Verilog Scheduling and Event Queue

Consider

```
always_ff @( posedge clk ) c = reset ? 0 : c + 1;
always_ff @( posedge clk ) over_th = c + 1'd1 > threshold;
```

Is `over_th` computed using the new or old `c`?

(Answer: either one, and so code is unreliable.)

Terminology

Event:
Sort of a to-do item for simulator. May include running a bit of Verilog code or updating an object's value.

Event Queue:
Sort of a to-do list for simulator. It is divided into time slots and time slot regions.

Time Slot:
A section of the event queue in which all events have the same time stamp.

Time Slot Region:
A subdivision of a time slot. There are many of these. Important ones: active, inactive, NBA.

Scheduling:
Determining when an event should execute. The when consists of a time slot and a time slot region.

Update Events:
The changing of an object's value. Will cause *sensitive* objects to be scheduled.

Time Slot Regions

Rationale:

"Do it now!" is too vague. Need to prioritize.

SystemVerilog divides a time slot into 17 *regions*.

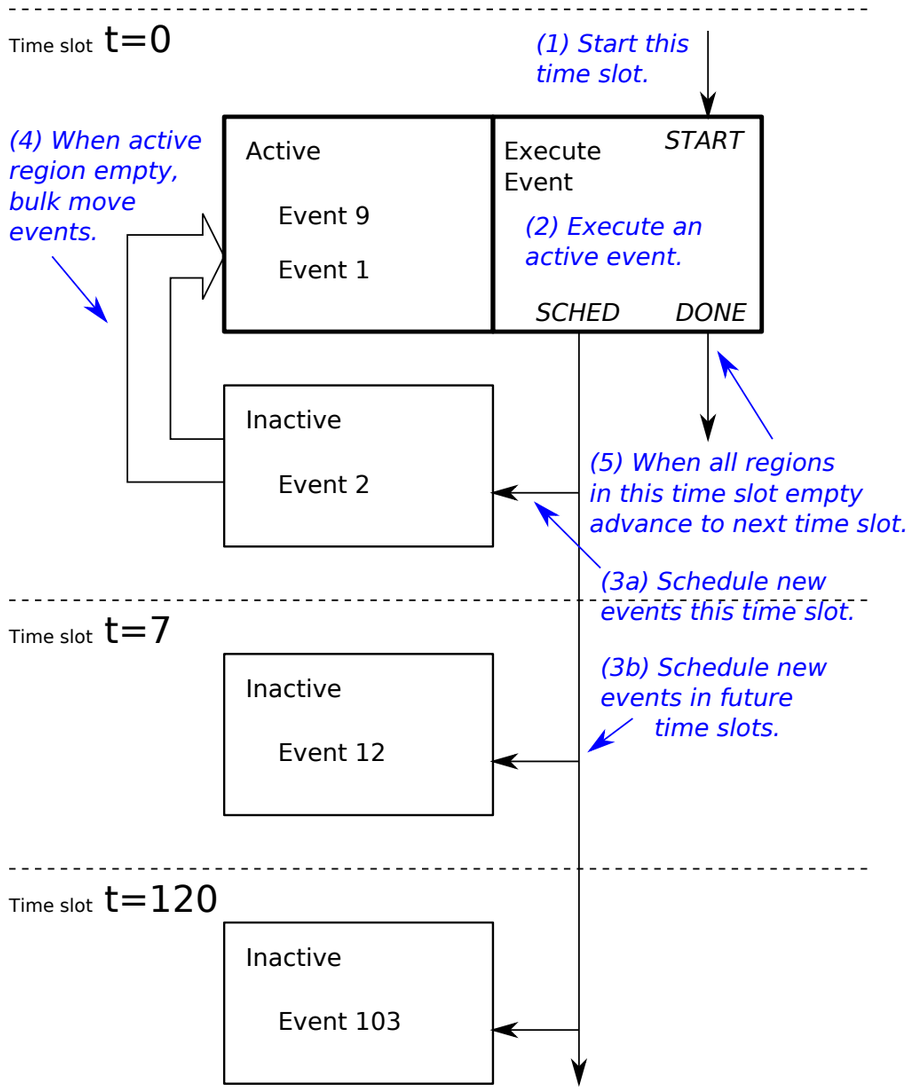
Some Regions

Active Region:

Events that the simulator is currently working on. Only the current time slot has this region.

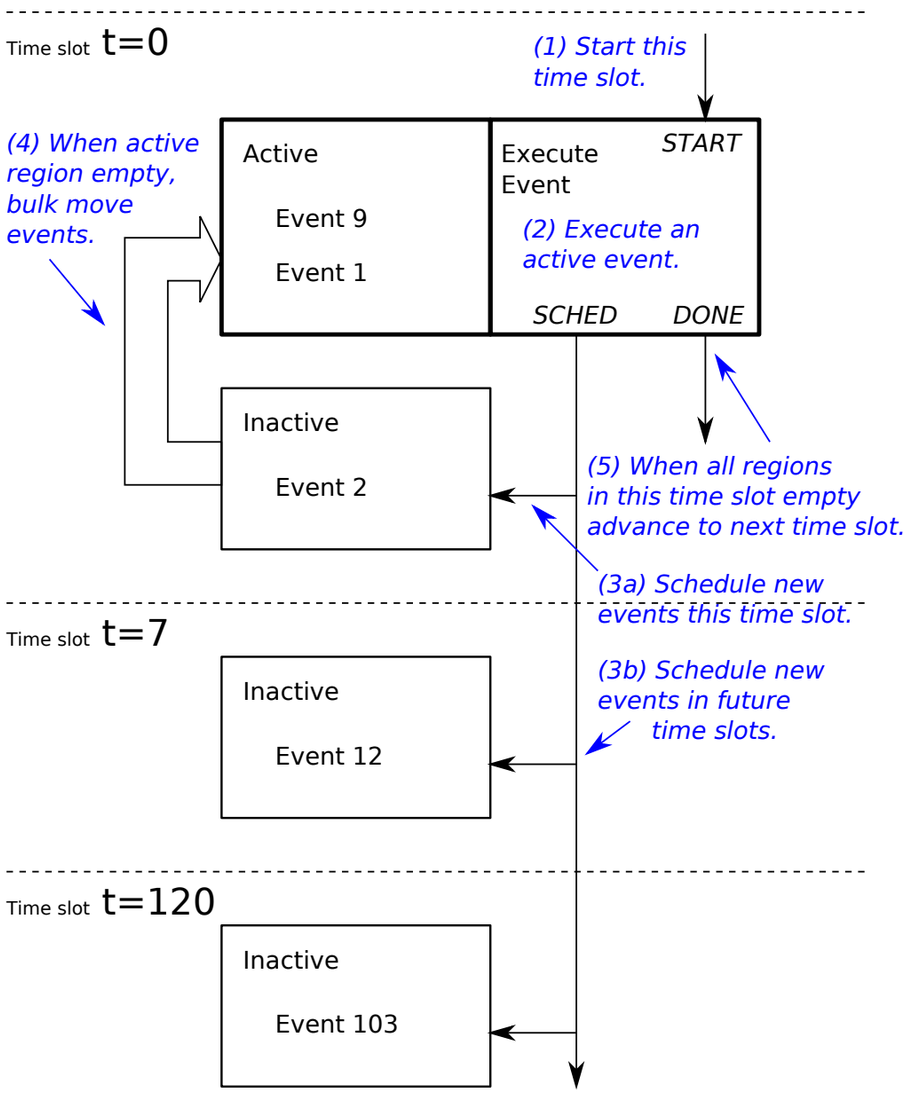
Inactive Region:

Contains normally scheduled events. Current and future time slots have this region.



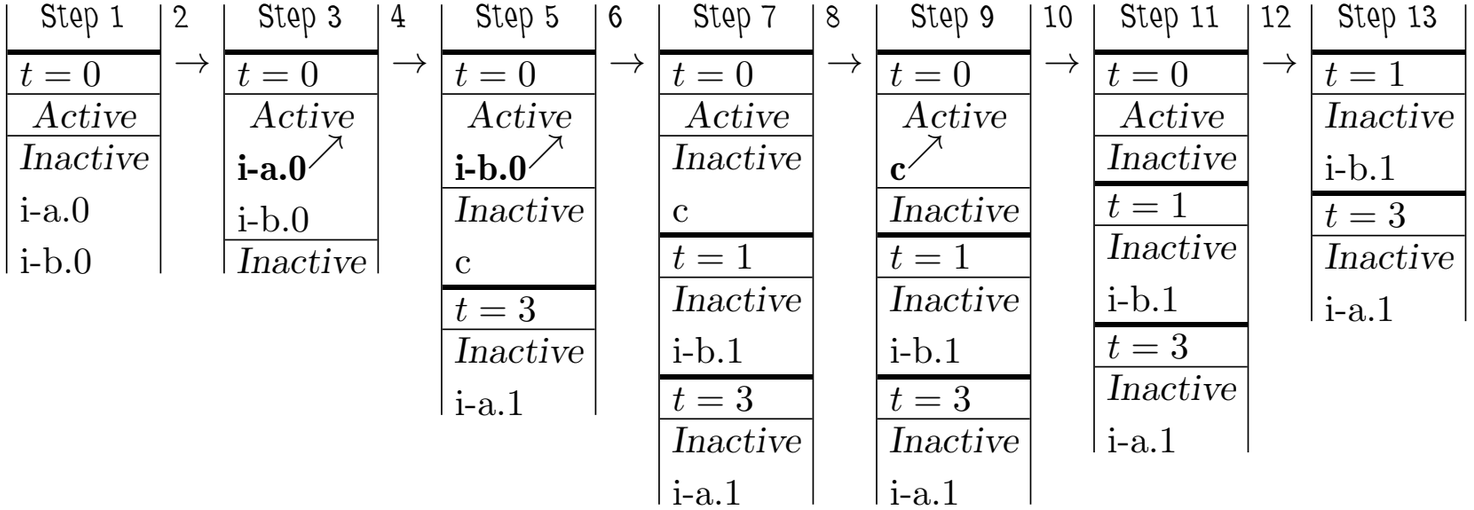
EE 4755 Lecture Transparency. Formatted 17:46, 11 December 2019 from lsli-event-q-TeXize.

eq-3


Time slot t=0
(1) Start this time slot.
(4) When active region empty, bulk move events.
Active
Event 9
Event 1
Execute Event
START
(2) Execute an active event.
SCHED
DONE
Inactive
Event 2
(5) When all regions in this time slot empty advance to next time slot.
(3a) Schedule new events this time slot.
Time slot t=7
Inactive
Event 12
(3b) Schedule new events in future time slots.
Time slot t=120
Inactive
Event 103

eq-4

Event Queue Example



1: Verilog puts all `initial` blocks in $t = 0$'s inactive region.

2: Active region is empty, and so inactive copied to active.

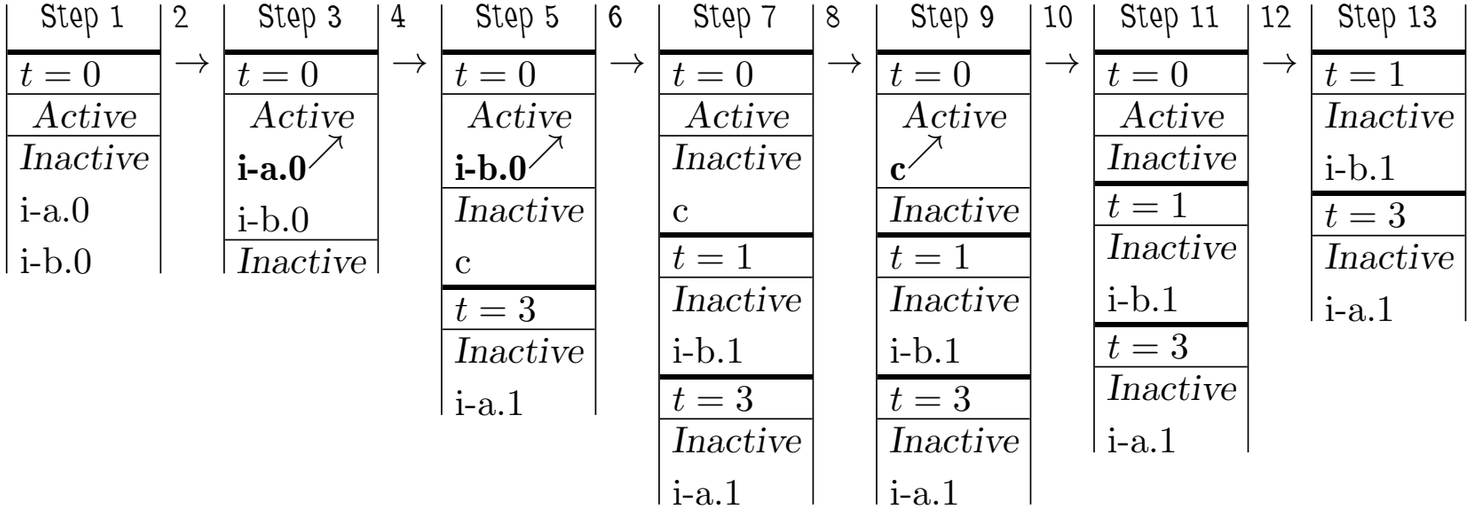
3: Event `i-a.0` executes and schedules event `c` for $t = 0$ ...
... and `i-a.1` for $t = 3$.

4: Event `i-a.0` removed from active region (it is now not scheduled anywhere).

```
initial begin
   // i-a.0
   a = 1;
   #3;
   // i-a.1
   a = 2;
end

initial begin
   // i-b.0
   b = 10;
   #1;
   // i-b.1
   b = a;
end

assign c = a + b;  // c
```


EE 4755 Lecture Transparency. Formatted 17:46, 11 December 2019 from lsli-event-q-TeXize.
eq-5

Event Queue Example



5,6: Event `i-b.0` executes and schedules `i-b.1` for $t = 1$.

7,8: Since active region is empty, inactive region is bulk-copied to active region.

9: Event `c` executes.

10-12: Since all regions in time slot 0 are empty, move to next time slot, $t = 1$.

```
initial begin
   // i-a.0
   a = 1;
   #3;
   // i-a.1
   a = 2;
end

initial begin
   // i-b.0
   b = 10;
   #1;
   // i-b.1
   b = a;
end

assign c = a + b;  // c
```

Some More Regions

NBA Region:

Update events from non-blocking assignments.

Postponed Region:

Events scheduled using `$watch` system task.

Event Types

Evaluation Event

Indicates that a piece of code is to be executed or resumed.

Sometimes just referred to as events, or resume events.

All events from the previous event queue example were evaluation events.

Update Event

Indicates that the value of an object is to be changed.

Update events are created by executing non-blocking assignments.

Event Scheduling

Event Scheduling:
The placing of an event in the event queue.

Types of Scheduling

Initially Scheduled Events
Events scheduled when simulation starts, such as for `initial` blocks.

Time-Delay Scheduled Events
Events scheduled when execution reaches a time delay.

Sensitivity-List Scheduled Events
Events scheduled when certain object values change.

Non-Blocking Assignment (NBA) Scheduled Update Events
Update events scheduled when a non-blocking assignment is reached.

Event Scheduling

Time-Delay Scheduled

When a delay control, *e.g.* `#12`, is encountered...
... schedule (put) a *resume* event in inactive region ...
... of future (t+delay) time step.

Time-Delay Scheduled Example:

```
b++;
// Label L1
#4;            // Schedule resume event for L2 at time t+4.
// Label L2;
a = b;
```

Sensitivity List Scheduled

When an object in a sensitivity list changes . . .
. . . schedule a resume or check-and-resume event for code associated with sensitivity list . . .
. . . in the inactive region of current time step.

Explicit Event Examples:

```
// Label: L0
@( x );                 // Put a check-@-condition event in sensitivity list of x ..
                        // .. event will resume at L1 if condition satisfied ..
                        // .. meaning any change in x.
// Label: L1
@( posedge clk );       // Put a check-@-condition event in sensitivity list of clk ..
                        // .. event will resume at L2 if condition satisfied ..
                        // .. meaning 0->1 transition.
// Label: L2
wait( stop_raining );   // Put a check-wait-condition event in sensitivity list of stop_raining ..
                        // .. event wil resume at L3 if stop_raining != 0.
// Label: L3
```

Live-In of `always_comb` or `always @*`:

```
always_comb x = a + b;  // Put an execute event in sensitivity list of a and b.

always_comb begin       // Put an execute event in sensitivity list of ..
  y = d + e;            // d, e, and f, BUT NOT y. (y is not live in).
  z = y + f;
end
```

Continuous assignment:

```
  assign x = a + b;     // Put an execute event in sensitivity list of a and b.
```

Primitive ports:

```
and myAndGate(x,e,f);  // Put an execute event in sensitivity list of e and f.
```

NBA Scheduled Update Events

When a non-blocking assignment is executed . . .
. . . the value of the right-hand-side is computed . . .
. . . and an update event is scheduled in the NBA region of the current time step . . .
. . . when the update event executes the left-hand-side variable is updated with the value.

```
always_comb begin
  y <= a + b;  // Schedule an update-y event in NBA region, keep executing.
  e = y + g;   // Uses old y.
end
```

Permanently Scheduled

When a `$watch(OBJ)` system task is executed . . .
. . . a watch event is scheduled in the postponed region of every time step . . .
. . . when the watch event executes the value of `OBJ` is printed on the console.

Example: Non-blocking assignments

Show the state of the event queue for the module below ...
... starting at $t = 10$ and given the external events described below.

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```

Example's Sensitivity Lists and Update Events

Sensitivity List For Example Code

`clk`: Due to `@(posedge clk)`. If $0 \rightarrow 1$ schedule alf.

`a`: Due to `always_comb`. Any change, schedule alc.

`f`: Due to `always_comb`. Any change, schedule alc.

Update Events For Example Code

Execution of `a <= a + c` ...
... will result in scheduling an update event (for `a`).

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```

Conditions and External Events

Example Problem Assumptions:

Queue is initially empty at $t = 10$.

At $t = 10$ `j` changes.

At $t = 12$ `clk` changes from 0 to 1.

At $t = 14$ `f` changes.

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```

Event Queue Changes

Step 1: Queue is empty.

Step 2: At $t = 10$ `j` changes.

Step 3: No change, because `j` is not in a sensitivity list.

Step 1	2	Step 3
$t = 10$	→	$t = 10$
Active		*Active*
Inactive		*Inactive*
NBA		*NBA*

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```

Event Queue Changes

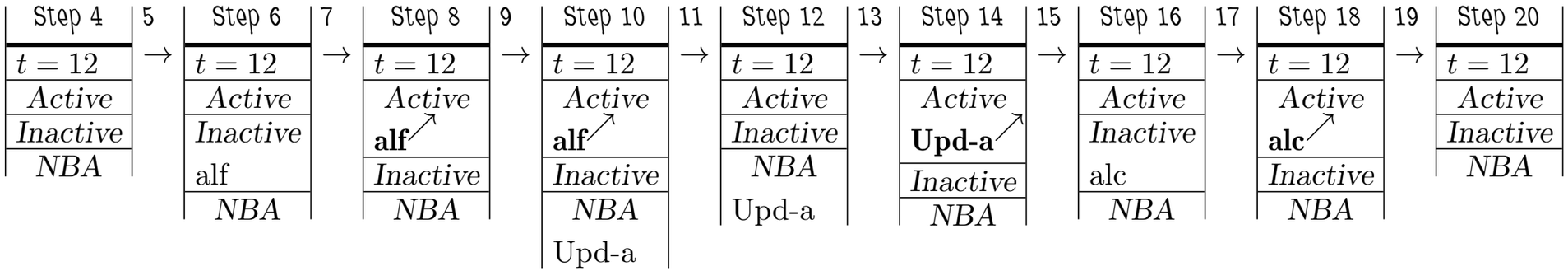
Step 5: At $t = 12$ `clk` changes from 0 to 1 ...
... scheduling alf in inactive region in Step 6.

Step 7: Since active region empty ...
... inactive copied to active.

Step 8: alf starts execution.

Step 10: Execution of `a<=a+b` ...
... results in scheduling Upd-a in NBA region.

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```



Event Queue Changes

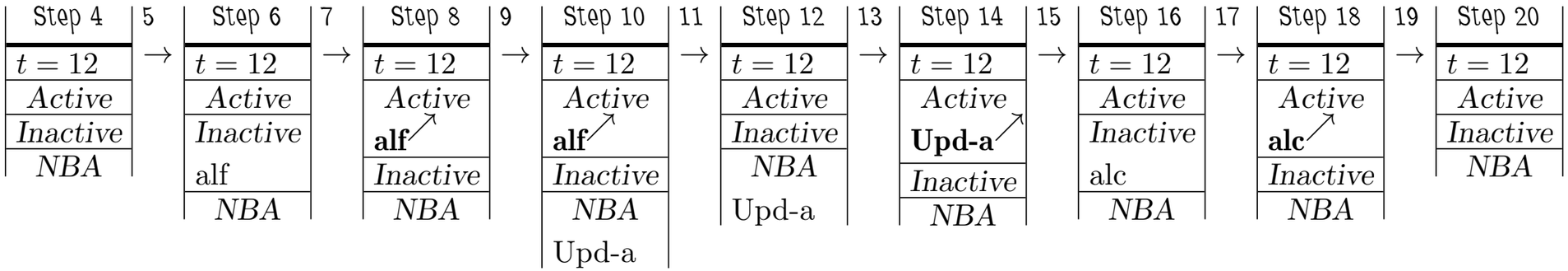
Step 11: alf finishes leaving active region empty.

Step 13: Next non-empty region, NBA, copied to active region.

Step 14-16: Upd-a causes alc to be scheduled.

Step 17-20: alc moved to active region, runs, finishes.

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```



eq-20

Event Queue Changes

Step 22: `f` changes, scheduling alc.

Step 23-26: alc moved to active region, executes finishes.

Step 27: If nothing else happens simulation ends.

Step 21	22	Step 23	24	Step 25	26	Step 27
$t = 14$	→	$t = 14$	→	$t = 14$	→	$t = 14$
Active		*Active*		*Active*		*Active*
Inactive		*Inactive*		alc		*Inactive*
NBA		alc		*Inactive*		*NBA*
		NBA		*NBA*		

```
module misc #( int n = 8 )
   ( output logic [n-1:0] a, g, e,
     input uwire [n-1:0] b, c, j, f,    input uwire clk );

   logic [n-1:0] z;

   always_ff @( posedge clk ) begin   // Label: alf
      a <= b + c;
      z = a + j;
      g = z;
   end

   always_comb                        // Label: alc
      e = a * f;
endmodule
```

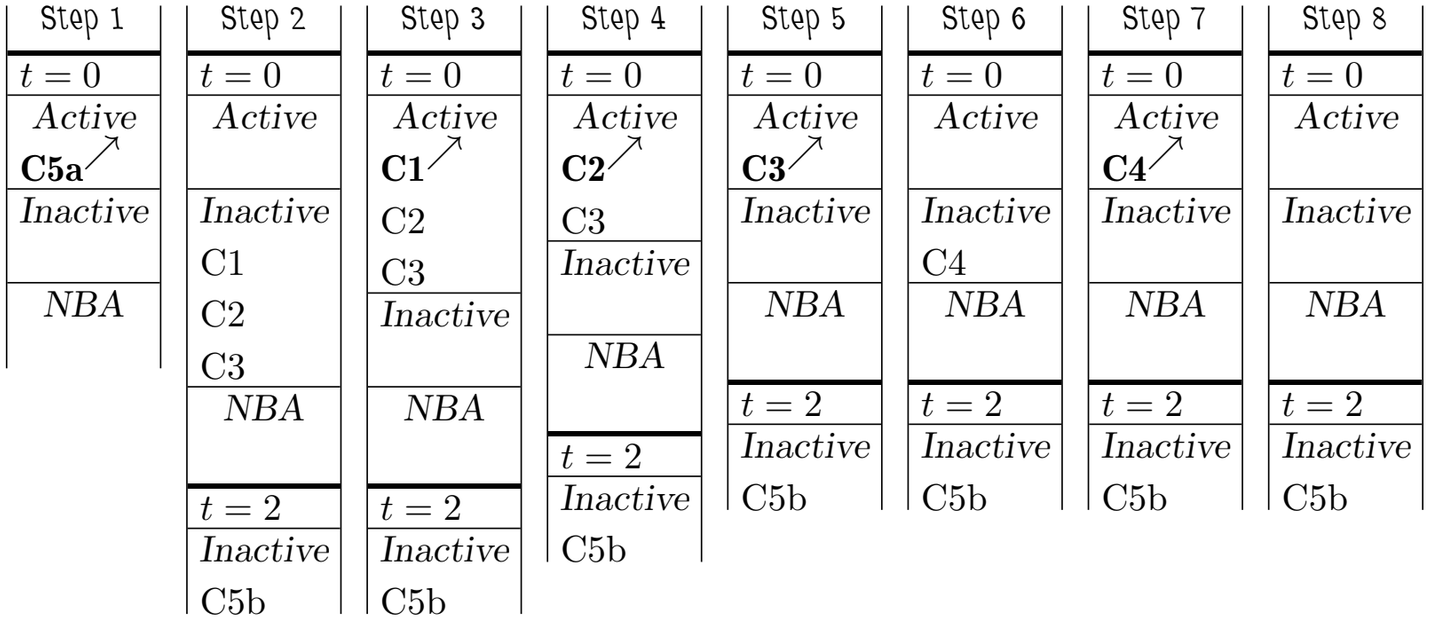
Example: Non-Blocking Assignments 2

Consider the code to the right.

```
module eq;
   logic [7:0] a, b, c, d, x, y;
   logic [7:0] x1, x2, y1, y2, z2;

   always_comb begin          // C1
      x1 = a + b;
      y1 = 2 * b;
   end

   assign x2 = 100 + a + b;  // C2
   assign y2 = 4 * b;        // C3
   assign z2 = y2 + 1;       // C4

   initial begin
      //                          C5a
      a = 0;
      b = 10;
      #2;
      //                          C5b
      a = 1;
      b <= 11;
      #2;
      //                          C5c
      a = 2;
      b = 12;
   end
endmodule
```


EE 4755 Lecture Transparency. Formatted 17:46, 11 December 2019 from lsli-event-q-TeXize.

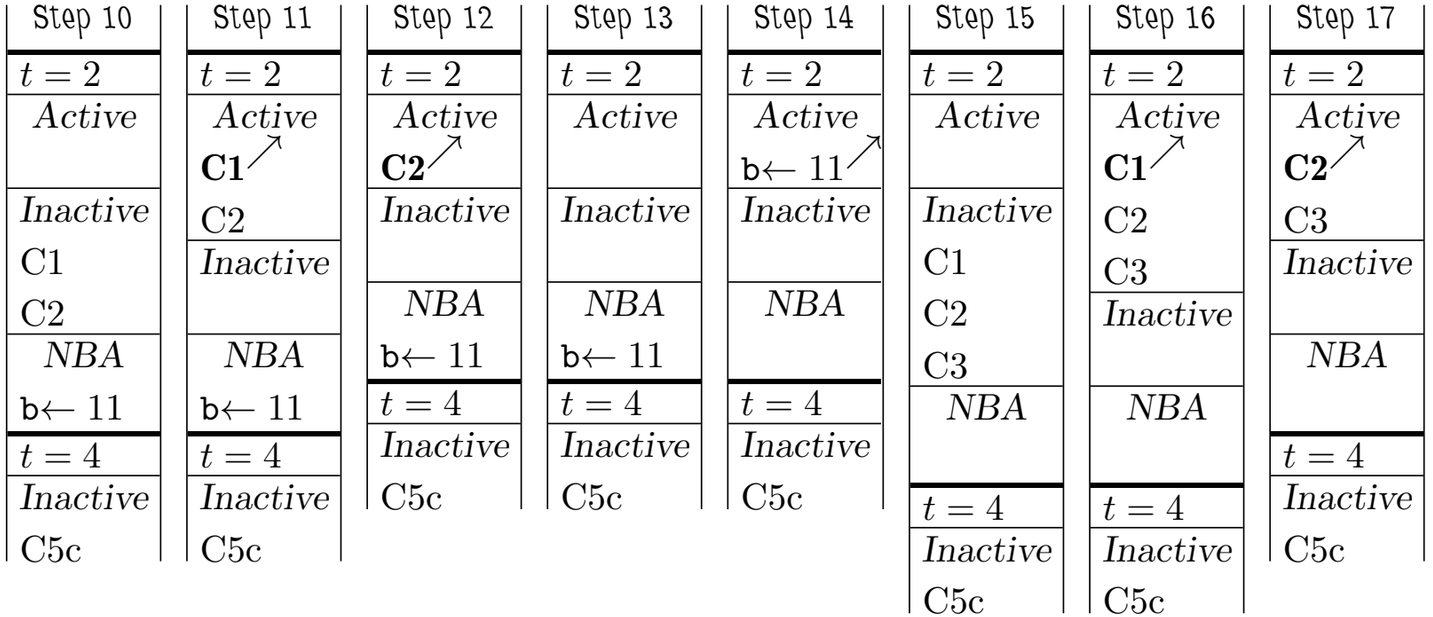
eq-22




```
module eq;
logic [7:0] a, b, c, d, x, y;
logic [7:0] x1, x2, y1, y2, z2;

always_comb begin          // C1
   x1 = a + b;
   y1 = 2 * b;
end

assign x2 = 100 + a + b;   // C2
assign y2 = 4 * b;         // C3
assign z2 = y2 + 1;        // C4

initial begin
   //                      C5a
   a = 0;
   b = 10;
   #2;
   //                      C5b
   a = 1;
   b <= 11;
   #2;
   //                      C5c
   a = 2;
   b = 12;
end
endmodule
```


EE 4755 Lecture Transparency. Formatted 17:46, 11 December 2019 from lsli-event-q-TeXize.

eq-23




```
module eq;
   logic [7:0] a, b, c, d, x, y;
   logic [7:0] x1, x2, y1, y2, z2;

   always_comb begin          // C1
      x1 = a + b;
      y1 = 2 * b;
   end

   assign x2 = 100 + a + b;   // C2
   assign y2 = 4 * b;         // C3
   assign z2 = y2 + 1;        // C4

   initial begin
      //                          C5a
      a = 0;
      b = 10;
      #2;
      //                          C5b
      a = 1;
      b <= 11;
      #2;
      //                          C5c
      a = 2;
      b = 12;
   end
endmodule
```

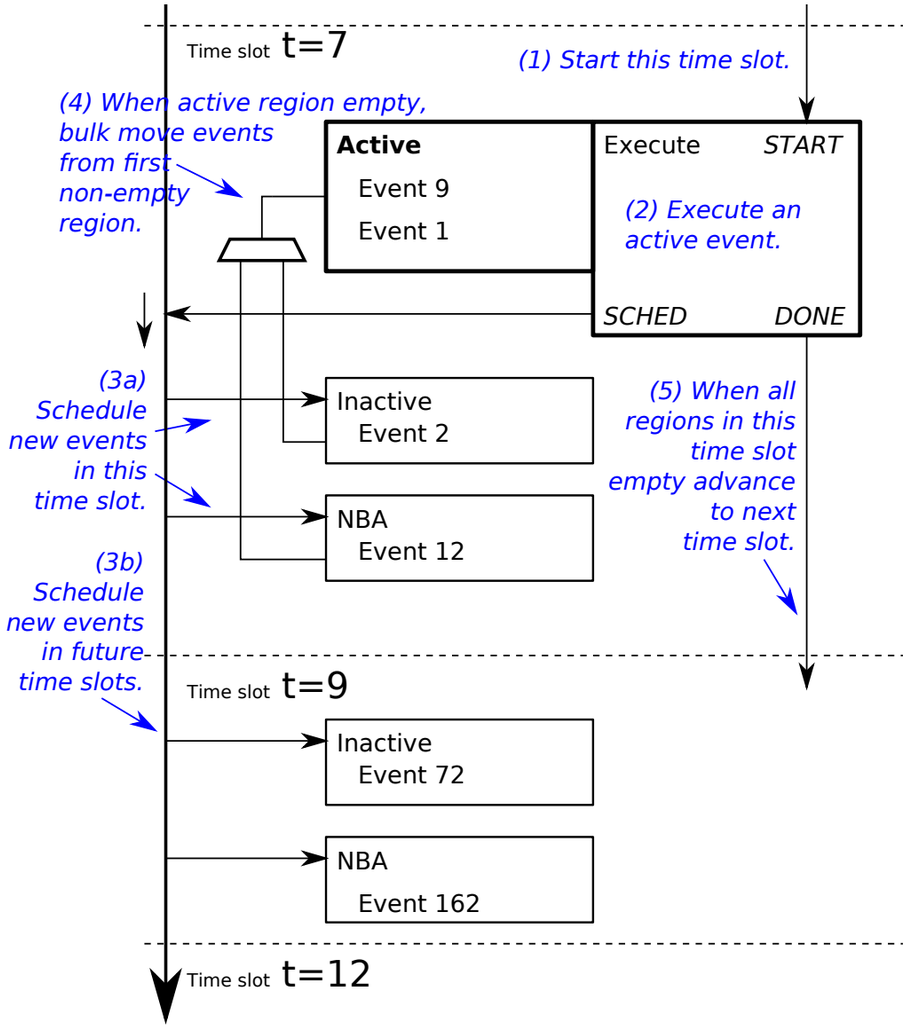

EE 4755 Lecture Transparency. Formatted 17:46, 11 December 2019 from lsli-event-q-TeXize.

eq-24

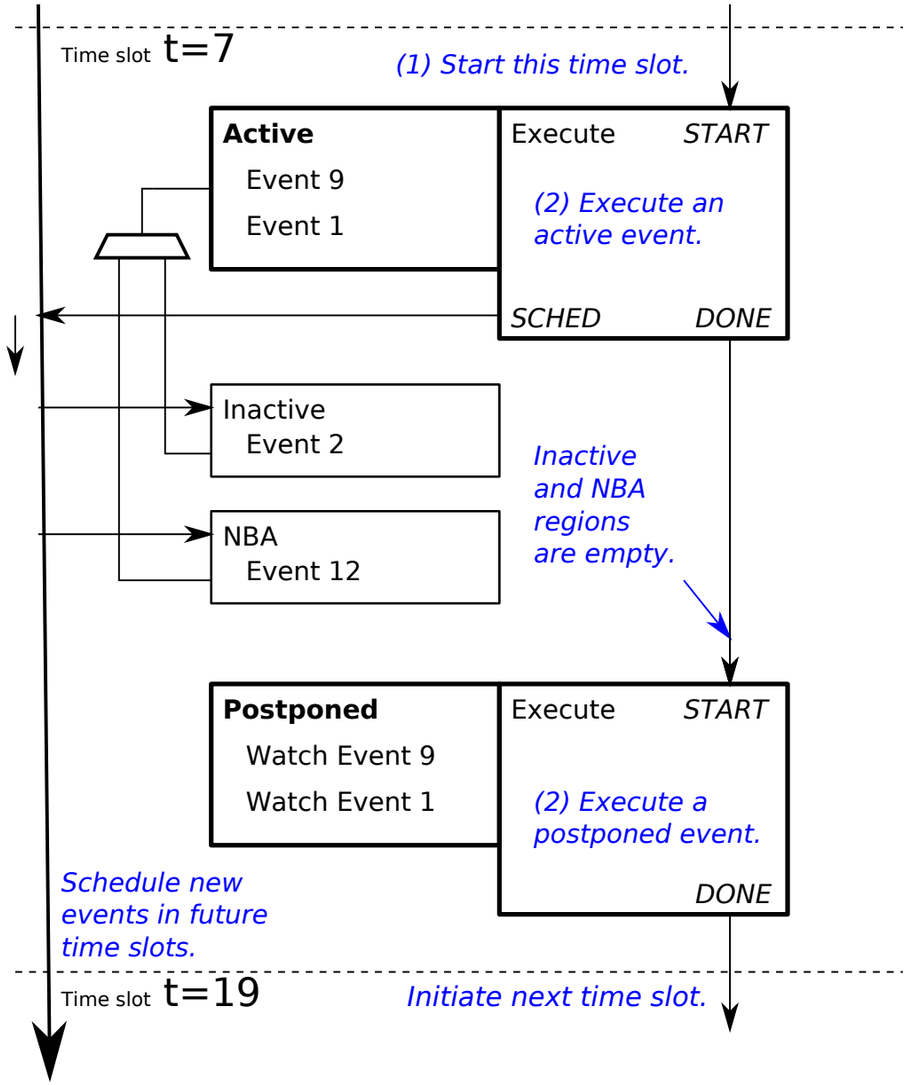
Step 18	Step 19	Step 20	Step 21	Step 22
$t = 2$	$t = 2$	$t = 2$	$t = 2$	$t = 4$
Active **C3** ↗	*Active*	*Active* **C4** ↗	*Active*	*Active* **C5c** ↗
Inactive	*Inactive* C4	*Inactive*	*Inactive*	*Inactive*
NBA	*NBA*	*NBA*	*NBA*	*NBA*
$t = 4$	$t = 4$	$t = 4$	$t = 4$	
Inactive C5c	*Inactive* C5c	*Inactive* C5c	*Inactive* C5c	

```
module eq;
   logic [7:0] a, b, c, d, x, y;
   logic [7:0] x1, x2, y1, y2, z2;

   always_comb begin          // C1
      x1 = a + b;
      y1 = 2 * b;
   end

   assign x2 = 100 + a + b;  // C2
   assign y2 = 4 * b;        // C3
   assign z2 = y2 + 1;       // C4

   initial begin
      //                          C5a
      a = 0;
      b = 10;
      #2;
      //                          C5b
      a = 1;
      b <= 11;
      #2;
      //                          C5c
      a = 2;
      b = 12;
   end
endmodule
```

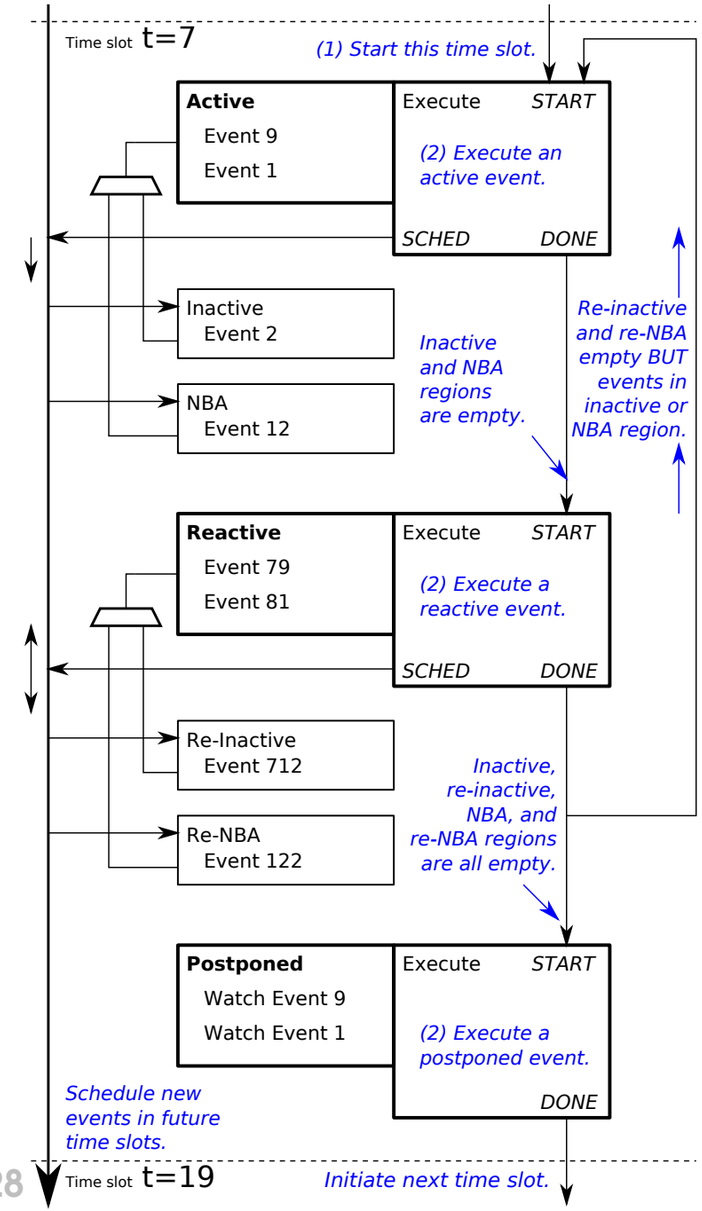

eq-25


Time slot t=7
(1) Start this time slot.
(4) When active region empty, bulk move events from first non-empty region.
Active
Event 9
Event 1
Execute START
(2) Execute an active event.
SCHED DONE
(3a) Schedule new events in this time slot.
Inactive
Event 2
NBA
Event 12
(5) When all regions in this time slot empty advance to next time slot.
(3b) Schedule new events in future time slots.
Time slot t=9
Inactive
Event 72
NBA
Event 162
Time slot t=12

eq-26


Time slot t=7
(1) Start this time slot.
Active
Event 9
Event 1
Execute START
(2) Execute an active event.
SCHED DONE
Inactive
Event 2
NBA
Event 12
Inactive and NBA regions are empty.
Postponed
Watch Event 9
Watch Event 1
Execute START
(2) Execute a postponed event.
DONE
Schedule new events in future time slots.
Time slot t=19
Initiate next time slot.

eq-27


Time slot t=7
(1) Start this time slot.
Active
Event 9
Event 1
Execute START
(2) Execute an active event.
SCHED DONE
Inactive
Event 2
NBA
Event 12
Inactive and NBA regions are empty.
Re-inactive and re-NBA empty BUT events in inactive or NBA region.
Reactive
Event 79
Event 81
Execute START
(2) Execute a reactive event.
SCHED DONE
Re-Inactive
Event 712
Re-NBA
Event 122
Inactive, re-inactive, NBA, and re-NBA regions are all empty.
Postponed
Watch Event 9
Watch Event 1
Execute START
(2) Execute a postponed event.
DONE
Schedule new events in future time slots.
Time slot t=19
Initiate next time slot.

eq-28