Digital Design with Verilog

Verilog

Lecture 12: Modeling Memory

Learning Objectives

- Modeling of
  - Read-Only Memory in Verilog
  - Read-Write Memory in Verilog

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.

  - Primarily used for simulation purpose

  - Sometime for the synthesis of small-sized memories.

Read-Only Memory in Verilog

- One of the way is to use a case statement to define the contents of each location in memory based on the incoming address.

```
module rom_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address);

 always @ (address)
   case (address)
     0       : data_out = 4'b1110;
     1       : data_out = 4'b0010;
     2       : data_out = 4'b1111;
     3       : data_out = 4'b0100;
     default : data_out = 4'bXXXX;
   endcase

endmodule
```

Behavioural Models of a 4x4 Asynchronous ROM

Read-Only Memory in Verilog

- A second approach is to declare an array and then initialize its contents.

```
module rom_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address);

  reg[3:0] ROM[0:3];

  initial
    begin
      ROM[0] = 4'b1110;
      ROM[1] = 4'b0010;
      ROM[2] = 4'b1111;
      ROM[3] = 4'b0100;
    end

  always @ (address)
    data_out = ROM[address];

endmodule
```

Read-Only Memory in Verilog

```
module rom_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        Clock);

 always @ (posedge Clock)
   case (address)
     0       : data_out = 4'b1110;
     1       : data_out = 4'b0010;
     2       : data_out = 4'b1111;
     3       : data_out = 4'b0100;
     default : data_out = 4'bXXXX;
   endcase

endmodule
```

Behavioural Models of a 4x4 synchronous ROM

Read-Only Memory in Verilog

```
module rom_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        Clock);

  reg[3:0] ROM[0:3];

  initial
    begin
      ROM[0] = 4'b1110;
      ROM[1] = 4'b0010;
      ROM[2] = 4'b1111;
      ROM[3] = 4'b0100;
    end

  always @ (posedge Clock)
    data_out = ROM[address];

endmodule
```

Behavioural Models of a 4x4 synchronous ROM

$readmem(h|b)

Verilog allows you to initialize memory from a text file with either hex or binary values:

- `$readmemh("hex_memory_file.mem", memory_array, [start_address], [end_address])`
- `$readmemb("bin_memory_file.mem", memory_array, [start_address], [end_address])`

Function Arguments

- `hex_memory_file.mem` - a text file containing hex values separated by whitespace (see below)
- `bin_memory_file.mem` - a text file containing binary values separated by whitespace (see below)
- `memory_array` - the name of Verilog memory array of the form: `reg [n:0] memory_array [0:m]`
- `start_address` - where in the memory array to start loading data (optional)
- `end_address` - where in the memory array to stop loading data (optional)

https://timetoexplore.net

$readmem(h|b)

Four 16-bit data values in hex:

Verilog:

```
reg [15:0] ex1_memory [0:3];
$readmemh("ex1.mem", ex1_memory);
```

Memory file contents:

```
dead
beef
0a0a
1234
```

https://timetoexplore.net

$readmem(h|b)

Sixteen 8-bit data values in hex (mixing spaces and newlines):

Verilog:

```
reg [7:0] ex2_memory [0:15];
$readmemh("ex2.mem", ex2_memory);
```

Memory file contents (including comment):

```
ab cd ef 01  // this is a comment
ef 22 1e 00
9f ff 13 e6
ce b7 28 8f
```

https://timetoexplore.net

$readmem(h|b)

Six 3-bit values in binary:

Verilog:

```
reg [2:0] ex3_memory [0:5];
$readmemb("ex3.mem", ex3_memory);
```

Memory file contents:

```
001 101 111 111 101 001
```

https://timetoexplore.net

$readmem(h|b)

Six 16-bit values in hex starting at array position 4:

Verilog:

```
reg [15:0] ex4_memory [0:255];
$readmemh("ex4.mem", ex4_memory, 4);
```

Memory file contents:

```
dead beef 0a0a 1234 abab 987e
```

https://timetoexplore.net

Read/Write Memory in Verilog

- In a simple read/write memory model, there is an output port that provides data when reading (data_out) and an input port that receives data when writing (data_in).

- Within the module, an array signal is declared with elements of type reg.

- To write to the array, signal assignment are made from the data_in port to the element within the array corresponding to the incoming address.

Read/Write Memory in Verilog

- To read from the array, the data_out port is assigned the element within the array corresponding to the incoming address.

- A write enable (WE) signal tells the system when to write to the array (WE == 1) or when to read from the array (WE == 0).

Read/Write Memory in Verilog

```
module rw_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        WE,
    input  wire [3:0]  data_in);

  reg[3:0] RW[0:3];

  always @ (address or WE or data_in)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

Behavioural Model of a 4x4 Asynchronous R/W Memory

Read/Write Memory in Verilog

```
module rw_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        WE,
    input  wire [3:0]  data_in,
    input  wire        Clock);

  reg[3:0] RW[0:3];

  always @ (posedge Clock)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

Behavioural Model of a 4x4 synchronous R/W Memory

Read/Write Memory in Verilog

```
module ram_sp_sr_sw (
        clk, address, data, cs, we, oe );
        parameter DATA_WIDTH = 8 ;
        parameter ADDR_WIDTH = 8 ;
        parameter RAM_DEPTH = 1 << ADDR_WIDTH;
        //--------------Input Ports-----------------------
        input                   clk         ;
        input [ADDR_WIDTH-1:0] address     ;
        input                   cs          ;
        input                   we          ;
        input                   oe          ;
        //--------------Inout Ports-----------------------
        inout [DATA_WIDTH-1:0]  data        ;
        //--------------Internal variables----------------
        reg [DATA_WIDTH-1:0] data_out ;
        reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1];
        //--------------Code Starts Here------------------
        // output : When we = 0, oe = 1, cs = 1
        assign data = (cs && oe && !we) ? data_out : 8'bz;
        // Memory Write Block
        // Write Operation : When we = 1, cs = 1
        always @ (posedge clk)
        begin : MEM_WRITE
           if ( cs && we ) begin
               mem[address] = data;
           end
        end
        // Memory Read Block
        // Read Operation : When we = 0, oe = 1, cs = 1
        always @ (posedge clk)
        begin : MEM_READ
          if (cs && !we && oe) begin
            data_out = mem[address];
          end
        end
endmodule
```

References

- Chapter 10, Introduction to Logic Circuits & Logic Design with Verilog by Brock J. LaMeres

- Disclaimer: “I don’t claim the ownership of all the slides, some of the material is picked up from various publicly available sources on the internet”.

Thank you