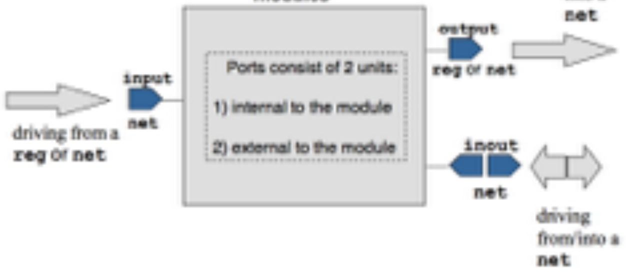
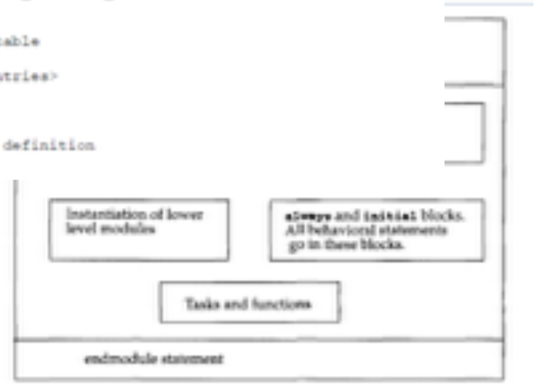
ASIC Design Flow: Func. Specs., HDL, Synthesis, Floorplanning, place & route, verify in circuit.
FPGA: Func. Specs., HDL, Synthesis, place & route, download and verify.
A Verilog simulator consists of a compiler, user interface, and event-based scheduler.
Design Block, Stimulus Block
Abstractions Levels: Algorithmic, data flow, gate, switch, RTL.
signed 8 bit: -8sd104. Constant integers are signed by default. If width is specified, it's not signed

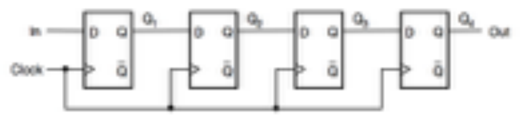
```
my_bit = bus[152];
my_bit = bus[i];                // selects ith b
my_byte = bus[87:80];
my_byte = bus[80+:8];           // selects bits
my_byte = bus[87-:16];          // selects bits
my_byte = bus[byteNum*8-:8];  // variable part
my_byte = bus[120:127];         // Illegal!
```

Port connections when modules are instantiated within other modules


driving into a net
output
reg or net
input
net
driving from a reg or net
Ports consist of 2 units:
1) internal to the module
2) external to the module
inout
net
driving from/into a net

```
//UDP name and terminal list
primitive <udp_name> (
<output_terminal_name>(only one allowed)
<input_terminal_names> );


//Terminal declarations
output <output_terminal_name>;
input <input_terminal_names>;
reg <output_terminal_name>;(optional; only for sequential
                                              UDP)


// UDP initialization (optional; only for sequential UDP
initial <output_terminal_name> = <value>;


//UDP state table
table
    <table entries>
endtable


//End of UDP definition
endprimitive
```


Instantiation of lower level modules
always and initial blocks. All behavioral statements go in these blocks.
Tasks and functions
endmodule statement

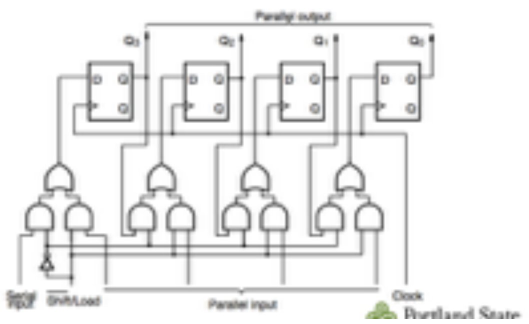


A Shift register using 4 D flip-flops

```
module shiftreg4 (
    input In, Clock;
    output Out;
);
    reg [1:4] Q;

    assign Out = Q[4];
    always @(posedge Clock)
        Q <= {In, Q[1:3]};

endmodule
```

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

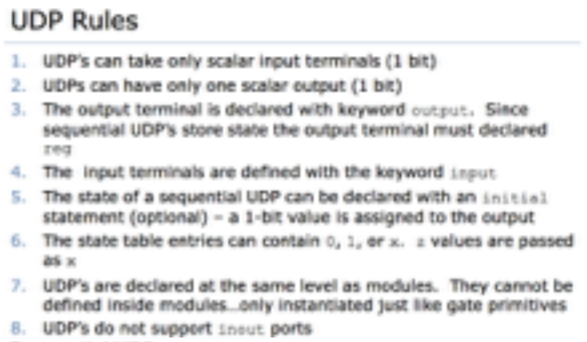
Loads input or shifts on clock edge



```
module shift4 (
    input [3:0] R,
    input L, w, Clock,
    output reg [3:0] Q
);

    always @(posedge Clock)
        if (L)
          Q <= R;
        else begin
          Q[0] <= Q[1];   // Could also be Q <= {w, Q[3:1]};
          Q[1] <= Q[2];
          Q[2] <= Q[3];
          Q[3] <= w;
        end

endmodule
```

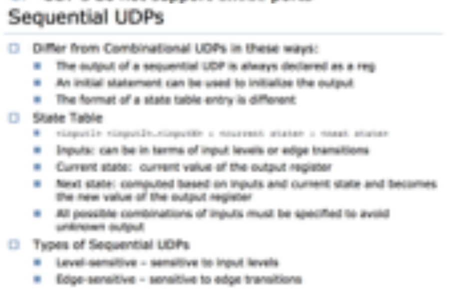
Functions	Tasks
A function can enable another function but not another task.	A task can enable other tasks and functions.
Functions always execute in 0 simulation time.	Tasks may execute in non-zero simulation time.
Functions must not contain any delay, event, or timing control statements.	Tasks may contain delay, event, or timing control statements.
Functions must have at least one input argument. They can have more than one input.	Tasks may have zero or more arguments of type input, output, or inout.
unctions always return a single value. They cannot have output or inout arguments.	Tasks do not return with a value, but can pass multiple values through output and inout arguments.

~| Nor, ^ XOR, ^~ Xnor. **casex —> treats all x and z as don’t cares (don’t check them). casez** —> treats all z as don’t cares.
Sequential block —> begin end, **parallel block** —> fork join. **Named Blocks**
Two Types of UDP: Combinational and sequential.



UDP Rules

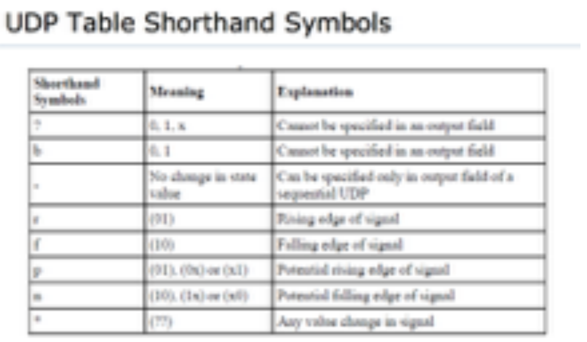
1. UDP's can take only scalar input terminals (1 bit)
2. UDPs can have only one scalar output (1 bit)
3. The output terminal is declared with keyword `output`. Since sequential UDP's store state the output terminal must declared `reg`
4. The input terminals are defined with the keyword `input`
5. The state of a sequential UDP can be declared with an `initial` statement (optional) – a 1-bit value is assigned to the output
6. The state table entries can contain `0`, `1`, or `x`. `z` values are passed as `x`
7. UDP's are declared at the same level as modules. They cannot be defined inside modules...only instantiated just like gate primitives
8. UDP's do not support `inout` ports



Sequential UDPs

- Differ from Combinational UDPs in these ways:
  - The output of a sequential UDP is always declared as a reg
  - An initial statement can be used to initialize the output
  - The format of a state table entry is different
- State Table
  - `<input1> <input2>...<inputN> : <current state> : <next state>`
  - Inputs: can be in terms of input levels or edge transitions
  - Current state: current value of the output register
  - Next state: computed based on inputs and current state and becomes the new value of the output register
  - All possible combinations of inputs must be specified to avoid unknown output
- Types of Sequential UDPs
  - Level-sensitive – sensitive to input levels
  - Edge-sensitive – sensitive to edge transitions

UDP has exactly one output. min of 9 in for seq UDP and 10 in for comb UDP. **Events:** Named, Regular, Event OR Control, Level-Sensitive Control. wait() is used for level-sensitive. **Stimulus vectors – provide inputs to the model under test. Reference vectors –** provide expected output from the model under test...used to check (either manually or automatically) that the model is performing as expected. **`timescale** <ref time units>/<time precision>.100ns/1ns means sim. time tick is 100 ns and sim. time is rounded of to 1 ns increments. **I/O: Open up to 3 files at most. $readmem<b/h>(file_name, mem_name, start_address, end_address). $random(<seed>). Seed is integer, time, or reg. returns a 32-bit signed integer. {$random()} to generate positives. `ifndef <flag>** - compile code between `ifndef and the next `endif only when <flag> is not defined. **`ifdef <flag>** - compile code between `ifdef and the next `endif only when <flag> is defined. **`else** – optional. Specifies path if `ifndef or `ifdef is not taken. Each `ifndef or `ifdef can only have one `else associated with it. **`elsif** – optional. Specifies conditional path if `ifndef or `ifdef is not taken. Each `ifndef or `ifdef any number of `elsif associated with it. **3 kinds of events:** Regular, Non-blocking, Monitor events. **Races:** Read-write, write-write, always-initial.



UDP Table Shorthand Symbols

Shorthand Symbols	Meaning	Explanation
?	0, 1, x	Cannot be specified in an output field
b	0, 1	Cannot be specified in an output field
-	No change in state value	Can be specified only in output field of a sequential UDP
r	(01)	Rising edge of signal
f	(10)	Falling edge of signal
p	(01), (0x) or (x1)	Potential rising edge of signal
n	(10), (1x) or (x0)	Potential falling edge of signal
*	(??)	Any value change in signal