Session #9

Sub-programs

- Verilog has two sub-program modeling constructs:
  - Tasks
  - Functions
- Tasks and functions provide means of splitting Verilog code into small parts which can be frequently used in during modeling.

Tasks

Task definition properties:

- A task can be declared only within the module definition
- Enclosed within the task and endtask keyword
- A task can contain parameter, input, output, inout, event, and register declarations
- All register data types can be used in task definition (***reg***, ***integer***, ***real***, ***realtime***, ***time***)
- Net declaration is ***illegal***.
- If more than one statement is given, then begin-end block must be used.
- Tasks can contain time-control statements
- Tasks return value through variables declared as inout or output

Syntax:

```
task identifier;

  parameter declaration;

  input declaration;
  output declaration;
  inout declaration;

  register declaration;

  event declaration;

  statement_or_block;

endtask
```

Just specify the name of the task to enable a task in a particular part of a Verilog code.

Tasks

Example 1:

```
task first_task;

  parameter size = 4;
  input a;
  integer a;
  inout [size-1:0] b;
  output c;
  reg [size-1:0] d;
  event e;

  begin
   d = b;
   c = |d;
   b = ~b;
   if (!a) -> e;
  end

endtask
```

Example 2:

```
reg a, b;
task my_task; // definition
begin
   a = 1'b1;
   b = 1'bx;
end
endtask
my_task; // task enabling
```

Task enabling:

```
integer x;
reg a, b, y;
reg [3:0] z;
reg [7:0] w;

first_task(x, z, y);
first_task(x, w[7:4], w[1]);
first_task(1, {a, b, w[3], x[0]}, y);
```

Example 3:

```
task negation;
  inout data;
  data = ~data;
endtask

task my_nor;
  input a, b;
  output c;
  begin
   negation(a);
   negation(b);
   c = a & b;
  end
endtask
```

Functions

```
function type_or_range identifier;

  parameter declaration;
  input declaration;
  register declaration;
  event declaration;

  statement_or_block;

endfunction
```

Just specify the name of the function to enable a function in a particular part of a Verilog code.

Function definition properties:

- Can be declared only inside a module declaration.
- Enclosed within ***function*** and ***endfunction*** keywords
- The returned type or range should follow the keyword ***function***
- Permitted declarations: range, returned type, parameters, input arguments, registers and events
- Time-control statements, task enable and net declarations are ***illegal***.
- A function specified without a range or a type defaults to a one-bit register for return value
- At least one input declaration
- Returns only one value by function name (return value should be assigned to register with the same name as function)

Examples

Example 1:

```
function [3:0] f1;
  input [1:0] data;
  input enable;
. . .
endfunction

wire [3:0] b;
reg [7:0] data;
reg d1, d2, en;

assign b = f1({d1,d2}, en);

initial begin
  data[7:4]  = f1(b[1:0], 1'b1);
end
```

Function Enabling:

```
real a;
wire [15:0] b;
wire c, d, e, f;
assign b = negation ({4{c, d, e, f}});

initial begin
  a = multiply(1.5, a);
  $display(" b=%b ~b=%b", b, negation(b));
end
```

Example 2:

```
function real multiply;
input a, b;
real a, b;
  multiply = ((1.2 * a) * (b * 0.17)) * 5.1;
endfunction
```

Example 3:

```
function [15:0] negation;
input [15:0] a;
 negation = ~a;
endfunction
```

Tasks Vs. Functions

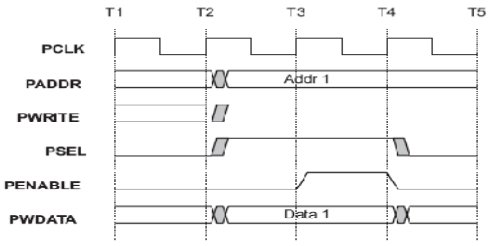
Tasks:

1. Can contain time-control statements
2. Can enable other tasks and functions
3. Return value through arguments declared as ***inout*** or ***output***
4. Zero or more values can be returned
5. Can have zero or more arguments declared as ***input***, ***inout***, and ***output***

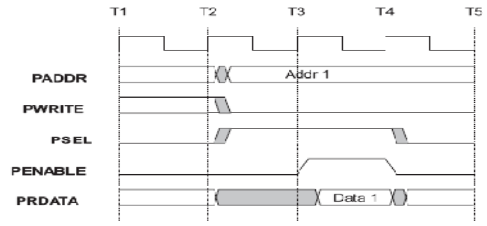
Functions:

1. Can't contain time-control statements
2. Can't enable tasks
3. Return value by name
4. Only one value can be returned
5. At least one input argument
6. Can't have any ***inout*** or ***output*** arguments

Case Study: Bus Transactions



Write transfer: APB master



Read transfer: APB master

How can a master, slave and arbiter be designed for this specs?

Exercise 1

- Write a task which can read from a FIFO and write another task which can write to a FIFO.
- Verify the same using the given simulator

Assume the FIFO you have designed in the previous sessions

45 Minutes