An on-chip debugging method based on bus access

Zhang Peng, Fan Xiaoya, Huang Xiaoping
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
Xteeq734801864@qq.com

***Abstract*—Supporting the online debugging is one of the design goals of SoC. Usually the function unit and the debugging structure are tightly coupled, thus it is hard to reuse the debug structure in other systems. This paper presents an on-chip debug method for SoC bus architecture. The system reuses On Chip Bus(OCB) as the transmission path for debugging data and debugs the units in system in form of bus access through debug interface. To implement debugging of embedded processor, Debug Support Unit and Debug Handle Unit is designed. The method fits for mainstream SoCs. It includes various debug functions and few limitations of debug interface, which is also very efficient. SoC of AMBA architecture with debug architecture of this kind has been implemented, and the debug function has been verified. The experiment indicates that the design satisfies the demand of SoC debugging.**

***Index Terms*—On-chip debug, SoC, On-chip bus**

I. INTRODUCTION

The development of computer architecture and VLSI technology enables the integration of more than several million transistors in a single chip to make a SoC (System-on-Chip). SoC makes the IC works as a complex system with low power consumption, but makes the verification and debug the most critical bottleneck in the chip design flow as well. Paper [1] indicates that about 70 to 80 percent of the design cycle is spent in verification and debug. So efficient debug method is essential for SoC development, it shortens the design cycle.

On Chip Debug (OCD) debugs SoC by means of inserting debugging circuit in system. Compared with other debug methods, there is lower interference on running applications, better real-time characteristics and lower development cost in OCD. Traditional OCDs are based on the IEEE 1149.1 JTAG [2] Standard, which was originally developed for boundary scan and internal device tests during the chip production, paper [3, 4] are the examples. They extend boundary scan instructions defined by JTAG and insert several test scan chains in system. When debugging, the debug host controls the write and read of the inserted scan chain by extending scan chain instructions. The state of system is modified and obtained in this way. But related literatures [5~7] indicate there are several disadvantages in traditional OCD method.

(1)Debug clock is separated from system clock. This complicates the control of clock signal [5]. As the debug clock is slower than system clock [6], debug data can't be transferred with high speed.

(2)With the increasing of debugging demand and the observation points, the resource of debugging circuit increases. Generally, more than 5% of the total area of the circuit is occupied. The debugging circuit may influence place and route of IC, even affect the performance of the system [7].

(3)Since the debug structure is tightly coupled with the target system, it's hard to reuse it in other systems.

To solve the problems, the paper presents an OCD method based on bus access. The debug structure transfers debug data through On Chip Bus, changes the mode of system when debug trigger event happens. Debugging of peripherals and embedded processor are of the same form. The debug structure can be easily reused, and it is area saving and efficient.

II. THE ARCHITECTURE OF SYSTEM WITH DEBUG STRUCTURE

The architecture of SoC with debug structure is shown in picture 1. Target SoC is a typical SoC interconnected by OCB. The processor and debug interfaces (UART, JTAG, USB and PCI) are master devices. RAM, FLASH, other peripherals and Debug Support Unit (DSU) are slave devices. Debug Handle Unit (DHU) is embedded in processor.

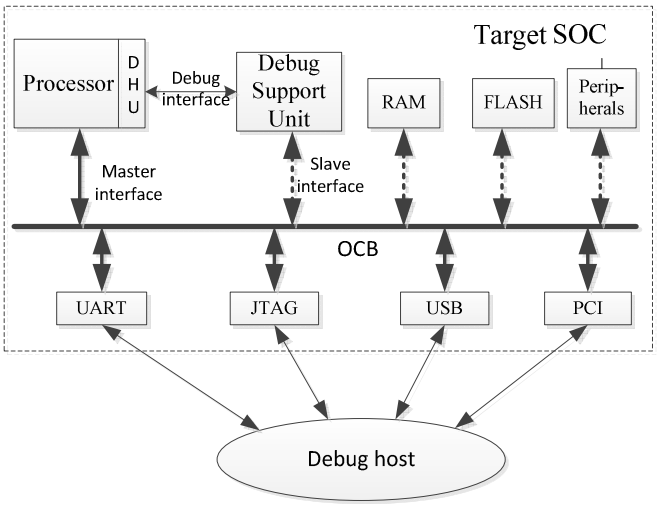

Fig. 1. SoC with debug structure integrated

978-1-4799-1027-4/13/$31.00 ©2013 IEEE

The system in figure 1 implemented the debugging of peripherals (memory included) and processor. When debugging the peripherals, the debug host sends debug command to debug interface. As a master, the debug interface accesses the peripherals through OCB. By reading and writing data, the interface capturers and modifies the state of peripherals, thus implements the debugging of peripherals. Processor debugging is the main goal of SoC debug. Since the processor is usually a master in the system, debug interface can't access the processor immediately. A slave interface must be appended to the processor. So we design the DSU and DHU. The debug of processor will be described in detail in the next chapter.

III. DEBUG OF EMBEDDED PROCESSOR

The debug structure implements three debug functions for processor. First, set debug trigger conditions and change the state of processor when trigger event happened. Second, capture or modify the state of processor when the processor is in debug mode. Third, single step debug and trace debug.

To make sure that debug behavior won't interfere the running of program, a debug mode was added to the processor. The processor state can only be accessed when the processor is in debug mode. Under some particular circumstances, the processor enters the debug mode. All the other debug functions are based on the setting of debug mode. The single step behavior is the transition of processor state between debug state and run state. The trace debug function implements the trace of two sets of significant signals: bus access signals, instruction sequence and their results. The record can only be accessed at debug state.

DSU and DHU are the essential cores for processor debug. Figure 2 shows the structure of DSU and the processor with DHU.

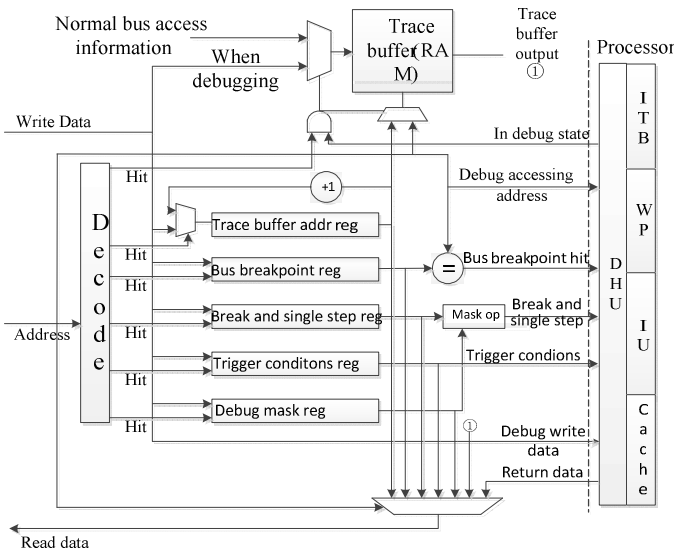

Fig. 2. Structure of DSU and Processor

AS shown in figure 2, DSU mainly contains the following components: address bus decoder, debug control registers, bus trace buffer and read data multiplexer. The processor consists of the following components: Integer Unit (IU), Cache, Watch point registers (WP), Instruction trace buffer (ITB), DHU and so on. The decoder in DSU decodes the address signal and generates the hit signals for the series of debug control register, bus trace buffer and processor access. The multiplexer selects the correct data according to the access address. The DHU controls the state transition of processor and handles the accessing of processor state. The implementation of each debug function will be introduced below

A. Debug trigger conditions and debug state

1) Debug Trigger Conditions

The trigger conditions of debug includes: software breakpoint, hardware watch point, general instruction exception and bus breakpoint. Bus breakpoint refers to a bus access address; it is saved in bus breakpoint register showed in figure 2.DSU tells whether the bus breakpoint is hit by comparing the address signal and the bus breakpoint register. The result is submitted to the processor.

By configuration, the trigger conditions can come into effect selectively. For example, when the program hits the breakpoint hits, the processor enters into debug mode or when memory access error happens, the processor enters into debug mode or if any trap occurs, the processor enters into debug mode. The configuration information is saved in debug trigger condition register. The trigger conditions are detected by DHU.

The trigger conditions belong to the generalized interruptions [8] except for bus breakpoint. If the bus breakpoint hit is defined as a kind of breakpoint detected trap, all of the trigger condition will generate the corresponding interrupt vector. DHU just needs to check whether the trigger conditions related interrupt vector equals to the actual interrupt vector to detect the trigger conditions.

2) Debug State

In general, there are two states of processor, namely the running state and the trap state. If trap occurs at running state, the processor state changes into trap state. At trap state, processor saves state register and changes the Program Counter (PC) according to the trap type. After the processor changes the context, processor goes back to running state. Figure 3 shows the state machine with debug state.

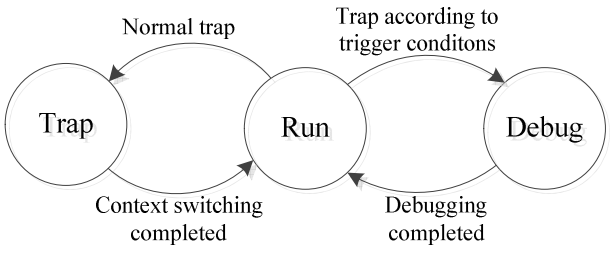

Fig. 3. State machine of processor with debug state added

As shown in Figure 3, processor state goes into debug state when the trap corresponding to trigger conditions occurs. At

debug state, the processor pipeline is held and the processor state can be accessed by the DSU. In order to keep the processor stopping at the instruction which triggers the debug behavior, some of the component in pipeline will be changed or used. There are mainly two points.

①Change the PC generating component. When processor is at debug state, keep the output of PC generating component at the debug trigger instruction's PC.

② Make use of the annulling instruction function of pipeline. After the processor enters into debug state, annul all the instructions which enter the pipeline.

For the sake of recovery from debug state, something like sport saving during exception handling should be done. During exception handling, the program field is usually saved in ISA defined registers or memory. Different from exception handling, debug behavior doesn't call exception handler, there is no need to change context. By just saving PC and NPC(next PC) in registers which are not visible for program, recovery from debug state can be easily accomplished. After debug behavior, processor stops annulling instruction and the trigger instruction and the next instruction will come into pipeline, at the same time, processor state goes back to running state.

B. Accessing of processor state

In debug state, processor state can be captured and modified. To make accessing of processor state like accessing of slaves, map the state related resource in the address space of DSU. When the accessing address is in the scope of state related resource, DSU submits the lower part of the address to DHU. DHU decode the lower part of the address to determine the exact component accessed.

The state of processor includes special function registers, general purpose registers, cache, hardware breakpoint registers and instruction trace buffer. Some of them are made up of registers, such as special function registers. The data saved in registers can be easily read out or modified immediately. Some of them are made up of SRAM, such as cache and general purpose registers. Address and control signals are needed to access them. During normal running, address and control signals are generated by specific component in the pipeline. During debugging, these signals should be generated by DHU according to the access address and control signals provided by DSU. So a multiplexer is necessary to switching signals.

C. Single step debugging and trace debugging

If single step signal in figure 2 is valid, single step debugging is activated. After the recovery from debug state, by making the second instruction the pipeline fetched (the first instruction is not annulled) generate a hitting breakpoint event, the processor goes into debug state again. Thus only one instruction is executed, single step debugging is realized.

Tracing behavior is performed when processor is running normally. When the system is running normally, once there is a bus transaction, DSU sample the bus signals and store them in bus trace buffer showed in figure 2.Once a valid instruction (not annulled for some reasons such as branch predication missing) is executed, the PC instruction type and the result are stored in the instruction trace buffer before being written back. At debug state, the history information can be obtained by accessing the bus trace buffer and instruction trace buffer.

IV. IMPLEMENTATION AND PERFORMANCE

A. Implementation

The Northwestern Polytechnical University has designed a 32-bit processor of SPARCv8 ISA which has been embedded in a SoC. The system bus is the AMBA AHB. The processor takes LEON3[10] processor as a reference. There are seven stages in the pipeline, namely Instruction Fetch (IF), Instruction Decode (ID), Register Access (RA), Execution (EX), Memory Access (MEM), Exception (TE) and Write Back (WB). The system's debug method is the method introduced above. We defined two debug related processor state dsu1 and dsu2. After the processor has entered dsu2, the processor state can be accessed.

Figure 4 and Figure 5 shows the timing of significant signals during state transiting of processor. restart_addr is the invisible register used for store PC and NPC. te_tt is the trap type register of TE stage. dbgi_dbreak signal is the debug request signal. annual_all is the annul instruction signal in the pipeline. Figure 4 is the timing diagram of entering dsu2 for the sake of breakpoint detected trap (te_tt = 0x0B). When the trigger event is detected (te_tt= 0x0B), the PC and NPC are stored in restart_addr successively; the annual_all signal is asserted at the same moment; the processor state changed from run to dsu1, at last the state changed into dsu2; the if_pc keeps the value of trigger instruction's PC. The process of exiting debug is shown in figure 5. The stop of debug is implemented by invalidating signal dbgi_dbreak. After the invalidation of dbgi_dbreak, annual_all is invalidated and the state of processor changed from dsu2 to run. The processor starts running from the breakpoint. The process of other event triggered debug is similar to the process above, so there is no need to describe them in detail.

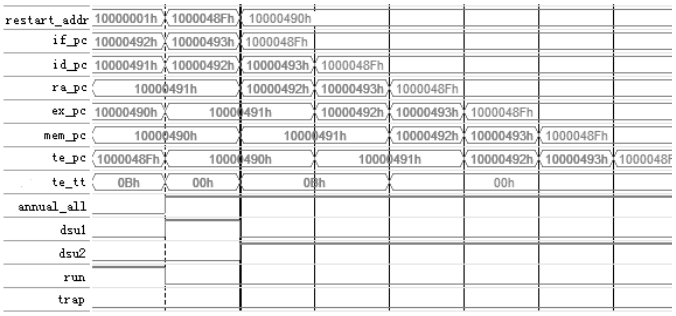

Fig. 4. The processor state transited from run to dsu2 for the sake of breakpoint detected trap.

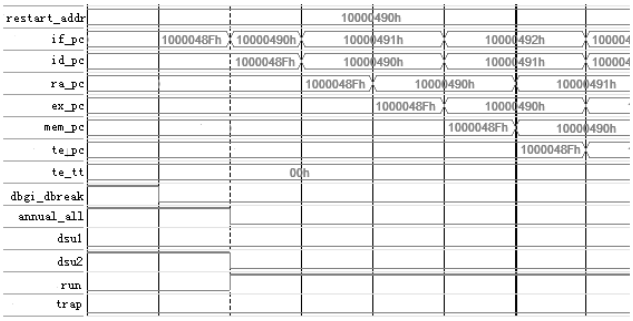

Fig. 5. The processor state transited from dsu2 to run for the invalidation of dbgi_dbreak.

B. Performance

The design has been implemented on FPGA of ALTERA and XILINX. On FPGA board DE2-115, the resource usage of the system and DSU is shown in table 1. From the data in table I, we conclude that the resource DSU used is only about 1.5% of the whole system. The max frequency of DSU is as high as 284.41MHz; it is much faster than the whole system.

TABLE I. RESOURCE USAGE OF DSU AND THE SYSTEM

Resource type	DSU/FPGA	system/FPGA
Combinational ALUTs	355 / 114480 (< 1 %)	24252 / 114480 (21 %)
Dedicated logic registers	255 / 114480 (< 1 %)	16758 / 114480 (15 %)

Table II shows the time required by all kinds of debug behaviors, among which the time of communication between debug interface and debug host is not included. The communication time is decided by the actual interface. The average time required by debug behaviors is about 2.7 system clock cycle.

TABLE II. TIME REQUIRED BY ALL KINDS OF DEBUG BEHAVIORS

Debug behaviors	System clock cycles needed
Start debug	2
Exit debug	1
Set the value of registers in DSU	3
Accessing special registers in processor	2
Accessing register file	4
Accessing AHB trace buffer	4

Table III compares the scan chain method and the method introduced in this paper in different aspects. The method proposed is much better than traditional method.

TABLE III. COMPARISON BETWEEN TWO METHODS

Item	Scan chain method	Method based on bus access
Debug interface	Usually there must be a JTAG interface.	Many common interfaces can be selected such as USB and RS232
Debug clock	Test clock, clock control logic is needed. Usually the test clock has lower frequency than system clock.	System clock, debug is faster and clock control is simple.
Debug function	Memory management, peripherals debug, processor debug e.g.	Have better performance on peripheral debug.
Reusability	Debug structure is tightly coupled with the target system. Any change of the system leads to the change of scan chains.	Demand of new system can be met by changing the accessing address, The reusability is better.
Resource usage	The resource used increases with the increasing of watch point. Usually more than 5% of the resource is occupied.	The resource used is nearly fixed. Less than 3% of the resource is used. It is resource saving.
IC implementation	There must be a test clock tree. The place and route of scan chain registers must be taken into account separately.	The place and route of the whole chip is uniform. It is easy to make IC.

V. CONCLUSIONS

Online debug is essential for SoC design. The paper proposed a debug method based on bus access. Peripherals and embedded processor are targets of debug. By mapping the state related components in the address space of DSU, processor state is accessed like the accessing of peripherals through OCB. Debug is triggered by trigger event set before debug according to user demands. Single step debug and trace debug are supported. Various debug functions meet the demands of SoC debug. The performance and reusability of the method is much better than traditional scan chain method.

ACKNOWLEDGMENT

This work is supported in part by National Natural Science Foundation of China (NSFC, Grant No.60773223, No. 61003037 and 61173047) and Basic Research Project of Northwestern Polytechnical University(JC201212 and JC20110224).

REFERENCES

[1] PurviD.Mulani. “SoC Level Verification using System Verilog :Emerging Trends in Engineering and Technology”. IEEE CA: CylinkCorporation., pp..378-380, 2009

[2] IEEE Standard 1149.1-2001, “IEEE Standard Test Access Port and Boundary-Scan Architecture.” IEEE ,2001.

[3] CHANG Zhiheng, XIAO Tiejun, SHI Shunbo. “Design of on-chip-debugger and debug system based on JTAG “Computer Engineering and Applications, vol.48,2012, pp.78-82

[4] PEND Degang, ZHOU Huiling, LIU Miao. “Research of Embedded Debug Based on JTAG”. Instrumentation, vol3. 2008, pp. 24-26

[5] GOELSK,VERMEULENB. “Hierarchical data invalidation analysis for scan based debug on multiple clock system chips” //Proc of IEEE International Test Conference. 2002, pp.1103-1110.

[6] Gao Jianliang, “Han yinhe.Survey on post silicon debug for multi core processor” Application Research of Computers, vol30, 2013, pp.322-324

[7] LI Shaoqing,DENG Qinxue. “Integrated Design of the Fault Test Structure and the Debugging Structure” COMPUTER ENGINEERING&SCIENCE. vol28, 2008, pp. 99-100

[8] Luo Lei.”Embedded Real-time Operation System and Application Development (Third Edition)” BEIHANG UNIVERSITY PRESS, Jan.2011

[9] SPARC International Inc. “The SPARC Architecture Manual Version 8”.535 Middlefield Road Suite 210Menlo ParkUSA: SPARC International, 1992.

[10] Aeroflex Gaisler. “GRLIB IP Core User’s Manual” Kungsgatan 12 SE-411 19 GoteborgSweden: AeroflexGaisler, 2010.