Advanced JTAG-based On-Chip Debugging Unit Design for SoC

Yeonsang Yun, Seungyoul Kim, Youngdae Kim and Younggap You
Dept of Electrical and Computer Engineering, Chungbuk Nat'l University, Chongju City, Korea 361-763
Tel : +82-43-271-2480 Fax : +82-43-271-2480 E-mail: ysyun@hbt.chungbuk.ac.kr

Abstract: An on-chip debugging unit is proposed aiming performance enhancement of JTAG-based SoC systems. The proposed unit comprises a JTAG module and a core breaker. The IEEE 1149.1 standard has been modified and applied to the new JTAG module. The proposed unit eliminates redundant clock cycles included in the TAP command execution stage reducing overall debugging time. TAP execution commands are repeatedly issued to perform debugging of complicated SoC systems. Simulation on the proposed unit shows some 14% performance enhancement and 50% gate count reduction compared to the conventional ones.

Keywords: SoC Debugging, JTAG, TAP controller.

1. INTRODUCTION

Relative debugging work loads tend to increase even though overall design cycles of SoC designs become shorter as design technology advances. Cost of test and debugging has been reported to take 50 to 70% of system design cost [1].The cost increase is mainly due to the migration of circuit functions from a higher level of system boards to lower levels of chips or circuit modules within a chip. JTAG-based test replaces conventional bed-of-nails of the PCB level [2-4]. An on-chip debugging unit comprises the JTAG-based test function of a target processor, which provides with not only the output monitoring but also several other features such as applications of test input, pipelining of test commands and debugging [5,6].

The performance of contemporary JTAG-based debugging system relies on host computer performance. The serial and parallel data transfer speed of host computer, for example, is a main limitation of the debugging performance. One bit output may be monitored using a 400MHz data channel, but 10Gbit data traffic should be used to monitor 32bit bus output real time which is extremely difficult without a new revolutionary interface techniques.

Substantial amount of research has been devoted to alleviate this data traffic problem of JTAG-based debugging systems. Debugging systems may be modified to accommodate new features reducing data traffic or simplifying debugging unit hardware, and thereby enhances debugging performance [7, 8].

This paper addresses the performance enhancement and simplification of the JTAG based debugging system modifying the conventional standard of IEEE 1149.1. Section II describes the overall features of a JTAG-based test and suggests some revision of debugging unit. Section III proposes a new debugging unit design, and Section IV describes the implementation details of the proposed debugging unit. Finally Section V concludes with some evaluation results.

2. DEBUGGING SYSTEM

A debugging system performs sequencing of test commands with accompanying test data and motoring internal data changes. Bi-directional data traffic is implemented between the host computer and target unit. A typical structure of the debugging unit is shown in Figure 1 consisting of three components: a host computer, a protocol converter and a test target. Test data from a GUI-based host computer is sent to the test access port (TAP) of a target via protocol converter. A TAP receives four signals of TCK(test clock), TMS(test mode select), nTRST(test reset) and TDI(test data input), and sends out TDO(test data output) [5].

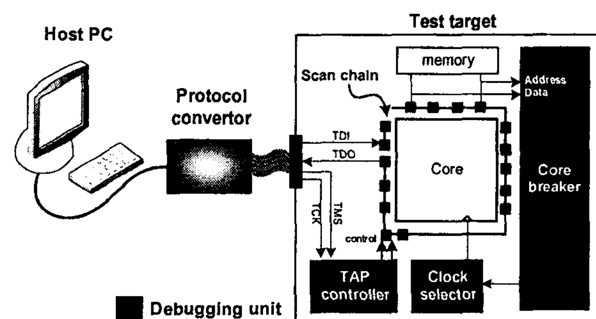

Fig. 1. Structure of a JTAG-based debugging system

Test target comprises debugging unit in addition to the original functional core. The debugging unit includes a JTAG module (TAP controller and scan chains) and a core breaker. The scan chain shifts test input to TDO pin, and is controlled by the TAP controller. Core breaker allows breaking program execution by designating breaking points. The core stops when the addresses stored in the core breaker are accessed during its normal operation. The stopped core is then in a pipelining mode controlled by the TAP controller. The clock selector transfers from the normal system clock to the debugging clock.

Overall debugging flow is shown in Figure 2: it comprises normal and debugging modes. The core breaker compares normal addresses to the pre-programmed addresses: it transfers to the debugging mode when it finds a "match." Then the TAP controller controls the scan chain and clock of the core to complete one cycle of debugging consisting of three stages: applying the incoming test data, pipelining of test sequences and shifting of intermediate results to TDO pins.

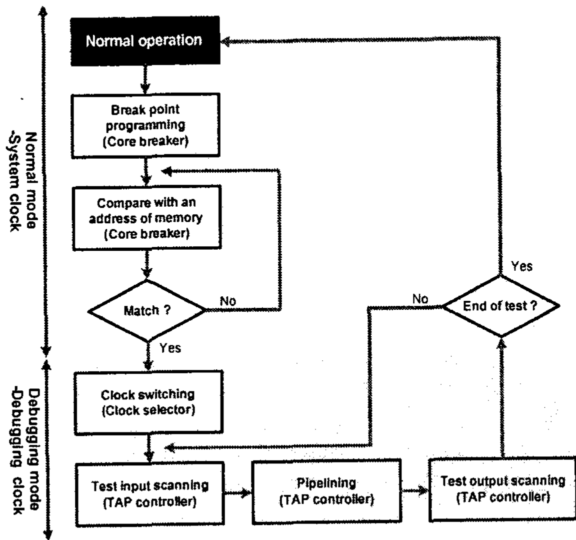

Fig. 2. Debugging flow

We now turn to the derivation of TAP controller revision for enhanced performance. The conventional TAP controller is a finite state machine having 16 states as shown in Figure 3. State transition occurs when the TCK changes from '0' to '1' and takes a flow following the TMS signal.

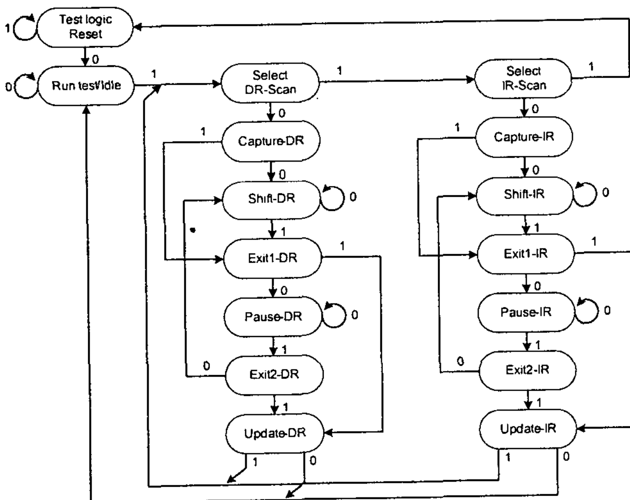

Fig. 3. State diagram of the conventional TAP controller[5]

IR (instruction register) consists of four shift register stages to transfer 4bit TAP instruction to the TAP instruction decoder. The instruction decoder generates DR select signal which select a specific scan chain data to shift toward the TDO pin. Figure 4 shows output signals from the TAP controller, which are used to control scan cells of IR and DR.

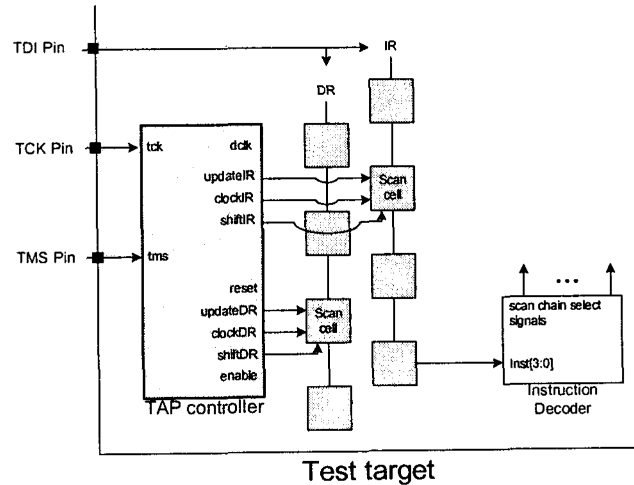

Fig. 4. Output signals from a conventional TAP controller

TAP instruction input stage is to transfer a 4bit TAP controller to the instruction decoder. Figure 5 shows a timing of the TAP input stage of a conventional TAP controller generating clockIR and updateIR signals to control scan cells of IR. Total 10 clocks are used to decode a 4bit TAP instruction. Redundant clock cycles are found in the conventional system while less than five clocks are used for storing 4bit data into a register in general.

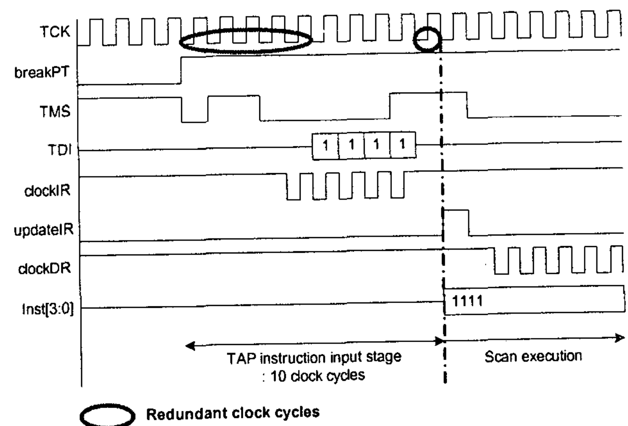

Fig. 5. Timing of a conventional TAP controller

3. PROPOSED TAP CONTROLLER

The proposed TAP controller is to eliminate the redundant clock cycles found in a conventional one. The original state diagram has been simplified as shown in Figure 6 by eliminating states to control IR. Two changes are expected from this state simplification: reduction of the number of clocks and increased debugging rates. The state transition to TEST-LOGIC-RESET state uses four clocks in the simplified version while the original one uses five clocks. Accidental TEST-LOGIC-RESET state is blocked during test since all the state transitions are controlled by the TMS signal.

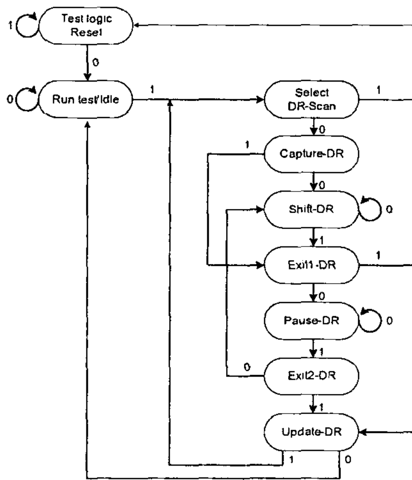

Fig. 6. Simplified state diagram of the proposed TAP controller

Substantial debugging rate increase is expected through this simplification. The minimum cycling period of RUN-TEST-IDLE state is reduced. The simplified state diagram shows three clocks for the minimum cycling whereas four clocks are used in the conventional ones yielding 25% improvement. Control signals for IR are eliminated as shown in Figure 7. A 4bit shift register replaces IR while the original TAP instruction decoder is used.

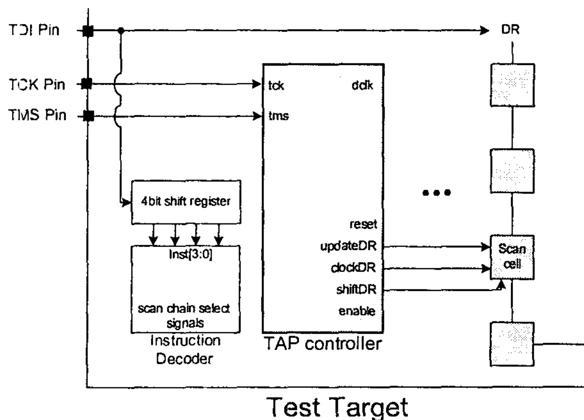

Fig. 7. Signals from the proposed debugging unit

Total time to perform the TAP instruction transfer is 5 clocks in the proposed controller, which is shorter than 10 clock periods of the conventional one. The TMS signal becomes the enable signal indicating the incoming bits from TDI form a TAP instruction. The TAP controller resets with four or more TCK clocks keeping the TMS signal at '1'. The shifted incoming bits are stored in the instruction register with the update signal from the newly introduced modulo-4 counter.

TAP instructions are inputted without any TAP controller intervention since IR is removed. Figure 9 illustrates the input circuit which takes 4-bit instructions through the TDI pin. The TAP instructions are recognized in the TEST-LOGIC-RESET state for the proposed structure whereas they are handled in the SHIFT-IR state for the conventional one. Instruction decoder takes in turn the incoming 4-bits with the update signal from the modulo-4 counter.

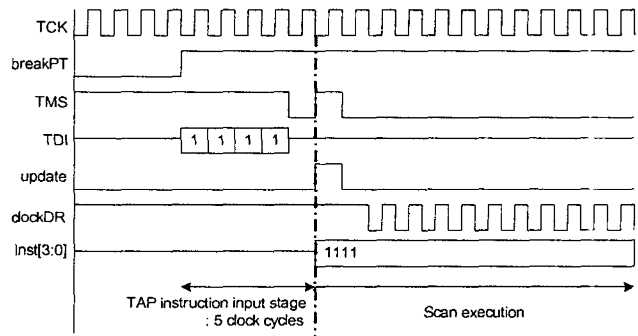

Fig. 8. Timing of the proposed TAP controller

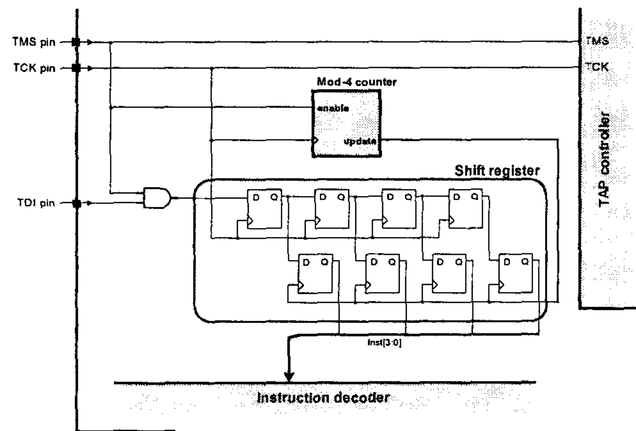

Fig. 9. The proposed TAP instruction input circuit

4. IMPLEMENTATION DETAILS

The proposed TAP controller comprising the debugging unit has been tested with a special target core designed for performance evaluation purposes as shown in Figure 10. The test target core features

essential elements of a 32bit RISC processor with a minimal instruction set. Other debugging unit components strictly comply with the IEEE 1149.1 standard otherwise specified.

Core breaker carries debugging point information during a normal operation. The debugging point is specified with a 32bit address of main memory. The TDI takes the address bits and send them to core breaker through the scan chain 2. Debugging mode begins with the high value

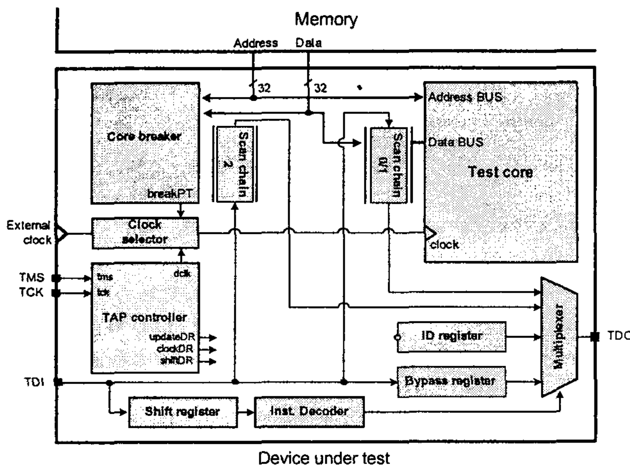

Fig. 10. The proposed JTAG test core with debugging unit

of breakPT signal which is set when the processor accesses the address stored in the core breaker. The TAP controller clock dclk takes over the role of the test core clock. This dclk clock signal goes high only when the TAP controller enters the RUN-TEST-IDLE state. The test core is controlled by the TAP controller. The scan chains 0 and 1 are to send test instruction to the test core and verify its output.

The proposed JTAG-based test target supports six TAP instructions described in Table 1: three of them (IDCODE, BYPASS and EXTEST) comply with IEEE 1149.1 and the remaining three are newly introduced in this design. Instructions IDCODE and BYPASS are to connect ID register and Bypass register to the TDI and TDO pin, respectively. EXTEST verifies internal scan chains by connecting scan chain 0 to TDI and TDO. SC1 is to shift and output 32bit data from the data bus of the test core. SC0 and SC2 are to input data from TDI to test core data bus and core breaker, respectively.

Table 1 TAP instructions and features

TAP instr.	Binary	features
IDCODE	0000	Identification of target ID values
BYPASS	0001	Skip target test (1 clock delay)
EXTEST	0010	Verification of external I/O
SC0	0011	Output of data from Scan chain 1 through TDO
SC1	0100	Scan chain 0 selection from TDI
SC2	0101	Selection of Scan chain 2 from TDI

Test vector sequences in a simulation run are shown in Figure 11. The illustrated example shows that the test core processes instructions until it finds the memory address 0x04(hexadecimal) and then verifies data at 0x01. It takes the instruction SC2 (0101) through the TDI pin in the step 1, and program 0x04. In the step 2, the test core stops after the execution of the instruction at 0x04, and input the test instruction (0xC00000001) instead of the execution of the instruction at 0x05. In the step 3, the instruction let the core load data at 0x01 into the register R0. In the step 4, data at 0x01 is moved to the memory bus and shifted toward the TDO.

Debugging step	Test vector	Description
Step 1. Break point programming	A. TDI << 0101 B. TDI << 0x00000004	A. Select scan chain 2 B. Shift 32 bits address
Core halt		
Step 2. Input test instructions	A. TDI << 0100 B. TDI << 0xC0000001	A. Select scan chain 0 B. LDR R0, x"00000001"
Step 3. Execute pipeline	TAP controller - Repeat Run test/Idle state	Input dclk to the test core
Step 4. Output scanning	TDI << 0011	Select scan chain 1

Fig. 11. Input test vector of debugging steps

Comprehensive simulation has been performed to verify the debugging unit function and to evaluate its performance. A debugging unit is attached to the target test core. The proposed circuitry has been designed using HDL, and processed using the commercial simulation tool (Modelsim 5.7d). Partial signal waveform of the step 1 and the step 2 are illustrated in Figure 12.

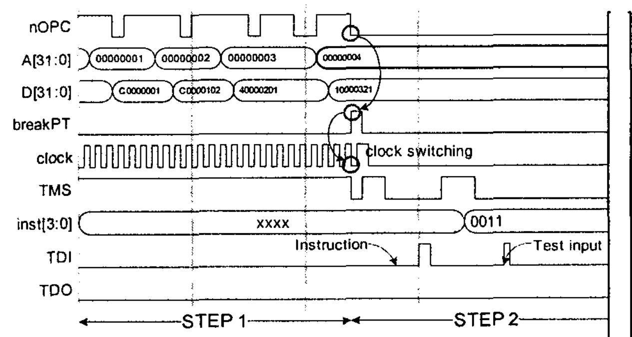

Fig. 12. Simulation results

Overall simulation results are summarized in Table 2. The simulation results do not include the step 1, where the controller selects the debugging startpoint. The simulation run on the proposed debugging unit shows some saving of five clock period at the step 2 and the

step 4 with respect to the conventional one. It is due to the simplified state transition of the TAP controller. Additional one clock period reduction is observed for repetitive Run test/Idle states. The number of the clock signal dclk reduces to three clocks, and thereby results in six clock saving in the step 3. This reduction in the number of clocks contributes 14% performance enhancement with respect to the conventional scheme.

Table 2 Simulation results

Debugging steps	Step 2	Step 3	Step 4	Overall	Relative performances
Conventional design	45	25	47	117	100%
Proposed design	40	19	42	101	114%

The circuit synthesis results are given in Table 3. The proposed unit is constructed using less gates than conventional units even though slight gate count increase in scan chains. About 50% size reduction is observed in the TAP controller whereas about 3% increase in scan chains due to the newly introduced modulo-4 counter.

Table 3 Circuit synthesis results

	Test core	Debugging unit			Cell library
		Core breaker	TAP controller	Scan chains	
Convent'l Design	14,363	1,730	330	1,894	Equivalent gate count with Xilinx library
Proposed Design	14,363	1,730	166	1,944	

5.CONCLUSION

The simplification of the TAP controller addresses the debugging speed issues of complex SoC designs. Elimination redundant state of the TAP controller is the base of this simplification resulting in a smaller but faster controller. Eliminating IR and about half of states from the TAP controller behavior does not impede the normal JTAG operations. Simulation results show 14% performance enhancement for a 32bit RISC type target core with the proposed on-chip debugging unit. The number of gates of the TAP controller reduces by half.

Multiple IP cores are accommodated A SoC system accommodates multiple core IP's, and the number of scan chains increase to debug these IP's. The TAP instructions handle larger number of bits, or introduce multiple steps keeping the fixed instruction lengths of 4bits. The multi-step approach has been employed in the debugging unit of ARM processors introducing scan chain select register. This multi-step instruction increases instruction traffic. The proposed debugging unit contributes to speed up debugging by eliminating redundant clocks.

References

[1] B. Hailpern, P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, pp. 4-12, 2002.

[2] A.J. Albee, M. Ellis, "Basic boundary-scan for in-circuit test," *IEEE Proceedings of ETC 1994*, pp. 349-354, Apr. 1993.

[3] K. P. Parker, *The Boundary Scan Handbook*, Kluwer Academic Publishers, 2003.

[4] J. Beck, R. Rose, "Integrated test logic for video IC's," *IEEE Proceedings of Test Conference 1998*, pp. 744-751. Sept. 1998.

[5] IEEE Std 1149.1-2001, *Test Port and Boundary-Scan Architecture*, IEEE, 2001.

[6] Advanced RISC Machines, *ARM7TDMI Data Sheet*, Document Number: ARMDDI0029E, http://www.arm.com.

[7] I. Huang, C. Kao and H. Chen, "A retargetable embedded in circuit emulation module for microprocessors," *IEEE Design & Test of Computers*, vol. 19, pp.28-38, Aug. 2002.

[8] C. MacNamee, D. Heffernan, "Emerging on-chip debugging techniques for real-time embedded systems," *IEE Computing & Control Eng.*, vol. 11, no. 6, pp. 295-303, Dec. 2000.