Lecture - 3

☆ Recap:
- module
- assign statement
- NET type variables
- REGISTER type variables

☆ Register Data Type
- Register is a variable that can hold a value
- Register cannot be used with "assign" statement as LHS of assign statement should be of type "NET".
- Register statement not necessarily maps to a hardware register during synthesis.

* Types of Register data types in Verilog:
i) reg
ii) integer
iii) real
iv) time (not used in synthesis, only used for simulation)

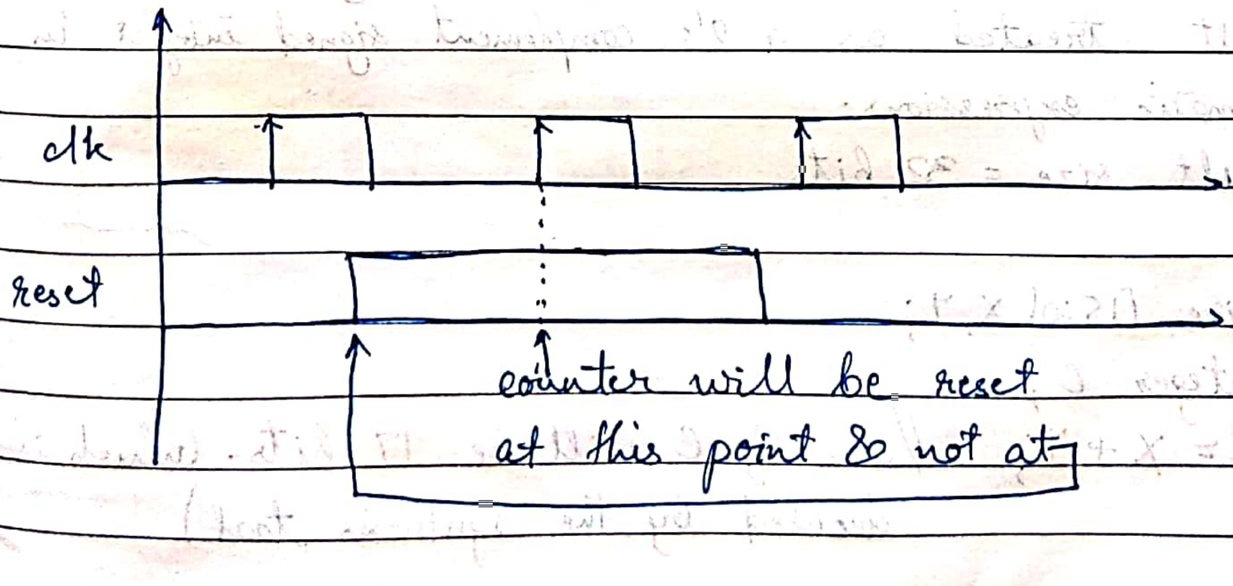
i) "reg" data type
- default value of a "reg" type is 'X'.
- declaration explicitly specifies the size.
  Eg: reg x,y; // single bit register variables
      reg [15:0] bus; // A 16 bit bus
- reg is treated as an unsigned no. in arithmatic expressions
- reg must be used when we model actual sequential ~~model~~ elements like counters, shift registers, etc.

★ Eg: 32 bit counter with synchronous reset

```
module simple_counter(clk, reset, count);
    input clk, reset;
    output [31:0] count;  ──> reg [31:0] count;
    always @(posedge clk)
    begin
        if (reset)
                count = 32'b0;
        else
                count = count + 1;
    end ~~end~~
endmodule
```

synchronous reset as reset will work only at +ve edge of the clock.

Note: In the above code, "count" is of type "reg". so we cannot use "assign" statement like "assign count = 32'b0;" but we can directly use count = 32'b0;



A 32 bit asynchronous counter

```
module simple_counter (clk, rst, count);
    input clk, ~~resetrst~~ rst;
    output [31:0] count;
    reg [31:0] count;
    always @ (posedge clk or posedge rst)
    Begin
        if (rst)
            count = 32'b0;
        else
            count = count + 1;
    end
endmodule
```

ii) "integer" data type
- general purpose register data type
- more convenient to use in loop counting.
- default treated as a 2's complement signed integer in arithmatic expressions.
- default size = 32 bits
- Eg:

```
    wire [15:0] X, Y;
    integer C;
    C = X + Y;   //size of C will be 17 bits. (which is
                   decided by the synthesis tool)
```

iii) "real" data type
- used to store floating point numbers
- when a "real" value is assigned to an integer, the real no. is rounded off to the nearest integer.
- Eg:

```
    real e, pi;
    initial            //"initial" is used in testbench
    begin
        e = 2.718;
        pi = 314.159e-2;
    end
```

Another code :

```
    integer x;
    initial
    begin
        x = pi;        // 'x' gets only 3 values
    end
```

iv) "time" data type
- In verilog, simulation is carried out w.r.t a logical clock called simulation time.
- "time" data type can be used to store simulation time.
- Eg:

```
    time curr_time;
    initial
    .......
        ~~current~~ curr_time = $time;
    end
```

✰ Vectors

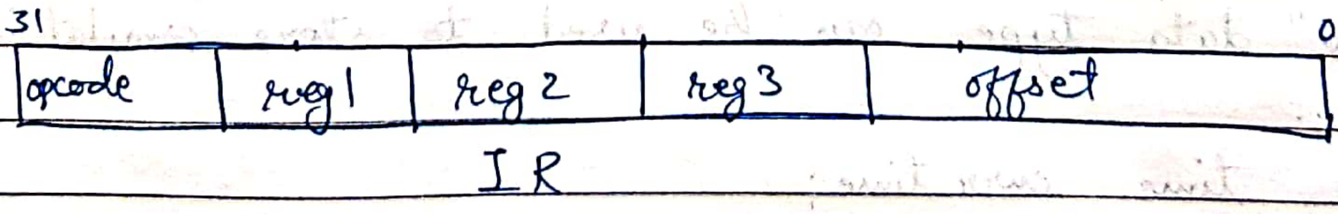
- NETs or "reg" type variables can be declared as vectors of variable bit widths.
- Eg:

```
wire x,y,z;          //single bit values
wire [7:0] sum;      //MSB is sum[7], LSB is sum[0]
reg [31:0] MDR;
reg [1:10] data;     //MSB is data[1], LSB is data[10]
```

* Using sections of a vector

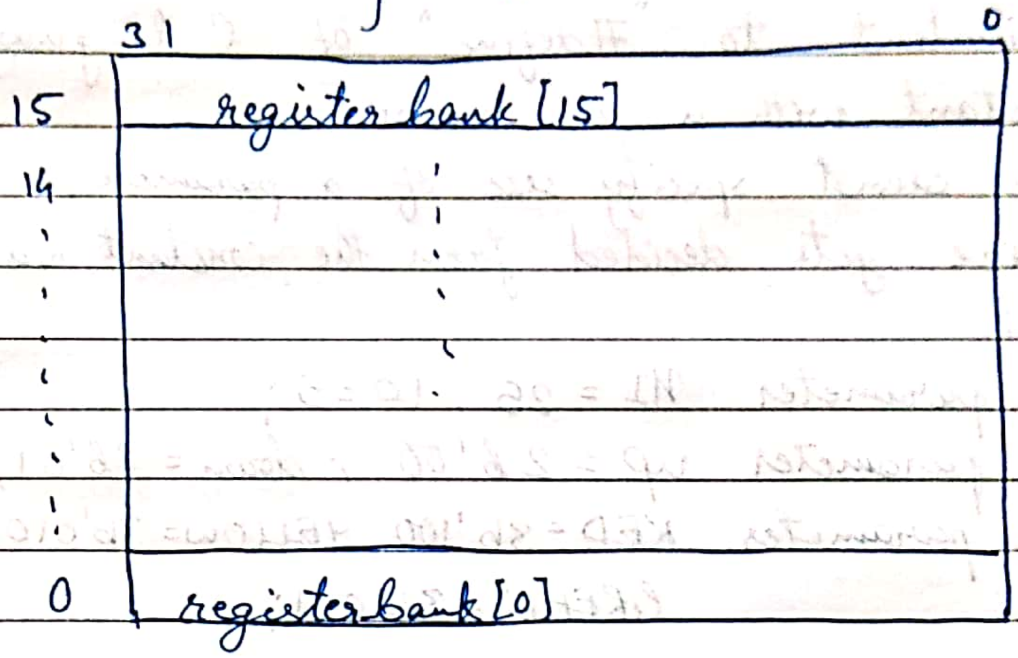
- Eg:

```
reg [31:0] IR;
reg [5:0] opcode;
reg [4:0] reg1, reg2, reg3;
reg [10:0] offset;
opcode = IR[31:26];
reg1 = IR[25:21];
reg2 = IR[20:16];
reg3 = IR[15:11];
offset = IR[10:0];
```



✯ Multidimensional Arrays & Memories

• Eg:



reg [31:0] register_bank [15:0] ; // 16 32-bit register

• Eg: $\underset{\substack{\uparrow \\ \text{32 bit by default}}}{\underline{\text{integer}}}$ matrix $\underset{\substack{\uparrow \\ \text{8 rows}}}{\underline{[7:0]}}\underset{\substack{\uparrow \\ \text{16 columns}}}{\underline{[15:0]}}$;

• Eg: reg mem_bit [0:2047] ; // 2Kb memory
reg [7:0] mem_byte [0:1023] ; // 1KB memory

✯ Specifying Constant Values

• can be specified in sized or unsized form

• Eg:

```
4'b0101     // 4 bit Binary no. 0101
1'b0        // logic 0 (1 bit)
12'hB3C     // 12 bit hex no.
12'h 8XF    // 12 bit hex no. (1000 xxxx 1111)
25          // signed integer
```

✰ Parameters

- Equivalent to "#define" of C language i.e. a constant with a given name
- we cannot specify size of a parameter
- size gets decided from the constant value itself
- Eg:

```
parameter HI = 25, LO = 5;
parameter up = 2'b00, down = 2'b01, steady = 2'b10;
parameter RED = 3'b100, YELLOW = 3'b010,
          GREEN = 3'b001;
```

- In above examples, it is easier to understand the code if HI, LO, up, down, steady, RED, etc. are used rather than the corresponding values.

- Eg: N bit counter

```
module counter (clear, clock, count);
  parameter N=7;
   input clear, clock;
   output [0:N] count;
  reg [0:N] count;
   always @(negedge clock)
      if (clear)
           count <= 0;          // <= vs = will be
      else                      // covered later
           count <= count + 1;
   end
endmodule
```

NOTE: Any variable assigned inside "always" block must be of type reg.

Scanned with CamScanner