# VERILOG

## TIMING CHECKS AND USEFUL SYSTEM TASKS



ROHIT KHANNA

# Timing Checks

- Timing check statements are placed in specify blocks to verify timing performance of a design.
- Although all timing check begins with $ sign they are not categorized as system tasks.
- No system task is allowed inside specify block.
- No timing check statement can be used outside specify block.

# Timing Checks

- Timing check performs the following operations
  - Records occurrence time of a data or reference event.
  - Waits for occurrence second data or reference event.
  - Compares the elapsed time to the specified limit.
  - Reports violation in design if any.

- Verilog supports following timing checks

| | | | |
|---|---|---|---|
| $setup | $recovery | $skew | $width |
| $hold | $removal | $timeskew | $period |
| $setuphold | $recrem | $fullskew | $nochange |

# $setup

Syntax:

$setup(data_event, reference_event, timing_check_limit);

- ❖data event is a data signal.
- ❖reference event is a clock signal.
- ❖timing check limit is setup time.

- Violation is reported if

$T_{reference\ event} - T_{data\ event} < timing\ check\ limit$

Example:

$setup(d, posedge clk, 5);

# $hold

Syntax:

$hold(reference_event, data_event, timing_check_limit);

- reference event is a clock signal.
- data event is a data signal.
- timing check limit is hold time.

- Violation is reported if

$T_{data\ event} - T_{reference\ event} <= timing\ check\ limit$

Example:

$hold(posedge clk, d, 5);

# $setuphold

- It combines functionality of $setup and $hold into a single timing check

Syntax:

$setuphold(reference_event, data_event, setup timing_check_limit, hold timing check limit );

- Violation is reported if

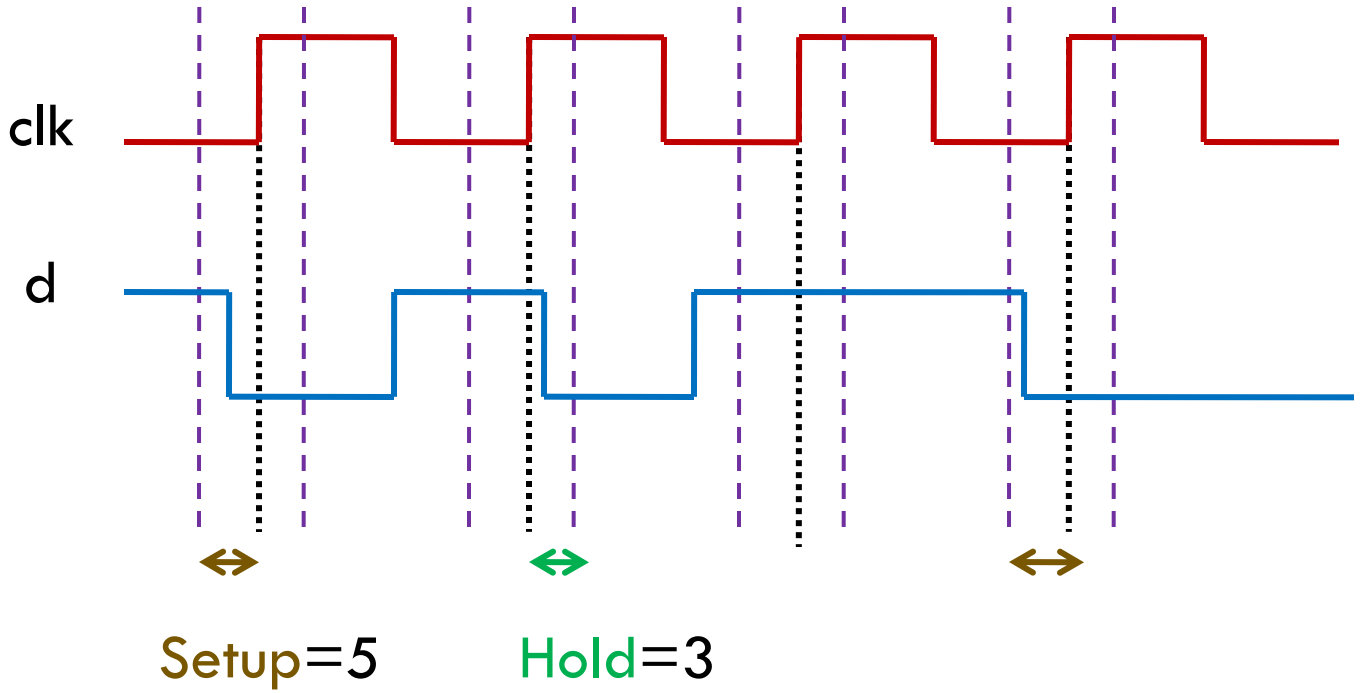
$T_{reference\ event} - T_{data\ event} <$ setup timing check limit

$T_{data\ event} - T_{reference\ event} <=$ hold timing check limit

Example: $setuphold(posedge clk, d, 5, 3);

# Waveform

$setuphold (posedge clk, d, 5, 3);



# Example

```
module dff (din, dout, clk);
input din, clk;
output reg dout;

always @ (posedge clk)
dout<=din;

specify
$setup(din, posedge clk, 5);
$hold(posedge clk, din, 3)
endspecify

endmodule
```

# $recovery

Syntax:

$recovery(reference_event, data_event, timing_check_limit);

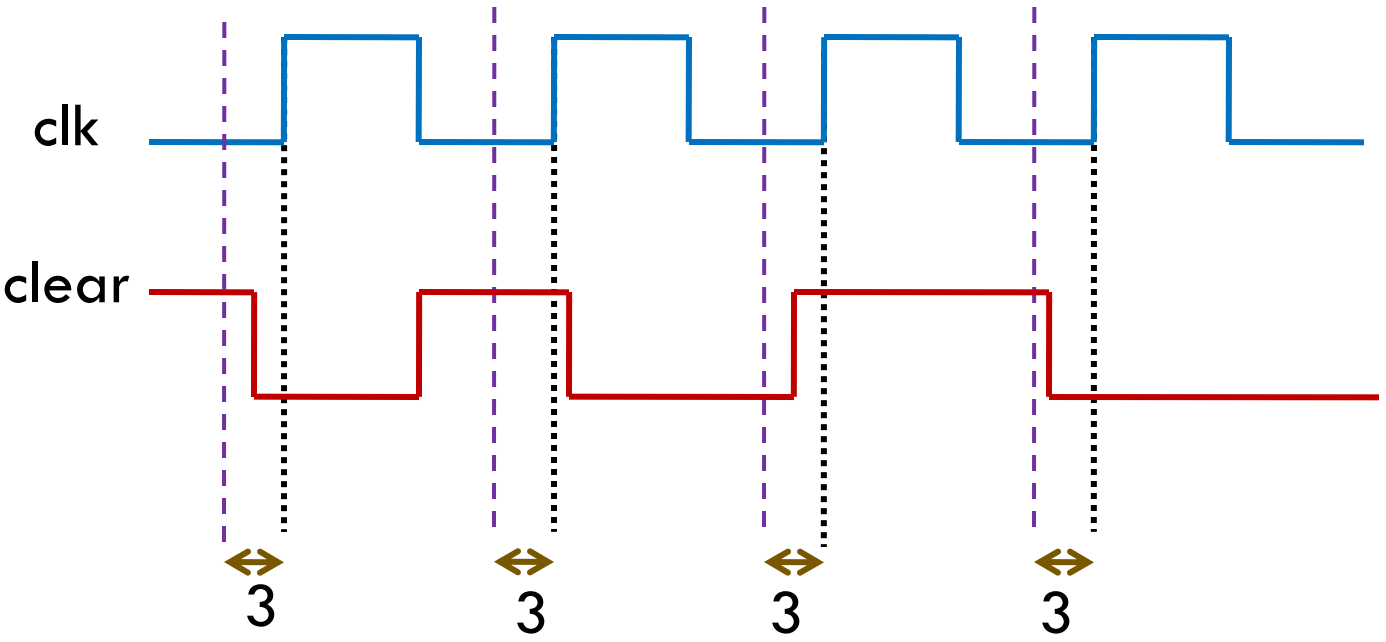
- reference event is usually a control signal like clear, reset, or set.
- data event is a usually a clock signal.
- timing check limit is non negative constant expression.

- Violation is reported if

$T_{data\ event} - T_{reference\ event} <= timing\ check$

# Waveform

$recovery(posedge clear, posedge clk, 3);




Recovery Violation occurs at 3rd rising edge of clock

# $removal

Syntax:

$removal(reference_event, data_event, timing_check_limit);

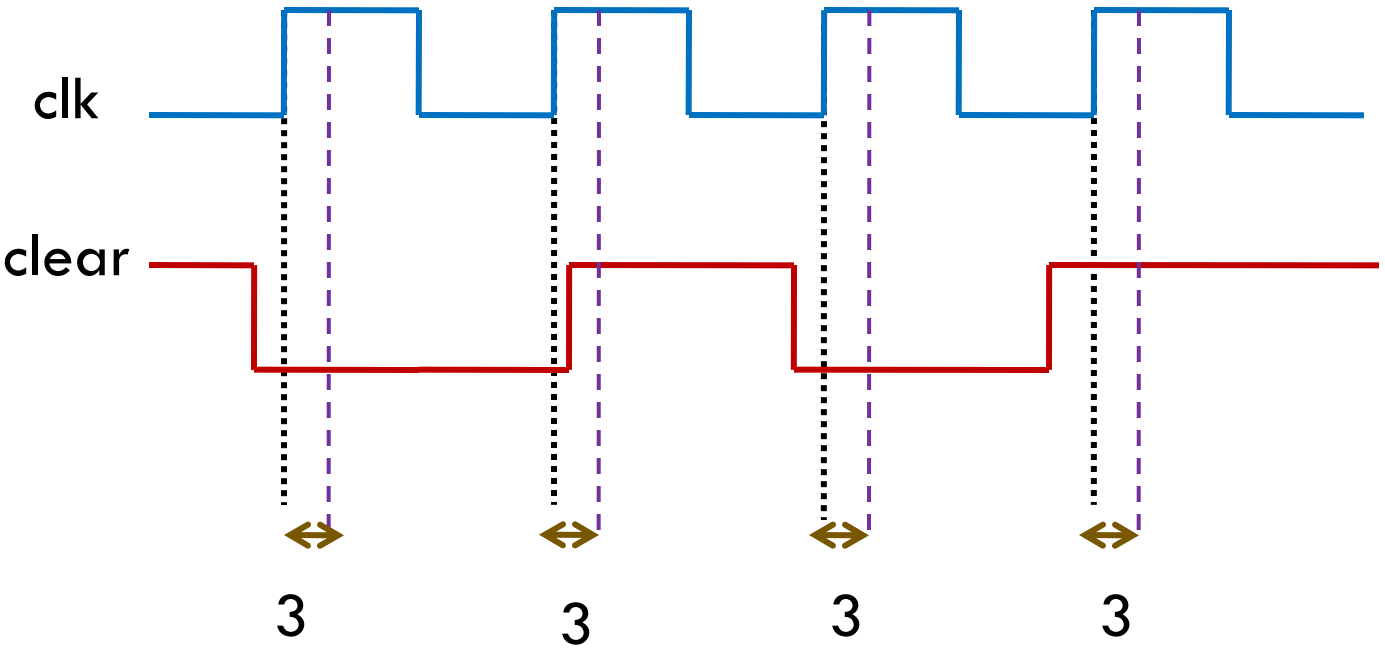
- reference event is usually a control signal like clear, reset, or set.
- data event is a usually a clock signal.
- timing check limit is non negative constant expression.

- Violation is reported if

$T_{reference\ event} - T_{data\ event} < timing\ check\ limit$

# Waveform

$removal(posedge clear, posedge clk, 3);




Removal Violation occurs at 2nd rising edge of clock

# $recream

- The $recrem timing check combines the functionality of the $removal and $recovery timing checks into a single timing check.

Syntax:

```
$recream(reference_event, data_event,
  recovery_timing_check_limit, removal_time_check_limit);
```

Example:

```
$recream (posedge clear, posedge clk, TREC, TREM);
//Equivalent to the following statements
$recovery(posedge clear, posedge clk, TREC);
$removal(posedge clear, posedge clk, TREM);
```

# Example

```
module dff (din, clr, dout, clk);
input din, clk, clr;
output reg dout;

always @ (posedge clk, posdege clr)
if (clr) dout<=0 else dout<=din;

specify
$recovery(posedge clr, posedge clk , 2);
$removal(negdge clr, posedge clk, 3)
endspecify

endmodule
```

# $skew

Syntax:

$skew(reference_event, data_event, timing_check_limit);

- reference event is signal.
- data event is skewed version of signal.
- timing check limit is the maximum allowed skew.

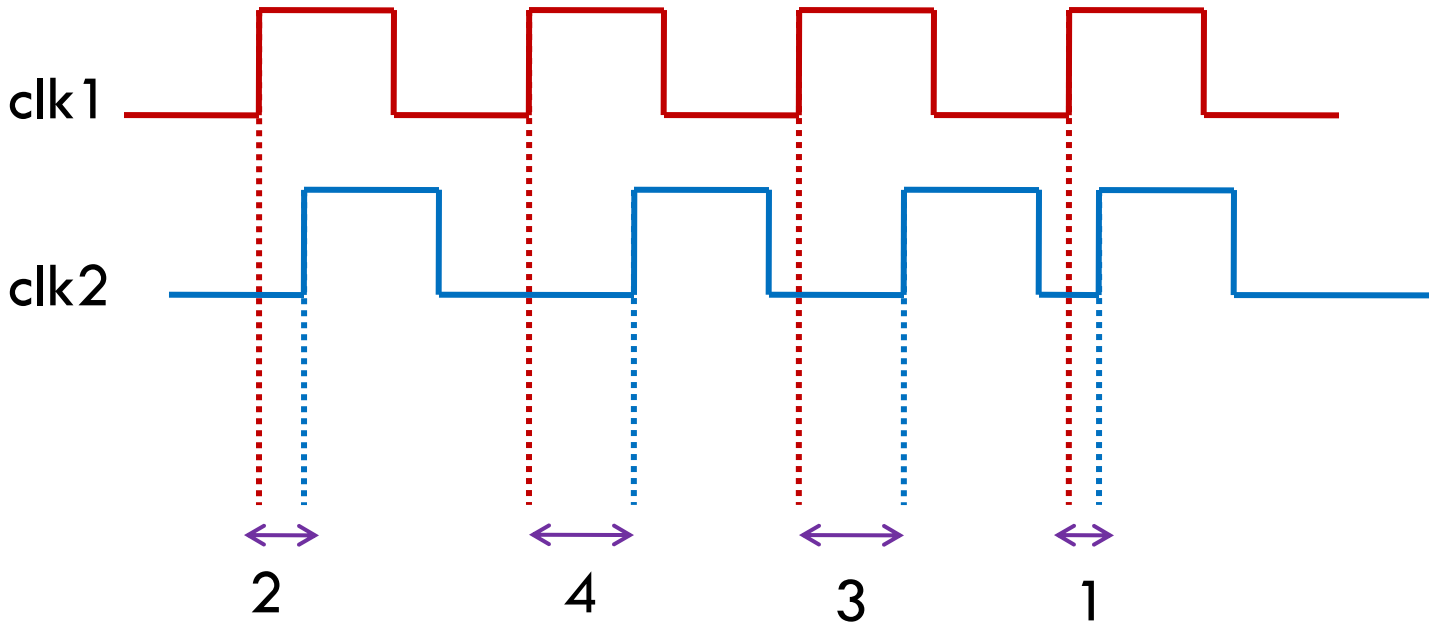
- Violation is reported if

  $T_{data\ event} - T_{reference\ event} > timing\ check\ limit$

- It checks that skew is not more than the specified limit with respect to a signal.

# Waveform

$skew (posedge clk1, posdege clk2, 3 );




Skew Violation occurs for skew=4

# $period

Syntax:

$period(reference_event, timing_check_limit);

- edge triggered event on a signal.
- data event is a not specified explicitly, it is derived as next same edge of reference event.
- timing check limit is time period of the signal.

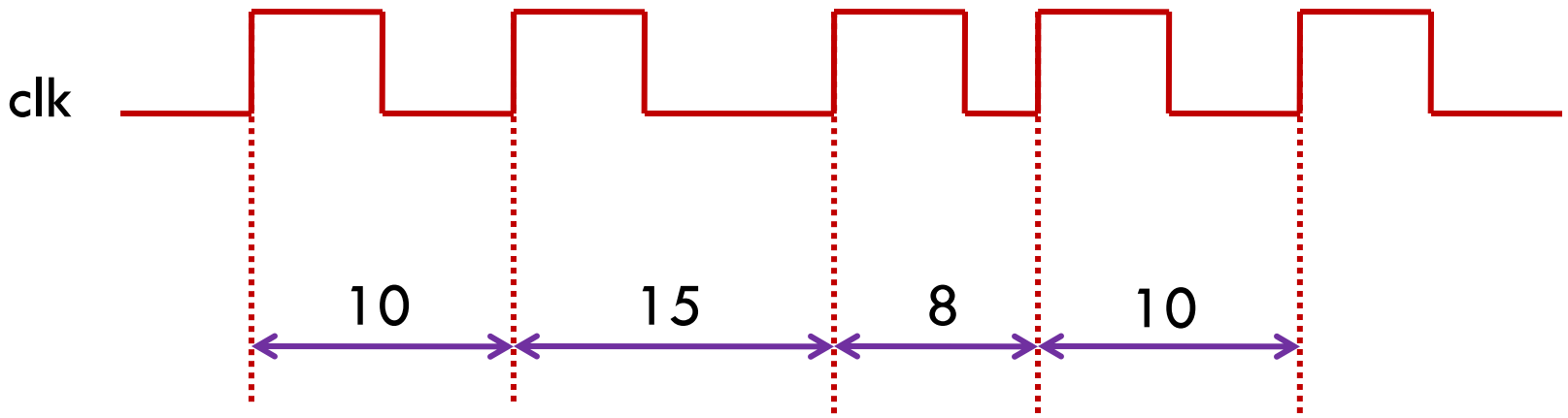
- Violation is reported if

$T_{data\ event} - T_{reference\ event} < timing\ check\ limit$

- It checks whether reference signal time period less than timing check limit.

# Waveform

$period (posedge clk, 10);




Period Violation occurs for period=8

# $width

Syntax:

$width(reference_event, timing_check_limit);

- edge triggered event on a signal.
- data event is a not specified explicitly, it is derived as next opposite edge of reference event.
- timing check limit is pulse width value.

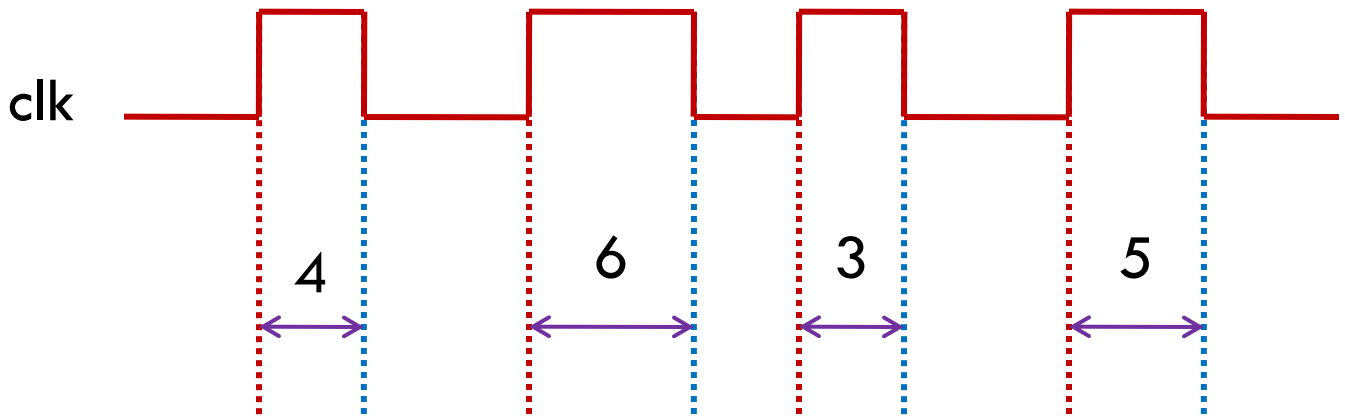
- Violation is reported if

$$T_{data\ event} - T_{reference\ event} < timing\ check\ limit$$

- It checks whether reference signal has width less than the specified timing check limit.

# Waveform

$width (posedge clk, 5);




Width Violation occurs for width=4, 3

# Random Number Generator

- Random numbers are used for providing random test inputs which helps in finding hidden bugs in the design.
- $random system task is used to generate random numbers

  Syntax : $random[(seed_value)];

- Seed value is used to ensure that same random number sequence is generated each time it runs. Seed parameter can be a reg, integer, or time.
- $random returns 32 bit signed integer.

# $random

Examples:

```
integer seed1=0, seed2=2, seq1, seq2;

initial
forever
begin
seq1=$random(seed1);      //seed is inout argument
seq2=$random(seed2);      // both gives different sequence
#10;
end
```

# $random

Examples:

```
integer seq1, seq2, seq3, seq4, seq5, s1, s2;

always begin
seq1=$random(s1)>>>1;                  //Arithmetic Right shift works
seq2=$unsigned($random(s2))>>>1; //Performs Logical Right shift
seq3=$random % 50;     //random number between -49 and 49
seq4=$unsigned($random) % 100; //numbers between 0 to 99
seq5={$random} % 35;  //random number between 0 to 34
#10;
end
```

# File Operations

- $fopen system task is used to open a file.

  Syntax : file_handler=$fopen("name of the file");

- $fopen returns a 32-bit value called multichannel descriptor.

- Only one bit of a multichannel descriptor is set to 1.

- Each bit of multichannel descriptor is used to represent different channel, except bit 31(reserved).

# File Operations

- Standard output (Transcript) has a multichannel descriptor with bit 0 set to 1(default condition).
- Standard output is also called as channel 0, which is open by default.
- Each successive $fopen call opens a new channel and return 32-bit multichannel descriptor with next MSB bit set to 1.
- Multichannel descriptor offers advantage of writing multiple files at same time.

# Example

```
// multichannel descriptor value=32'h0000_0001 (bit 0 set)
integer file1, file2, file3;

initial begin
 file1=$fopen("file1.txt");
 // file1= 32'h0000_0002 (bit 1 set)
 file2=$fopen("file2.out");
// file2= 32'h0000_0004 (bit 2 set)
 file3=$fopen("file3.c");
// file2= 32'h0000_0008 (bit 3 set)
 end
```

# File Operations

- Once a file is opened, it can be written with help of following system task:
  - $fwrite
  - $fdisplay
  - $fstrobe
  - $fmonitor

Syntax : $fmonitor(file_descriptor, p1, p2,…, pn);

- file descriptor specifies which files are to be written.
- p1, p2, …, pn can be variables, constants or strings.

# File Operations

- file descriptor can be a single file handler incase data has to be written to one file.

```
$fmonitor(file1, "Value of a=%d and b=%d", a, b);
```

- file descriptor can be bitwise operation of several file handlers, incase data has to be written to multiple files.

```
$fmonitor((file1 | file2), "Value of a=%d and b=%d", a, b);
```

- Once file has been written, it can be closed using $fclose system task .

Syntax: `$fclose(file_handler);`

# Example

```
integer filed1, filed2, filed3, filed4, filed5; // Five file descriptor

initial begin
filed1= file1 | 1;                          // 2 | 1 =>32'h0000_0003
$fdisplay(filed1, "Hello");
//writes to file1.txt and Transcript
filed2= file2 | file3;                      // 4 | 8  => 32'h0000_000C
$fdisplay(filed2, "World");
//writes to file2.out and file3.c
```

# Example

```
filed3= 8;                              // 4 =>32'h0000_0004
$fdisplay(filed3, "are");
//writes to file3.c
filed4= file1 |file2 | 1;               // 2 | 4 | 1=> 32'h0000_0007
$fdisplay(filed4, "you");
//writes to file1.txt, file2.out and transcript
filed5= file3 & 1;                      // 8 & 1=> 32'h0000_0000
$fdisplay(filed5, "Hello Again");
//does not write to any of the files
end
```

# Memory Initialization

- Verilog provides a system tasks to initialize memories from a file.
- $readmemb and $readmemh are systems tasks used to initialize memories.

Syntax:

```
$readmemb("file_name", memory_name);
$readmemb("file_name", memory_name, start_address);
$readmemb("file_name", memory_name, start_address,
                                    end_address);
```

- $readmemb or $readmemh are used if the values present in file are in binary or hexadecimal format respectively.

# Example

```
module example;
integer i;
reg [7:0] mem [0 : 7];    //8 x 8 Memory declaration
initial
$readmemb("file.txt", mem);

initial
for (i=0; i<=7; i= i + 1)
$display("Value at location %d is =%b", i, mem[i]);
endmodule
```

# Memory Initialization

- In initialization file, data is separated by whitespace.
- Data can contain X and Z.
- Optionally address can be specified with @address. here address has to be in hexadecimal format.
- If there are multiple entries for same address, the last entries will be considered.
- If there is conflict between (start address, end address) and address provided in initialization file, tool will report warning.

# Data File – Example1

```
//White spaces are used to separate data
//Data specified in binary ($readmemb)
1001_0110       // Data at Location 0
1011_0100       // Data at Location 1
1010_0001       // Data at Location 2
1111_0010       // Data at Location 3
0111_0011       // Data at Location 4
1101_1110       // Data at Location 5
1001_1101       // Data at Location 6
0101_0111       // Data at Location 7
```

# Data File – Example2

```
//Binary Data specified along with address($readmemb)

@2  1111_0110      // Data at Location 2
    1001_0100      // Data at Location 3
    1011_0001      // Data at Location 4

@6  0011_0010      // Data at Location 6
    1010_1111      // Data at Location 7

@0  1110_0111      // Data at Location 0
```

# Data File – Example3

```
//Data specified in Hex($readmemh)

a1      // Data at Location 0
12      // Data at Location 1
67      // Data at Location 2
ff      // Data at Location 3
3f      // Data at Location 4
97      // Data at Location 5
```

# Data File – Example4

```
//Hex Data specified along with address ($readmemh)

@2  f3      // Data at Location 2
    97      // Data at Location 3
    ab      // Data at Location 4

@6  12      // Data at Location 6
    53      // Data at Location 7
```