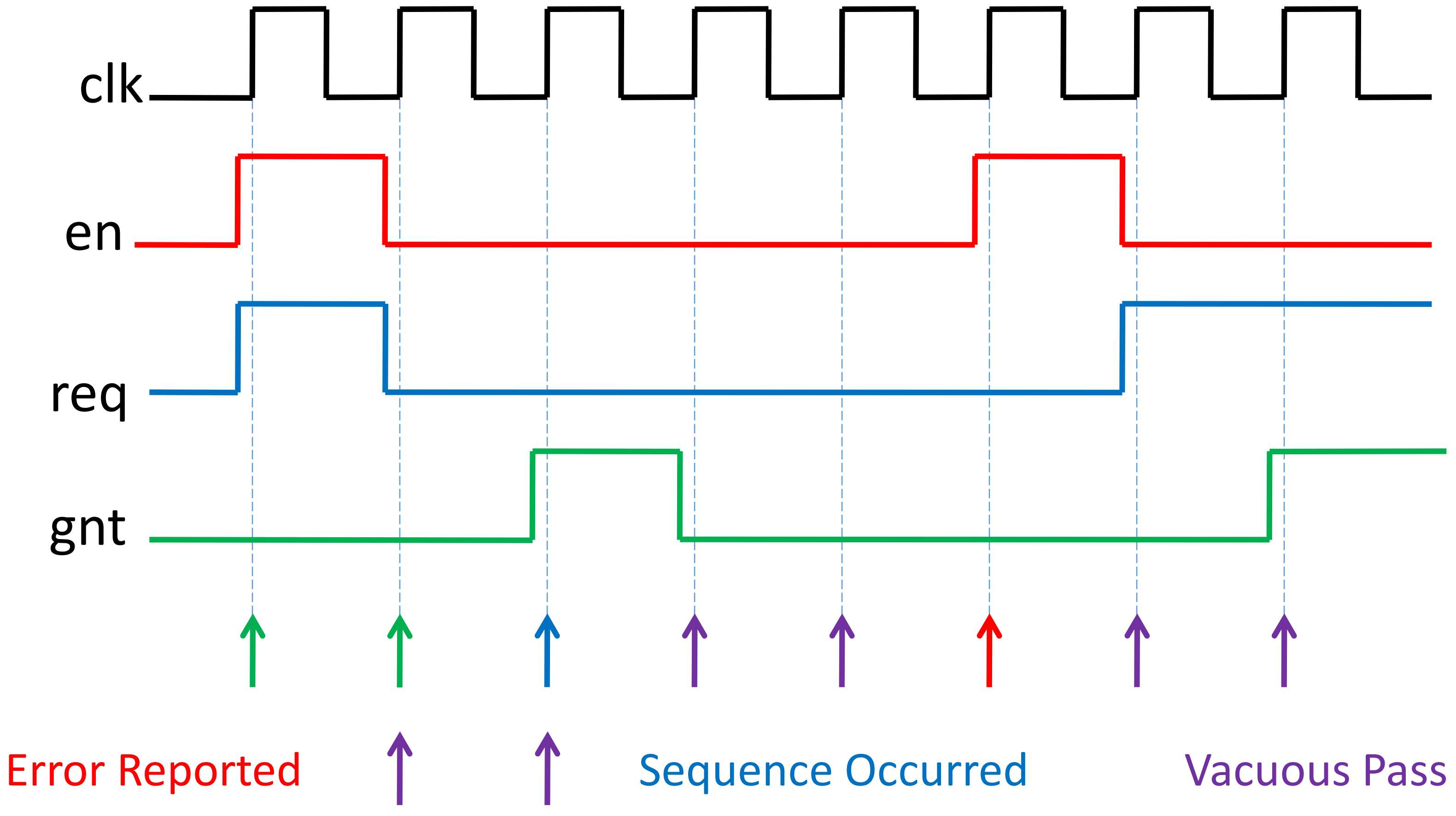
```
module test;
bit clk;
logic en=0, req=0, gnt=0;

always #5 clk=~clk;
property abc;
@ (posedge clk) en |-> req ##2 gnt;
endproperty

assert property(abc)
$info("Sequence Occurred");
//program block
endmodule
```

```
program assert_test;
initial begin
#4 en=1; req=1;
#10 en=0; req=0;
#10 gnt=1;
#10 gnt=0;
#20 en=1;
#10 en=0; req=1;
#10 req=0; gnt=1;
#10;
end
endprogram
```

Example



Error Reported

Sequence Occurred

Vacuous Pass

Evaluation in progress

Example

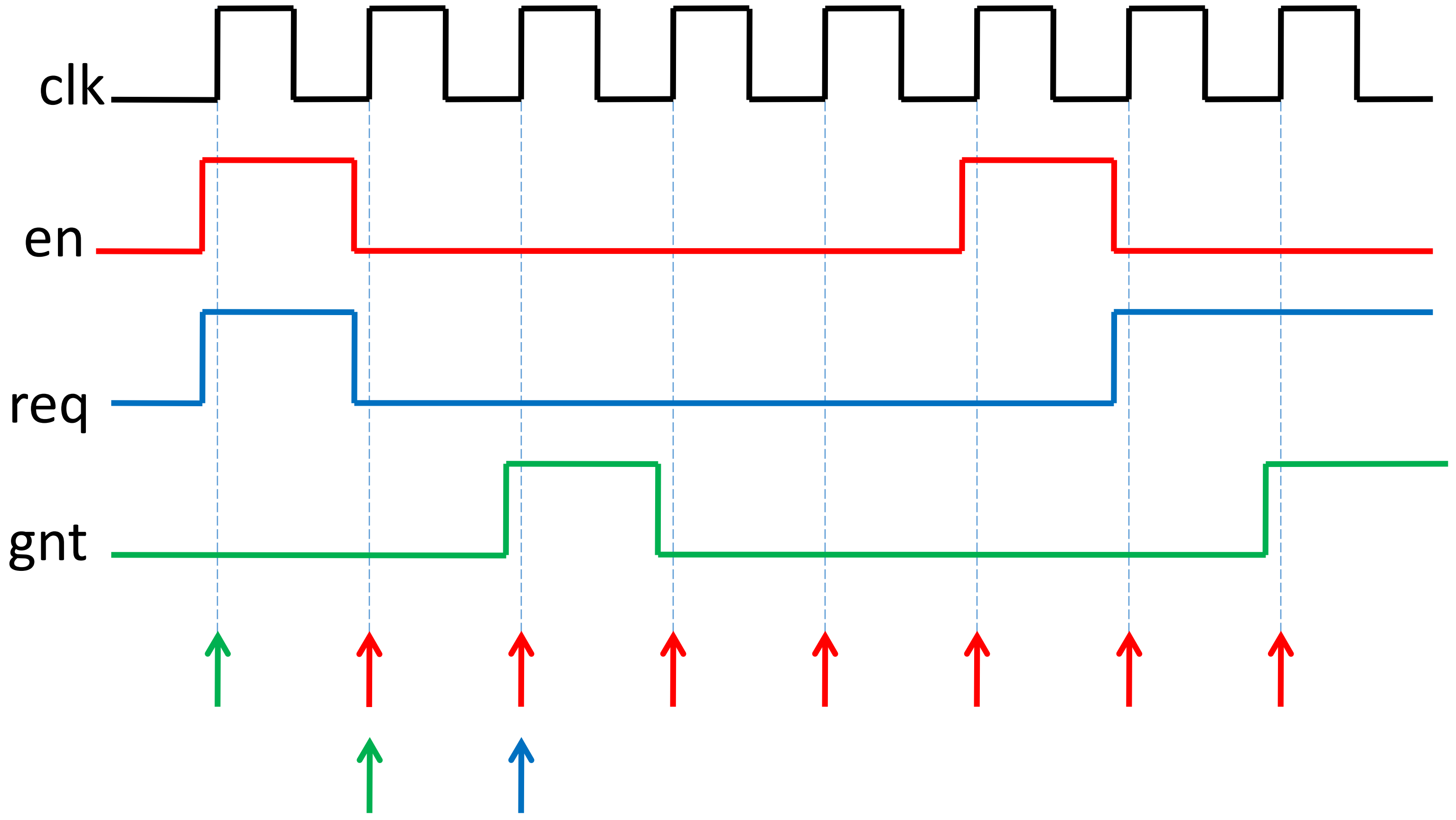
```
module test;
bit clk;
logic en=0, req=0, gnt=0;

always #5 clk=~clk;
property abc;
@ (posedge clk) en ##0 req ##2 gnt;
endproperty

assert property(abc)
$info(“Sequence Occurred”);
//program block
endmodule
```

```
program assert_test;
initial begin
#4 en=1; req=1;
#10 en=0; req=0;
#10 gnt=1;
#10 gnt=0;
#20 en=1;
#10 en=0; req=1;
#10 req=0; gnt=1;
#10;
end
endprogram
```

Example



Error Reported

Evaluation in progress

Sequence Occurred