TestBench Regarding Time and Events

CECS360

CSULB

Verilog Data Types

- Up till now we have been dealing with two main data types - Registers and Nets - with a practical application using **reg** and **wire** when modeling hardware
- In addition to modeling hardware there are other uses for variables in an HDL model - we now will consider **integer** and **time**
- A **time** variable is used for storing and manipulating simulation time quantities when timing checks are required for diagnostics and debugging
- An **integer** is a general purpose variable for manipulating quantities that are not regarded as hardware registers

Integer and Time

- The **time** variable is typically used in conjunction with the **$time** system function. The time variable is unsigned 64 bits
- An **integer** is a signed variable supporting two’s complement operations and is 32 bits
- Arrays of time and integer variables are allowed

```
integer a[1:64];                  // array of 64 integers
time change_history[1:1000]; // array of 1000 times
```

Real Numbers

- Verilog supports real number constants and variables
- Not all Verilog operators can be used with real number values
- Ranges are not allowed on real number variable declarations
- Real number variables default to an initial value of zero
- Real numbers can be specified in either decimal notations (14.72) or in scientific notation (39e8)
- Real numbers specified with a decimal point must have at least one digit on each side of the decimal point

Operators and Real Numbers

- The result of using logical or relational operators on real numbers is a single-bit scalar value
- Prohibited use includes edge descriptors, bit-select, memories
- Converting real numbers to integers by rounding a real number to the nearest integer - not truncation
- 35.7 and 35.5 -> 36, 35.2 -> 35
- Implicit conversion occurs when you assign a real to an integer

Parameters

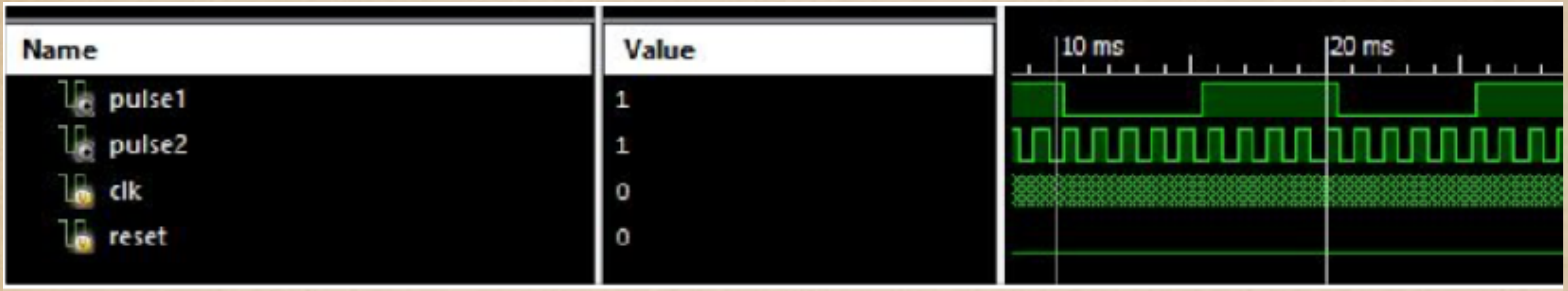
- Verilog parameters do not belong to either the register or the net group
- Parameters are not variable - they are constants
- Parameters may be ‘over ridden’ at compile time
- More convenient to use than `define since the variable defined must include

Measuring Elapsed Time

- Keeping track of elapsed time is achieved using multiple approaches
- The first item needed is of the test bench to identify when an event occurs
- This is accomplished by using an approach that is not acceptable when modeling synchronous logic but is perfectly acceptable for a test bench

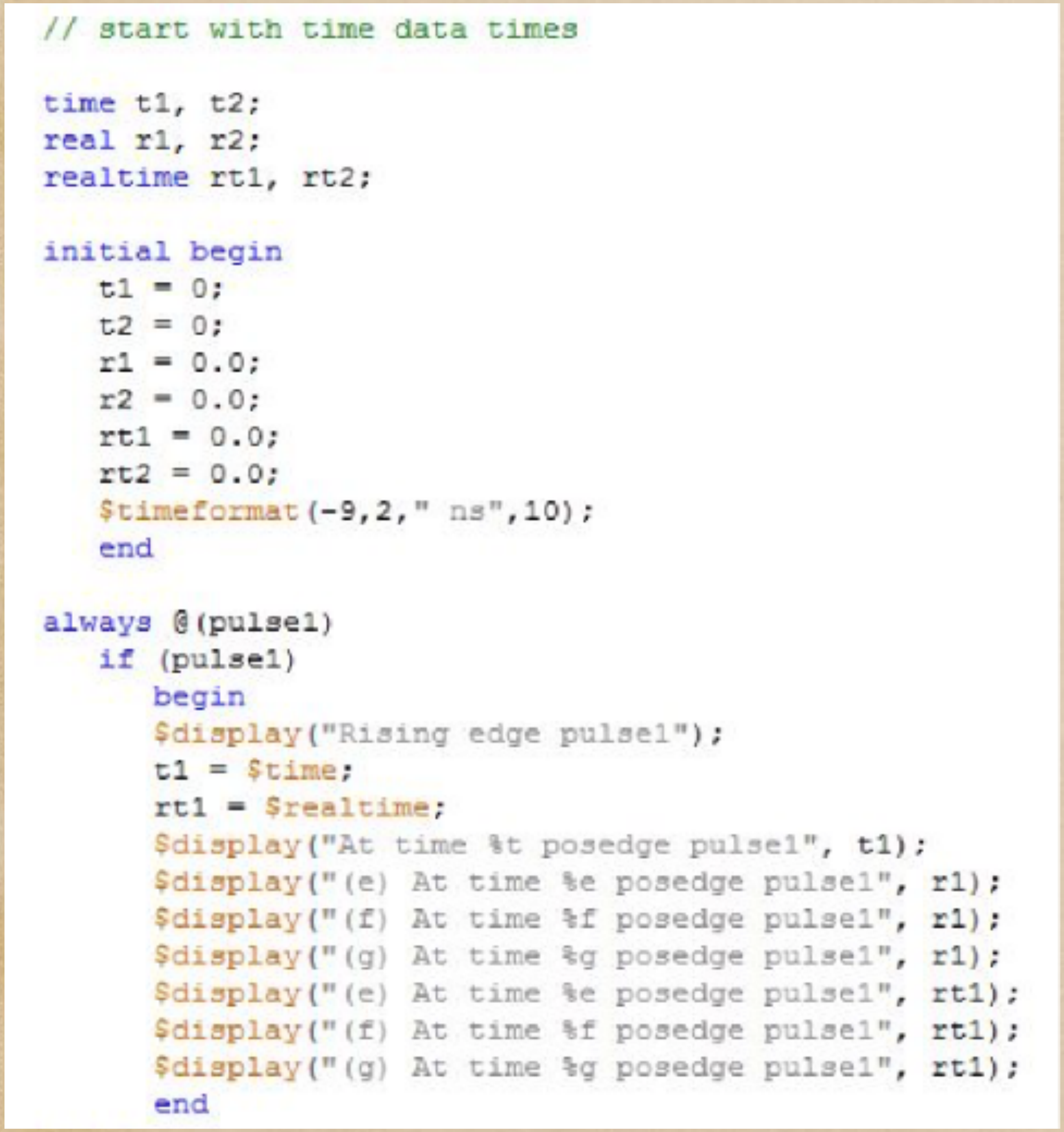
```
always @(signal)          // detect any transition
always @(posedge signal) // detect rising edge
always @(negedge signal) // detect falling edge
```

Detecting Transitions



- In the example case there are two signals - pulse1 & pulse2 which are switching at different frequencies
- The goal here is to define an approach where the switching characteristics of the signals may be quantified

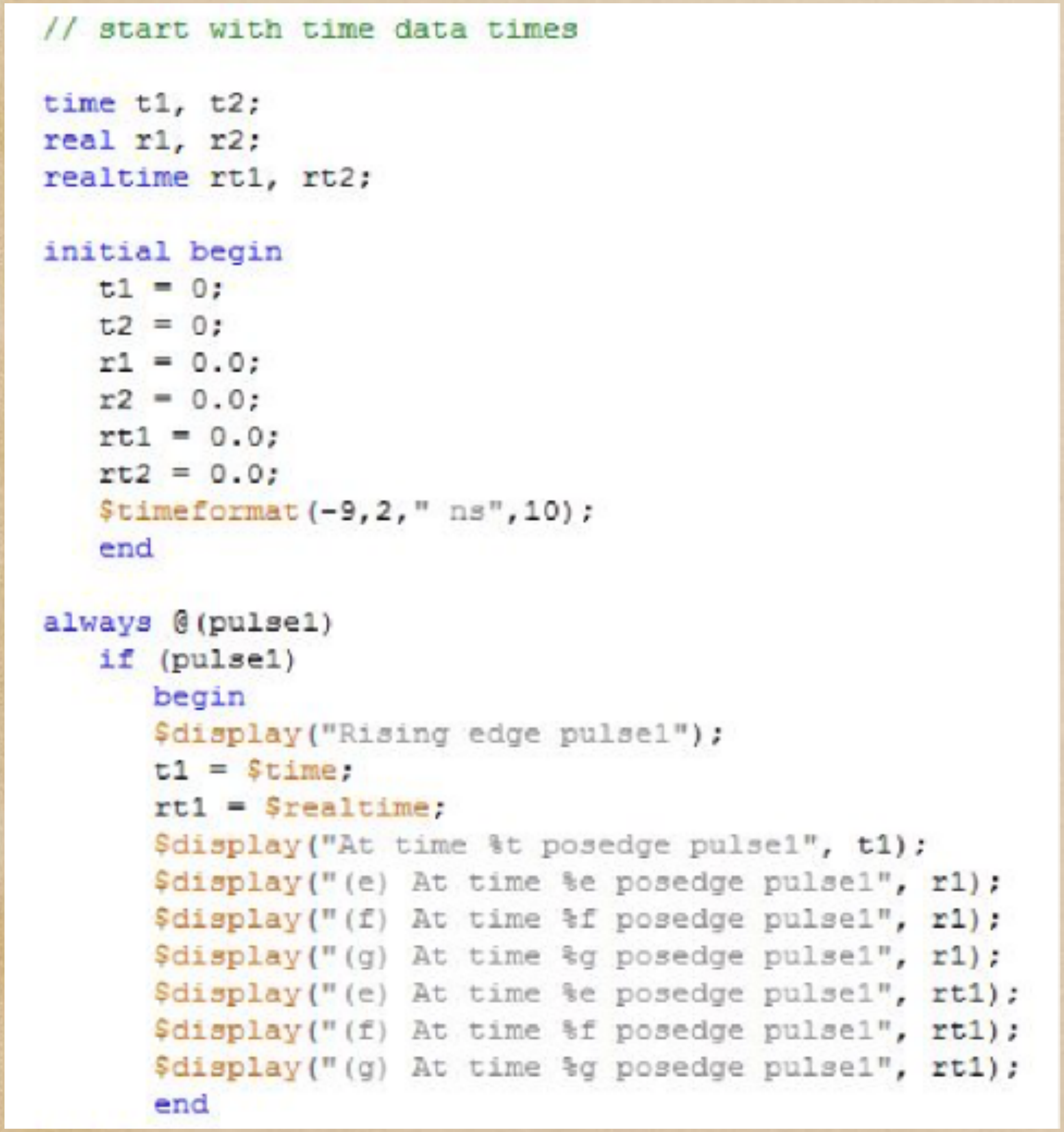
Test Bench Setting up Evaluation



```
// start with time data times

time t1, t2;
real r1, r2;
realtime rt1, rt2;

initial begin
   t1 = 0;
   t2 = 0;
   r1 = 0.0;
   r2 = 0.0;
   rt1 = 0.0;
   rt2 = 0.0;
   $timeformat(-9,2," ns",10);
   end

always @(pulse1)
   if (pulse1)
      begin
      $display("Rising edge pulse1");
      t1 = $time;
      rt1 = $realtime;
      $display("At time %t posedge pulse1", t1);
      $display("(e) At time %e posedge pulse1", r1);
      $display("(f) At time %f posedge pulse1", r1);
      $display("(g) At time %g posedge pulse1", r1);
      $display("(e) At time %e posedge pulse1", rt1);
      $display("(f) At time %f posedge pulse1", rt1);
      $display("(g) At time %g posedge pulse1", rt1);
      end
```

- The options presented are which data types to use when monitoring elapsed time
- There is time, realtime, and eventually just a reg
- $timeformat will assist when displaying simulation time using time variables
- -9:ns 2:10ps 10:spaces

Print Formatters for Reals

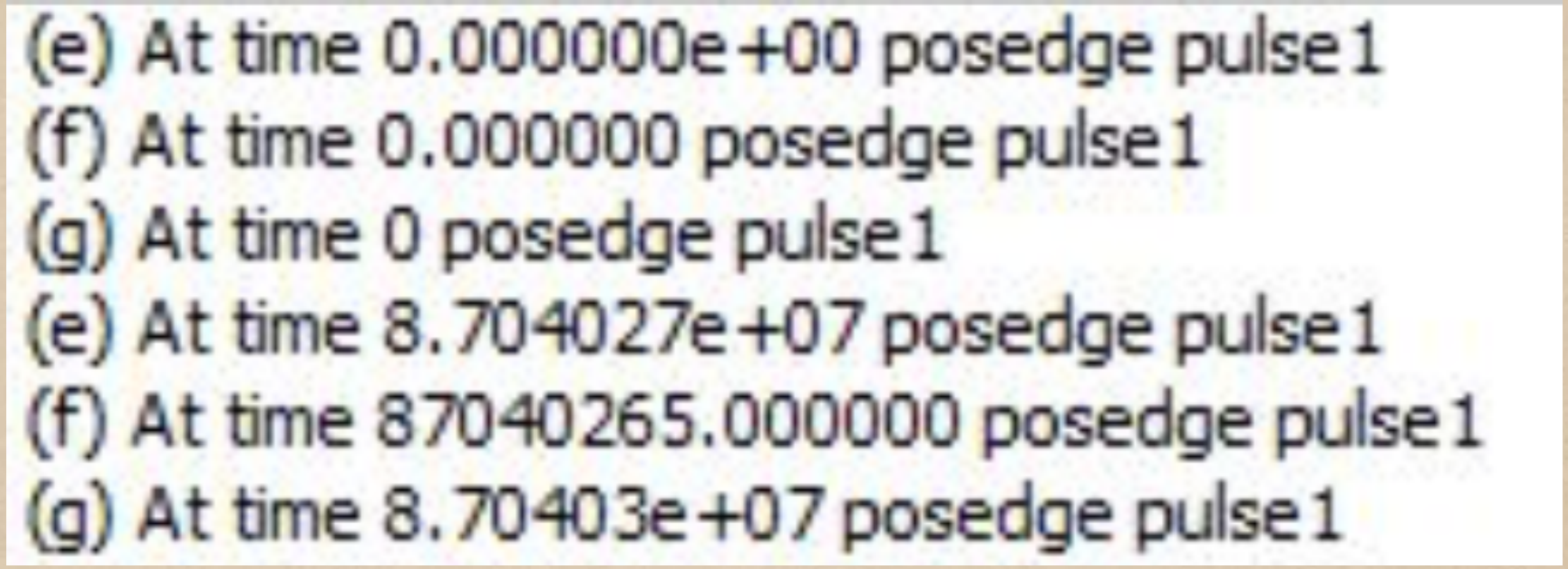


```
// start with time data times

time t1, t2;
real r1, r2;
realtime rt1, rt2;

initial begin
   t1 = 0;
   t2 = 0;
   r1 = 0.0;
   r2 = 0.0;
   rt1 = 0.0;
   rt2 = 0.0;
   $timeformat(-9,2," ns",10);
   end

always @(pulse1)
   if (pulse1)
      begin
      $display("Rising edge pulse1");
      t1 = $time;
      rt1 = $realtime;
      $display("At time %t posedge pulse1", t1);
      $display("(e) At time %e posedge pulse1", r1);
      $display("(f) At time %f posedge pulse1", r1);
      $display("(g) At time %g posedge pulse1", r1);
      $display("(e) At time %e posedge pulse1", rt1);
      $display("(f) At time %f posedge pulse1", rt1);
      $display("(g) At time %g posedge pulse1", rt1);
      end
```

- When using real variables there are different print formatters you may use: %e, %f, %g

- The next page will document the difference between these formatters

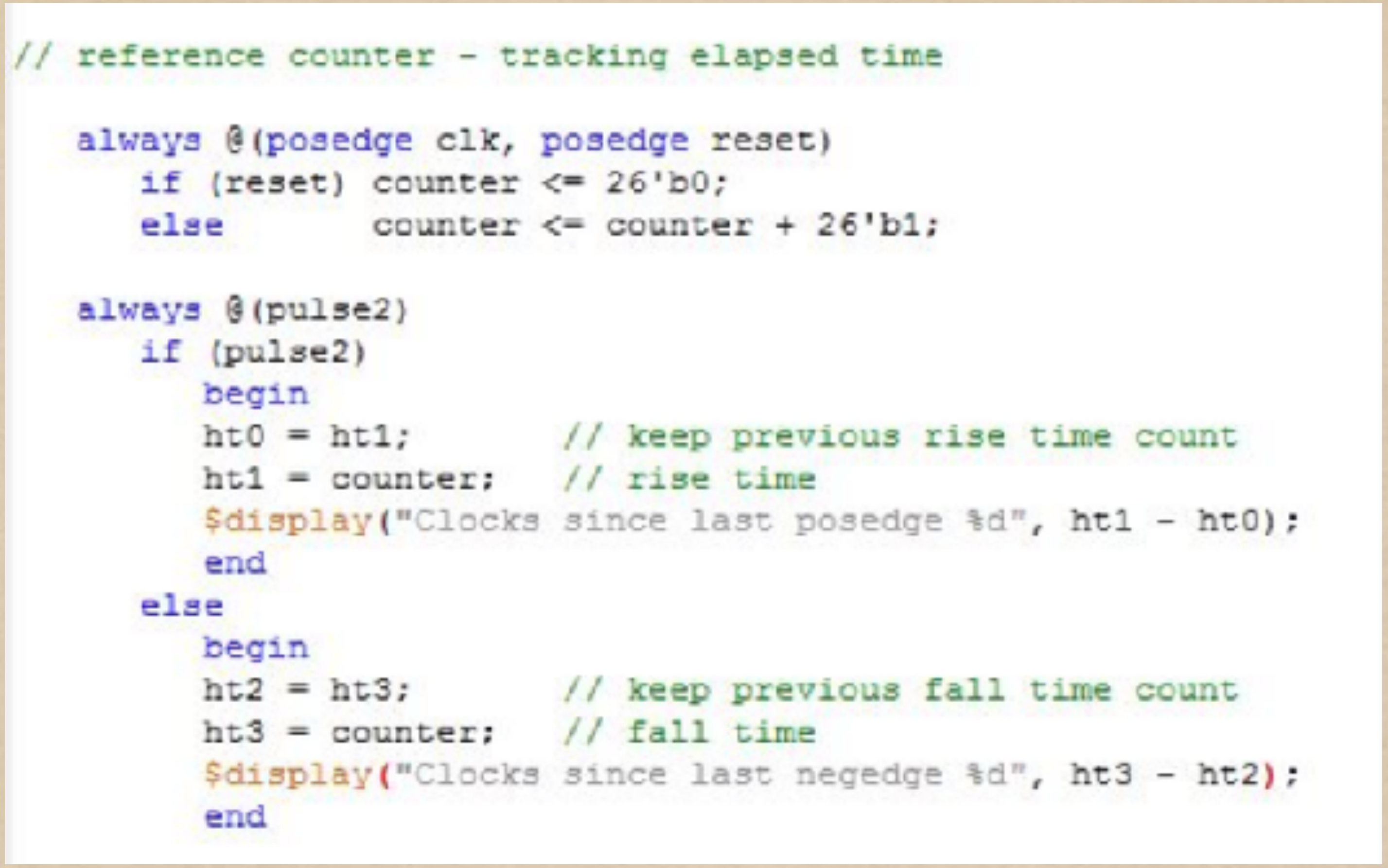
Real Numbers are Problematic



```
(e) At time 0.000000e+00 posedge pulse1
(f) At time 0.000000 posedge pulse1
(g) At time 0 posedge pulse1
(e) At time 8.704027e+07 posedge pulse1
(f) At time 87040265.000000 posedge pulse1
(g) At time 8.70403e+07 posedge pulse1
```

- The above demonstrates the difference between the formatters
- In my opinion none of the above are easily read nor manipulated
- Another approach will yield a simpler way to capture elapsed time and to monitor the durations
- This allows the test bench the ability to form opinions regarding what it is observing when compared to what it was expecting

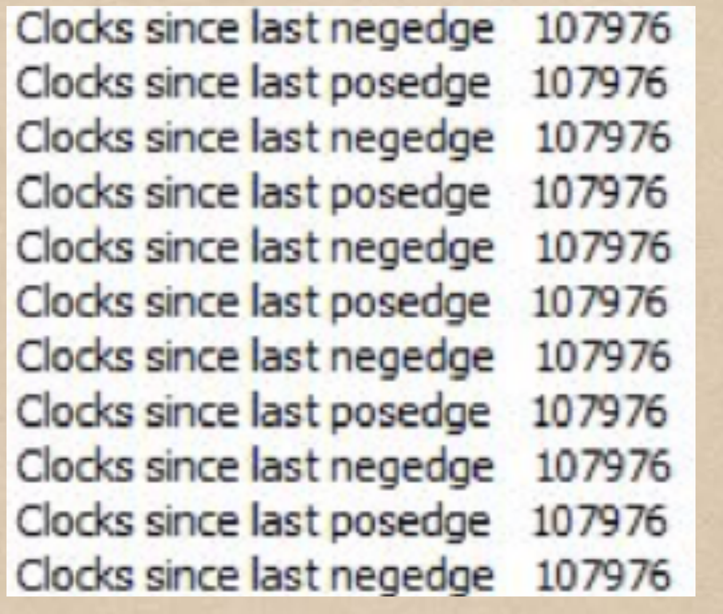
Technique for Counting Clocks



```
// reference counter - tracking elapsed time

   always @(posedge clk, posedge reset)
      if (reset) counter <= 26'b0;
      else       counter <= counter + 26'b1;

   always @(pulse2)
      if (pulse2)
         begin
         ht0 = ht1;        // keep previous rise time count
         ht1 = counter;    // rise time
         $display("Clocks since last posedge %d", ht1 - ht0);
         end
      else
         begin
         ht2 = ht3;        // keep previous fall time count
         ht3 = counter;    // fall time
         $display("Clocks since last negedge %d", ht3 - ht2);
         end
```

- The above demonstrates the use of reg to define a large counter that will be referenced whenever elapsed time needs to be measured
- Define the size of the register appropriate to the duration of the simulation
- When the events occur you are able to save the previous count and capture the current count
- The resultant math will yield the duration between the events which may then be evaluated

Counting Clocks Displayed



```
Clocks since last negedge   107976
Clocks since last posedge   107976
Clocks since last negedge   107976
Clocks since last posedge   107976
Clocks since last negedge   107976
Clocks since last posedge   107976
Clocks since last negedge   107976
Clocks since last posedge   107976
Clocks since last negedge   107976
Clocks since last posedge   107976
Clocks since last negedge   107976
```

- Although the example is not that exciting it does document the ability of the test bench to differentiate between rising and falling edges and to keep track of the difference in elapsed time

- If one wanted to also consider the duty cycle of the signal then the math could then also be applied to the difference between a posedge and a negedge event