ICEBERG: An Embedded In-circuit Emulator Synthesizer for Microcontrollers

Ing-Jer Huang and Tai-An Lu

Institute of Computer and Information Engineering

National Sun Yat-sen University

Kaohsiung, Taiwan, R. O. C.

ijhuang@cie.nsysu.edu.tw

Abstract

This paper presents a synthesis tool ICEBERG for embedded in-circuit emulators (ICE's), that are part of the development environment for microcontroller (or microprocessor)-based systems (PIPER-II). The tool inserts and integrates the necessary in-circuit emulation circuitry into a given RTL core of a microcontroller, and thus turning the core into an embedded ICE. The ICE, based on the IEEE 1149.1 JTAG architecture, provides standard debugging mechanisms, including boundary scan paths, partial scan paths, single stepping, internal resource monitoring and modification, breakpoint detection, and mode switching between debugging and free running modes. ICEBERG has been successfully applied to synthesize the embedded ICE for an industrial microcontroller HT48100 from its RTL core.

1. Introduction

An in-circuit emulator (ICE) is part of the development environment for microcontroller (or microprocessor)-based systems (called *target systems*). During the development of the hardware and software of the target systems, the ICE replaces the microcontroller and is inserted into the slot where the microcontroller should be. The ICE, while retaining the same functionality as the original microcontroller, provides extra debugging supports such as single stepping, break point setting and detection, internal resource monitoring and modification, *etc*.

For board level target systems, the microcontroller and its corresponding ICE usually adopt different design styles: while the microcontroller being a single chip implementation, the corresponding ICE is a board level design that may even span more than one board, such as [1] and [14]. Although such approach has been in practice for quite a long time, it may become insufficient for the development of target systems with higher degree of integration such as SOC (system-on-chip) chips. A SOC chip contains within a single die multiple components, such as microcontrollers, digital signal processors, application specific integrated circuits, A/D and D/A converters, RAM, ROM, I/O interfaces, analog devices, *etc*.

The reasons are that, first, it is difficult, if not impossible, to replace the embedded microcontroller with an external ICE by probes, as in the usual practice for the board level designs. Second, the board level implementation of ICE might not be able to run at the high clock rates that are demanded by high performance SOC chips. Therefore, embedding the ICE within the microcontroller becomes a feasible solution for SOC applications, such as the ARM7TDMI embedded microcontroller [11].

However, the design of ICE's, either stand along or embedded, has been mostly a manual and ad-hoc process. In this paper, we present a methodology and the corresponding tool ICEBERG to synthesize embedded ICE's for microcontrollers. The tool accepts as inputs the RTL Verilog code of a microcontroller and a set of ICE configuration parameters, and generates as output the RTL Verilog code of the embedded ICE for the given microcontroller.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the ICE architecture. Section 4 presents the ICE synthesis. Section 5 demonstrates the technique with an industrial embedded microcontroller HT48100. Section 6 provides the conclusions for this work.

2. Related Work

The approaches to the design of ICE's are primarily ad hoc. Most of the ICE's are board level designs. They are built around extended versions of the microcontroller chips ([1] and [14]). The extended chip makes available to the outside world its internal information such as data bus and the program counter, and certain control signals through extra I/O pins that do not exist in regular chips. The other components on the circuit board of the ICE are responsible for the debugging support: such as interfacing between the host and the microcontroller chip, monitoring and recording the chip status, and controlling the behavior of the chip, *etc*. As for the single chip solutions, the ARM7TDMI processor [11] is one of the first embedded ICE's available. It is built around the JTAG architecture [7].

Scan paths have been widely adopted as a standard approach to the debugging and testing of synthesized hardware. Works in this area include Touba and Pouya [9], Fernandez and Sanchez [10], Sievert *et al*. [16], Zak Jr. and Hill [17], *etc*. Some of the works, such as [16] and [17], access the scan paths via the TAP controller [7].

Koch *et al*. [6] study the synthesis of breakpoints for a different

DAC 99, New Orleans, Louisiana

purpose. Their goal is to maintain a one-to-one correspondence between a HDL (hardware description language) code such as the VHDL code of a certain circuit and the emulation of such circuit in programmable logic devices such as FPGA. A breakpoint set in the HDL code can be mapped to a machine state in the FPGA emulation. The purpose is to debug the HDL source of the circuit, such as a microcontroller, itself. On the other hand, the purpose of ICE is to debug the hardware and software of the target system that is built around the microcontroller.

On the commercial side, some cell library venders provide cells such as scan registers and the TAP controller and the necessary tool to integrate them into a circuit (e.g., [15]). However, it is not sufficient to support the synthesis of ICE's. To design an ICE, the designers have to manually integrate these cells and build necessary control mechanism into the microcontroller core.

Compared with the related work, our tool aims at providing a total solution by not only synthesizing the individual components such as the scan paths, the TAP controller and the breakpoint detection unit, but also integrating these components into a given RTL core of a microcontroller.

3. ICE Achitecture

Similar to the ARM7TDMI microcontrollers, our ICE is based on the JTAG/IEEE 1149.1 architecture. In this section we first describe the JTAG architecture and then describe the necessary modifications to the JTAG architecture and the operation modes in order to support the ICE functionality.

3.1. JTAG/IEEE 1149.1 Architecture

The JTAG/IEEE1149.1 architecture is a framework for standardized design-for-testability of integrated circuits for module-level (e.g., board-level) testing [7]. It allows the inputs and outputs of the digital logic of the integrated circuit to be accessed from outside modules. The architecture places a boundary-scan cell adjacent to each component's input/output pin in order to observe and control the component's signals at its boundaries. The advantage is that the controllability and observability of a module containing many components is vastly improved while the input/output overhead of the module consists of only three extra input pins TCK (test clock), TMS (test mode select) and TDI (test data in), and one extra output pin TDO (test data out).

The major components of the JTAG/IEEE1149.1 architecture are the TAP (Test Access Port) controller, boundary-scan registers and some other registers, shown in Figure 2. The TAP controller controls the behavior of the architecture via the TAP instructions and the TCK and TMS signals. The TAP instructions are fed into the instruction register of the TAP controller through the TDI input pin. The boundary-scan chain register is a multiple-bit shift-register consisting of the boundary-scan cells interconnected in serial fashion with access to the component's input/output pins and internal logic. The boundary-scan register can be written through the TDI pin and read from the TDO pin.

3.2. Building an ICE upon the JTAG architecture

In the sub sections we first describe the necessary modifications to the JTAG architecture in order to support the ICE functionality. We then introduce the execution modes of ICE. Finally, we present the ICE organization and its operations.

3.2.1. Necessary modifications of the JTAG architecture

In order to implement the ICE functionality, several components in the JTAG architecture need modifying, and furthermore, some new components are also necessary. Table 1 summarizes the components that are modified or new to the JTAG architecture for the support of ICE functionality. And the connection of these components is shown in Figure 2.

Classification	Components
Original 1149.1 JTAG components	● TAP controller, ● Boundary Scan Registers, ● Bypass Register
Modified 1149.1 JTAG components	● Instruction Register, ● Instruction Decode Logic, ● Data Register Selector
New Components	● Core Data Register, ● Breakpoint Scan Register, ● Scan-Chain Select Register, ● Breakpoint Detection Unit

Table 1. ICE component classification

The new components in our architecture include additional scan-chain registers and the breakpoint detection unit. The *core data register* is a scan chain of the internal registers that the designer wants to observe. The *breakpoint scan register* is a scan chain used to shift breakpoint and its configuration parameters into the breakpoint detection unit. The *scan-chain select register* is also a scan chain used to select the desired scan-chain that users want to input or shift data.

3.2.2. Breakpoint Detection Unit

The Breakpoint detection unit (BDU) is the central part of the ICE. The BDU monitors the value(s) on the address bus and/or the data bus of the microcontroller. Once the value matches some target values, BDU stops the normal operation of the microcontroller, and the control is taken over by the TAP controller. The target values are stored in the internal registers (called *breakpoint registers*) of the BDU through the TAP controller by the user before the debugging is activated.

Two types of matches are supported: *data independent* and *data dependent*. The data independent match happens when the value of the address bus matches at least one of the target values in the breakpoint registers. The data dependent match happens only when both the values of the address bus and the data buses match the target values.

To provide further flexibility in debugging, the values can be *masked* before they are compared with the target values. The masking mechanism makes it possible to stop the microcontroller under some sophisticated conditions such as stopping when the address is word aligned, or when the data is even, or only when both conditions are met.

To achieve the masked data dependent matching, a breakpoint register consists of four fields: *target address*, *target address mask*, *target data*, and *target data mask*. To save the size of the BDU, the designer can decide whether to remove the masking and/or the

dependency mechanism by eliminating the corresponding mask and/or target data fields.

Figure 1 illustrates the relationship between the microcontroller core and the BDU. The signals *address_in* and *data_in* are controlled by the TAP controller. They are used to assign the breakpoint value to a breakpoint register that is specified by the *address_in* signal. If the microcontroller is pipelined, there are many concurrent activities in the microcontroller. In this case, the BDU should be enabled only at some specific timing in order to avoid picking up the wrong information. The signal *en_check* is used to turn on the BDU at the right cycle. When a breakpoint is matched, the signal *breakpoint* is enabled to force the ICE into the TAP mode (to be explained in Section 3.2.3).

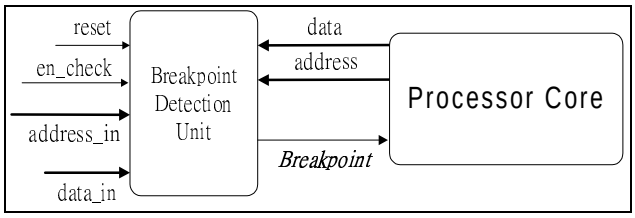


Figure 1. Connection between processor core and BDU

Depending on the memory configuration of the microcontroller core, there are some alternative designs for the BDU. If the program and data memory are separated (Harvard architecture), and the user want to detect both instruction and data breakpoints, the BDU must have breakpoint registers for both instruction and data. If the program and data memory share the same data and address bus (Voneumann architecture), then the BDU must know if the microcontroller core is in the fetch stage or execute stage. Therefore, each breakpoint register in the BDU requires an extra 1-bit field to record whether the target is for instruction or data.

In summary, the BDU can be customized by the following parameters: (1) the width of the breakpoint register, (2) with or without data dependence, (3) with or without masking, (4) the number of breakpoints, (5) Harvard or Voneumann memory architecture. These parameters have impacts on both performance and cost of the BDU.

3.2.3. ICE operation modes

Three operation modes are necessary for the embedded ICE, in order to properly control the behaviors of the microcontroller and the JTAG mechanism.

- *Monitoring mode*

In the monitoring mode, the microcontroller core is active as in the normal case. In addition, the BDU keeps monitoring the microcontroller's status, and stops the microcontroller core when the breakpoint is reached. The TAP controller is in the idle state. The microcontroller's core clock is used as the system clock, and therefore the system runs at the normal speed.

- *TAP mode*

In the TAP mode, the microcontroller core is stopped; that means the core's state will not change util the mode switch signal (*mode*) is enabled. In the TAP mode, the TCK clock of the TAP controller, which is usually slower than the core clock, is used as the system clock. During this mode, the user can observe and control the internal status of the microcontroller core through executing TAP instructions in the TAP controller.

There are two types of registers in the microcontroller core: one includes the original internal registers and the other includes the modified internal registers that are replaced by the shift-scan registers. In the TAP mode, the original internal registers are disabled. On the other hand, the shift-scan registers are controlled

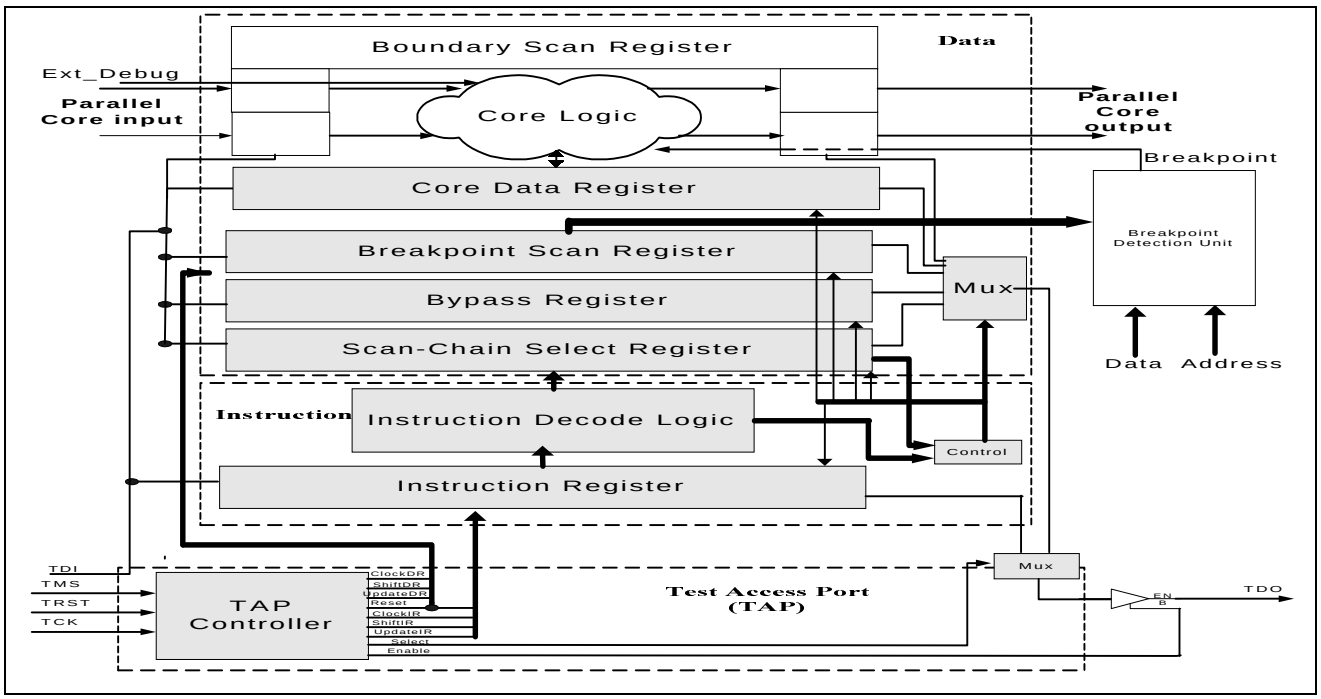


Figure 2. ICE Block Diagram

by the *ClockDR* signal that is generated by the TAP controller.

- *Free running mode*

In the free running mode, the microcontroller core is active and ignores the breakpoint signal, acting as the bare microcontroller without the ICE circuitry. The processor will run continuously until the external debug request occurs, in which case the ICE enters the TAP mode and waits for the TAP instructions for further operations.

Figure 2 illustrates how the ICE circuitry is integrated with the microcontroller core. The gray components are JTAG related components, including scan chains and the TAP controller. There are many scan chains that can be selected by the related TAP instruction. The core logic in the figure (the "cloud") is the microcontroller core. Its I/O boundary is replaced by the boundary scan registers. Some of the internal registers, which the designer decides to make observable and controllable to the outside world, are replaced by the core data register chain. The address and the data buses are connected to the BDU for comparison. The BDU is configured by the parameters stored in the breakpoint scan register. The detail connections between the microcontroller core and the BDU are shown in Figure 1. The *Ext_Debug* control signal switches the microcontroller between the free running mode and the TAP mode.

4. Synthesis of embedded ICE

In this section we present the synthesis framework and the synthesis flow for the embedded ICE described in the previous section.

4.1. Synthesis framework of microcontrollers

Figure 3 depicts the synthesis framework of our hardware/software co-design system for microcontrollers. At the left side is the behavioral synthesis tool PIPER-II [3] that produces the pipelined design at the RTL level for a given instruction set specification. In addition, it also produces an instruction-reordering table that contains constraints to guide the compiler backend to generate optimal software code to run efficiently in the synthesized pipelined design. The synthesized RTL design is ready for further synthesis by commercial gate level tools.

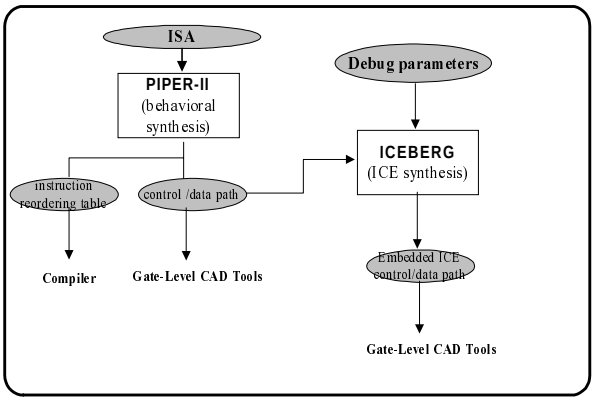


Figure 3. Hardware/software co-design system for microcontrollers

The ICEBERG at the right side is the embedded ICE synthesis tool. ICEBERG takes in the synthesized RTL design as the input and integrates ICE circuitry into the RTL design according to the ICE configuration parameters specified by the designer. The parameters include the number of the breakpoints, data dependency and masking of the breakpoints, the data bandwidth, and the internal registers to be made observable. The output is the RTL level design of the embedded ICE that is synthesizable by commercial gate level tools.

4.2. Synthesis flow of ICE

The synthesis flow of ICE consists of the following three major steps.

Register extraction and test register insertion: The first step is to search for all the registers in the original microcontroller core and chain all registers together to form a full scan chain, if the designer selects the full scan mode. Otherwise, selects the registers that are specified by the designer and forms a partial scan chain. This scan chain is the core data register in Figure 2. Synthesis tools constructed for this step are primarily pattern matching and replacement tools.

JTAG/BDU circuit insertion: In this step the ICE components, including the TAP controller and the BDU are synthesized according to the designer's specification. Synthesis tools constructed for this step are primarily parameterized module generators.

ICE integration: The last step is to integrate the ICE components and the microcontroller core. The major challenge is to activate the TAP controller, the BDU and the microcontroller core at the right timing since their operation styles are quite different. The TAP controller is basically a finite state machine that executes sequentially. The microcontroller core may be a pipelined design in which multiple instructions are executed concurrently at different portions of the hardware. The BDU is a combinational circuit that keeps monitoring the data and address buses continuously.

Figure 4 shows how the hardware components, the data and control paths of the microcontroller core, the single step counter, the BDU and the TAP controller are integrated. The BDU, single step counter, the TAP controller, and the control signals with their glue logic which provide the necessary integration mechanism are synthesized and integrated by the synthesis tool ICEBERG. In addition, ICEBERG also automatically modifies the data and control paths to accommodate and generate the ICE related control signals.

The data and address buses of the data path are monitored by the BDU. The validity of the values on the buses is indicated by the control signal *Enable_BDU_check*, which is asserted by the control path. The signal is constructed by analyzing the RTL code of the microcontroller core for the specific timing and the pipeline stage location of the memory access.

Once the breakpoint is detected, a flip-flop BF is set by the BDU. The value of the flip-flop is combined with two control signals of the TAP controller *RESTART* (reset the BDU and enter the monitoring mode) and *RESTARTF* (ignore the BDU and enter the free-running mode) and one control signal from the control path *Flush* (flush the pipeline registers and pre-fetched instructions

under pipeline hazards) to generate the *Breakpoint* signal to the control path, indicating a breakpoint is reached. The *Breakpoint* signal can also be asserted by the single step counter in the control path to mark the end of the execution of an instruction during the single stepping mode.

Once the *Breakpoint* signal is received by the control path, it completes the execution of the current instruction and raises the *Mode* signal at the very beginning of the execution of the next instruction. The signal prevents the storage elements in the data path of the microcontroller core from being updated, thus "halting" the microcontroller. The user can then access the internal status of the microcontroller through the TAP controller. The *Mode* signal is constructed by combining the *Breakpoint* signal with an internal signal of the control path that indicates the end of instruction execution.

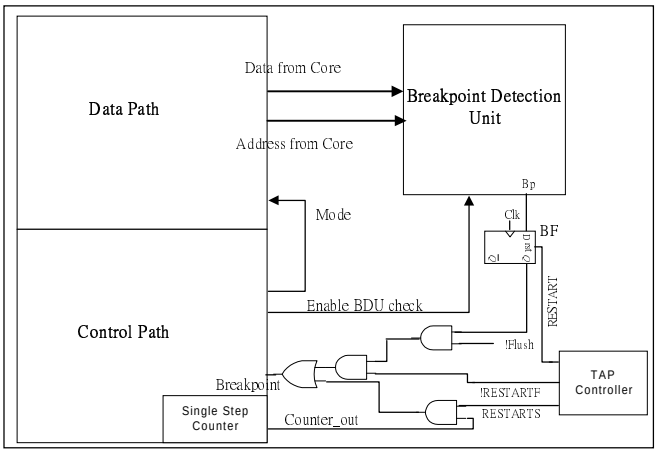


Figure 4. ICE integration

5. Synthesis of embedded ICE for the industrial microcontroller HT48100

The HT48100 microcontroller is a high performance RISC-like microcontroller specifically designed for I/O control applications, such as remote controller, fan/light/washing machine controllers, scales, and various subsystem controls. There are 64 instructions, which are executed in 1 or 2 machine cycles. The microcontroller adopts a two-staged pipeline structure. Each stage takes one cycle for execution, with each cycle being divided into four internal phases. The program memory ROM is 14-bit wide and the data memory RAM is 8-bit wide. There are 18 bi-directional I/O lines to communicate with the outside world. HT48100 supports one external interrupt input, which is processed by the interrupt control unit (ICU), and one 8-bit programmable timer/event counter (TEC) with overflow interrupting. A watchdog timer (WDT) is incorporated to prevent the software malfunction or sequence jumping to an unknown location with unpredictable results.

We have synthesized, using our synthesis tool PIPER-II, the HT48100 core from its instruction set architecture specification and also performed hardware/software co-design exploration [3]. The synthesized RTL core is called HT-SYN. In this paper, we further integrate the ICE circuitry into HT-SYN and turn it into an embedded ICE, called HT-ICE. HT-ICE includes boundary-scan

Design Name	HT-SYN	HT-ICE
Gate Count	4732	8089 (+71%)

Table 2. Gate count comparison between original design and ICE design

registers, a breakpoint detection unit of three breakpoints with data dependency. The instruction register, the program counter, stack registers, RAM data/address registers and ROM data/address registers are replaced with the test data registers. Both the synthesized RTL designs of HT-SYN and HT-ICE are further refined into gate level by Synopsys Design Compiler with COMPASS 0.6 μm 1p3m Library. Both have been verified with gate level simulation.

Table 2 shows the gate count statistics. The ICE circuitry adds 3,357 extra gates to the microcontroller core, resulting in 71% overhead. The overhead seems high because HT48100 is a compact design of an 8-bit microcontroller, which requires only less than 5000 gates. The percentage of area overhead will decrease significantly for higher bit-width microcontrollers such as the ARM7 core, which has about 190,000 gates.

How does the added ICE circuitry impact the microcontroller speed? Table 3 shows the critical path delay for HT-SYN and HT-ICE, measured by Synopsys's Design Compiler. For HT-ICE, the delay varies under different modes: monitoring mode (in which the processor core is active and the ICE monitors the system status), free running mode (in which the processor is active and the ICE does not monitor the system status), and TAP mode (in which the processor is inactive and the ICE communicates with the outside world or configures the debugging parameters through the TAP controller). The delays in the monitoring and free running modes involve the same critical path in the microcontroller core, whereas the delay in the TAP mode involves the critical path in the TAP controller. Note that we didn't attempt to optimize the critical paths during the synthesis.

Table 3 shows that the delays of monitoring and free running modes in HT-ICE are the same and are only 19% slower than the

Design	Critical Path Delay (ns)
HT_SYN	22.77
HT-ICE (monitoring mode)	27.12 (+19.1)
HT-ICE (free running mode)	27.12 (+19.1)
HT-ICE (TAP mode)	34.65 (+77.14%)

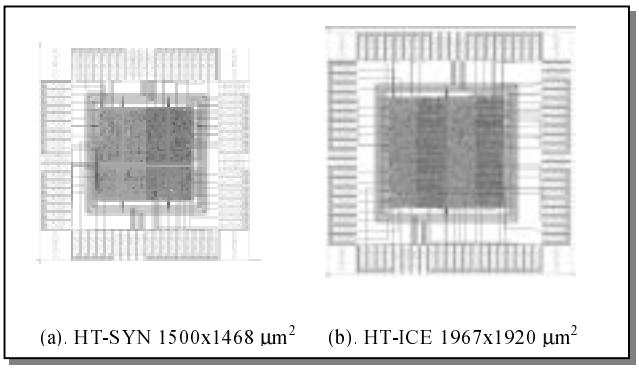
Table 3. Delay Comparison between the core without ICE (HT-SYN) and the core with ICE (HT-ICE)

delay in HT-SYN. *The result is quite satisfactory since conventional board-level ICE's are usually significantly much slower than the microcontroller cores.* In addition, the slow-down in our embedded ICE can be further reduced by carefully optimizing the ICE-involved critical paths during logic synthesis.

The delay of the TAP mode is 34.65 ns (29M Hz), which is 77% slower than the delay in HT-SYN. This is acceptable since in this mode the microcontroller is inactive; only the TAP controller is operative. The TAP controller communicates with the outside world or configures the ICE, in which situations a higher speed is

not necessary. In our current implementation, we do not attempt to optimize the delay in the TAP controller. Should a higher speed is necessary, further efforts can be taken to improve it.

Figure 5 shows the layouts of HT-SYN and HT-ICE with COMPASS 0.6 μm 1p3m standard cell library, respectively. The HT-ICE core is about 71% larger than the HT-SYN core. However, since the HT48100 microcontroller is an I/O bounded design, we expect that the final die sizes of HT-SYN and HT-ICE will be roughly the same, since HT-ICE has only five extra I/O pins (for ICE control) than HT-SYN does. The result implies that HT-ICE may replace HT-SYN as the standard product, no matter whether the ICE functionality is needed or not for a specific customer since they both require about the same die size.



(a). HT-SYN 1500x1468 μm^2 (b). HT-ICE 1967x1920 μm^2

Figure 5. Layouts of HT-SYN and HT-ICE

6. Conclusions

In this paper we have motivated the need for embedded ICE's for microcontroller-based systems with higher performance or higher degree of system integration such as SOC (system-on-chip) chips. We have described an embedded ICE architecture and its components that are based on the IEEE 1149.1 JTAG architecture. A synthesis tool ICEBERG has been constructed to automatically insert and integrate the ICE circuitry into a given RTL code of a microcontroller.

We have demonstrated the application of the synthesis tool by synthesizing the embedded ICE for an industrial microcontroller HT48100. The embedded ICE has been verified at the gate level. Both the gate level implementation and the standard cell-based VLSI implementation have been accomplished. The chip size analysis shows that embedded ICE's are comparable to and may even replace their corresponding bare microcontrollers for higher bit widths (such as 16-bit or above) or I/O pin-bounded designs, since in these designs the ICE area overhead becomes insignificant. On the other hand, the delay overhead introduced by the added ICE circuitry is kept within 20%, which is a lot better than the delay overhead of the board-level implementation of ICE's. The delay overhead can be further kept down by performing timing optimization. The speed of the embedded ICE makes it possible to perform "true" real-time emulation in system development.

Our future research directions include expanding the functionality of the embedded ICE such as providing the option of adding the trace buffer, improving the ICE architecture to further reduce the speed overhead, and simplifying the interface between the TAP controller and the host machine that controls the ICE.

Acknowledgement

The authors would like thank Mr. Guo-Chen Yu, Mr. Guo-Ming Lin, Mr. Chang-Chin Yeh and Mr. Bao-Lung Chang (former emplyee) of Holtek Semiconductor Inc. for their technical support.

Reference

[1]. HT48100 Development Data Book, Holtek Microelectonics, Dec. 1994.

[2]. I-J Huang and A. Despain, "Synthesis of Application Specific Instruction Sets," *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*,1994.

[3]. Ing-Jer Huang, Li-Rong Wang, Yu-Min Wang, "Synthesis and Analysis of an Industrial Microcontroller," *In Proceedings of Asia And South Pacific Design Automation Conference (ASP-DAC'97)*, 1997.

[4]. David W. Knapp, *Behavioral Synthesis, Digital System Design Using Synopsys Behavioral Compiler*, 1996.

[5]. "Concepts of Emulation And Analysis, Edition 1," Hewlett Packard, Nov. 1990

[6]. Gernot Koch, Udo Kebschull, Wolfgang Rosensitel, "Breakpoints and breakpoint Detection in Source Level Emulation," *International Symposium of System Synthesis,* 1996.

[7]. IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1.1a-1993.

[8]. Colin M. Maunder and Rodham E. Tulloss, "The Test Access Port and Boundary-Scan Architecture," *IEEE Computer Society Press Tutorial*, 1990.

[9]. Nur A. Touba and Bahram Pouya," Testing Embedded Cores Using Partial Isolation Rings"

[10]. V. Fernandez and P. Sanchez, " Partial Scan High-Level Synthesis," *IEEE ED&TC*, 1996.

[11]. "The ARM7TDMI Debug Architecture," *Application Note 28*, Dec. 1995.

[12]. ARM 7TDMI Data Sheet, Advanced RISC Machines Ltd., 1995.

[13]. Neil H.E. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed., P 505-508

[14]. ICE Production Information, Microtek International, *http://server3.microtek.com.tw/mice/product.html*.

[15]. Design Ware, Synopsys Corp., 1998.

[16]. K. Sievert, *et al*., "On-chip Emulation and Debugging for Embedded Microcontrollers using the IMS ScanDebugger," European Design and Test Conference, pp. 229-232, 1995.

[17]. R. Zak Jr. and Jeffrey Hill, "An IEEE 1149.1 Compliant Testability Architecture with Internal Scan," *Proceeding of Int'l Conference on Computer Design*, 1992.