Practical Methods and Tools for Embedded Macro Test

by
Bernd Koenemann, Dean Adams, Brion Keller, Randy Kerr, David Pruden, and Ronald Walther,
IBM
Contact: berndk@us.ibm.com
Reference: http://www.chips.ibm.com/services/testbench

Abstract

Modern semiconductor technologies have long broken through the million-gate threshold. Today's tools and methodologies have to handle designs with well over 10 million gates and more than 1 million flip-flops. Chips of that size often combine many different types of design elements including a variety of macros from a rich set of library components. To meet aggressive development schedules, ASICs and custom processor designs alike depend on a high degree of automation in all aspects of design and test.

This presentation will provide an overview of a highly automated ASIC test development methodology that enables a vector-less sign-off flow. The specific focus will be on the methods and tools used for automating the tests for embedded macro functions like processor cores, register files, dense SRAM/eDRAM/ROM macros, and other types of macros. The macro test tools in particular facilitate the access of embedded macro interfaces from the surrounding, scan-based chip logic, and the automatic migration/translation of macro-level test data to top-level test data. Key features of the macro test tool flow and the input languages for describing the macro test interface and macro-level test data will be shown.

Extended Summary

Background

A recent announcement slated the availability of ASIC design kits for a new technology with 40 million wireable gates in a .11µm drawn (.08 µm effective) CMOS process with 7 levels of copper interconnect and low-k dielectric for later this summer. The design kits will continue using a highly successful vector-less sign-off methodology. Vector-less sign-off means that the ASIC customers do not have to provide any test vectors for manufacturing test or design verification.

For test, the customers follow a very robust LSSD Design-For-Test (DFT) methodology using DFT Synthesis and DFT checking tools provided in the design kits. The customer needs no additional test tools. All test data generation is performed post-sign-off by the manufacturing organization using a completely automated, essentially push-button, process.

The ASIC library contains a very rich set of embeddable cores to enable System-on-Chip (SoC) designs. The library elements include microprocessors, memories (e.g., SRAM, eDRAM, ROM, etc.), data compression (e.g., JPEG, ALDC, etc.), interconnect (e.g., SONET, HSSL, etc.), interfacing (e.g., PCI, USB, etc.), mixed-signal elements (e.g., PLLs, DAC/ADC, etc.), and more.

To be successful, the test development methodology must make it easy to integrate the cores with the rest of the design and to automatically generate a composite set of test data for the complete design. While covering the complete ASIC test methodology is beyond the scope of this presentation, a brief summary of some key methodology elements is useful for establishing a context.

Reduced Pin Count Testing (RPCT)

The number of signal I/Os on the chips can exceed the number of channels available on the Automatic Test Equipment (ATE). The DFT Synthesis tools in the design kit insert special boundary scan cells as needed to establish I/O wrap-back paths that allows for at-speed driver/receiver testing without contacting the pad. At wafer probe only a fixed subset of I/Os, called test I/Os, is contacted while the other I/Os are tested through the I/O wrap-back path. The ATE used for final test has a large number of low-cost DC-only channels in addition full function channels for the test I/Os. That permits the testing of the chip and package pad connections as well as DC parametric testing of the I/O drivers and receivers. Most other testing, including the test of embedded macros, is performed through the test I/Os. The I/O wrap tests and the I/O parametric tests are automatically generated.

Logic Test

The on-chip logic is tested using a full-scan Level Sensitive Scan Design (LSSD) methodology. The front-end DFT flow automatically replaces any functionally edge-triggered flip-flops with scannable LSSD Shift-Register Latches (SRLs) that are driven by LSSD clock splitters. The clock splitters preserve the original edge-triggered clocking while adding multi-phase level-sensitive test mode clocking capabilities. DFT Synthesis handles scan chain insertion and balancing. The insertion of the clock splitters into the clock trees is optimized during physical design, as is the ordering of SRLs in the LSSD scan chains.

To simplify Automatic Test Pattern Generation (ATPG) and to minimize the use of test wrappers and other isolation/partitioning hardware overhead, all LSSD logic is combined for flat full-chip ATPG. The ATPG tool can generate full-chip tests for Stored Pattern ATE and for Weighted Random Pattern (WRP) ATE depending on pre-defined scripts for the different technology/package options. In the WRP case, the tests are stored in a highly compressed test data format that is expanded in real-time by custom hardware in the WRP ATE.

Embedded Memories

The ASIC library contains high-performance register file macros that are implemented with scannable register elements. From a test perspective, these register files are treated not as RAM macros, but as are implemented as scannable, LSSD-compatible logic and merged into the rest of the logic for full-chip ATPG.

Dense SRAM macros as well as ROM and embedded DRAM macros are tested by memory Array BIST (ABIST) techniques. For certain memory types, a single Multiple-ABIST (MABIST) controller can optionally be shared by several compatible memory macros. The ABIST/MABIST controllers generally contain a number LSSD registers for programming different modes of operation (e.g., for testing, for test algorithms, burn-in, diagnostics, etc) and result/status feedback. The memory blocks themselves are surrounded by boundary scan to provide observability/controllability when testing the surrounding logic. The ABIST/MABIST logic and the memory wrappers are merged into the rest of the logic for full-chip ATPG.

Embedded Cores

Newer logic hard cores, including micro-processors, are designed with LSSD for and full-chip ATPG. The logic model and existing scan chains for the core are picked up by the DFT synthesis tools and merged with the scan structure of the rest of the chip. Synthesizable logic cores follow the DFT insertion flow described earlier and are also merged for full-chip ATPG.

Some older legacy processor cores require the application of broadside functional tests in addition to LSSD scan tests. To facilitate the application of these functional tests to the embedded core from the chip level and to test the surrounding logic, a special test wrapper must be inserted around the core. The test wrapper uses muxing-techniques to connect the core I/Os to the RPCT test I/Os for accessing the core for test. The embedded macro test tools described later on are used for verifying/controlling the access paths and for translating the core test patterns to the chip-level.

The embedded macro test tools also are used for verifying/controlling the correct connection of the ABIST/MABIST controllers, and to generate the chip-level test data needed for programming, and running the BIST engines as well as for receiving the BIST status and result information.

In addition, many chips contain other types of macros/cores like PLLs, ADCs, DACs, special I/Os, and so on. Non-logic cores can be separated from the surrounding logic by an integrated scannable wrapper. The wrapper logic becomes part of the full-chip logic model used for ATPG. In addition, the core may require direct access to certain core ports from the chip boundary for test. In that case, the embedded macro test tools can be used to verify/control the necessary access paths.

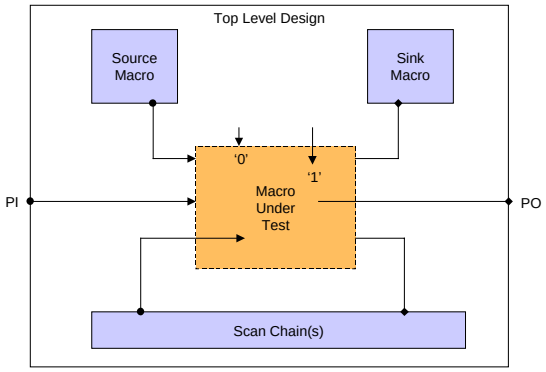
The Embedded Macro Test Approach

Brief Overview of the Concept

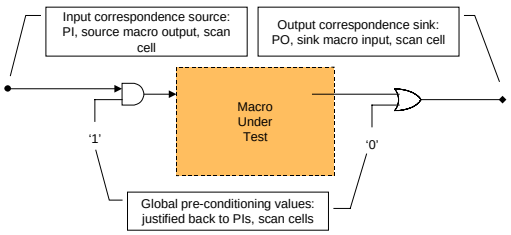
The embedded macro test approach and set of tools were originally developed many years back for testing embedded memories on LSSD chips and packages. The basic concept is very simple and powerful. To provide test stimuli to the macro inputs, a logic path is sensitized between each relevant macro data input and a corresponding controllable net (input correspondence point) in the top-level design. Likewise, to receive test responses from the macro outputs, a logic path is sensitized between each macro output and a corresponding observable net (output correspondence point) in the top-level design.

In a scan-based top-level design, Primary Inputs (PIs) and scan cell outputs are considered controllable nets, while Primary Outputs (POs) and scan cell inputs are observable nets. With BIST, some ATE control/observe functions are actually moved into the BIST controllers, and certain BIST controller inputs/outputs may qualify as macro output/input correspondence points in specific cases.

This general structure is illustrated in the figure below.



Path sensitization between the macro I/Os and their respective correspondence points requires that the side-inputs of all logic gates on each path are pre-conditioned to non-controlling values. In addition, some macro inputs may also need to be pre-conditioned to certain values to put the macro into the proper state for testing. The local pre-conditioning values must be justified back to suitable top-level controllable nets such that the pre-conditioning can be applied from the top level as illustrated in the following figure:



The embedded macro test concept described here requires that the pre-conditioning and correspondence path sensitization is independent of the data values transported along the correspondence paths. Hence, the actual test data values need not be known, and are not taken into account, at the time the correspondence solution is generated. The correspondence solution in turn must guarantee that arbitrary test data values can be transported to/from the macro interface safely.

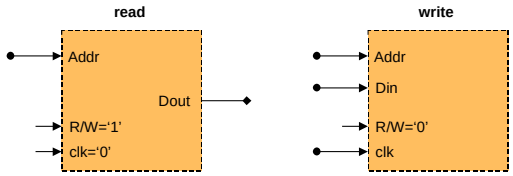
The data-independent correspondence solution in essence defines a top-level test pattern template with actual values for pre-conditioning and "placeholders" for the macro test data variables. Generating the derived test data involves reading the macro-level data values and "sticking" them into the appropriate "placeholders" in the top-level pattern template.

Summary of Some Key Features

Macros and Macro Operations

Any module in the design hierarchy can be marked as an embedded macro. The macro itself can have an internal structure that is visible in the chip-level netlist. The macro interface consists of all or some of the module I/Os, and optionally internal nets in the visible macro-internal structure.

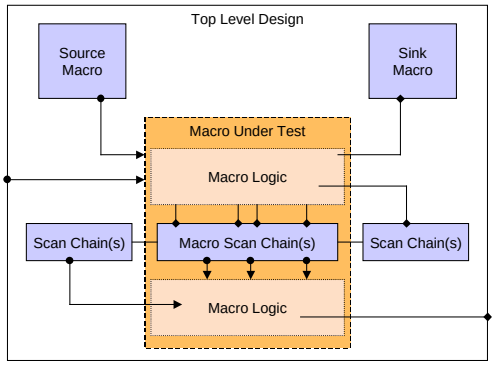
Each macro can have several operations defined. Each operation is a named interface configuration that may involve different macro interface nets. A memory macro, for example, may have a read operation and a write operation defined as illustrated below:



Each of the illustrated operations uses only a subset of the macro I/Os. Finding a separate correspondence solution for each operation often is easier and results in a smaller solution footprint than having to find a solution for all macro I/Os simultaneously. It should be noted that the macro-level test data must match the operations defined for the macro (no test vector must require access to I/Os from more than one operation at a time). It also must be possible to switch between operations without destroying the test flow.

Macro-Internal Scan Chains

If a macro contains one or more internal scan chains that are completely visible in the chip-level netlist, then the associated macro-internal scan cells can be used as correspondence points just like any other scan cell in the design:



This simple, yet elegant approach makes it possible to use the macro test tools to, say, find and access BIST programming/result registers in the middle of chip-level scan chains.

Macro Grouping

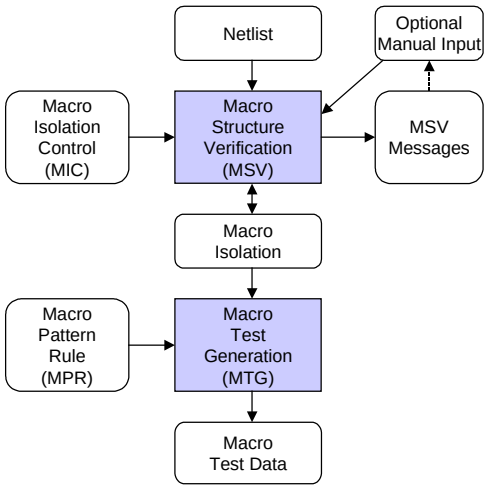
On chips with many macros it can be desirable to group several macros for simultaneous access. If the macros meet certain compatibility requirements (e.g., the macros use the same operations and test data), it may be possible to share input correspondence points.

Embedded Macro Test Tools Overview

The embedded macro test methodology uses a data-independent access scheme. Hence, the processing is performed in two separate steps. First, a Macro Structure Verifiation (MSV) tool generates correspondence solutions for all relevant macros and macro operations. The correspondence solutions include the necessary pre-conditioning data and global pattern templates for the actual test data. Second, a Macro Test Generation (MTG) tool combines the local macro-level test data with the correspondence solutions to generate the global test data for the macros.

The data-independent approach makes it possible to defer the actual data generation until very late in the game. The ASIC designers use MSV to generate/verify the correspondence solutions without needing access to the macro-level test data. The pattern translation is performed post-sign-off by manufacturing.

The basic flow is illustrated below.



Macro Structure Verification (MSV)

The MSV tool reads the chip-level netlist and test mode information. It will find all macros that must be processed for the given test-mode. Each such macro must have an associated Macro Isolation Control (MIC) file that defines the macro-specific operations and access requirements. MSV will then try to find a pre-conditioning and correspondence solution for each macro operation. For each macro data input/output, the tool must find a suitable correspondence point and sensitize a path. The path sensitization is performed by an Automatic Test Pattern Generation (ATPG) utility that justifies the path side-inputs back to appropriate pre-conditioning values. The MIC file describes the nature of the allowed path and pre-conditioning constraints for each macro pin. The MIC file also contains information about macro grouping and sharing constraints.

A small excerpt from a MIC file for an ABIST controller is shown below.

```
MIC_DEF ReleaseDate=98/04/03 ReleaseTime=14:16:57;

ALGORITHM
   NAME                 = BIST_MFG,
   MACRO_TYPE           = macvim_1P_MABIST, macvim_2P_MABIST,
   OPERATIONS           = INITIALIZE_MABIST, etc.

OPERATION
   NAME                 = INITIALIZE_MABIST,
   PINGROUPS            = TEST_ENABLE, BIST_INIT_ALL, BIST_INIT_MODE,
   REQUIRE_STAB         = YES,
   MULTIGROUP           = SEVERE,
   PRECONDITION_TYPE    = PI_LATCH; etc.

PINGROUP
   NAME                 = TEST_ENABLE,
   PINS                 = TESTM1/1, TESTM3/0, GATE/1,
   CORRESP_TYPE         = ANY; etc.

INNER_NETS
   NAME                 = BIST_DATA_OUT,
   NET_NAME_PREFIX      = "dout_",
   NETS                 = 000:127,
   NET_TYPE             = OUTPUT; etc.
```

The PINGROUP and INNER_NETS statements define named groups of macro I/O ports and internal nets and specify the required correspondence type for each pin. The OPERATTONS statements define the associated PINGROUPS/INNER_NETS as well as pre-conditioning and grouping constraints. The ALGORITHM statements define a set of operations and the name of a Macro Pattern Rule (MPR) file for the associated test data.

In the case MSV fails to achieve/verify a correspondence objective, detailed error messages are written for analysis and debug.

If desired or necessary, the user can guide the global correspondence solution. For example the user can specify specific correspondence points and/or pre-conditioning data rather than letting MSV find all correspondence points and pre-conditioning data by itself.

The correspondence solutions are generated in binary and in text formats.

Macro Test Generation (MTG)

The MTG tool reads the binary form of the correspondence solution and the associated Macro Pattern Rule (MPR) files. The MPR files contain the macro-level test data that are to be "plugged" into the global pattern templates defined by the correspondence solutions.

The resulting macro test data are then written into the test data file for the chip. The test data file is highly structured and the macro tests are clearly defined within the composite test data for the complete chip.

Macro Pattern Rule (MPR) File

The MPR files actually are C programs that use MTG-specific commands to assign data values to the PINGROUPS/INNER_NETS defined in the operations for the ALGORITHM represented by the MPR. The concept is illustrated by the simple MPR excerpt shown below:

```
/*   ALGORITHM = WALKING_ONES  */

void WALKING_ONES(void) {

  char ZERO[]="00";
  char ONE[]="11";
  char msg_text[120];
  int message_number=1;
  char severity_code='S';
  int i;

  /* Write each address to zero, and then read it to make sure */
  MtgAssignString( ZERO, "DATA_IN");
  MtgAssignString( ZERO, "DATA_OUT");

  for (i=0; i<8; i++) {
    MtgAssignValue(i,"ADDRESS");
    MtgExecuteOperation( "WRITE", 1);
    MtgExecuteOperation( "READ", 1);
  } /* endfor */

and so on . . .
```

The code example uses an MtgAssignString command to assign a pre-defined value string ZERO to PINGROUPS named DATA_IN and DATA_OUT. It then uses a simple loop to increment an address value named that is assigned to a PINGROUP named ADDRESS by an MtgAssignValue command. For each address value, OPERATIONS called READ and WRITE are invoked by an MtgExecuteOperation command. The value behind the comma in the arguments is a repetition count. Repetition counts are very useful, for example, for ABIST algorithms where the bulk of the test consists of repeating some BIST clock sequence for a large number of times.

Summary

Originally developed for broadside and serial access to embedded memory array macros, the methods and tools described here are in daily use in a very successful ASIC test development flow. The primary use today is to manage the chip-level access to memory array BIST (ABIST) controllers, some legacy cores, and special macros like PLLs. The data-independent approach and the two-step tools flow lend themselves to a vector-less ASIC sign-off flow.

Acknowledgements

The embedded macro test approach described in this paper is the product of many years of development and application by several generations of highly talented and dedicated IBM engineers.