ECS Transactions, 27 (1) 185-190 (2010)
10.1149/1.3360617 ©The Electrochemical Society

A Case Study of Hierarchical Diagnosis for Core-Based SoC

Eric Wang[a], Yu Huang[b], Wu-Tung Cheng[b], Wu Yang[b], and James Fu[b]

[a]Freescale Semiconductor, 288 ZhuYuan Rd, SuZhou, JiangSu, P.R.C, 215011
[b]Mentor Graphics Corporation, 8005 S.W. Boeckman Rd., Wilsonville, OR 97070, USA

In this paper, a silicon debug case study was given in the context of a hierarchical diagnosis flow for core-based SoC. We discuss (1) how to design a simple core wrapper that supports at-speed test, (2) how to map the failures collected from the chip level to core level, and (3) how to perform failure analysis and silicon debug under the guidance of diagnosis results.

1 Terminology and Introduction

The terminology used in this paper is briefly discussed below.

SoC: Designs that integrate a complete system onto one chip are called System-on-a-Chip (SoC) designs.

***Core*:** In SoC designs, the design process involves an IC that is often made up of large pre-defined and pre-verified reusable building blocks or intellectual property (IP) blocks, such as digital logic, processors, memories, analog and mixed signal circuits. The IC building blocks are called cores or embedded cores [1]. In order to shorten the design cycle and improve the system integration efficiency, the modules with more complex functions and required to guarantee high quality of routing and timing are often delivered to the SoC design company as hard IP cores. Typically, a pre-generated and validated test pattern set for a hard IP core is also delivered together with the core. Core-based SoC design strategy has become more and more popular because designing a multi-million gate system with pre-designed and pre-verified reusable cores is the most effective way to reduce design time and cost. As the complexity of SoCs keep increasing and process technology keeps shrinking, how to rapidly diagnose test failures, analyze diagnosis results and identify the root cause of the defects become more and more important for silicon debug and yield improving.

Test Access Mechanism (TAM): On-chip hardware infrastructure to transport test stimuli from the SoC pins to the embedded cores. TAMs also transport test responses from the embedded cores to the SoC pins [1]. In prior publications, several TAM architectures were proposed for SoC testing [2-6]. They can be classified into two categories: (1) using existing functional paths inside cores and (2) inserting additional paths outside of cores dedicated to test purposes. It is also possible to use a combination of these approaches. The method proposed in [2] belongs to the first category. Most of the TAMs proposed in the literature belong to the second category, i.e. the test access paths are established outside of embedded cores. The advantages of using these TAMs include (1) structured DFT, (2) simple control protocols and (3) easier diagnosis. The major TAM techniques in this category include parallel access [3], serial access [4], Test Bus [5], and TestRAIL [6].

Wrapper: A wrapper is a thin shell around the core that provides interface between SoC pins and core terminals during test [7] [8]. It also allows adjusting the bandwidth for test thus providing the flexibility to better utilize the limited number of SOC pins for test. Controlled by the instruction register, a core can be set into functional mode, test mode and bypass mode. In test mode, it can be accessed from TAM through wrapper boundary cells. The functional input / output / bi-directional terminals of a core connect with these wrapper cells. In addition, if a core has internal scan chains, the scan input and output terminals of the core can also be connected to the wrapper cells. This leads to multiple choices for configuring a wrapper. In the next section we will first review the IEEE standard wrapper design and then describe the wrapper used in our SoC.

2 Core Wrapper Design

The IEEE 1500 core wrapper [8] is illustrated in Figure 1. Its components are briefly described as follows.

(1) Wrapper Serial Port (WSP) has a set of serial terminals that could be sourced from chip-level pins or from an embedded controller such as an IEEE 1149.1-based (JTAG) controller. The WSP is used to load and unload instructions and data into and out of the IEEE 1500 registers. In addition to the wrapper serial input (WSI) and wrapper serial output (WSO) terminals shown in Figure 1, the WSP contains wrapper serial control (WSC) terminals used to control the operation of all IEEE 1500 registers.

(2) Wrapper Parallel Port (WPP) is a user-defined set of wrapper terminals providing a parallel interface to the IEEE 1500 wrapper. These terminals are used when the wrapper is configured into parallel mode.

(3) Wrapper Instruction Register (WIR) enables all IEEE 1500 wrapper operations. This register is loaded via the WSP with instructions that select an IEEE 1500 data register. The WIR can optionally be interfaced to the core for establishing test mode or functional operation.

(4) Wrapper Bypass Register (WBY) provides a bypass path for the WSI-WSO terminals of the WSP. The WBY is the default data register between WSI and WSO and should be selected by the current wrapper instruction when no other data register is selected. The WBY is intended to provide a minimum length scan path through the wrapper, so that when several IEEE 1500 wrappers are serially chained together in a SoC, the wrappers that do not require a data register to be accessed can be bypassed with a short scan path through their WSI/WSO terminals.

(5) Wrapper Boundary Register (WBR) is the data register through which test data stimuli are applied and pattern responses are captured. This register allows internal testing of the core, as well as testing of external connectivity to other cores and SoC integration circuitry, in response to an instruction loaded into the WIR.

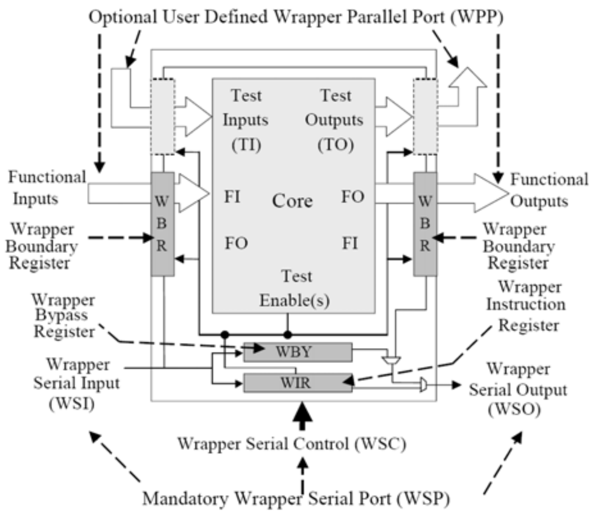

Optional User Defined Wrapper Parallel Port (WPP)
Test Inputs (TI)
Test Outputs (TO)
Functional Inputs
FI
Core
FO
Functional Outputs
W B R
W B R
Wrapper Boundary Register
FO
FI
Wrapper Boundary Register
Test Enable(s)
Wrapper Bypass Register
Wrapper Instruction Register
WBY
WIR
Wrapper Serial Input (WSI)
Wrapper Serial Output (WSO)
Wrapper Serial Control (WSC)
Mandatory Wrapper Serial Port (WSP)

Figure 1: Standard Components of the IEEE 1500 wrapper

Although IEEE 1500 wrapper is very flexible for controlling and testing complicated SoC with many cores, we decide to design a more simplified wrapper for our SoC.

(1) We do not need WBY for our SoC design. Our SoC design only has 4 hard IP cores, and hence the total testing time is not a big problem for us. Therefore we decide to apply serial core testing for simplicity. That is to say we test one core at a time. When testing a core, its WPP was connected to the IO pads at chip level directly. The core test scheduling becomes very simple.

(2) Instead of using a dedicated WIR, we use a few control signals to directly control the core into test mode or functional mode. In Figure 2(a) we illustrate the conceptual view of the core wrapper used for our SoC. In Figure 2(b) and 2(c), we show the detailed implementation of a wrapper cell. To support launch-by-capture at-speed test, we design our wrapper with 2 flops for each wrapper cell. As it can be seen, in functional mode, "*test_mode*" is set to 0 and the wrapper cell is in transparent mode. The IP signals can pass through the wrapper cell and communicate with other modules in the SoC. In test mode, "*test_mode*" is set 1. The core input signals come from the port Q of the flip-flops in the wrapper cell, and the core output signals go to port D of the flip-flops. By using two flops per wrapper cell, core IOs can be used as launched points for at-speed test since the proposed wrapper cells can have two values in two clock cycles. Under the control of the ATPG test vectors, when "*wrapper_se*" is 1, these flip-flops are connected as a scan chain to load the input signals and unload the output signals; when "*wrapper_se*" is 0, the flip-flops are used to update the input signals and capture the output signals.

The above-mentioned simplified wrapper satisfies our test requirements for the design at hand and reduces the silicon area overhead which is introduced by complicated IEEE 1500 wrapper.

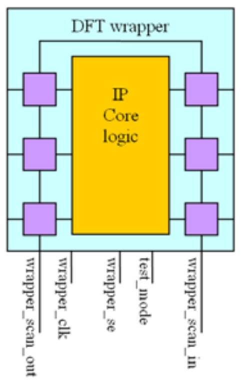


Figure 2(a):
Conceptual View of Wrapper

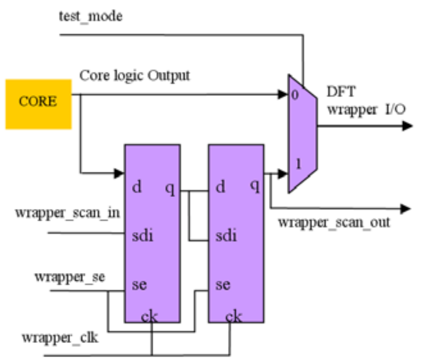


Figure 2(b):
Wrapper Cell at Core Output Side

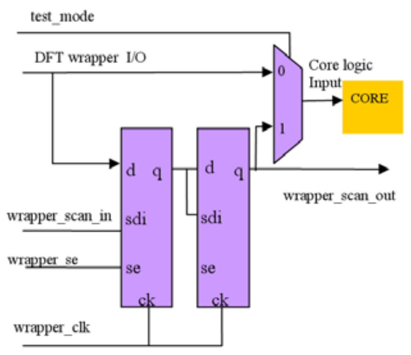


Figure 2(c):
Wrapper Cell at Core Input Side

In Figure 2(b), we illustrated the core output side and in Figure 2(c), we illustrated the core input side. Note that the internal scan chains of a core can be accessed through WPP of the wrapper, which is not illustrated in Figure 2.

3 Hierarchical Diagnosis

As we know the hierarchical nature of SoC designs makes the hierarchical ATPG and diagnosis the most straightforward and effective approaches. Since many prior papers discussed how to perform hierarchical ATPG for core-based SoC designs, in this section we will only focus on how to run hierarchical diagnosis. It requires two steps to run hierarchical diagnosis.

The first step is to identify if the failures come from cores or come from glue logic between cores. If the failures come from cores, we need identify which core(s) caused the failures. This is straightforward if we always maintain a core level database to indicate a test cycle at a particular pin. So, it requires maintaining two mapping relations in one database:

(1) The mapping between the primary IO port names at core level and chip level in test mode.

Note that it is a simple task if we always statically allocate a core to a fixed TAM during the entire testing of this particular core. If someone dynamically allocates cores to TAM to achieve optimal test time, like the methodology proposed in [9] [10], the mapping relation must be also dynamically tracked. Hence it will add another dimension of test time / cycle number to the name mapping database.

(2) The mapping between the test cycle numbers at core level and chip level in test mode.

Note that this mapping is not only related to a test scheduling scheme, it also related to the test setup procedure at the power on phase or between testing of two cores. If any post shift procedures are used, the mapping has to consider it as well. After the test scheduling is done, and the test setup procedure is determined, we need carefully calculate the mapping relation between test cycle numbers at core level and at chip level.

The second step is relatively easy. Suppose we have identified the failing core and translate the failure data log from chip level to core level, based on the information stored in the two mapping database. We can run core level diagnosis as if it is a failing chip, by using core netlist, core patterns and translated core level failure data log.

The above mentioned diagnosis flow can be iteratively applied from the upper level down to lower level, if hierarchical cores are used. Essentially hierarchical diagnosis can provide us with better diagnosis resolution and shorter run time, compared to diagnosing the entire chip.

4. Case Study

In this section, we share our experiences with one industrial case study to illustrate the application of the proposed hierarchical diagnosis flow. The design we manufactured is a SoC integrated with 4 hard IP cores, which are manufactured by 90nm CMOS process. Each core has 27947 scan cells, 46 scan chains, and about 1500 PIs/POs. The entire SoC design has about 7 million gates in total.

Since the hard IP used to be applied into multiple projects, to avoid duplicated efforts to generate the test pattern, we can just map the IP level patterns to the chip level. This method not only improves the reusability and reliability of the patterns, but also reduces the pattern debug effort in SoC level.

The core level patterns can be mapped to chip level by applying the flow shown in Figure 3. It also takes advantage of the mapping database proposed in the previous section, but in an opposite way compared to diagnosis flow. Firstly, we map the primary IO port names in hard IP cores to chip level. Then, we schedule the order of the cores to be tested, and determine a power-on setup pattern sequence according to the chip reset sequence and boot up mode. Finally, the chip level patterns are obtained by merging the setup patterns and the patterns at each hard IP core level, based on their scheduled order.

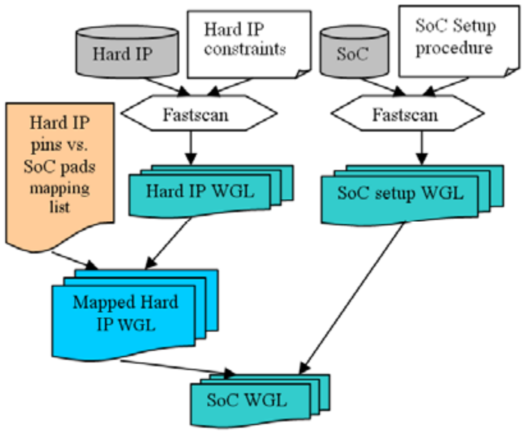


Figure 3: Test Pattern Integration Flow

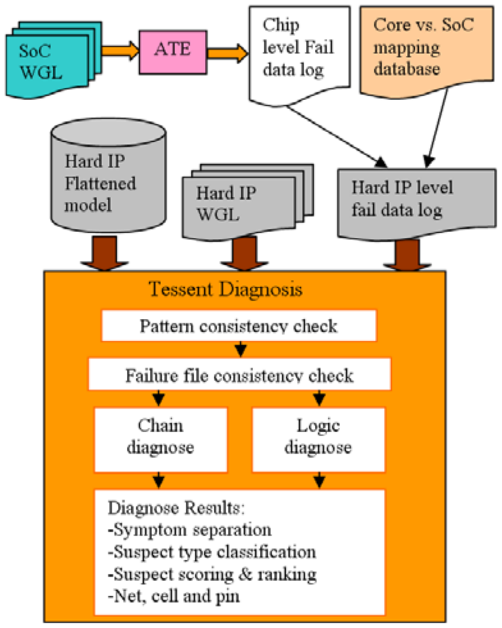


Figure 4: Core Diagnosis with Tessent Diagnosis

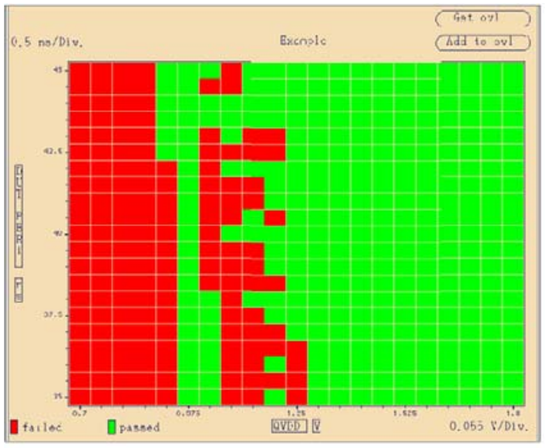
The first silicon failed at-speed test. We have to identify the root cause of the test failures as soon as possible, and provide guidance to make the design function. Good EDA tools can help engineers run diagnosis at logic and physical level, which will greatly reduce the effort of silicon debug. In our silicon debug phase, firstly, we apply the proposed hierarchical diagnosis flow and identify the failing IP core. Secondly, Mentor Graphics's diagnosis tool Tessent Diagnosis (its old name is YieldAssist) was used to do the logic diagnosis at core level.

The input data for Tessent Diagnosis include the flattened design model, core level ATPG test patterns and failure data log which is translated from SoC level. The flattened model is generated during ATPG, which includes design netlist, test signal constraints and the circuit model of some modules. WGL, parallel STIL, ASCII and Binary are the acceptable test pattern formats for Tessent Diagnosis. Since the ATE vendors have different data log format, it's required to convert them to a unified format which can be read into Tessent Diagnosis.

This case proves the benefits of applying hierarchical diagnosis, which is only requiring core level test patterns and flattened model for diagnosis. Since the size of hard IP flatten model is much smaller than that of SoC, the runtime of Tessent Diagnosis is very short. Figure 4 illustrates our flow of diagnosing hard IP cores with Tessent Diagnosis.

In our case study, according to the Tessent Diagnosis report, a few particular nets are highlighted as the suspects. Meanwhile, we correlate the diagnosis report with the STA report, which indicates that the suspected nets given by Tessent Diagnosis happen to be on a timing critical path in on-chip clock generating module. So we know that the fault is caused by timing issues on the critical path. However, we still can not draw any conclusion if the root cause is design-related or process-related.

To further investigate the root cause, we draw shmoo plots with different voltage and clock frequencies. The shmoo plots using an internal on-chip clock and using an external clock are shown in Figures 5(a) and 5(b) respectively.



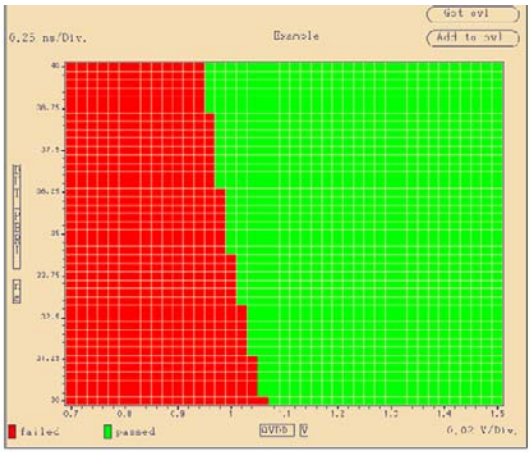
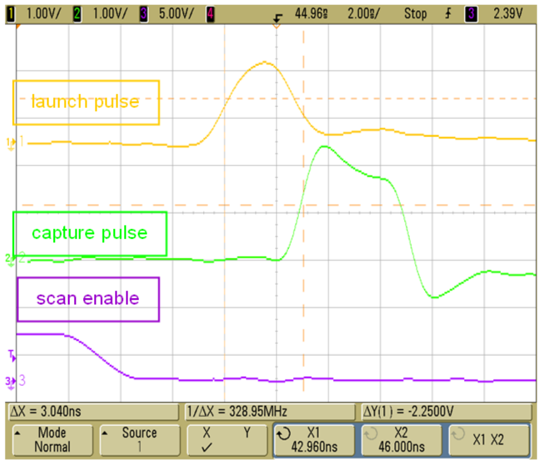


Figure 5(a):
Shmoo Plot with Internal On-Chip Clock Source

Figure 5(b):
Shmoo Plot with External Clock Source

Comparing the two Shmoo plots shown in (5.a) and (5.b), we found that using internal on-chip clock will cause intermittent failures at 1.15V, whereas using external clocks this is not happening.

To find out why we get intermittent failures at 1.15V when using internal on-chip clock, we exploit the micro probe measurement in failure analysis laboratory and observe that the clocks don't necessarily shift to the same side at the same time and it caused the phase to change between 2.9ns and 6.6ns. The two pictures illustrated in Figures 6(a) and 6(b) show the phase difference between launch and capture pulses is not stable around 1.15v. Figure 6(a) shows the minimum value of 3.04ns and Figure 6(b) shows the maximum value 6.6ns. However, the expected value should be around 12.7ns.



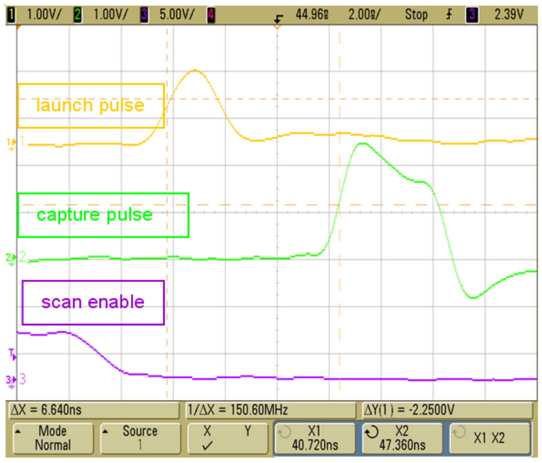


Figure 6(a): Minimum phase difference

Figure 6(b): Maximum phase difference

Based on these analyses, we finally can draw the conclusions that:

(1) The PLL generated clock phase variations lead to on-chip clock generator output pulse changes.
(2) The intermittent failures are caused by an unstable launch/capture frequency which is generated from on-chip clock generator.
(3) At lower or higher voltage rather than around 1.15v, the phase difference stabilizes to a ~12.7ns value, making the test to pass.

(4) The launch/capture pulse frequency lead to the AC scan failures. The timing critical paths in the tested IP are the most sensitive paths at that frequency.

After identified the root cause, we examined the PLL carefully. In the design phase we have to use PLL behavior model for simulation because it is an analog module. The behavior model may not as accurate as synthesis model that were used for other parts of the same design. This is why we can not detect the pattern failure during design phase with simulation.

To completely fix the problem, we need to create a better PLL model for future projects. However, for this project, we use external clock source as a workaround to speed up time-to-market. The conceptual view of the clock path select circuit is illustrated in Figure 7. As can be seen, When external_clk_sel=1 and cgm_en=0, the clk_root is input from external_scan_clk pad.

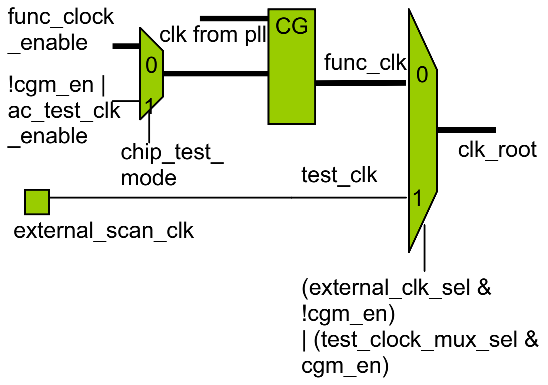


Figure 7: Bypass PLL to use External Clock Source

5 Conclusions

In this paper, we first proposed a hierarchical diagnosis flow such that diagnosis can be directly performed at core level instead of chip level. This can be achieved by maintaining a database translating between core IO names and chip level pads and translating core level failure cycle number to chip level failure cycle number. A silicon debug case study was also given in detail to explain how to correlate diagnosis results with shmoo plots, micro probe measurement to identify the root cause of the design problems.

References

[1] Y. Zorian, E.J. Marinissen and S. Dey, “Testing Embedded Core-Based System Chips,” Computer, Volume: 32 Issue: 6, pp. 52 –60, June 1999.
[2] I. Ghosh, N.K. Jha, and S. Dey, “A Low Overhead Design for Testability and Test Generation Technique for Core-Based System-on-a-Chip,” IEEE TCAD, Vol. 18, pp.1661-1676, Nov., 1999.
[3] V. Immaneni, and S. Raman, “Direct Access Test Scheme – Design of Block and Core Cells for Embedded ASICs,” pp.488-492, ITC 1990.
[4] N.A. Touba and B. Pouya, “Testing Embedded Cores Using Partial Isolation Rings,” pp. 10-15, VTS 1997.
[5] P. Varma and S. Bhatia, “A Structured Test Re-Use Methodology for Core-Based System Chips,” pp.294-302, ITC 1998.
[6] E.J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, “A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores,” pp.284-292, ITC 1998.
[7] E.J. Marinissen, S. K. Goel and M. Lousberg, “Wrapper Design for Embedded Core Test,” pp.911-920, ITC 2000.
[8] http://grouper.ieee.org/groups/1500/
[9] V. Iyengar, K. Chakrabarty, and E.J. Marinissen, “Test wrapper and test access mechanism co-optimization for system-on-chip,” pp. 1023 – 1032, ITC, 2001.
[10] S. Koranne, “Design of reconfigurable access wrappers for embedded core based SoC test,” pp.955 – 960, IEEE Trans. on VLSI Systems, Vol. 11, Issue 5, Oct. 2003.