==========

ReadMe File

TurboBIST-Memory Usage Description of Tool Options

SynTest Technologies, Inc.

August 2006

==========

This package includes one demonstration version of SynTest Design-for-Test (DFT) tools, TurboBIST-Memory, for generating Built-In-Self-Test (BIST) circuitry for embedded memory of SRAM or ROM types such that the memories embedded within a circuit can be tested at-speed from the circuit's Primary Inputs/Outputs. The package also includes a sample circuit sram_1p_256x8 and required directories, files, and run scripts for users to try. Users are encouraged to study all materials before running the tool.

netsrc/	The original design netlist
test.bcf	BIST Configuration File that contains user-specified memory BIST algorithm
sram.mdf	Memory Definition File that describes memory and memory BIST specification
Run_MBIST	Run script for TurboBIST-Memory flow for the sample circuit.

The information below describes

A. the software modules (commands) and options used in the run script.
B. Instructions for creating .mdf file
C. Instructions for creating .bcf file

A. Available scambist Options

This version of srambist software module supports 3 options.

-algorithm <algorithm name>

Algorithm used in BIST. When using this option, users have to specify the algorithm name and the bcf file using –bcf_file.

-bcf_file <bcf file name>

BIST Configuration File used.

-scan_bist_results

Scan the result registers in the BIST controller. The result register includes Finish and BistFail signals.

B. Memory Description File (*mdf*)

All memories listed in ***mdf*** file will share one BIST controller. An ***mdf*** file contains %GLOBAL, %MEMORY_GROUP and %MEMORY blocks with the following syntax. %MEMORY further contains %PORT {} sub-section for each port.

```
%GLOBAL
{
        -----------------
}
%MEMORY_GROUP
{
        %GROUP -----;
        -------------
}

%MEMORY SRAM_NAME
{
        -----------------
}
```

1. The %Global Section

The %GLOBAL section specifies the memories or BIST global attributes. It has the following keywords for specification.

%TIMESCALE <string>

This item is used to define the simulation resolution in Verilog. It is optional, and if the option -vhdl is used, this item is ignored.
Ex. %TIMESCALE 1ns/10ps;

%CLK_CYCLE <number> ;

This item specifies the BIST clock cycle time in the testbench and synthesis script, which is an optional attribute. Its default value is set as 20. The clock cycle time should be the same as the memory clock; otherwise, the BIST might fail. If the memory is asynchronous, the clock cycle time will indicate the test clock cycle time.
Ex. %CLK_CYCLE 10;

2. The %Memory_Group Section

The %MEMORY_GROUP section describes how memories can be grouped and tested. The memories in one group will be tested in parallel, and different groups can be tested at separate times. Users can use the ***MemGroupSel*** signals to control the test sequence when testing different groups.

If the %MEMORY_GROUP section is omitted and there is more than one %MEMORY section, then all the memories will be tested at the same time, and no ***MemGroupSel*** signal is generated. If the users want ***srambist*** to generate the ***MemGroupSel*** signal, at least one %GROUP section has to be created.

%GROUP

%GROUP <memory 1 name>{hier_instance_name}, <memory 2 name>{} .....

The <hier_instance_name> is the memory hierarchical instance name.
Ex. %GROUP mem_s1p{top.a.b}, mem_s2p;

NOTE:

If the %MEMORY_GROUP section is omitted, and there is more than one %MEMORY section, all the memories will be tested at the same time, and no ***MemGroupSel*** signal is generated. If the users want ***srambist*** to generate the ***MemGroupSel*** signal, at least one %GROUP section has to be created.

If one memory module is used to create multiple instances, do not write the same %MEMORY section multiple times. Simply use the %GROUP.

3. The %Memory Section

Items defined in this section are the memory attributes. All the valid items inside the %MEMORY block, as described below, have to be end with ";".

%TYPE <memory type> ;

Valid memory type is SRAM only. This item is mandatory.
Ex. %TYPE SRAM;

%DATA_BITS <number> ;

This item specifies the memory data bus width. This item is mandatory.
Ex. %DATA_BITS 8;

%ADDR_BITS <number> ;

This item specifies the memory address bus width.
Ex. %ADDR_BITS 10;

%HIGH_ADDR <bits string> ;

This item defines the highest memory address.
Ex. %HIGH_ADDR 8'b11111111;

%LATENCY <number> ;

This item describes when the BIST should strobe the data output from the memory for comparison, and it is mandatory.
Ex. %LATENCY 0;

%CLOCK <clock name>;

This item specifies the memory clock name. If there is only one clock in the memory, the item can be specified either in the memory section or in the port section. If there are different clocks for different ports in the memory, then each port clock must be specified in each port section. If the clocks are specified here, all the clocks in the %PORT section will be ignored. If no clock is specified, the memory will be assumed as an asynchronous memory.
Ex. %CLOCK CLK;

If the memory is triggered at the clock's falling edge, a minus sign needs to be added before the clock name to indicate that it is an active low signal.
.Ex. %CLOCK -CLK;

%SELECT <chip select name > ;

This item specifies the chip select signal name, which will not overwrite the %SELECT in the %PORT section. If there are port select signals in the memory, they will be defined in each port section. .
Ex. %SELECT CE;

If the chip select is active low, a minus sign needs to be added in front of the name.
Ex. %SELECT -CE;

%OTHER_INPUT

%OTHER_INPUT <name1> , <name2> , ...;

All the inputs that are not listed in other keywords are specified here. For a 4 bits bus signal "A", it can be written as A[3:0].
Ex. %OTHER_INPUT pwn, A[3:0];

%OTHER_OUTPUT

%OTHER_OUTPUT <name1> , <name2> , ...;

All the outputs that do not belong to any other keywords must be specified here.

%PORT

The port section defines all the port signals. If there are many ports in the memory, each port section must be specified individually.

Ex. %PORT a
```
    {
      ----
    }
```

%TYPE r|w|rw;

Available types are r (read port), w (write port) and rw (read/write port). This item is mandatory.
Ex. %TYPE rw;

%ADDRESS <address name> ;

This item defines the port address name.
Ex. %ADDRESS A;

If the addresses are not written in the bus style, they can be written as the following:
%ADDRESS A7, A6, A5, A4, A3, A2, A1, A0;

%DATA_IN < input data name > ;

This item specifies the input data signal name.
Ex. %DATA_IN D;

If the input data are not written in the bus style, they can be written as the following:

%DATA_IN D7, D6, D5, D4, D3, D2, D1, D0;

%DATA_OUT < output data name > ;

This item specifies the output data signal name.
Ex. %DATA_OUT Q;

If the output data are not written in the bus style, they can be written as the following:
%DATA_OUT Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0;

%SELECT <port select name> ;

This item specifies the port select signal name. For more information, please refer to %SELECT in the %MEMORY section. Only one signal can be specified here, and no bus signal is accepted.

%CLOCK <port clock name> ;

This item specifies the port clock signal name. For more information, please see %CLOCK in the %MEMORY section.

%OUTPUT_EN <output enable name> ;

This item defines the port output enable signal. Users can define multiple signals in this item, and the bus signal is also acceptable.
Ex. %OUTPUT_EN OE;
Ex. %OUTPUT_EN OEA[1:0];

If this signal is active low, a minus sign will be added.
Ex. %OUTPUT_EN -OE;

%READ_EN <read enable name> ;

This item defines the port read enable name. Users can define multiple signals in this item, and the bus signal is also acceptable.
Ex. %READ_EN RE;
Ex. %READ_EN RE[1:0];

If this signal is active low, a minus sign will be added.
Ex. %READ_EN -RE;

%WRITE_EN <write enable name> ;

This item defines the port write enable name. Users can define multiple signals in this item, and the bus signal is also acceptable.
Ex. %WRITE_EN WE;
Ex. %WRITE_EN WE[1:0];

If this signal is active low, a minus sign will be added.
Ex. %WRITE_EN -WE;

NOTE: This version only supports at most 2 memory modules or 2 memory instances in the mdf file.

NOTE: If a line begins with //, it will be ignored by the ***srambist***.

C. Creating User-Specified BIST Algorithm

Users can generate their own algorithms by writing the algorithms in the BIST Configuration File (BCF). Users can describe many algorithms in one single .bcf file but users can choose only one of the algorithms in the .bcf file to generate a BIST logic at one time.

Users can use the following keywords, constructs, and examples to describe their algorithm(s).

%ALGORITHM

Each algorithm has to begin with the keyword " %ALGORITHM". This is followed by the name of the algorithm; for example, in the code below, the algorithm name is 5N.

Ex.

```
%ALGORITHM 5N
{
        ------
}
```

Within the %ALGORITHM section, users can define as many March operations preferred. However, if the algorithm length is increased, users need to make sure memory BIST can still operate at desired speed.

NOTE : This version only allows to specify at most 13N operations.

%MARCH

Ex. %MARCH U { %W(0) ; }

With the %MARCH command, the operations inside the bracket are performed on every address.

When using %MARCH U, the operations are performed from the lowest address to the highest address. When using %MARCH D, the actions are performed from the highest address to the lowest address.

%W(0), %W(1)

The %W (0) operation will write a pattern 0 in the memory while %W (1) will write a pattern 1 in the memory.

%R(0), %R(1)

The %R(0) operation will read the data from the memory and compare it to the background pattern 0. Every "read" operation will also perform a comparison with the expected data and data output from the memory.

All operations within the same bracket are performed to one address first and then advance to the next address. For the following algorithm, %ALGORITHM 5N,

```
{
        %MARCH  U { %W(0) ; }
        %MARCH  U { %R(0) ; %W(1) ; }
        %MARCH  D { %R(1) ; %W(0) ; }

        %REPEAT_PAT 0 ( 0000, 0101 ) ;
        %REPEAT_PAT 1 ( 1111, 1010 ) ;
}
```

the following operations are performed:

1. Write pattern 0 from the lowest address to the highest address.

2. Read data from the memory and compare with the value "0". Then, write data pattern 1 to the same address. This step is repeated from the lowest address to the highest address.

3. Read data from the memory and compare with the value "1". Then, write pattern "0" to the same address. This step is repeated from the highest address to the lowest address.

% REPEAT_PAT

After specifying the March operation, users need to specify background patterns. srambist supports up to 28 background patterns (0, 1, A-Z). %REPEAT_PAT specifies the background patterns. The syntax for %REPEAT_PAT is written as the following:

%REPEAT_PAT [0/1/A-Z] (<first set of background data pattern>, <second> .....);

For example:
%REPEAT_PAT 0 (0000) ;

This command defines the background pattern 0 as 0000. If %DATA_BITS is larger than the bit length in %REPEAT_PAT, srambist will fill in the data pattern to the length of %DATA_BITS by repeating the bit string automatically.

Below is an example when the value of %DATA_BITS is 8.

Background Pattern Specification	Generated background patterns
%REPEAT_PAT 0 (0) ;	00000000
%REPEAT_PAT 0 (01) ;	01010101
%REPEAT_PAT 1 (1111);	11111111

Users can generate more than one set of background patterns. For example:

If Background Pattern Specification Is	Initial Set of Patterns	Second Set of Patterns
%REPEAT_PAT 0 (0, 01);	00000000	01010101
%REPEAT_PAT 1 (1, 10);	11111111	10101010

If there is more than one set of background patterns, the BIST will use the first set of background patterns for the algorithm first; after the first set of background patterns is completed, it will use the second set of background patterns for the algorithm, and so on.

NOTE: If the 13N March algorithm is used and two background pattern sets are created, the total time to test will be 13N * 2 = 26N.

Besides REPEAT_PAT 0 and REPEAT_PAT 1, users can specify 26 more background patterns A to Z. For
%REPEAT_PAT A (0, 01);
%REPEAT_PAT B (1, 10)

The algorithm can be written as follows.

```
{
        %MARCH  U { %W(A) ; }
        %MARCH  U { %R(A) ; %W(B) ; }
        %MARCH  D { %R(B) ; %W(A) ; }

        %REPEAT_PAT A ( 0, 01 ) ;
        %REPEAT_PAT B ( 1, 10 ) ;
}
```