

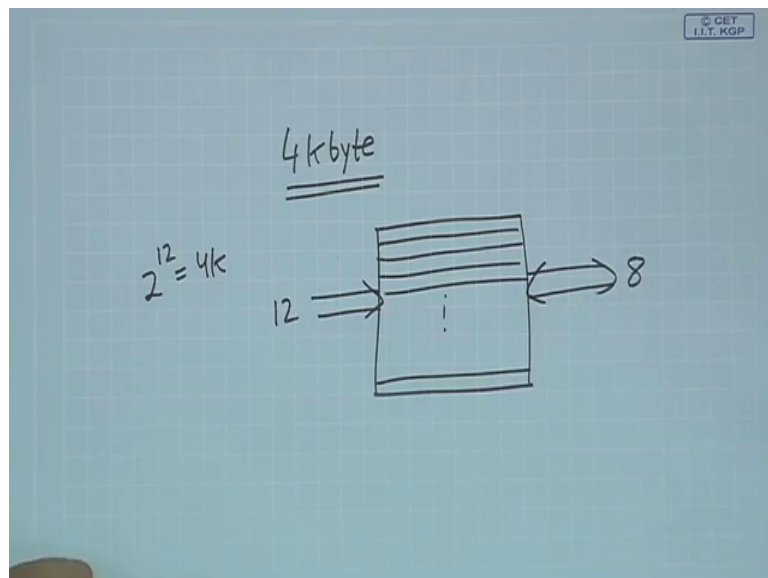
Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 30
Modeling Memory

So, far we have seen how we can model various sequential and combinational circuits using Verilog language. And you may know the kind of variables that we have used for most of these examples, where either single bit variables or it was a vector. Well in some very specific cases we use 2 dimensional matrices or arrays. Now, I mean in this lecture and the next, we shall be primarily focusing on how to model some specific kinds of 2 dimensional data elements. In this lecture we shall be looking at how to model memory elements.

Now, conceptually speaking, a memory element can be regarded as a 2 dimensional array of storage cells.

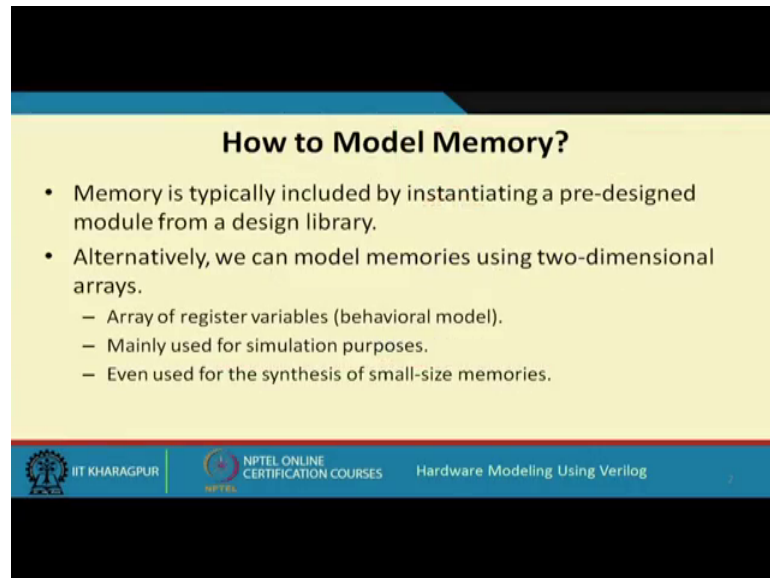
(Refer Slide Time: 01:17)



For example, let us say if I talk about a, let us say 4 kilobyte memory chip or a memory cell, what does this mean. So, it is a box, inside which there is storage space for 4 kilobytes of data. Byte means there are 8 data lines available to me, and 4 k means there are 2 to the power 12 is 4 k. So, there are 12 address lines available to access 1 of the memory words, and the size of the memory word is 8 bits fine.

So, how you can declare such a memory cell? I mean as an array, and how we can use it in a Verilog code, we shall be seeing through this lecture.

(Refer Slide Time: 02:14)



How to Model Memory?

- Memory is typically included by instantiating a pre-designed module from a design library.
- Alternatively, we can model memories using two-dimensional arrays.
 - Array of register variables (behavioral model).
 - Mainly used for simulation purposes.
 - Even used for the synthesis of small-size memories.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, the question is how to model memory. Now the first thing I would like to say is that. Well, when you are doing a professional design which will be finally, targeted to a hardware. So, when your aim is to synthesize the hardware into an AC call an FPGA, then the kind of memory blocks that we use, are not exactly what we shall be talking about now. There we will have to pick up some existing memory cells from a library

For instance, when you are using a, some kind of f p j, then there are some predesigned memory cells available in the library, you can pick them up, you can put them in your design and you can create whatever size memory you need.

Similarly, for an normally the way memory elements are laid out and fabricated are quite different from the way normal gates are laid out; that is why the layout of a memory is entirely separate and they are typically available as a prefabricated element in the library fine

So, that is what the first point says, that in a well design where your target is to synthesis, memory will be typically included by instantiating some existing module from a library. Now the examples that we shall be showing today, that as an alternative, we can also model memories using a 2 dimensional array.

Now, you see memory conceptually consist of a number of words. Each word consists of a number of bits. So, each word, we can regard as some kind of a register. So, a memory array can be regarded as an array of registers, where each register has a particular width; number of bits ok.

So, memories can be modeled as an array of register variables. Of course, this is a behavioral model, and we shall be looking into this and this is quite useful for simulation purposes, to check whether the rest of your design is correct or not. And also for memories which are relatively smaller in size, there also you can use this kind of behavioral model, but for larger memories as it said, you have to take some module from a library, design library fine.

(Refer Slide Time: 04:58)

The slide is titled "Typical Example" and contains two code snippets and a list of bullet points. The first code snippet shows a basic Verilog module declaration for a memory array. The second code snippet shows the same module with an initial block that initializes specific memory locations. The bullet points describe the memory word type and access notation.

```
module memory_model ( ..... )
...
  reg [7:0] mem [0:1023];
...
endmodule
```

```
module memory_model ( ..... )
  reg [7:0] mem [0:1023];
  initial begin
    mem[0] = 8'b01001101;
    mem[4] = 8'b00000000;
  end
endmodule
```

- Each memory word is of type [7:0], i.e. 8 bits.
- The memory words can be accessed as mem[0], mem[1], ..., mem[1023].

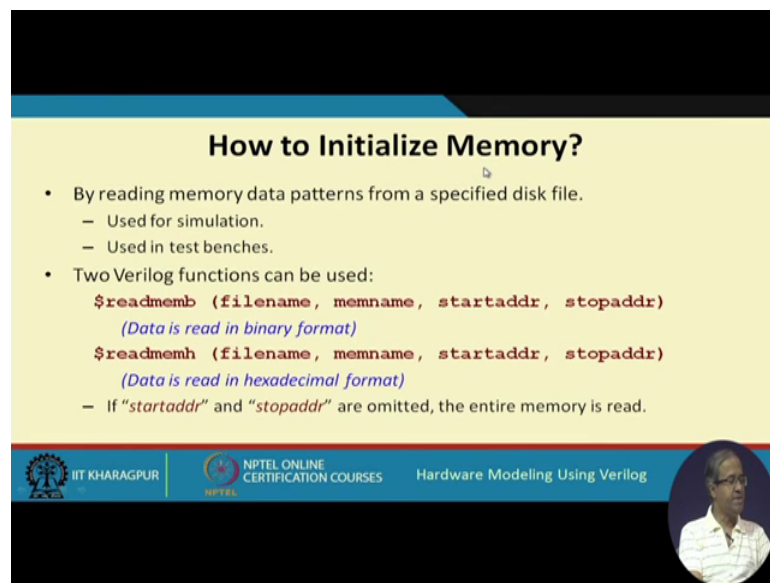
The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and the text "Hardware Modeling Using Verilog". A small circular portrait of a man is visible in the bottom right corner.

So, a very simple example; so a memory module can be declared as follows: so it is a registered array; reg 7 2 0 mem 0 1 2 2 3 means, there are 1 0 2 4 memory words, and each of the word is an 8 bit vector. So, each memory word will be 8 bits, and to access the individual memory words there are 1024 of them, we will have to, just access them using the array notation mem 0 mem 1 up to mem 1023, right

So, just 1 very simple example how we can initialize some memory content, some particular addresses in memory in some data: so this is the same declaration. Well in an initial block just an example. So, I have written mem 0 equal to. This will actually be a bit, this b has this thing. So, this is an 8 bit number 0 1 0 0 1 1 0 0. This is stored in

memory address 0, and mem 4 means this all 0 pattern, this is stored in memory location 4. So, this is how we can initialize a memory word inside a verilog module. Similarly you can access it as follows; mem 0 mem 1 mem 2 on the right hand side, same way talking about initializing.

(Refer Slide Time: 06:49)



How to Initialize Memory?

- By reading memory data patterns from a specified disk file.
 - Used for simulation.
 - Used in test benches.
- Two Verilog functions can be used:
 - `$readmemb (filename, memname, startaddr, stopaddr)`
(Data is read in binary format)
 - `$readmemh (filename, memname, startaddr, stopaddr)`
(Data is read in hexadecimal format)
 - If “startaddr” and “stopaddr” are omitted, the entire memory is read.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Memory in general, there are some other methods available that are quite useful for simulation, like you can read some data patterns from a file in the disk. Like suppose I have a large memory, whether or a large number of memory locations just writing the way I just showed, just assigning values to the different addresses, that way your code will become very long if there are thousand memory locations, there will be thousand lines of the code

But as an alternative what you can do. You can store the data that you want to load in the memory in a file beforehand, and then there is a function called using that function you can load that entire content from that file into that defined 2-dimensional memory array. So, there are 2 functions available, and these functions are useful for simulation, particularly in the test benches. These 2 functions are read memory binary and read memory hexadecimal.


So, in the first function, the data will be stored in the files in binary format, in 0 1 format, while in the second 1 they will be stored in hexadecimal format. So, you see there are 4 parameters; first 1 is the name of the file from where you have to read, then the name of

the array memory where you want to store, and start address and stop address they indicate, that well I mean you may not be wanting to initialize the whole of the memory, may be only a part of the memory. So, the starting and the ending address you can specify as a third and 4th arguments. Now if you omit this third and 4th argument, then the entire memory will be read, right.

(Refer Slide Time: 08:46)

Example 1: Initializing a memory from file

<pre>module memory_model (.....); reg [7:0] mem[0:1023]; initial begin \$readmemh ("mem.dat", mem); end endmodule</pre>	<pre>module memory_model (.....); reg [7:0] mem[0:1023]; initial begin \$readmemb ("mem.dat", mem, 200, 50); end endmodule</pre>
---	---

 Hardware Modeling Using Verilog

So, just a couple of examples; so here we had saying read memory in hexadecimal from this file into this variable mem, mem is the memory since we have not specified start and end. So, here we will store the entire contents of the 1024 locations from the file.

Here you see, there is another example where we have specified start and end, but start is greater than end, start is 200 end is 50. So, here memory will be initialized by reading from the file, starting from address 200 onward; 200 then 199 then 198 197, down up to 50, but you can specify the other way round also 50, 200 you can also write. In that case it will start from address 500 and go up up to 200, right. So, this is how we can initialize a memory define memory from data stored in a file, ok.

(Refer Slide Time: 09:56)

Example 2: Single-port RAM with synchronous read/write

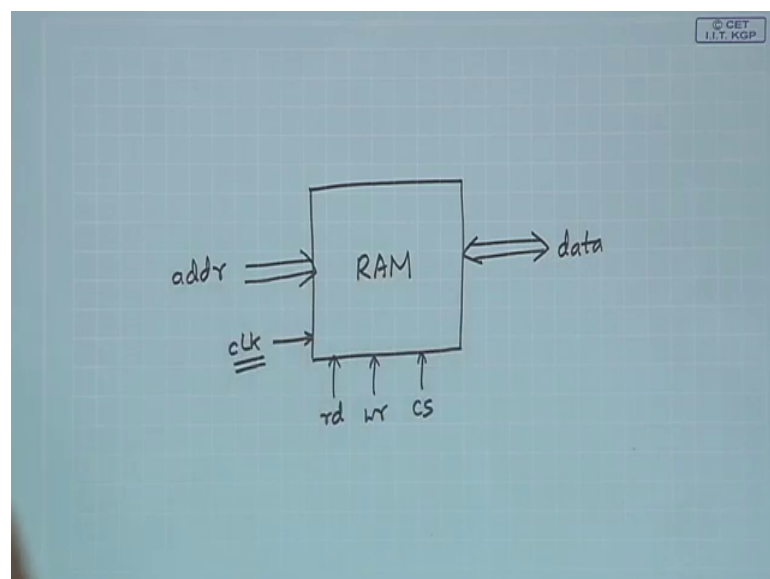
```
module ram_1 (addr, data, clk, rd, wr, cs);
input [9:0] addr;    input clk, rd, wr, cs;
inout [7:0] data;
reg [7:0] mem [1023:0];    reg [7:0] d_out;

assign data = (cs && rd) ? d_out : 8'bz;
always @(posedge clk)
    if (cs && wr && !rd) mem[addr] = data;
always @(posedge clk)
    if (cs && rd && !wr) d_out = mem[addr];
endmodule
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Here there is some examples. This is an example module declaration where we have used a single port ram with synchronous read write. So, what does this mean? This means we are trying to design.

(Refer Slide Time: 10:13)



A random access memory which can be read and written both; so here I am assuming that there are some address lines, which we call as a D D R. there are some data lines which we are assuming to be bidirectional; that means, I can either right into the ram or read from the ram over the same lines.

Now, in addition there are some control signals; like there is a read signal, there is a write signal, there is a chip select signal. And of course, there will be a clock signal because it is synchronous, it has to be a clock also. Reading and writing has to be in synchronism with the clock.

Now, here the memories the data bus is bidirectional. It can be read and also it can be written; that is why you see in the declaration this data has been declared as inout. Well this is a data type which you have not seen so far, the main reason is that it is best avoided because inout the way it is handled by the simulators and the synthesizers is not consistent. It varies quite significantly from 1 system to the other. So, it is a good suggestion to the designer, you should avoid inout where possible

But in this example I am showing how inout can be used, you see inout is by default of type wire. So, this is my memory. So, again 1 kilobyte 1024 into 8, and I am using a variable dout where I want to write it. And from dout to the memory I am using an assign statement, this assign data, data bus. So, whenever I chip select is there and read is there. So, from dout, the data will come to this data bus data and if they are not enabled then data will be tristated 8 z ok.

And it is synchronous operation in term of read write, posedge clock, if you are selecting the chip write is active, but not read; that means, you are doing to write, data is written into mem a D D R and if it is read and not write, then you are reading into dout.

Now, you see you cannot directly read into data, because I said this inout by definition is a wire, and inside a procedural block you cannot have a wire variable on the left hand side; that is why I defined the temporary reg variable dout assigned it there, and in an assign statement from dout I put it back to data right.

So, this is 1 way of specifying a single port ram with synchronous read and write. With slight modification you can make asynchronous read and write. Like here you see say in the earlier design, you are using the positive edge of the clock for doing the reading and writing, but for.


(Refer Slide Time: 13:40)

Example 3: Single-port RAM with asynchronous read/write

```
module ram_2 (addr, data, rd, wr, cs);
  input [9:0] addr;   input rd, wr, cs;
  inout [7:0] data;
  reg [7:0] mem[1023:0];   reg [7:0] d_out;

  assign data = (cs && rd) ? d_out : 8'bz;
  always @(addr or data or rd or wr or cs)
    if (cs && wr && !rd) mem[addr] = data;
  always @(addr or rd or wr or cs)
    if (cs && rd && !wr) d_out = mem[addr];
endmodule
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Asynchronous here we are not using clock the clock signal is not there anymore, only address data read write and chip select. So, the other declarations are similar.

Now, you see here the 2 always blocks are activated by change in any 1 of the input variables; either address changes or data changes or read write chip select, the rest is the same. So, synchronous and asynchronous both memory model definitions are very similar. So, in 1 case we are using the clock edge to synchronize my reading and writing, and in the other case there is no clock, whenever the signals changes. So, in accordance to that we are doing either reading or writing depending on whether read is active or right is active fine.

Now, as it said this inout is a not a very good way of using it in a module. We shall see it a little later, but first let us see.

(Refer Slide Time: 14:46)

Example 4: A ROM / EPROM

```
module rom (addr, data, rd_en, cs);
input [2:0] addr; input rd_en, cs;
output reg [7:0] data;
always @(addr or rd_en or cs)
  case (addr)
    0: data = 22;
    1: data = 45;
    .....
    7: data = 12;
  endcase
endmodule
```

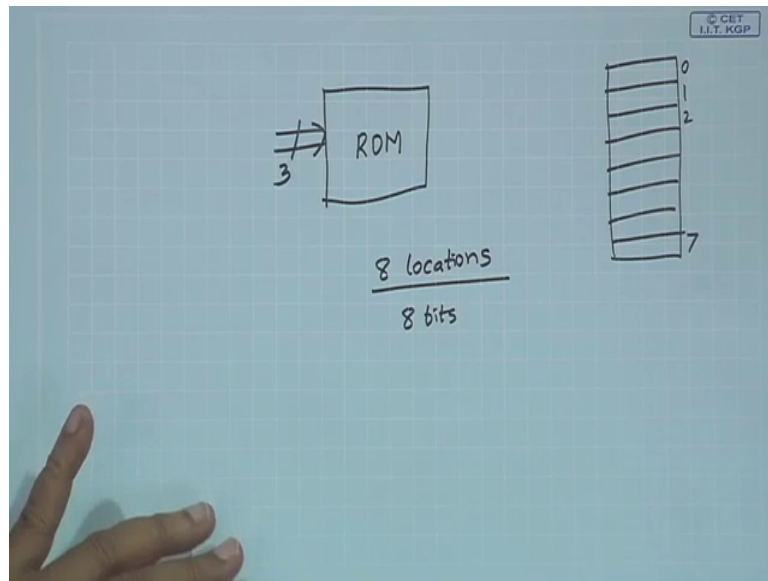
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, how we can model a memory where you are not writing only reading; that means, it can be either be a read only memory, or it can be a erasable programmable read only memory EPROM.

So, in A ROM or EPROM, we are assuming that we are storing some data in the memory that is fixed. We can only read from their. So, for A ROM when you model A ROM or EPROM, you need not use that kind of an array notation where you are storing and reading like that, rather you can use a case statement. So, here we have used a very small example.

Let us assume that my address is 3 bits. So, here we have taken a very small example.

(Refer Slide Time: 15:37)

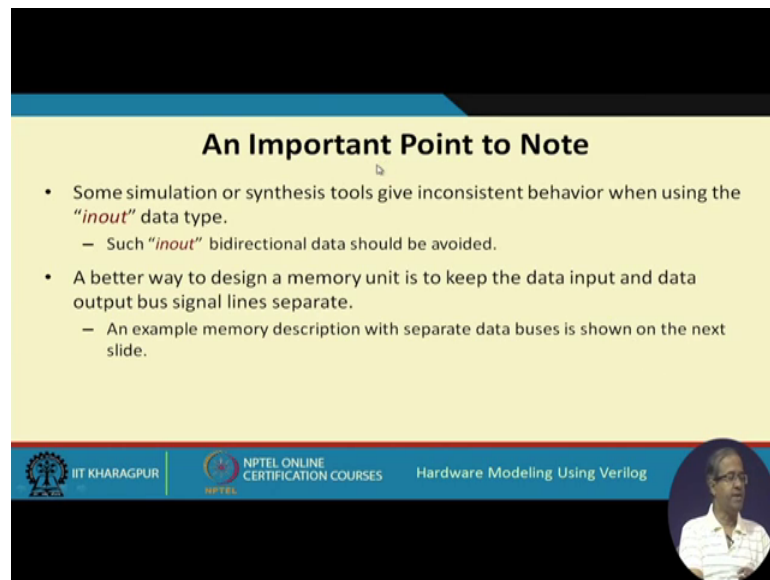


Let us say I have A ROM the address lines are 3, 3 bit address. So, inside the rom there will be 8 locations, and each locations I am assuming they are 8 bits in size. So, what I do, I just specify like a lookup table, there will be 8 locations. So, I store the 8 bit values, and depending on the address. So, 1 of these will be selected, address 0 1 2 up to 7 right. This is done by using a case statement.

So, you see, here there is a always block, this is activated when either address or there is a signal called read enable, because from A ROM you can only read you cannot write and chip select. So, when, means any of them these are active then you start the reading. There is a case a D D R. The address can be 0 1 up to 7 and you have stored some fixed data inside.

So, if you have given address equal to 0 data equal to 22, if it is 1 data equal to 45 and so on. There will be 8 lines like this. So, this is how you typically model A ROM or EPROM, where you do not change the data, data is fixed; that is why you are using a fixed case statement, right fine.


(Refer Slide Time: 17:12)



An Important Point to Note

- Some simulation or synthesis tools give inconsistent behavior when using the *"inout"* data type.
 - Such *"inout"* bidirectional data should be avoided.
- A better way to design a memory unit is to keep the data input and data output bus signal lines separate.
 - An example memory description with separate data buses is shown on the next slide.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

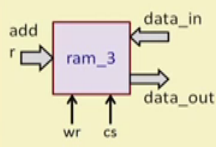


Now, the point that I was making this inout is a data type which is best avoided, because is it said some simulation or synthesis tools can give inconsistent behaviour. So, the experience designer says that do not use inout data type inside a verilog code, but does that mean that I will not use any bidirectional lines in my hardware. You see there are many instances where I do need bidirectional lines, memory is a very good example. So, I will use bidirectional lines, but I will not use inout data type in my verilog module in my verilog code, let us see how.

So, what we do. We keep the data input and data output lines separate, and then we will see how you can make it bidirectional. First let us do it in 1 step; first step says data input and output lines are kept separate.

(Refer Slide Time: 18:17)


Example 4



```
module ram_3 (data_out, data_in, addr, wr, cs);
    parameter addr_size = 10, word_size = 8,
              memory_size = 1024;
    input [addr_size-1:0] addr;
    input [word_size-1:0] data_in;
    input wr, cs;
    output [word_size-1:0] data_out;
    reg [word_size-1:0] mem [memory_size-1:0];

    assign data_out = mem[addr];
    always @(wr or cs)
        if (wr) mem[addr] = data_in;
endmodule
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



So, how does it look like, it is something like this. So, I have a memory a ram. So, I have my address, data in is separate data out are separate, 2 set of data buses and read ok.

There is no separate read line, what is meant is, right is like a read write both if right signal is high it means right. If right signal is low it means read. So, it is like a write slash read bar. So, I am just assuming that this kind of a signal is there.

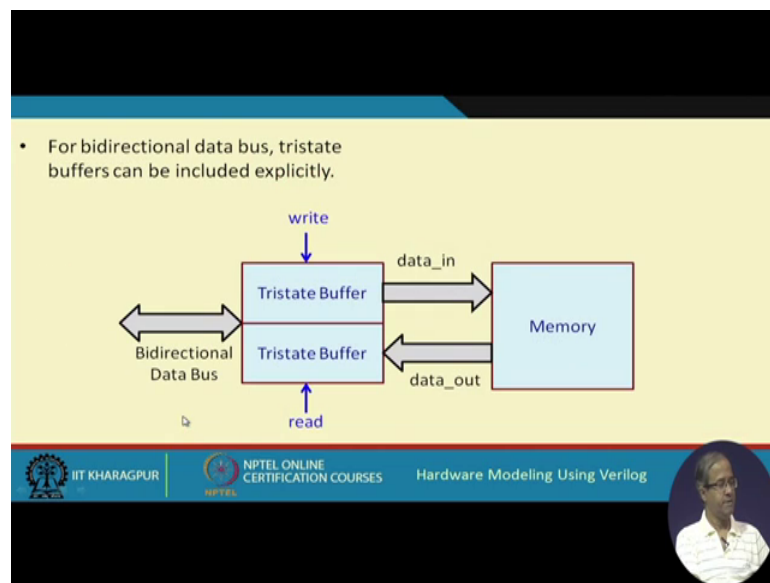
So, let us see how the module definition will look like. So, here we have used parameters to make our specification general. So, we have assumed address size is 10, it is a 10 bit address which means 1 k words; word size is 8 means 1 byte per location. And of course, memory size will be 1024×2^{10} .

So, the parameters are data out data in, these are the arguments address write and chip select. So, we have declared the variables; the address is of size 0 up to address size minus 1 10, data in, data in its an input. Again 0 up to word size minus 1 word size is 8. And similarly for the output, data out is an output that is it is an output. So, here, the size is again word size minus 1 8, and write and chips let the single bit inputs. And here we have defined our memory. this is number of bits per word 0 up to word size minus 1, and this is the total number of memory locations; it is 1024×2^{10} .

Now, the way we have implemented is very simple, writing we are doing in the always block. So, whenever right or chip select. That means, this changes, then inside it you

check if right equal to 1 then only you write mem a D D R equal to data in. Since mem is of type reg, there is no problem you can write it, but for reading the data we are not using always block. We are using an assign statement, because data out is declared just an output which is wire, it is not reg. So, mem a D D R equal to data out. So, from data out it will always be reading out, but whenever there is a right active, data in will be written inside right. This is the specification of the memory module here.

(Refer Slide Time: 21:13)



Now, let us see how we can make it bidirectional. You see this much we have implemented. We have implemented a memory module with separate data out and data in. Now what you do, we use 2 sets of tristate buffers. So, 1 we activate by write, other we activate by read. Now in the previous example there was no separate read signal. So, we can activate it by write bar; so when write is 0, and on the other side the 2 tristate buffers are connected together, and we have a single bus. So, you enable either this or this, both of them are not enabled together

So, when you are activating write, this is off, and whatever is coming here this will go through this into the data in, and whenever you are activating read this is off this is on. So, whatever is coming out of data out; that will be coming by this ok.

So, now you see in the module you are not declaring anything as inout, but by using tristate buffer you have implemented a bidirectional bus, sometimes you are moving here, how you are moving here. So, how you can declare you can declare it very simply

like this. Let us say this variable, let us call it bus here we defined it as tri, this is a tristate data out and data in are of type wires let us say, and we have 2 assign statements assign bus equal to, when read is active. When read is active data out will go to bus data out will go to bus, and otherwise it is tristate, this buffer is tristate.

And in the other assign statement data in equal to, depending on write bus, bus will go in or if write is not active then tristate. So, just a normal memory module with separate input and output data, and these kinds of tristate bus and enabling them will make the overall thing a bidirectional data bus enabled fine.

Now, just a simple test bench we are writing just to test this ram_3 module which we just wrote. So, we are just.

(Refer Slide Time: 23:41)

```
module RAM_test;
  reg [9:0] address;
  wire [7:0] data_out;
  reg [7:0] data_in;
  reg write, select;
  integer k, myseed;

  ram_3 RAM (data_out, data_in, address, write, select);

  initial
  begin
    for (k=0; k<=1023; k=k+1)
      begin
        data = (k + k) % 256; read = 0; write = 1; select = 1;
        #2 write = 0; select = 0;
      end
  end
end
```

The slide is titled "Simple Test Bench using ram_3". It contains Verilog code for a test bench. The code defines a module with several signals: a 10-bit register for 'address', a 7-bit wire for 'data_out', a 7-bit register for 'data_in', and two 1-bit registers for 'write' and 'select'. It also declares two integer variables, 'k' and 'myseed'. The main logic is in an 'initial' block with a 'begin' statement. Inside, there is a 'for' loop that iterates from k=0 to k=1023. In each iteration, a 'begin' block contains an assignment: 'data = (k + k) % 256; read = 0; write = 1; select = 1;'. This is followed by a delay of 2 time units: '#2 write = 0; select = 0;'. The 'end' statement concludes the 'begin' block, and another 'end' statement concludes the 'initial' block.

At the bottom of the slide, there are logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and the text "Hardware Modeling Using Verilog". A small circular portrait of a man is visible in the bottom right corner.

Showing a simple test bench; so these are the 2 parts of the test bench, let us see. Here we have instantiated the memory data out, data in, address, write and select.

So, here this is wire there is a type o wire. So, this address is of type reg, data out which will be coming out of the ram, this is of wire, data in will also go inside. So, that has to be of type reg, write and select you have to activate, chip select and write those are also type reg, and we have defined 2 integers in addition k and myseed for the purpose of writing the test bench.

Now, here you see what we do, in the first initial block we initialize the whole memory. How you do in a for loop, k equal to 0 up to k equal to 1023, we go in a loop and what you store, we store k plus k modulo 256. So, if the address, if k is 0 plus 0 is 0. So, if k is 1, 1 plus 1 is 2. If k is 10, 10 plus 10 is 20. If k is 100, 100 plus 100 is 200. If k is 200, 200 plus 200 is 400 modulo 256 divide by 256 and take the remainder. So, it will be 144 like that data will get stored. This is the initialization part.

So, when u store something in the data you activate read equal to 0 write 1 and select 1 so that this data will get written into the corresponding memory location. So, this gets written.

(Refer Slide Time: 25:56)

```

repeat (20)
  begin
    #2 address = $random(myseed) % 1024;
    write = 0; select = 1;
    $display ("Address: %5d, Data: %4d", address,
             data);
  end
end
initial myseed = 35;
endmodule

```

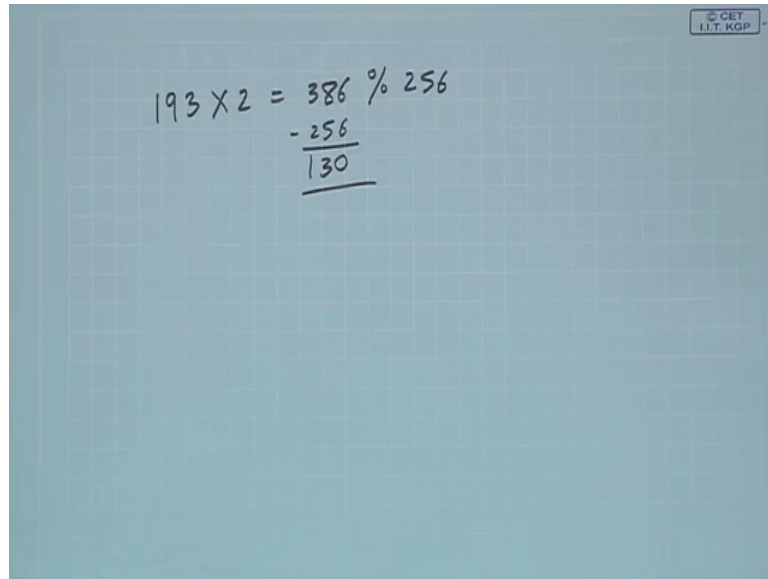
Address: 0, Data: 0
Address: 960, Data: 128
Address: 482, Data: 196
Address: 693, Data: 106
Address: 246, Data: 236
Address: 228, Data: 200
Address: 148, Data: 40
Address: 767, Data: 254
Address: 355, Data: 198
Address: 259, Data: 6
Address: 21, Data: 42
Address: 872, Data: 208
Address: 758, Data: 236
Address: 193, Data: 130
Address: 909, Data: 26
Address: 632, Data: 240
Address: 719, Data: 158
Address: 214, Data: 172
Address: 67, Data: 134
Address: 908, Data: 24

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, after this, now here you repeat 20 times, just I am randomly selecting 20 addresses, and I am reading them out and seeing what is there. So, I am just calling the random number with the seed mod 1024. So, an address in the range 0 up to 1023 will be generated; that is my address, write is 0; that means, I want to read select is 1, I display address data, so whatever is coming out

So, if I simulate it I see that my output is coming like this. So, repeat 20, there will be a 20 lines generated, this will be the random addresses, that the smaller 1 you check address is 21. So, what does address 21 contain k plus k. So, which is 42; so the others 193. So, 193 is how much, 193 multiplied by 2, just let us check 193.

(Refer Slide Time: 27:02)



© CET
I.I.T. KGP

$$193 \times 2 = 386 \% 256$$
$$\begin{array}{r} 386 \\ - 256 \\ \hline 130 \end{array}$$

Multiplied by 2 is 386. Now 386 mod 256 means what, divide by 256 and take the remainder. So, basically minus 256 which means 130; so you see in location here 193 data is 130. So, accordingly you can verify the operation.

Now, here there is 1 mistake that I see which I have missed out, here I have not put the address. We will have to put another line here. Here you will have to put address equal to k, this is missing, because where you will be writing. We will be writing it into address k right, data you are calculating, you are storing it into data right, and this write and select are activated, this will be data in actually, data in fine. So, you have declared it as data in. So, data in will be getting this data address, address will be k and then you write 1 select 1 it will be written, like this.

So, with this we come to the end of this lecture. So, in this lecture we have basically seen how we can model some memory devices as a 2 dimensional array, as an array of registers and at least from the simulation point of view, wherever we need memory elements in a larger design you can implement the memory systems like this. So, we shall see some examples later also, where you use these kind of memories in a larger system, and you can simulate them and see how it works.

And again I had said for smaller memory systems, even the synthesis tool will be able to synthesize the memory, but it will not be very efficient in terms of hardware. They will be implementing the, synthesis tool will be implementing the hardware in terms of flip

flops and arrays of flip flops, but in actual memory chip, the weight is constructed, it is much more compact, much more efficient in terms of number of transistors required per cell, per storage cell, ok.

Thank you.