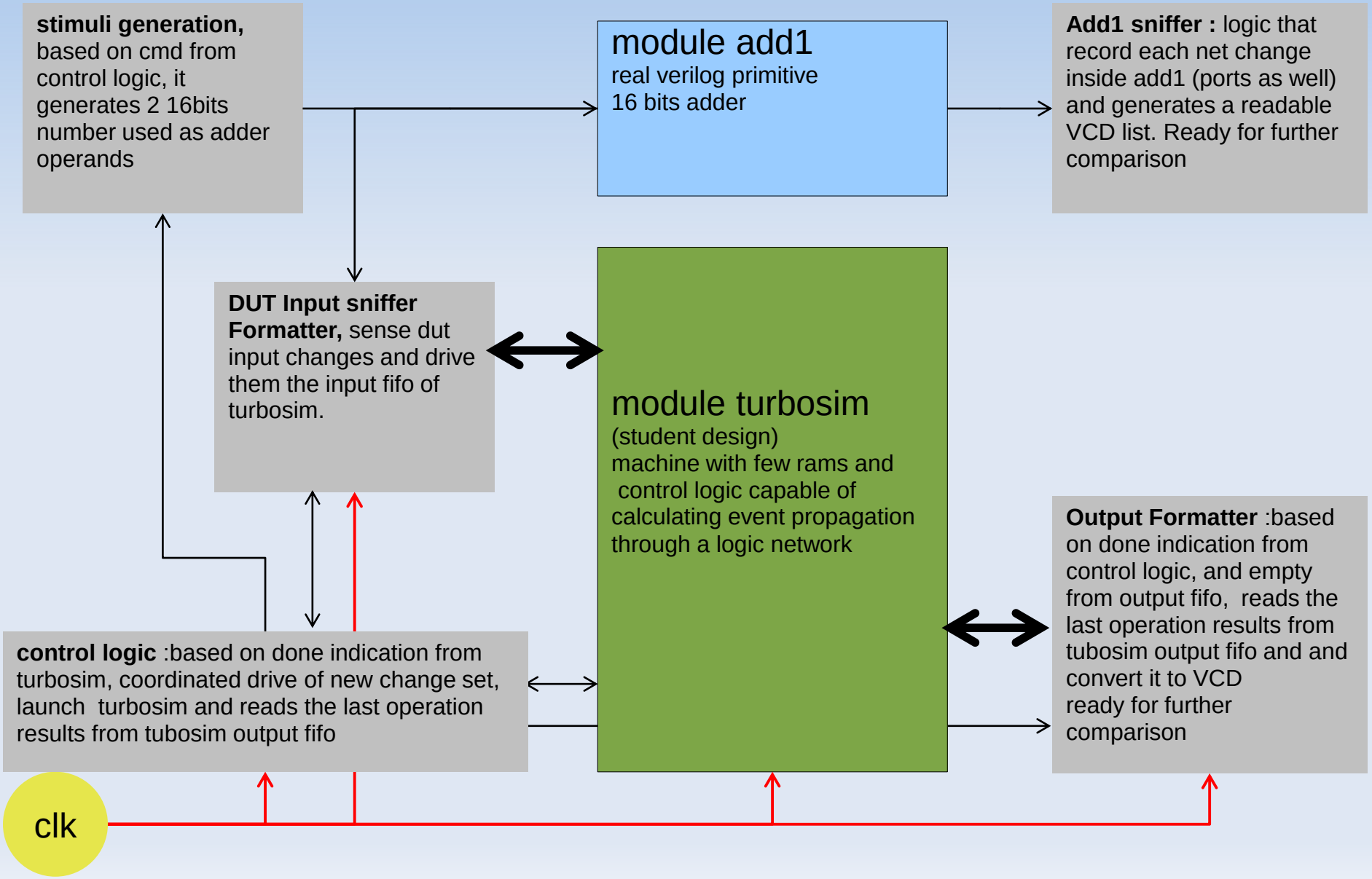
Top level view

module sim_project



Scope of project

- Support verilog primitive logic network (netlists).
- Single module, no hierarchy
- No feedback network, only feed-forward. (support is optional)
- No support for signal strength.
- Supports 4 values 0,1,X, Z
- Support single delay construct per instance (#0.433), 433 ps.
- Reference DUT (design under test) will be an arbitrary delay adder netlist.

Scope of project (1)

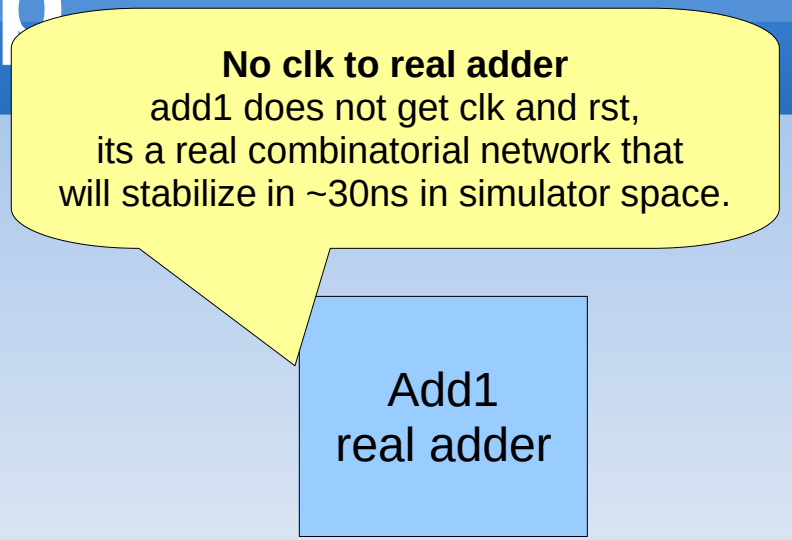
- Up to 32 input bits per DUT
- Well behaved netlists :
  - No input to output connection (only buf buf1(o,i))
  - Output can not be used as internal net ...
    or or1(o1,a,b);
    not not1(o2,o1); // forbidden
  - No support for multiple driven nets (contention)
  - Fully connected, no floating inputs or dandling outputs (of primitives)

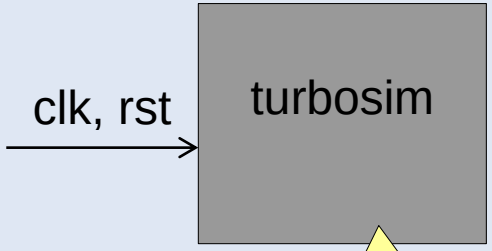
Netlist example

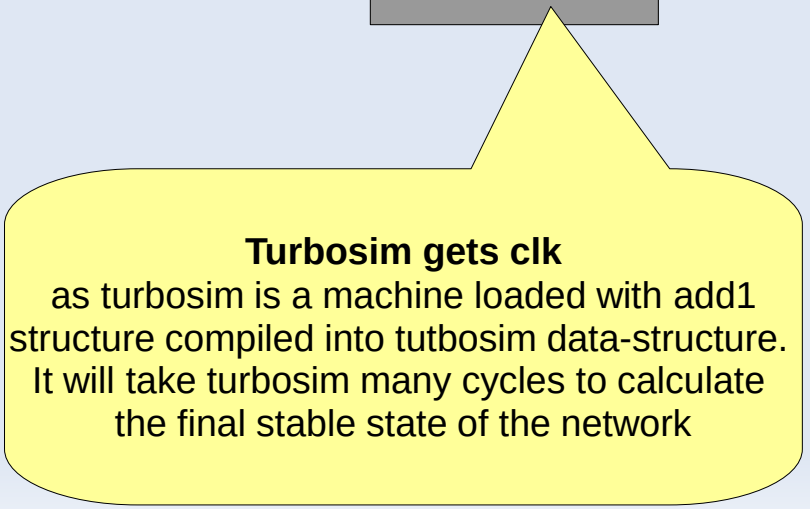
```
module add1 ( o, a, b );
  output [15:0] o;      // single signal in a line declaration
  input [15:0] a;
  input [15:0] b;
  wire   n1, n2, n3, n4, n5, n6, n7,
         n8, n9, n10, n11, n12, n13,
         n14, n15, n16, n17, n18, n19,
         n20, n21, n22, n23, n24, n25,
         n26, n27, n28, n29, n30, n31,
...
  not   #(1.482) U15 ( n87, n85 );
  // verilog primitive, net assignment by order only
  not   #(1.316) U16 ( n73, n71 );
  not   #(1.956) U17 ( n120, n118 );
  nor   #(1.449) U18 ( o[2], n50, n51 );
  and   #(0.225) U19 ( n51, n52, n53 );
  nor   #(0.646) U20 ( n50, n52, n53 );
  nand  #(0.592) U21 ( n53, n54, n55 );
  nor   #(1.762) U22 ( o[3], n43, n44 );
  and   #(0.331) U23 ( n44, n45, n46 );
  nor   #(0.068) U24 ( n43, n45, n46 );
  nand  #(0.466) U25 ( n46, n47, n48 );
  nor   #(1.628) U26 ( o[4], n36, n37 );
```

Test Setup

- Mostly provided by me
- The real adder module is instantiated.
- The student design Turbosim is instantiated.
- Testbench stimulate both design.
- Real adder internal and external nets change is recorded by the testbench, readable format. (code is in place)
- Turbosim results is read from turbosim output fifo
- Testbench translate the read data to a readable format, in a way that files can be compared side by side

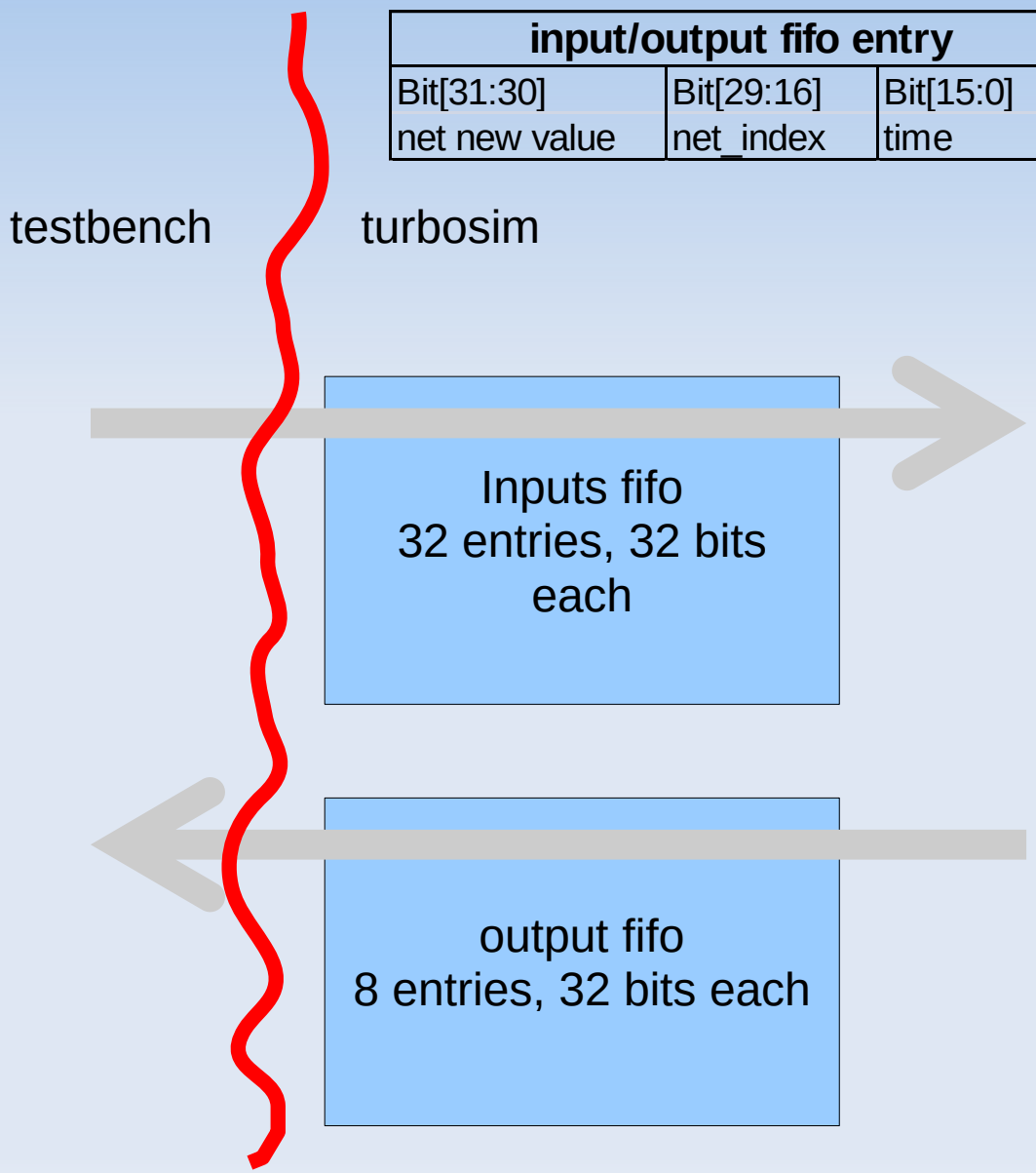






IN/OUT fifo data structure

input/output fifo entry		
Bit[31:30]	Bit[29:16]	Bit[15:0]
net new value	net_index	time



Net name to net index dictionary

```
# //    3    a[12]
# //    4    a[13]
# //    5    a[14]
# //    6    a[15]
# //    7    a[1]
# //    8    a[2]
# //    9    a[3]
# //   10    a[4]
# //   11    a[5]
# //   12    a[6]
# //   13    a[7]
# //   14    a[8]
# //   15    a[9]
# //   16    b[0]
.....

# //  122    n32
# //  123    n33
# //  124    n34
# //  125    n35
# //  126    n36
# //  127    n37
# //  128    n38
# //  129    n39
# //  130    n4
# //  131    n40
# //  132    n41
# //  133    n42
# //  134    n43
# //  135    n44
# //  136    n45
```

Input change set

- Input fifo used to communicate input net changes to the accelerator.
- Up to 32 external changes supported
- Changes should be written sorted by time.
- All inputs change at the same instant is the Basic usage model that must be supported.
- Supporting a gradual net change set is highly appreciated
- Supporting internal nets change (force implementation) is .....

```
1     3        0
0    16        0
1     4        0
1    19        0
.....
up to 32 events
```

```
1     3        0
0    16       14
1     4       17
1    19       17
.....
up to 32 events
```

Gradual change list :
net 3 change at time 0,
net 16 change at time 14,
net 4 change at time 17.
all times in pico seconds
net index to net name conversion is provided.

Cell data structure

```
// table may be used turbo sim to hold the cell functionality delay and connectivity
// table width is 88 bits
// where first 4 bits holds the primitive type
// next is 16 bits that holds the primitive delay in ps
// next is 12 bits the holds the net index of the output pin - pin0 bit 11 is set if exist
// next is 12 bits the holds the net index of the input pin  - pinX bit 11 is set if exist
//                  there are 4 such fields
// next is 4 bits for the current output value, set to x, R/W field
// next is 4 bits control that can be used for whatever, also R/W field
```

0	1	2	3	4	5	6	7	8			0	1	2	3	4	5	6	7	8
0_0327_871_84e_000_000_000_2_0	//	0	U10	|	not	807 ps	n24	n140	UC	UC	UC	val x, ctl
5_019e_833_811_834_000_000_2_0	//	1	U100	|	nand	414 ps	n116	b[10]	n117	UC	UC	val x, ctl
5_030c_834_835_836_000_000_2_0	//	2	U101	|	nand	780 ps	n117	n118	n119	UC	UC	val x, ctl
2_0510_84e_84f_850_000_000_2_0	//	3	U102	|	and	1296 ps	n140	n141	n142	UC	UC	val x, ctl
5_069c_84f_80b_879_000_000_2_0	//	4	U103	|	nand	1692 ps	n141	a[5]	n31	UC	UC	val x, ctl
5_0511_850_81b_851_000_000_2_0	//	5	U104	|	nand	1297 ps	n142	b[5]	n143	UC	UC	val x, ctl
5_04d1_851_852_87d_000_000_2_0	//	6	U105	|	nand	1233 ps	n143	n144	n35	UC	UC	val x, ctl
2_0314_8c0_826_827_000_000_2_0	//	7	U106	|	and	788 ps	n96	n104	n105	UC	UC	val x, ctl
5_00bb_826_802_82b_000_000_2_0	//	8	U107	|	nand	187 ps	n104	a[11]	n109	UC	UC	val x, ctl
5_015a_827_812_828_000_000_2_0	//	9	U108	|	nand	346 ps	n105	b[11]	n106	UC	UC	val x, ctl
5_00a2_828_829_82a_000_000_2_0	//	10	U109	|	nand	162 ps	n106	n107	n108	UC	UC	val x, ctl
0_00b3_869_849_000_000_000_2_0	//	11	U11	|	not	179 ps	n17	n136	UC	UC	UC	val x, ctl
2_02eb_849_84a_84b_000_000_2_0	//	12	U110	|	and	747 ps	n136	n137	n138	UC	UC	val x, ctl
....																			

Cell data structure format

88 bits																			
4b	16b	12b			12b			12b			12b			12b			4b	4b	
p_type	p_delay	net_idx_out			net_idx_in			net_idx_in1			net_idx_in2			net_idx_in3			cur_val	ctl	
		v	11b		v	11b		v	11b		v	11b		v	11b				

net data structure

```
// table may be used turbosim to hold the net connectivity
// table width is 84bits, 5x16+4
// where 5 = 1 driver + 4 loads
// where 16 = 12 + 4
// where 12 = cell index + bit 11 set if exist
// where 4  = pin index of driving/load cell
// where the last 4 bits reflects the number of loads on net
 0     1    2    3    4   5                |     0          1          2         3         4          5
000_0_84a_1_848_1_000_0_000_0_2 //    0 a[0]    |  no dpin    ,U167  pin1 ,U165  pin1 ,no load    ,no load    ,n loads 2
000_0_83f_1_8b3_1_000_0_000_0_2 //    1 a[10]   |  no dpin    ,U157  pin1 ,U99   pin1 ,no load    ,no load    ,n loads 2
000_0_808_1_840_1_000_0_000_0_2 //    2 a[11]   |  no dpin    ,U107  pin1 ,U158  pin1 ,no load    ,no load    ,n loads 2
000_0_811_1_841_1_000_0_000_0_2 //    3 a[12]   |  no dpin    ,U115  pin1 ,U159  pin1 ,no load    ,no load    ,n loads 2
000_0_843_1_81a_1_000_0_000_0_2 //    4 a[13]   |  no dpin    ,U160  pin1 ,U123  pin1 ,no load    ,no load    ,n loads 2
000_0_836_1_844_1_000_0_000_0_2 //    5 a[14]   |  no dpin    ,U149  pin1 ,U161  pin1 ,no load    ,no load    ,n loads 2
000_0_825_1_000_0_000_0_000_0_1 //    6 a[15]   |  no dpin    ,U133  pin1 ,no load     ,no load    ,no load    ,n loads 1
000_0_846_1_8a3_1_000_0_000_0_2 //    7 a[1]    |  no dpin    ,U163  pin1 ,U84   pin1 ,no load    ,no load    ,n loads 2
000_0_8a6_1_838_1_000_0_000_0_2 //    8 a[2]    |  no dpin    ,U87   pin1 ,U150  pin1 ,no load    ,no load    ,n loads 2

....
000_0_845_1_81e_1_855_2_000_0_3 //   31 b[9]    |  no dpin    ,U162  pin1 ,U127  pin1 ,U177  pin2 ,no load    ,n loads 3
87b_0_879_1_000_0_000_0_000_0_1 //   32 n1      |  U48   pin0 ,U46   pin1 ,no load     ,no load    ,no load    ,n loads 1
816_0_877_1_89d_2_876_1_000_0_3 //   33 n10     |  U12   pin0 ,U44   pin1 ,U79   pin2 ,U43   pin1 ,no load    ,n loads 3
887_0_886_2_000_0_000_0_000_0_1 //   34 n100    |  U59   pin0 ,U58   pin2 ,no load     ,no load    ,no load    ,n loads 1
88a_0_887_2_889_2_000_0_000_0_2 //   35 n101    |  U61   pin0 ,U59   pin2 ,U60   pin2 ,no load    ,no load    ,n loads 2
859_0_88a_1_000_0_000_0_000_0_1 //   36 n102    |  U180  pin0 ,U61   pin1 ,no load     ,no load    ,no load    ,n loads 1
830_0_88a_2_000_0_000_0_000_0_1 //   38 n104    |  U107  pin0 ,U106  pin1 ,no load     ,no load    ,no load    ,n loads 1
.....
```

Net data structure format – 84bits					
16b	16b	16b	16b	16b	4b
net_drive	net_load0	net_load1	net_load2	net_load3	load_num

Net drive/load format – 16b		
12b		4b
val	cell_idx – 11b	pin_idx

Event Queue data structure

- General structure of event Q
- Field width to be specified by student
- Filed encoding to be specified by student
- Design should maintain an empty container that holds the addresses of empty records.
- During init phase, this container should be field by ascending addresses
- Q size => 512 entries. ???
- Empty container should be fifo based
- Event Q should be dual or single port RAM

event queue entry necessary fields - 29 bits					
event type	new value	object index	time	prev event	next event
1b	2b	11b	16b	Irrelevant	irrelevant

Events flow

```
while (there are events) {
    if (no active events) {
        if (there are inactive events) {
            activate all inactive events;
        }
        else if (there are nonblocking assign update events) { // zero delay support ?
            activate all nonblocking assign update events;
        }
        else if (there are monitor events) {    // no support
            activate all monitor events;
        }
        else {
            advance T to the next event time;
            activate all inactive events for time T;
        }
    }
    E = any active event;
    if (E is an update event) {
        update the modified object;
        add evaluation events for sensitive processes to event queue;
    } else { /* shall be an evaluation event */
        evaluate the process;
        add update events to the event queue;
    }
}
```

utilities

- parse_verilog_netlist.pl <verilog module file>
- First level support – simple verilog, no special package support.
- Reads the verilog file and generates several files/tables/perl data structures, that are used by testbench and design.
- Also generates <module_name>.pl, perl representation of all data in module. This can be used by the student for further processing.

```
maxn@maxn-desktop:~/verilog_try$ lth
total 1000
-rw-r--r-- 1 maxn maxn 215245 2010-03-14 12:33 add1.pl
-rw-r--r-- 1 maxn maxn  22163 2010-03-14 12:33 cell_index_to_data_mem.v
-rw-r--r-- 1 maxn maxn  25942 2010-03-14 12:33 net_index_to_data_mem.v
-rw-r--r-- 1 maxn maxn  21360 2010-03-14 12:33 net_index_to_name_mem.v
-rw-r--r-- 1 maxn maxn   6502 2010-03-14 12:33 ref_dut_input_sniffer.v
-rw-r--r-- 1 maxn maxn  33334 2010-03-14 12:33 ref_dut_sniffer.v
-
```

Readable VCD format

- VCD – Value Change Dump

  Four-state: to represent variable changes in 0, 1, x, and z with no strength information.

  Extended: to represent variable changes in all states and strength information.

- Four-state VCD file:

  A VCD file is an ASCII file that contains header information, variable definitions, and the value changes for all variables specified in the task calls

Readable VCD format

- Examples:
  1. initial $dumpfile ("module1.dump") ;
  2. $dumpvars (1, top);
  3. $dumpvars (0, top);
  4. $dumpvars (0, top.mod1, top.mod2.net1);

Bonuses

- Force on wires.
  - Impact on events
- Change inputs during simulation.

Events (spec summary)

- Events types
- Events timing
- Stratified Event queue

Event Types

- **Update Event:**

  Every change in value of a net or variable in the circuit being simulated, as well as the named event

- **Evaluation Event:**

  Processes are sensitive to update events. When an update event is executed, all the processes that are sensitive to that event are evaluated in an arbitrary order. The evaluation of a process is also an event, known as an evaluation event.

Event Types: Update

Needed steps:

- Net D.S. (data structure)

  1. Change net's current value and figure out the driven gates

- Cell D.S.

  2. Read all driven cells

- EQ

  3. Insert Evaluation events of these cells

Event Types: Evaluation

Needed steps:

- Cell D.S.

  1. Change its current out put value and figure out the out put net index

- Net D.S. (data structure)

  2. read out put net entery

- EQ

  3. Insert Update events to all driven nets

Events Timing

- The term simulation time is used to refer to the time value maintained by the simulator to model the actual time it would take for the circuit being simulated.
- In order to keep track of the events and to make sure they are processed in the correct order, the events are kept on an *event queue*, ordered by *simulation time*.
- **Scheduling an event:**

  Putting an event on the queue.

Stratified Event queue (EQ)

- EQ is logically segmented into five different regions. Events are added to any of the five regions, but are only removed from the *active region.*
- *Active events:*

  occur at the current simulation time and can be processed in any order.
- *Inactive events:*

  occur at the current simulation time, but shall be processed after all the active events are processed.

Stratified Event queue (EQ)

Unsupported

- *Nonblocking assign update events:*

  have been evaluated during some previous simulation time, but shall be assigned at this simulation time after all the active and inactive events are processed.

- *Monitor events:*

  shall be processed after all the active, inactive, and nonblocking assign update events are processed.

- *Future events:*

  occur at some future simulation time. Future events are divided into future inactive events and future nonblocking assignment update events