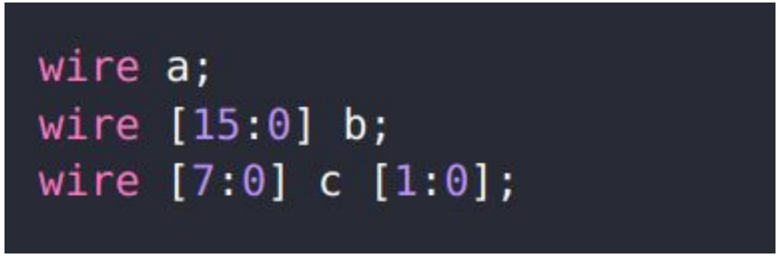
Data Types in Verilog

Wire datatype

We use a wire datatype when we want to connect two components (just like a real wire)

We cannot do multi assignment to a wire(i.e either its connected to constant voltage or some other wire)

Each Wire has its own unique name in a module



```
wire a;
wire [15:0] b;
wire [7:0] c [1:0];
```

- ‘a’ is the name of one wire transferring 1 bit
- ‘b’ is the name of a 16 bit wire
- ‘c’ is the name of 2 , 8 bit wires (an array of wires)

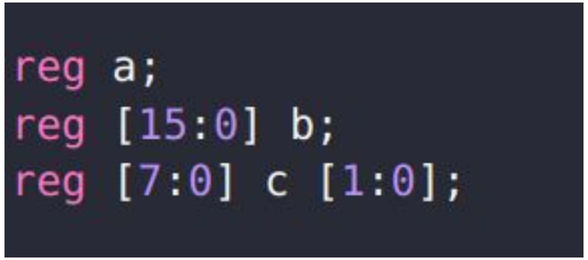
All input ports must be wires

Reg datatype

We use the Reg datatype whenever we want to store and use values (has memory capabilities). Need not always mean a real register in hardware.

Have to use inside a always block or initial block

Can be changed just by assigning it a new value



```
reg a;
reg [15:0] b;
reg [7:0] c [1:0];
```

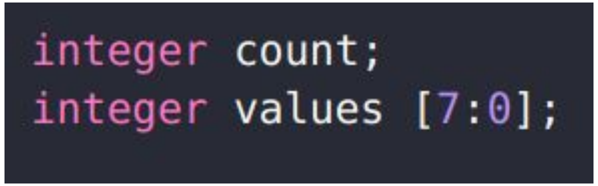
- ‘a’ is the name of one register storing 1 bit
- ‘b’ is the name of a 16 bit register
- ‘c’ is the name of 2 , 8 bit register (an array of registers)

Output ports can be regs or wires

Integer datatype

Acts just like an integer in C++

Only use in Testbench for testing, DO NOT USE in Hardware files.



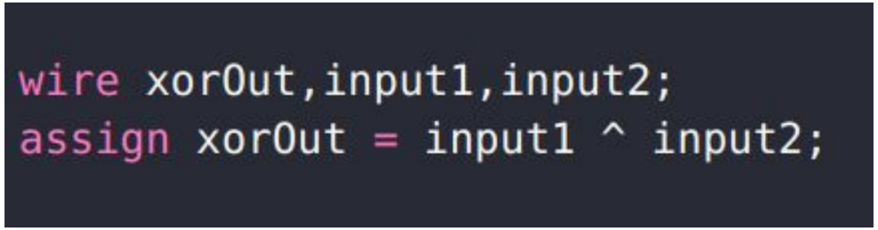
```
integer count;
integer values [7:0];
```

- 1 integer named count
- An array of 8 integers, named values

Assign keyword

The Assign statement connects a source to a destination

This assignment is continuous, i.e. used for setting wires and connecting wires



```
wire xorOut,input1,input2;
assign xorOut = input1 ^ input2;
```

Setting Reg values

We won’t be using the assign keyword when setting reg values.
Why is this? Well a simple way to explain it is that reg values are changed at discrete points of time (eg. every cycle, every rising edge..etc), whereas assign keyword does continuous assignment of the value to the reg which is not allowed.
However wires can be used for the same.

We usually set/change register values during a edge(either positive or negative) of the clock signal

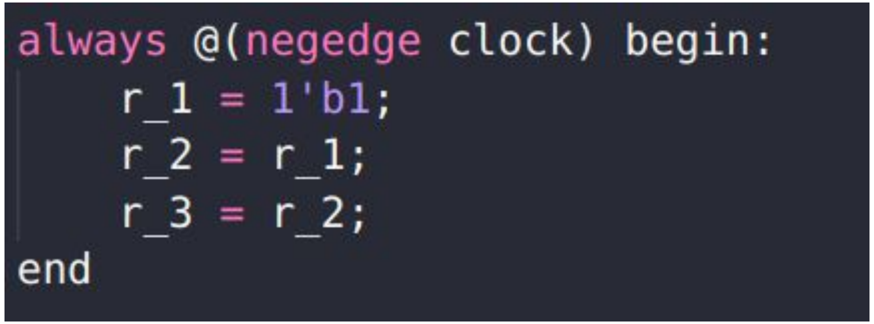
We can set values in 2 ways

- Blocking assignment (= operator)
- NonBlocking assignment (<= operator)

Blocking assignment

During a Blocking assignment, all statements after the assignment only take place after the blocking assignment is done, that is,

The assignment works in a sequential way as written in code.
This means that statement r_2 = r_1; gets executed after r_3 = r_2; and
 r_3 = r_2 is executed after r_2 = r_1, so....execution is sequential one after another;



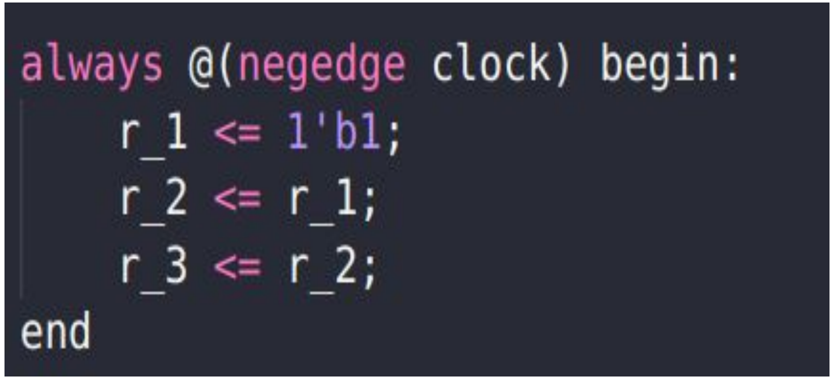
```
always @(negedge clock) begin:
    r_1 = 1'b1;
    r_2 = r_1;
    r_3 = r_2;
end
```

In this scenario, all three registers get assigned the value 1 in one cycle itself

Non Blocking Assignment

During a Non Blocking assignment, all statements after the assignment take place all at the same time, that is,

The assignment works in parallel with all the other assignments



```
always @(negedge clock) begin:
    r_1 <= 1'b1;
    r_2 <= r_1;
    r_3 <= r_2;
end
```

In this scenario, It takes 3 cycles for r_3 to be set to 1, since, in the first cycle, when r_1 is set to 1, that value is not used to set r_2 because it is running in parallel
The previous cycle value of r_1 is used in this case