Non-Overlapping Implication Operator

- If antecedent is true then Consequent evaluation starts in next clock cycle.
- If antecedent is false then consequent is not evaluated and sequence evaluation is considered as true this is called vacuous pass.

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

Example

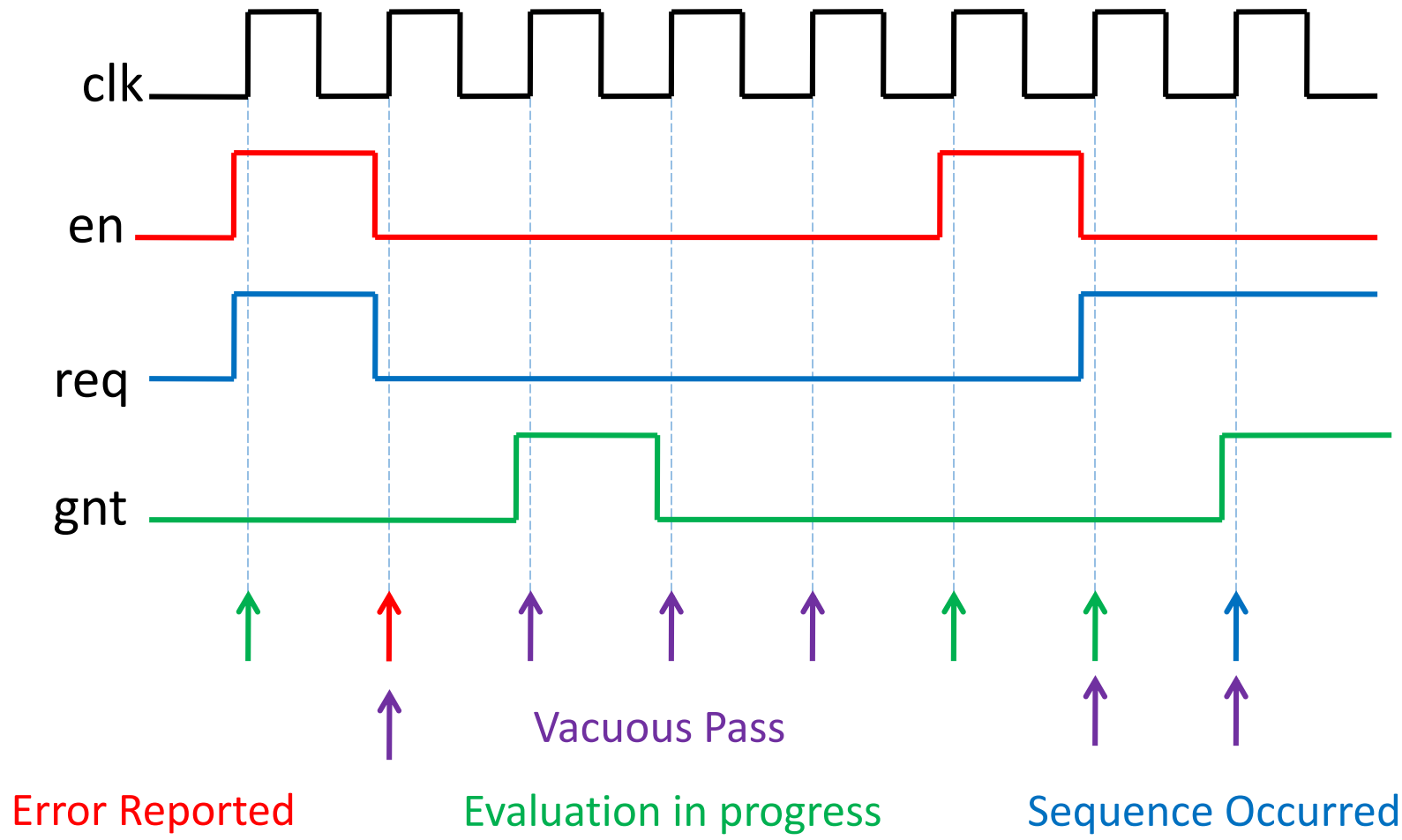
```
module test;
bit clk;
logic en=0, req=0, gnt=0;

always #5 clk=~clk;
property abc;
@ (posedge clk) en |=> req ##1 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


clk
en
req
gnt
Vacuous Pass
Error Reported
Evaluation in progress
Sequence Occurred

Example

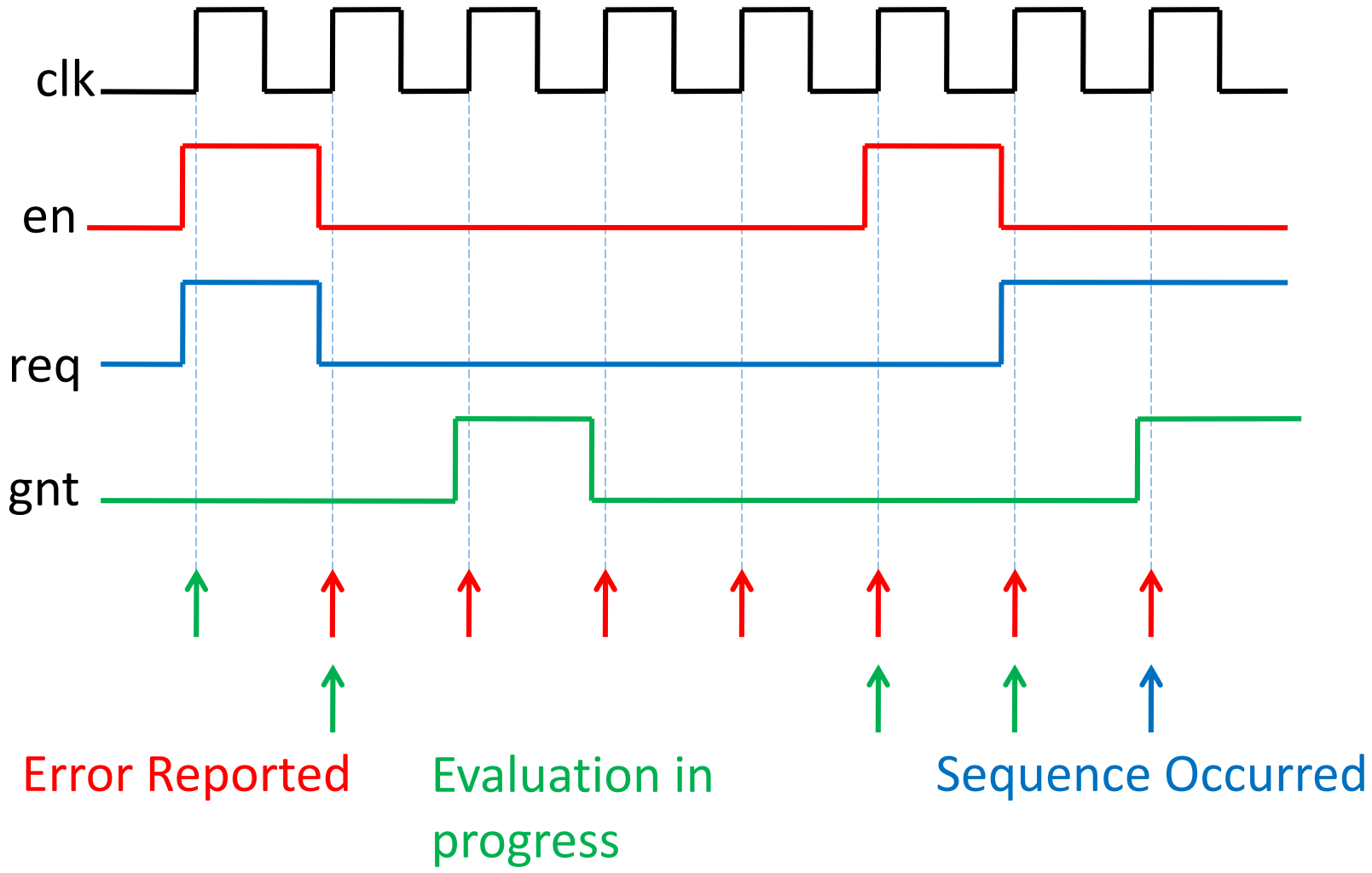
```
module test;
bit clk;
logic en=0, req=0, gnt=0;

always #5 clk=~clk;
property abc;
@ (posedge clk) en ##1 req ##1 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


clk
en
req
gnt
Error Reported
Evaluation in progress
Sequence Occurred

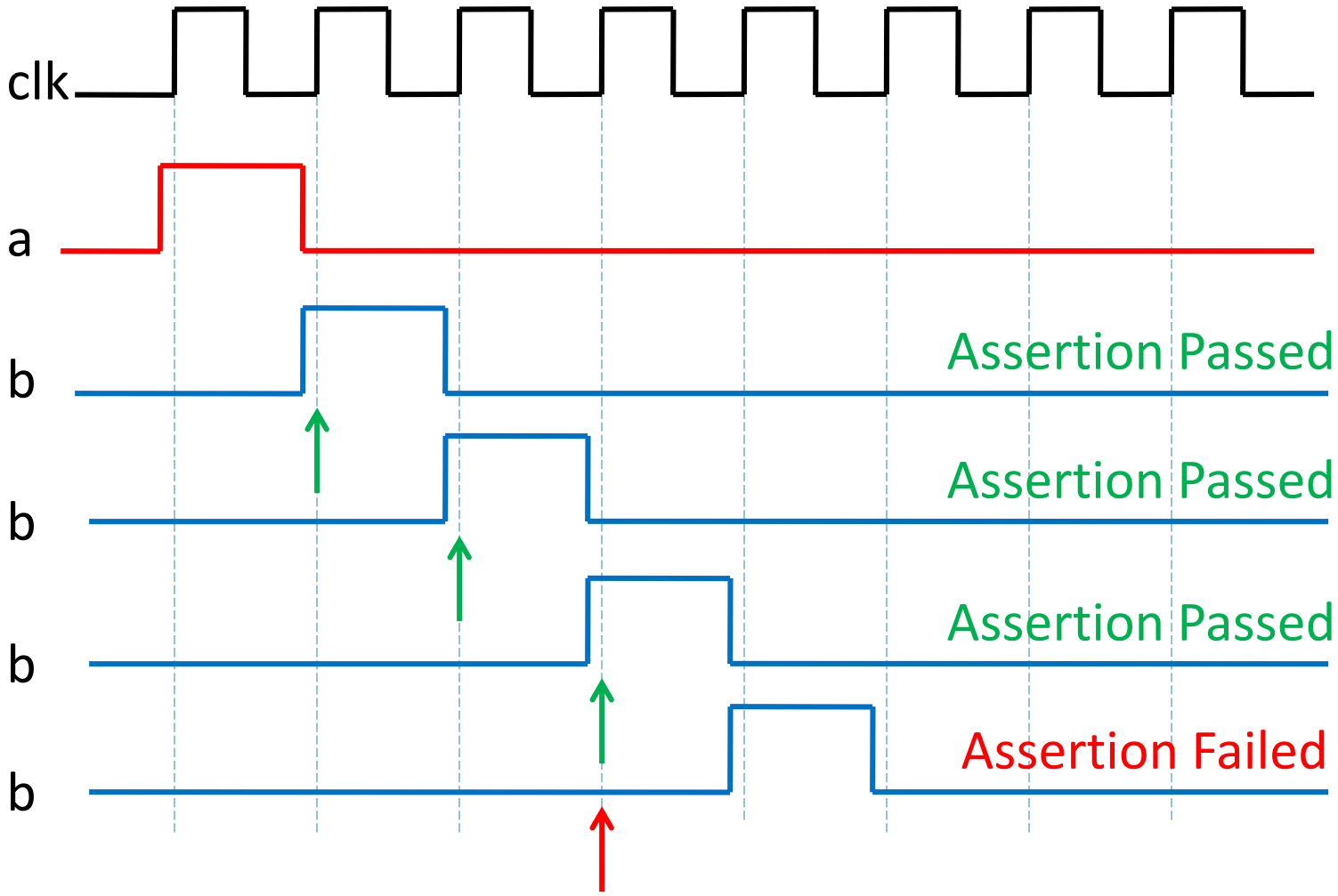
Sequence Operators

- ## n represents n clock cycle delay (or number of sampling edges).
- ## 0 means same clock cycle (overlapping signals).
- ## [min : max] specifies a range of clock cycles
  - min and max must be >=0.

```
sequence s1;
@ (posedge clk) a ## [1:3] b;
endsequence
```

Equivalent to: (a ##1 b) || (a ##2 b) || (a ##3 b)

Example


clk
a
b
Assertion Passed
b
Assertion Passed
b
Assertion Passed
b
Assertion Failed

Sequence Operators

- Sequence of events can be repeated for a range of count using [*m : n].
  - n should be more than 0 and cannot be $

```
sequence s4;
@ (posedge clk) a ##1 b [*2:5];
endsequence
```

Equivalent to :

```
(a ##1 b ##1 b ) ||
(a ##1 b ##1 b ##1 b) ||
(a ##1 b ##1 b ##1 b ##1 b) ||
(a ##1 b ##1 b ##1 b ##1 b ##1 b) ||
```

b must be true for minimum 2 and maximum 5 consecutive clock cycles after a is asserted

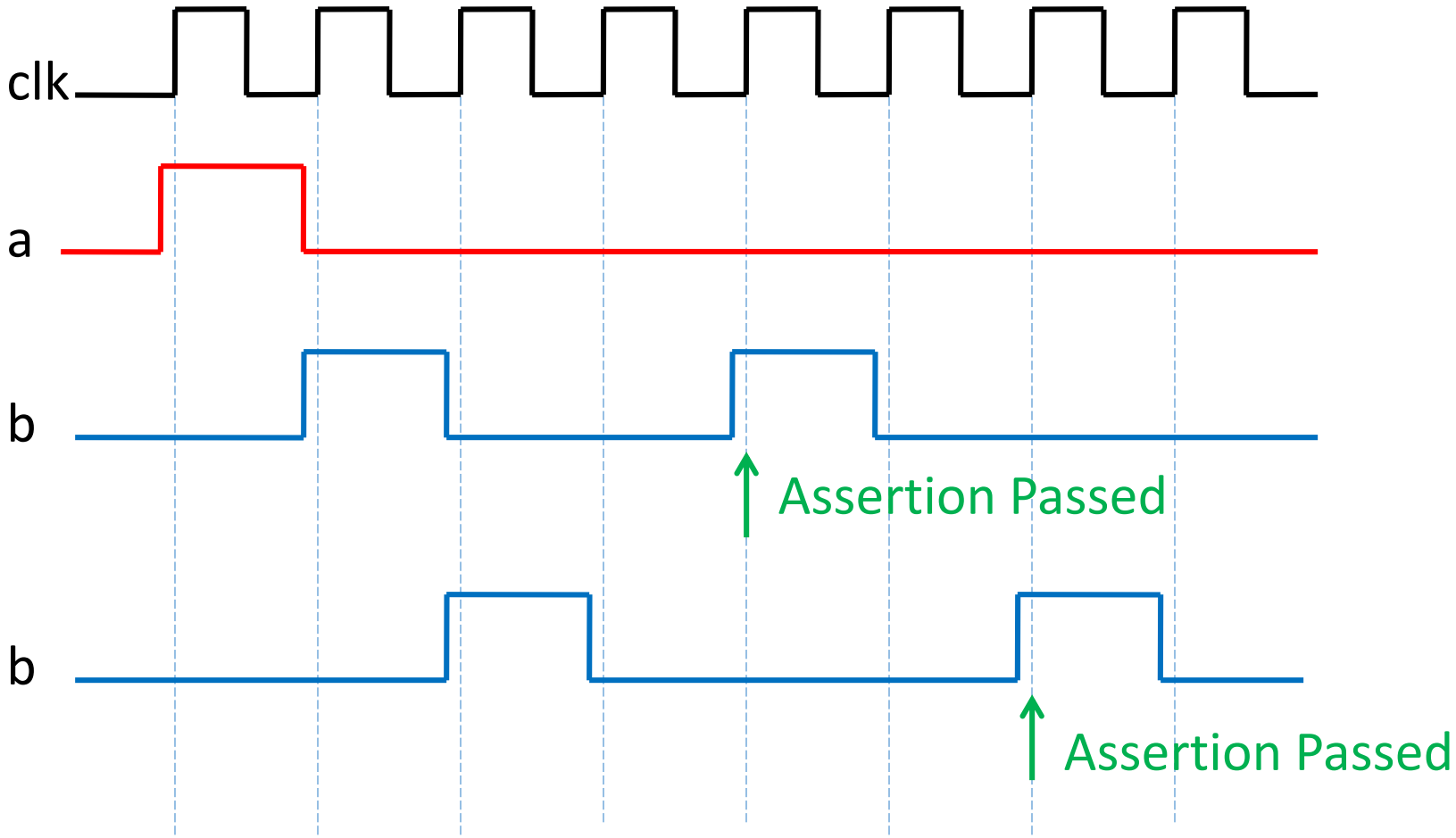
Sequence Operators

- [=m] operator can be used if an event repetition of m non-consecutive cycles are to be detected.
  - m should be more than 0 and cannot be $

```
sequence s5;
@ (posedge clk) a ##1 b [=2];
endsequence
```

b must be true for 2 clock cycles, that may not be consecutive.

Example



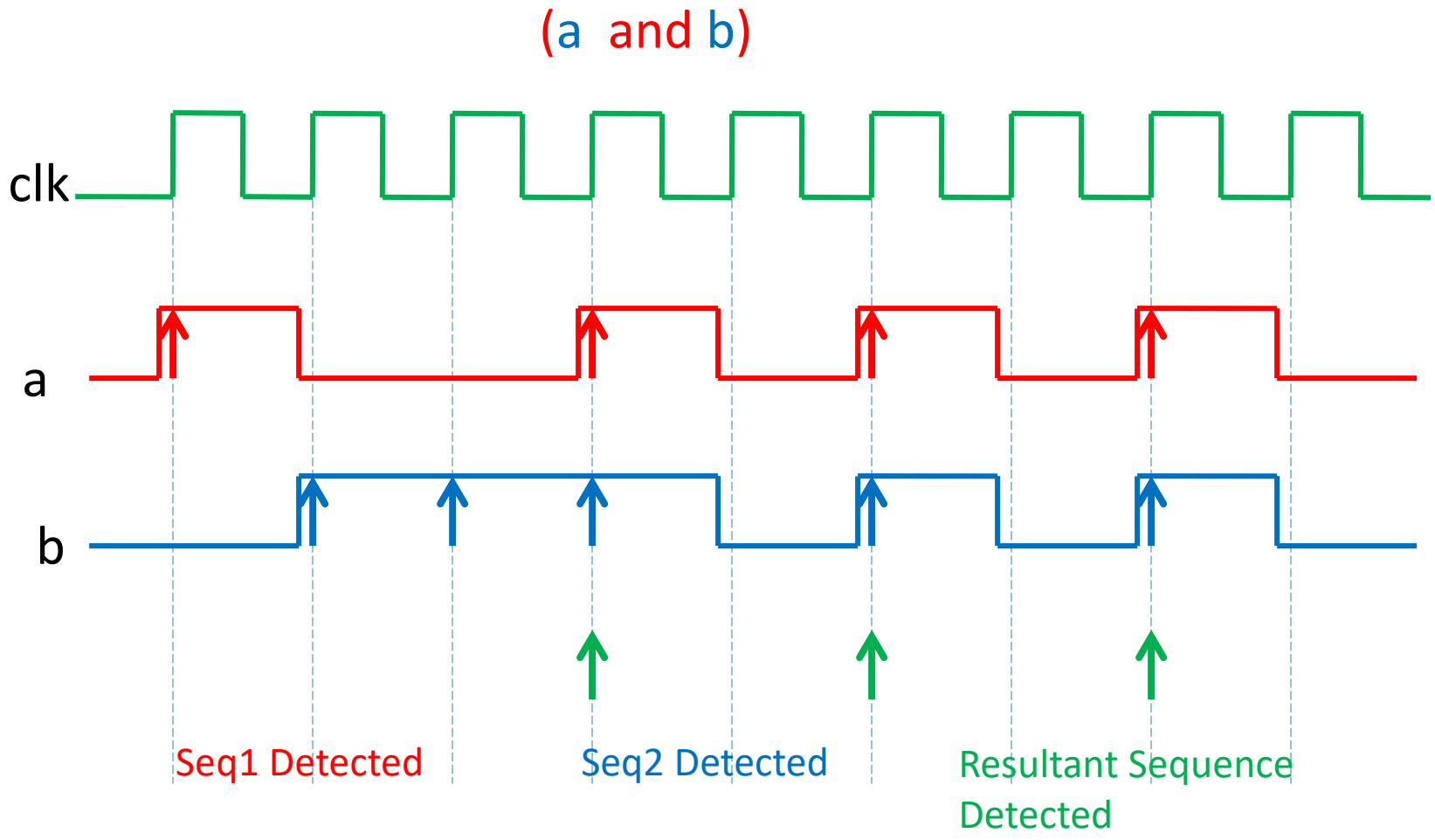
AND Operator

- Use and operator if two sequences needs to match.

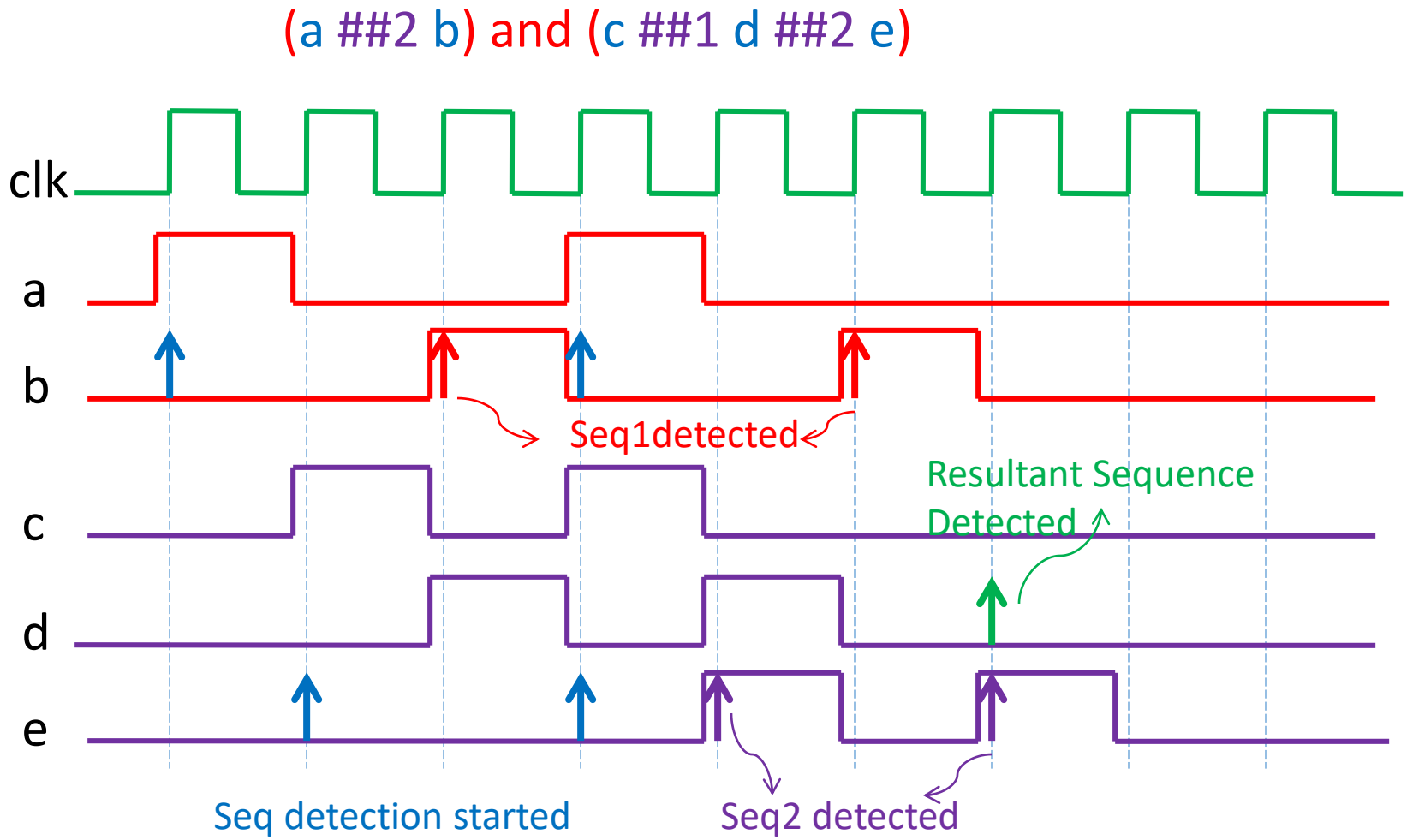
seq1 and seq2

- Following should be true for resultant sequence to matches:
  - seq1 and seq2 should start from same point.
  - Resultant sequence matches when both seq1 and seq2 matches.
  - The end point of seq1 and seq2 can be different.
  - The end time of resulting sequence will be end time of last sequence.

Example


(a and b)
clk
a
b
Seq1 Detected
Seq2 Detected
Resultant Sequence Detected

Example


(a ##2 b) and (c ##1 d ##2 e)
clk
a
b
Seq1detected
Resultant Sequence
Detected
c
d
e
Seq detection started
Seq2 detected

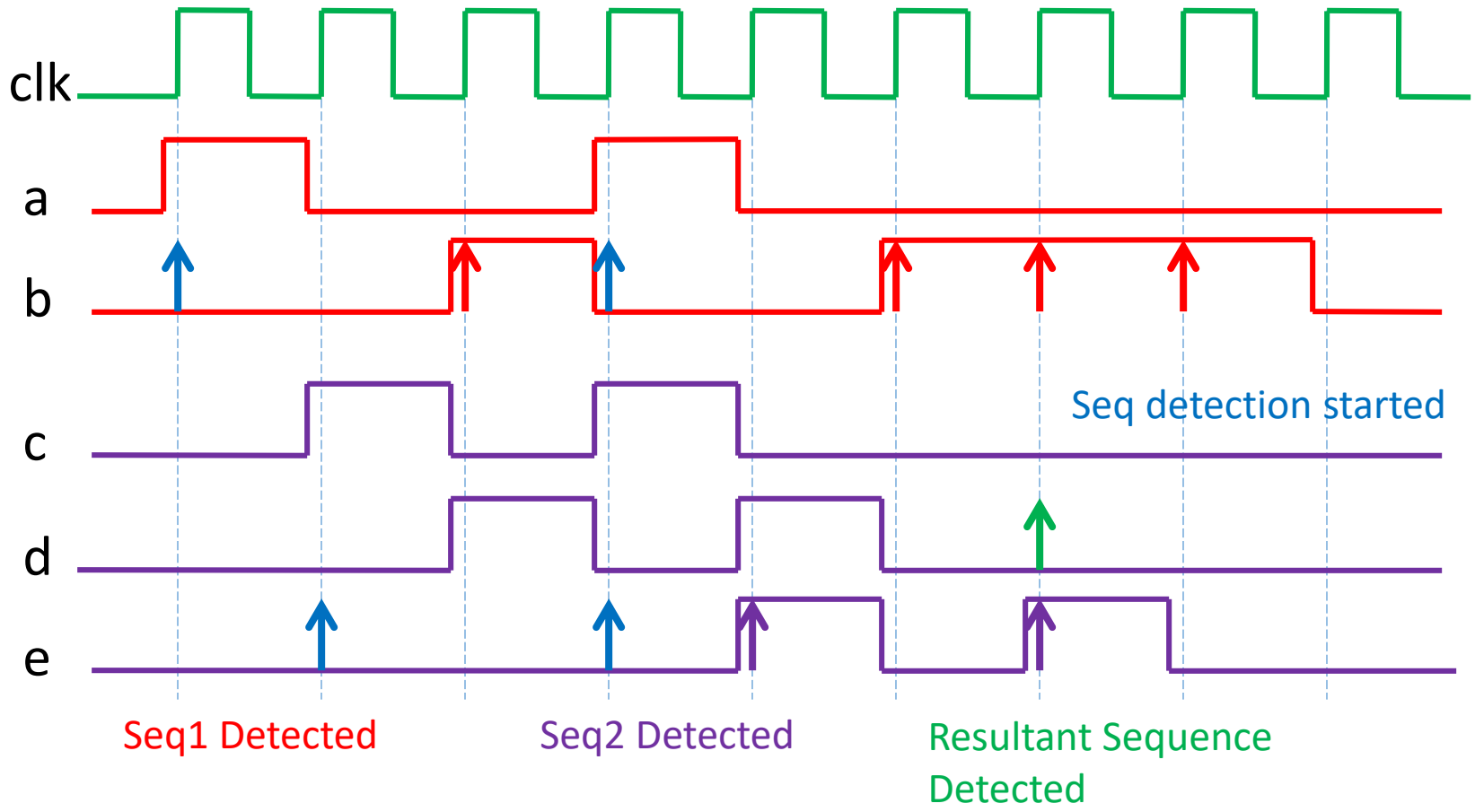
Intersect Operator

seq1 intersect seq2

- Following should be true for resultant sequence to matches:
  - seq1 and seq2 should start from same point.
  - Resultant sequence matches when both seq1 and seq2 matches.
  - The end point of seq1 and seq2 should be same.

Example

(a ##[2:4] b) intersect (c ##1 d ##2 e)



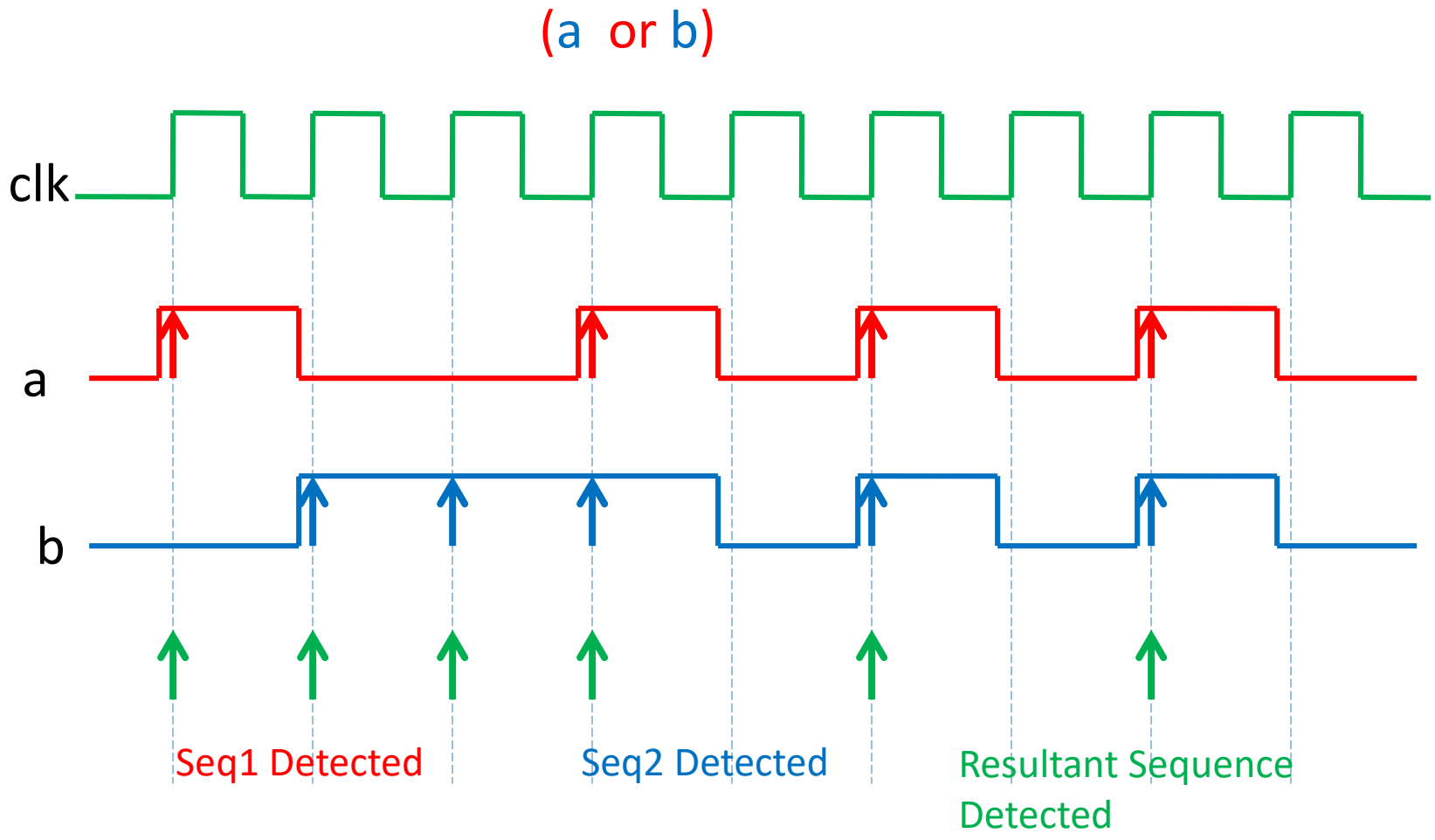
OR Operator

- Use or operator when at least one of the sequences needs to match.

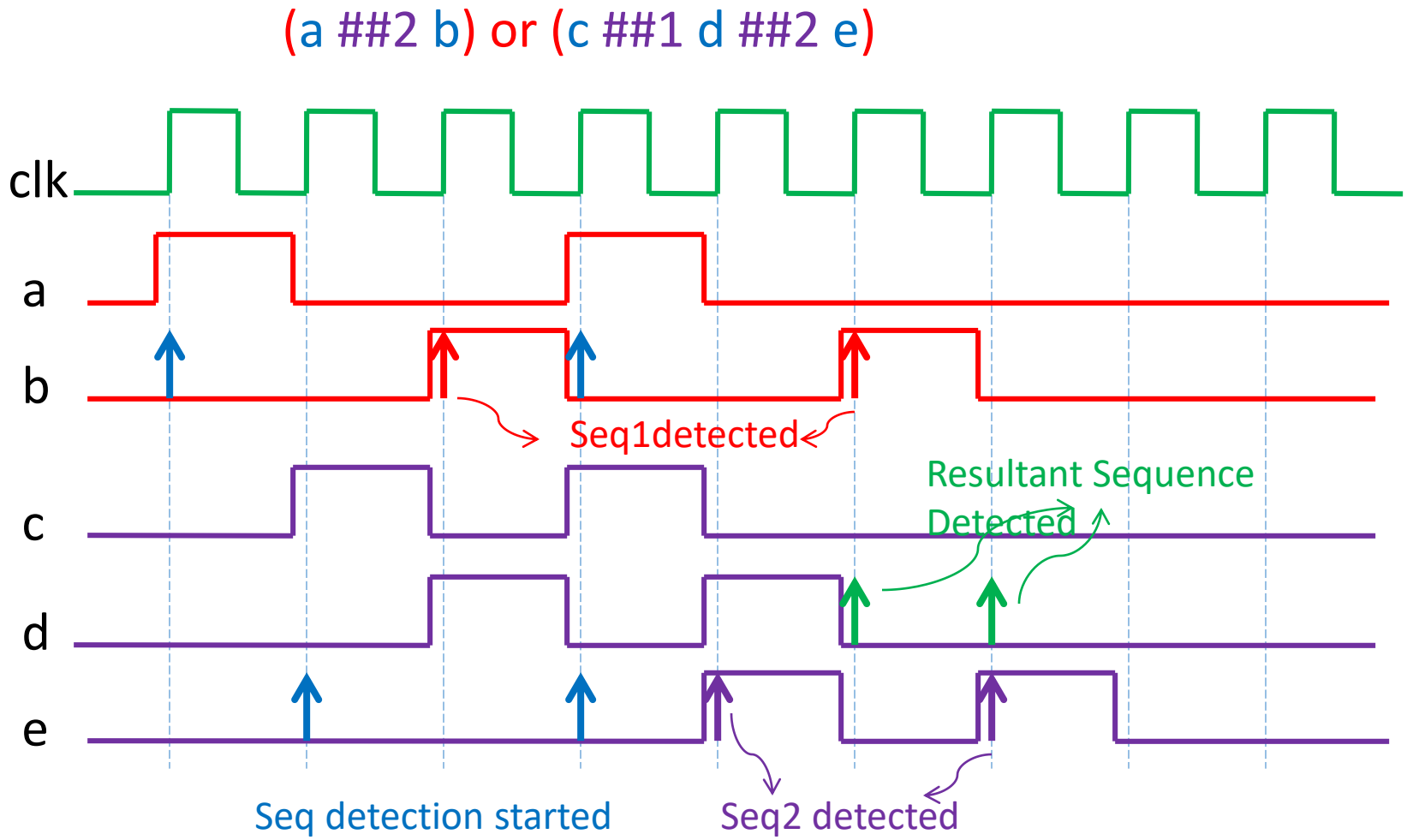
seq1 or seq2

- Following should be true for resultant sequence to matches:
  - seq1 and seq2 should start from same point.
  - Resultant sequence matches when either seq1 or seq2 matches.
  - The end point of seq1 and seq2 can be different.
  - The end time of resulting sequence will be end time of last sequence.

Example


(a or b)
clk
a
b
Seq1 Detected
Seq2 Detected
Resultant Sequence Detected

Example


(a ##2 b) or (c ##1 d ##2 e)
clk
a
b
Seq1detected
Resultant Sequence
Detected
c
d
e
Seq detection started
Seq2 detected

First_match Operator

- first_match operator matches only first of possible multiple matches for evaluation of sequence.

a ##[2:5] b

first_match(a ##[2:5] b)

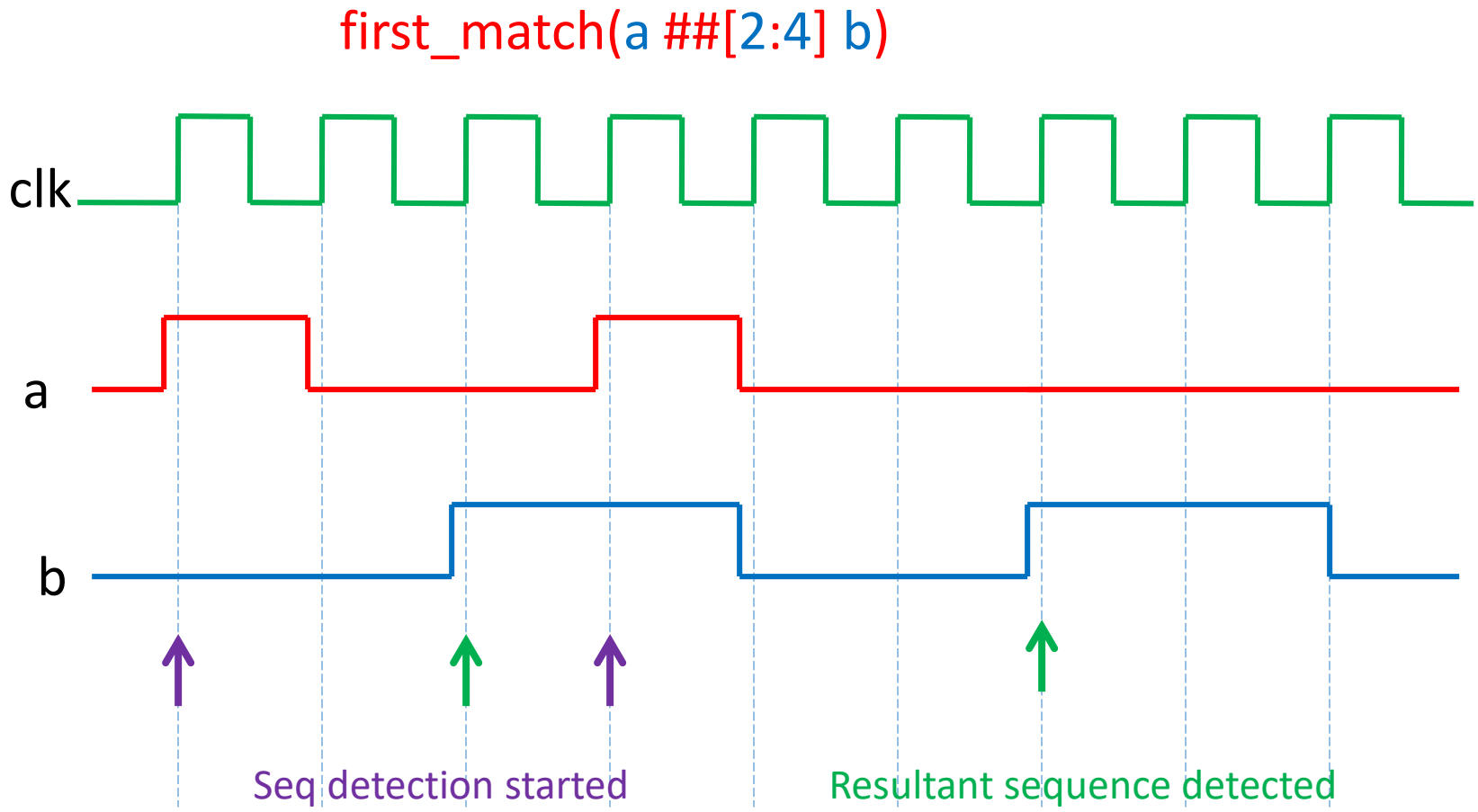
Equivalent to:

(a ##2 b) ||
(a ##3 b) ||
(a ##4 b) ||
(a ##5 b)

Sequence will match only one of the following options, whichever occurs first

(a ##2 b)
(a ##3 b)
(a ##4 b)
(a ##5 b)

Example


first_match(a ##[2:4] b)
clk
a
b
Seq detection started
Resultant sequence detected

not Operator

- not operator is used to check that a particular sequence should not occur.

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

    property nomatch;
    @(posedge clk) start |-> not (abc);
    endproperty
```

not Operator

- Example c ##1 d should not occur after a ##1 b.

```
property incorrect;
@(posedge clk) not (a ##1 b |=> c ##1 d);
endproperty
```

- Will report even if a ##1 b does not occur because of vacuous pass.

```
property correct;
@(posedge clk) not(a ##1 b ##1 c ##1 d);
endproperty
```

Sample Value Functions

- Special System Functions are available for accessing sampled values of an expression.
  - Functions to access current sampled value.
  - Functions to access sampled value in the past.
  - Functions to detect changes in sample values.

- Can also be used in procedural code in addition to assertions.

$rose, $fell

- $rose(expression [, clocking event])
  - Returns true if least significant bit changes to 1 with respect to value (0, X, Z) at previous clock else false.
- $fell(expression [, clocking event])
  - Returns true if least significant bit changes to 0 with respect to value (1, X, Z) at previous clock else false.
- Clocking event is optional usually derived from clocking event of assertion block or procedural block.

$rose vs @(posedge)

- @(posedge signal) returns 1 when signal changes from (0, X, Z) to 1 or (0 to X) or (0 to Z).
- $rose(signal) is evaluated to true when signal changes from (0, X, Z) to 1

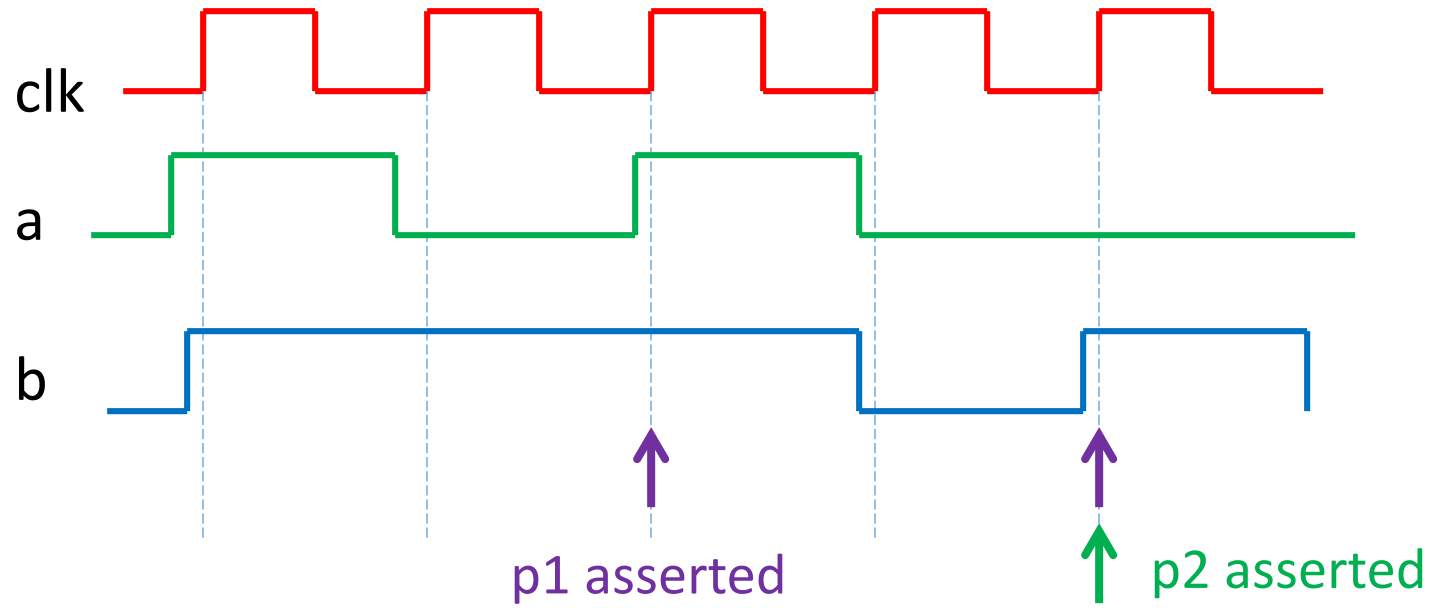
```
property p1;
@(posedge clk) a ##2 b;
endproperty
```

```
property p2;
@(posedge clk) a ##2 $rose(b);
endproperty
```

$rose

```
property p1;
@(posedge clk) a ##2 b;
endproperty
```

```
property p2;
@(posedge clk) a ##2 $rose(b);
endproperty
```



$stable, $past

- $stable(expression [, clocking event])
  - Returns true if value of expression did not change from its sampled value in previous clock else false.

- $past(expression [, no of cycles] [, gating expression] [,clocking event])
  - Used to access sampled value of an expression any number of clock cycles in past.
  - no of cycles defaults to 1.
  - gating expression for clocking event.
  - clocking event inferred from assertion or procedural block if not specified.

System Functions and Tasks

- Following System Functions and Tasks are available that can be used in assertions and procedural blocks:
  - $onehot (expression) Returns true if only one bit of the expression is high.
  - $onehot0 (expression) Returns true if at most one bit of the expression is high.
  - $isunknown (expression) Returns true if any bit of the expression is X or Z.
  - $countones (expression) Returns number of one's in the expression.

$asserton, $assertoff, $assertkill

- disable iff can be locally disable assertions.
- $asserton, $assertoff and $assertkill are used to control assertions of a module or list of instance.
- $asserton, resume execution of assertions, enabled by default.
- $assertoff, temporarily turns off execution of assertions.
- $assertkill, kills all currently executing assertions.

$asserton(level [, list of modules or instances])

- Design clock based calculator for +,-,*,/ for
- 3 bit inputs dt1 and dt2
- One bit sel input
- One bit enable input
- 8 bit output
- Write the assertion with following design rule:

When en is high then sel should high after 2 clk cycles dt1 and dt2 should high.

Dt1 and dt should be high for same clk cycle

When en is low sel should be low

- In provided verification environment of memory model try to implement following assertions
- Rd_en and wr_en should not get high at same time
- Wr_en is high then after one clk cycle wr_data should not be 0