DATA TYPES

Data Types

The data handled in Verilog fall into two categories:

(i) Net data type

(ii) Variable data type (register data type)

- The two types differ in the way they are used as well as with regard to their respective hardware structures.
- Data type of each variable or signal has to be declared prior to its use.
- The same is valid within the concerned block or module.

Net data type

- A net type, represents a physical connection between structural elements or a connection from one circuit unit to another.
- Its value is determined from the value of its drivers such as a continuous assignment or a gate output. If no driver is connected to a net, the net defaults to a value of z.
- Such a net carries the value of the signal it is connected to and transmits to the circuit blocks connected to it. If the driving end of a net is left floating, the net goes to the high impedance state.
- **Nets are one-bit values by default** unless they are declared explicitly as vectors.

- A net can be specified in different ways.

  1. Wire 2. Tri

- **wire:**
  - It represents a simple wire doing an interconnection
  - Only one output is connected to a wire and is driven by that.
  - The terms wire and net are often used interchangeably
- **tri:**
  - It represents a simple signal line as a wire.
  - Unlike the wire, a tri can be driven by more than one signal outputs.

Net Types

- wire
- tri
- wor
- trior
- wand
- triand
- trireg
- tri1
- tri0
- supply0
- supply1

A simple syntax for a net declaration is:

```
net_kind [ msb : lsb ] net1, net2 , . . . , netN ;
```

- where net_kind is one of the nets listed above, msb and lsb are constant expressions that specify the range of the net.
- The range specification is optional; if no range is specified, a net defaults to a size of one bit.
- ***Here are some examples of net declar***ations.

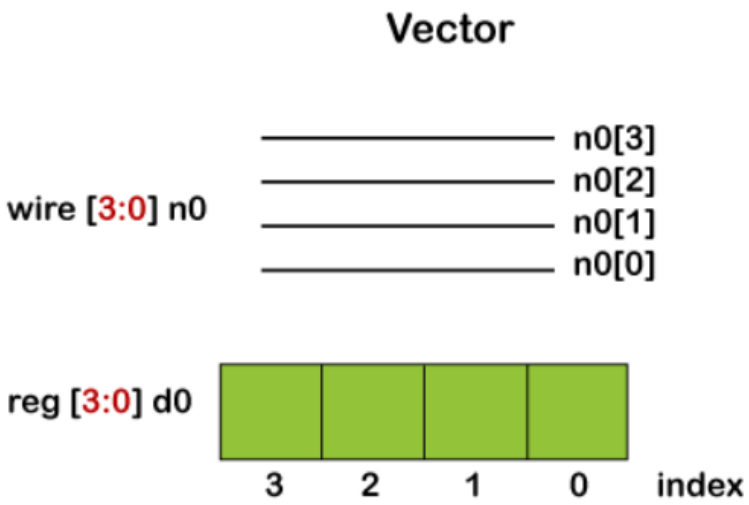
```
wire Rdy, Start;  // Two 1-bit wire nets.
wand [2:0] Addr;  // Addr is a 3-bit vector wand net.
```

Variable Data Type

- A variable is an abstraction for a storage device.
- It can be declared through the keyword **reg** and **stores**
- the value of a logic level are **: 0, 1, x, or z.**
- A net or wire connected to a **reg** takes on the value stored in the **reg** and **can be used as input to other circuit elements**.
- **But the output of a circuit cannot be connected to a reg**.
- The value stored in a reg is changed through a fresh assignment in the program.
- Other variable types of data are **time, integer, real, and realtime**

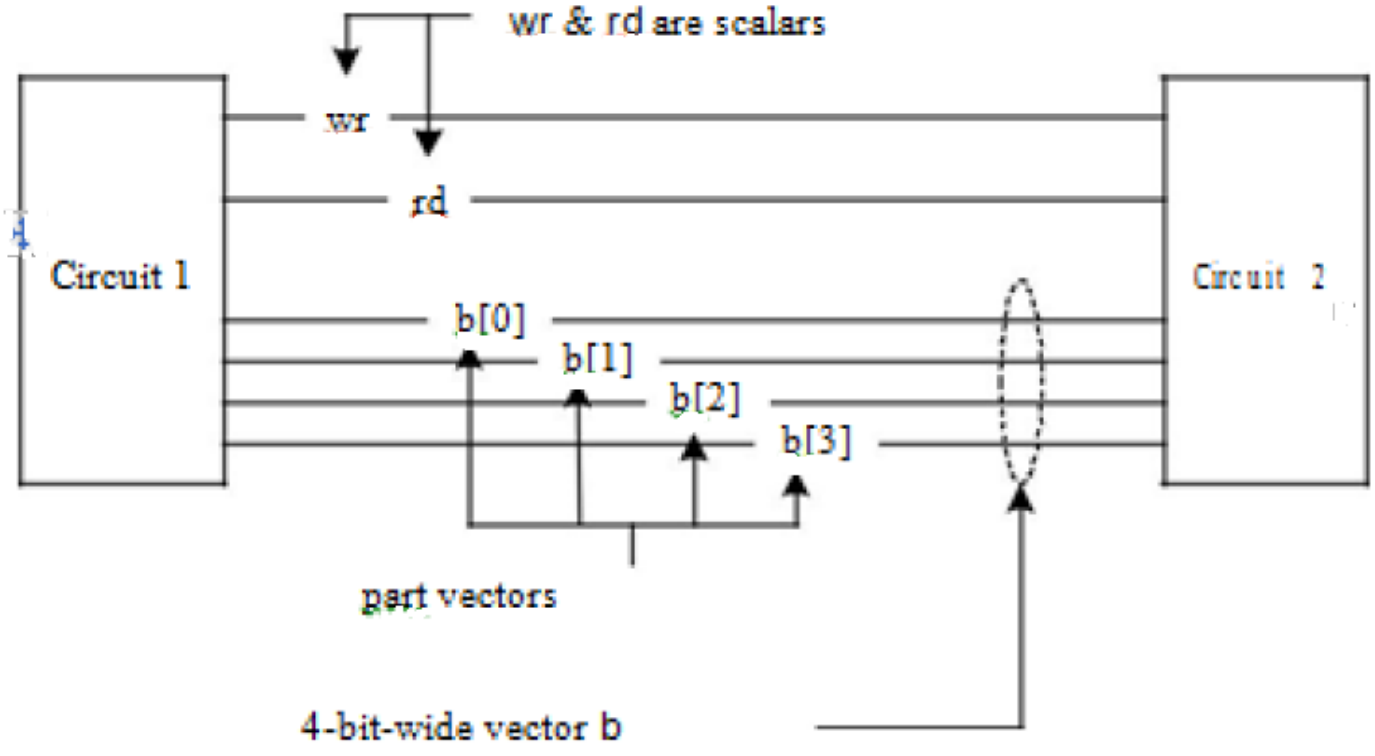
Scalars and Vectors

- Entities representing single bits — whether the bit is stored, changed, or transferred — are called **"scalars."**
- Often multiple lines carry signals in a cluster – like data bus, address bus, and so on. Similarly, a group of **regs** stores a value, which may be assigned, changed, and handled together. The collection here is treated as a **"vector."**



If the width of a data variable is not declared, scalar is assumed.

- Figure below illustrates the difference between a scalar and a vector.
- wr and rd are two scalar nets connecting two circuit blocks circuit1 and circuit2.
- b is a 4-bit-wide vector net connecting the same two blocks. b[0], b[1], b[2], and b[3] are the individual bits of vector b. **They are "part vectors."**



- The range gives the ability to address individual bits in a vector.
- The most significant bit of the vector should be specified as the left hand value in the range while the least significant bit of the vector should be specified on the right.

```
1 | wire        o_nor;          // single bit scalar net
2 | wire [7:0]  o_flop;         // 8-bit vector net
3 | reg         parity;         // single bit scalar variable
4 | reg  [31:0] addr;           // 32 bit vector variable to store address
```

- Whenever a range is not specified for a net or a reg, the same is treated as a scalar – a single bit quantity.
- In the range specification of a vector the most significant bit and the least significant bit can be assigned specific integer values. These can also be expressions evaluating to integer constants – positive or negative.
- Normally vectors – nets or regs – are treated as unsigned quantities. They have to be specifically declared as "signed" if so desired.

Examples

- ***wire signed[4:0] num*; // num is a vector in the range -16 to +15.**
- ***reg signed [3:0] num_1;*** // num_1 is a vector in the range -8 to +7.

Register Types

There are five different kinds of register types.

- reg
- integer
- time
- real
- realtime

Notes:

- Registers provide data storage. In Verilog, register simply means a variable that can hold a value.
- A Verilog register is not the same as a hardware register. A register does not need a driver.
- They also do not need a clock like a hardware register does. Values are changed by assigning a new value to the register.
- In Verilog, we do not explicitly declare hardware registers. The context of the description determines if a physical register exists.
- Registers are declared by the keyword reg.The default value for a reg data type is x.

```
reg start; // declares register "start"
reg reset, clock; // declares registers reset & clock
```

Time

- A special register which tracks simulation time. Verilog simulation is done with respect to simulation time.
- A special time register data type is used in Verilog to store simulation time.
- A time variable is declared with the keyword time.
- The width for time register data types is implementation-specific but is at least 64 bits.
- The system function $time is invoked to get the current simulation time.

```
time save_sim_time; // Define a time variable save_sim_time initial
save_sim_time = $time; // Save the current simulation time
```

Integer data type

- A general purpose register data type for manipulating integer values.
- They are declared by the keyword integer.
- Values are 32 bit signed values.
- These are rarely used in the description of a hardware module but are frequently useful in the Test Fixture.