崑山科技大學學報第四期 第 1~10 頁（民國 96 年 7 月）
Journal of Kun Shan University, Vol.4, pp.1~10（Jul, 2007）

A Novel Method for Real-Time Tracer Data Sequence Compression of FPGA-Based SoC On-Chip Debugger

Guo-Ruey Tsai and Min-Chuan Lin

Department of Electronic Engineering, Kun Shan University, Associate Professor

ABSTRACT

We have presented a two-level tracer data compressor design with pipeline structure for real-time data sequences compression of FPGA-based SoC on-chip debugger. A counter-based coarse level compression process is used for continuously repeating tracer data sequences, and the following fine level compression processes, including both fixed-length real-time fine compression and variable-length fine compression, are proposed to further compress the similarity bits between two neighbor tracer data of the de-repeated tracer data sequence. The two-level tracer data compressor not only is synchronous with the tracer sampling clock rate, but also needs less synthesized chip area. The compression ratio of such tracer data compressor is very high up to 100 times dependent of the tracer data channel arrangement. By the parametric hardware-description language module design, we can reconfigure flexible tracer data channel and data storage structure in order to match with different system requirements.

Keywords: SoC (System on Chip), Compression, de-Compression, logic state tracer, FPGA (Field Programmable Gate Array), on-chip debugger

1. Introduction

Embedding RISC (Reduced Instruction Set Computer) and DSP (Digital Signal Procession) cores, buses, and peripheral devices into a single FPFA chip presents a new challenge to the FPGA debugger design. Although the reconfigurable on-chip logic analyzers have been designed and proved to be effective, but they still suffer from large tracer data storage space requirement. The FPGAs are always lack of built-in SRAM, which leads to shorter trace data memory depth, so it is necessary to develop some real-time tracer data compressor in order to increase the FPGA memory usage. Developing a real-time tracer data

☆ A preliminary version of this paper was presented at the 2006 International SoC Design Conference, 26-27, Oct., Seoul, Korea.

compressor, we can enlarge trace data memory depth and make a more powerful universal In-Circuit-Debugger (ICD) for SoC system in a FPGA. If we can build in a universal ICD within the FPGA platform, as shown in Fig. 1, and, after finishing system debugging task, this temporary universal ICD will be removed and replaced by a Build-in-Self-Test (BIST) module, as shown in Fig. 2, then, this kind of universal ICD design should be compact enough to save the FPGA resource. Also, a compact real-time compressor should be designed for increasing the storage utilization for the tracer data when running the ICD, shown as Fig. 3.

The utilization of the block SRAM in FPGA chip is the key technique to the on-chip ICD design algorithm. Although current compression techniques [2-5] can achieve about 85% compression ratio, but most systems adopt off-line data compression, and use on-line hardware de-compression technique. Apparently, a real-time on-line tracer data compression and storage technique will be the key technique of extending the data memory depth of the In-Circuit-Debugger.

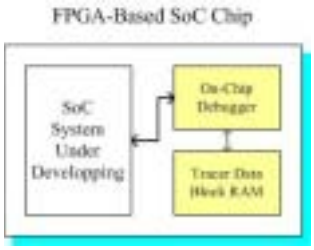

Fig. 1 A Universal In-Circuit Debugger

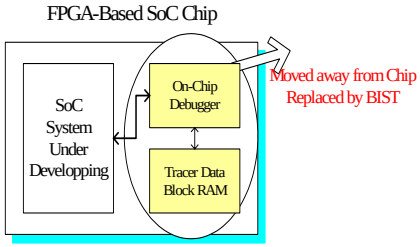

Fig. 2 The built-in ICD will be swept out after debugging.

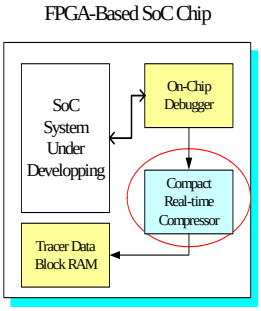

Fig. 3 a compact real-time compressor is designed to increase the storage utilization for the tracer data.

2. Real-Time Data Compression Methodology

A SoC system integrates the CPU (Central Procession Unit), memory, co-processor, and peripheral interface controller into a single chip. During the system function testing and debugging stage, we must solve sorts of problems, like glitch, stuck at logic state fault, and state transition error of FSM (finite state machine). Furthermore, we should trace the flow chart of CPU execution, and timing error from peripheral circuits. The stream tracer data always shows many duplicate data bits between two following data sets. Utilizing the high similarity of neighboring data, we propose a two-level tracer data compression method which is equipped with a compression algorithm under two different clock rates (tracer data sampling rate and data processing clock rate), in order to get more high data compression ratio.

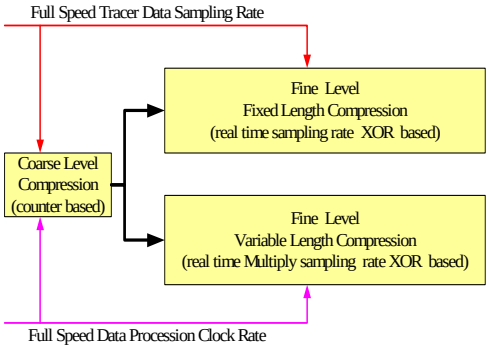

Fig. 4 Two-level compression algorithm includes first coarse level compression and the following fine level compression.

Fig. 4 demonstrates the two-level compression algorithm. At the coarse level compression process, we will use the counter to count and store the continuously repeating tracing data sequence. At fine level compression process, we utilize the similarity between neighboring data to estimate and filter out the duplicate data bits. Section 3 will introduces the counter-based coarse level compression process in detail. Section 4 will introduce a fine level compression process by fixed-length compression algorithm with full speed tracer data sampling rate. Section 5 will introduce another fine level compression process by variable-length compression algorithm with full-speed data procession clock rate.

3. Counter-based Coarse Level Compression

The main task of on-chip ICD or logic analyzer is to find out the glitch or state transition error of SoC system, therefore, the detection sampling rate of the ICD must be faster than system clock. In such a case, the normal signal tracing will produce many duplicate signals, shown as Fig. 5. In Fig. 5, we have the sampled 16 channel data sequence, such as 00AAh (repeated 9 times by sampling), 0099h (repeated 16 times by sampling), and so on. Assuming

the sampling rate is not 256 times faster than the data rate, we can take an 8-bit counter to evaluate the repeated tracer data sequence, and then get the compressed output data sequence like 0900AAh, 100099h, and so on. After coding the duplicate signal by the counter, we can generate the address and compressed data required for further data storage process.

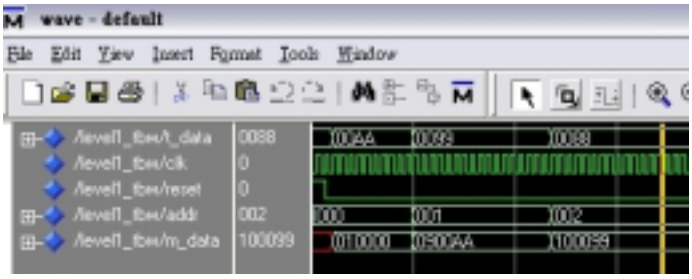

Fig. 5 Normal signal tracing shown in counter-based coarse compression.

Fig. 6 shows the data structure for counter-based coarse level compression. The output data tracer channel is (n+m)-bit width, where the counter is n-bit width, and the effective tracer data is m-bit width. The maximum compression can be calculated as the following equation:

$$\text{Compression ratio} = \frac{2^n \times m}{n+m} \qquad (1)$$

For example, we have originally 24 channel tracer data debugging system, and take an 8-bit data channel for counting, then the cost is to occupy an 8-bit tracing signal channel, and we can only have effective 16-channel for tracing data sequence. We can achieve the maximum compression ratio up to 170 (=256*16/24). With fixed counter width, the more tracing signal channel, the larger compression ratio. In fact, the counter width can adapt with the real situation, we will have the chance to choose a proper counter to achieve more satisfied compression ratio.

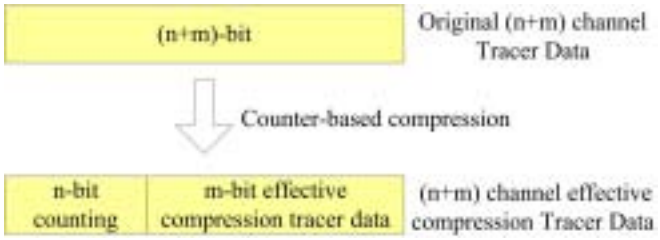

Fig. 6 Data structure of counter-based tracing data compression algorithm.

4. Level 2 Fixed-length Real-time Fine Compression Process

The coarse level compression is used to solve the duplicate sampling of the tracing data. There still exist some common properties between neighboring data stream which are output from the level 1 compression. We still have the chance for further compression. If we use an

initial data to act as the reference index, and compare the first retrieved data1 with initial data, we can calculate and get the quantity C01 of duplicate bits between MSB (Most Significance Bit) of two data sequence. Setting the bits of remaining LSB (Least Significance Bit) in tracer data1 as D01, we can integrate the C01 bits and D01 bits into a new tracing data which is the fine compressed version of the first retrieved data1. A compressed variable-length tracer data stream will be step-by-step set up according to the same algorithm, shown as Fig. 7.

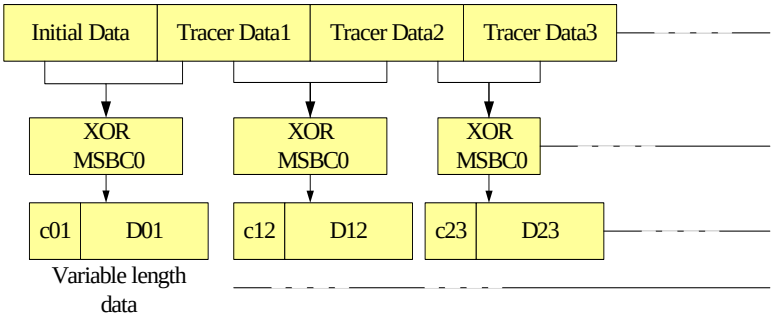

Fig. 7 Initial data as the reference index of the following fine data compression process to retrieve the first data shown as c01D01 and next data sequence.

For example, suppose a 32-bit debug channel with 32-bit tracer data sequences as the following:

Data#5 0000-0000-0000-0000-0000-0000-0000-0101(00000005h)
Data#6 0000-0000-0000-0000-0000-0000-0000-0110(00000006h)
Data#7 0000-0000-0000-0000-0000-0000-0000-0111(00000007h)

Relating Data#6 to data#5, we get 30-bit duplicate MSB bits (which are present in Data#5), and the bit number is expressed by C56=1Eh, and the bit number of remaining LSB in tracer Data#6 is represented by D56=~~0000-0000-0000-0000-0000-0000-0000-01~~10=10. The variable-length compressed data C56D56 will be 11110-10 (“1111010”) which is the compressed version of the 32-bit Data#6 (0000-0000-0000-0000-0000-0000-0000-0110 (00000006h)). By the same way, Data#7 will be compressed to a 6-bit variable-length data of 11111-1 (“111111”). When the sampling rate of the on-chip ICD is required to be synchronous with the tested circuit, the real-time compressor circuit must finish the data compression and storage process within the sampling period. Therefore, a fixed-length algorithm must be used for the required synchronous compression. A debugger memory infrastructure, shown in Fig. 8, is proposed, to rapidly real-time store the fixed-length data stream into the memory.

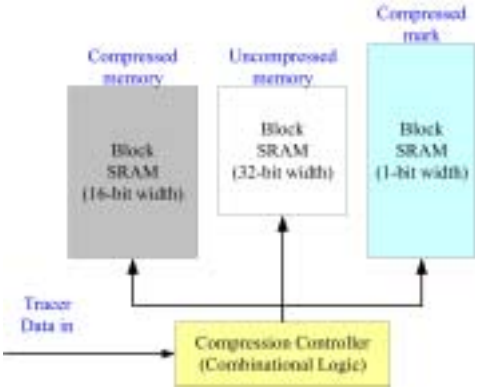

Fig. 8 The debugger memory infrastructure constituted by compressed memory, uncompressed memory, and 1-bit compression marker memory.

If the variable-length compressed data such as C01D01 and C12D12, are not larger than 16-bit, then we append some bits ‘0’ and extend them to 16-bit, store to the compressed memory zone, and mark ‘1’ to the corresponding address of the 1-bit compressing mark memory. Otherwise, the original data is stored onto the uncompressed memory zone (32 bits width), and a ‘0’ mark is stored onto the corresponding address of the 1-bit compressing mark memory. Fig. 9 shows the variable-length compressed data C56D56 (11110-10b), after appending 9 bits '0' such that the resulting tracer data length is 16-bit, the output compressed 16-bit data is “1111010000000000”, and the 1-bit compression bit is set to bit '1'. Fig. 10 shows the input compressed data sequences 0000000Ah, 00010932h, and 0000000Ch of which data length are larger than 16. The output of fixed-length algorithm will be stored onto the uncompressed memory zones and a ‘0’ mark is stored onto the corresponding address of the 1-bit compressing mark memory.

Fig. 9 the variable-length compressed data are changed to 16-bit fixed data length for real-time memory access.

Fig. 10 the variable-length compressed data are unchanged and 32-bit fixed-length data are processed for real-time memory access.

We have used Xilinx Virtex4 (xc4vsx25-10ff668) chip to implement the example system

under ISE8.1 environment. A 32-channel debugging data tracer consumes only 154 slices (1% of total 20480 slices), and speed up to 132MHz. In order to overcome the inherent gate delay latency of the FPGA place & routing circuit, we have designed a 3-stage pipeline structure to manage real-time fixed-length fine compression process. First-stage calculates duplicate bit of two neighboring data, second-stage generates fixed-length data from variable data length, and third-stage manages the data storage arrangement.

When the compressed data is fixed to 16-bit, the compression ratio is 50% above for 32-bit tracer data width. If the compression data width is fixed to 24-bit, then the compression ratio is above 75%.

5. Level 2 Variable-length Fine Compression Process

Storing variable-length data stream into data memory can not avoid bit-by-bit storing process, and a 32-bit wide uncompressed data will spend 32 clocks [6]. For 32-bit of debugging channel, the maximum variable-length compressed data of Fig. 9 will be 37-bit, so the data storing speed of the various length data must be 37 times larger than the sampling clock of the debugger at least. Here, we propose a FIFO (First-In-First-Out) ring buffer cooperating with finite state machine-based compression processor, shown as Fig. 11, which needs only 2 clocks for storing process.

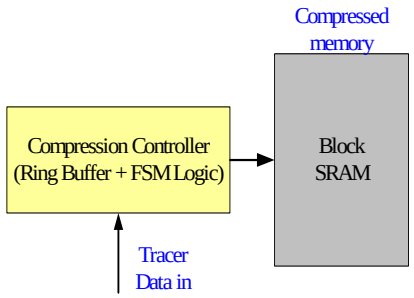

Fig. 11 FIFO ring buffer cooperating with finite state machine-based compression processor.

Fig. 12 shows the data structure of the ring FIFO data buffer, two dynamic data pointer, and FSM state transition process. At first FSM state, we put fixed-length data word into the ring FIFO data buffer, and then update last data pointer (pointer_l) according to Cxx duplicate bits count of Fig. 8. At second FSM state, we figure out effective data length in ring FIFO data buffer, if it is larger than memory access word then we proceed memory store access, and update first data pointer (pointer_s).

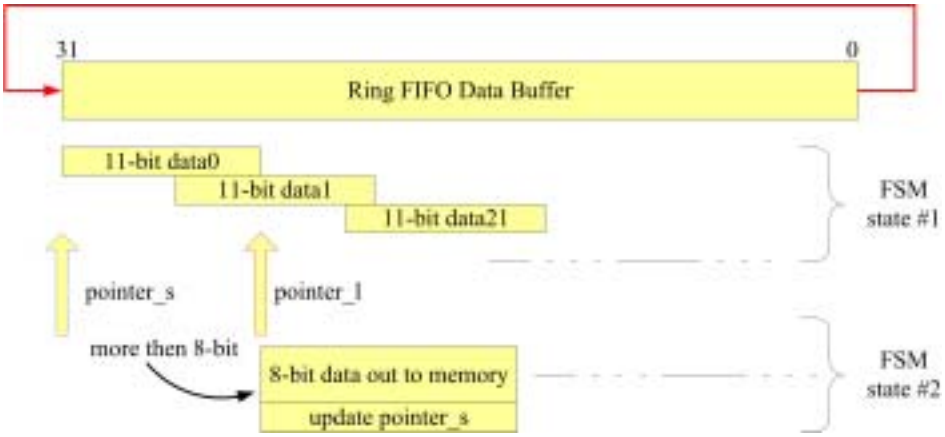

Fig. 12 Data structure of ring FIFO data buffer, two dynamic data pointer, and FSM state transition process.

For an 8-bit debugging tracing data sequence with 3-bit duplicate bits count, the maximum variable-length compressed data is 11-bit, so, ring FIFO data buffer will process 11-bit data word for input data as shown in Fig. 13. Suppose we have variable-length compression input data sequence, such as 1001110-0000, 1001110-0000, 101111-00000, 01101100-000, 1111-0000000, 1111-0000000, 1111-0000000, 1111-0000000, as shown in Fig. 13, then the simulation result shows series of compressed memory data byte as 1001110-1, 001110-10, 1111-0110, 1100-1111, (1100-1111, 1100-1111), 11111111, and so on.

The total original data sequence has 8x8=64 bits, and the total compressed data sequence will have less than 5x8=40 bits. The compression ratio is acceptable according to such a simulation result. The more data channel number is, the more data compression ratio will be achieved.

We have used Xilinx Virtex4 (xc4vsx25-10ff668) chip to implement the example system under ISE8.1 environment. An 8-channel debugging data tracer consumes only 119 slices (less than 1% of total 20480 slices), and speed up to 206MHz.

Combining duplicate bits elimination algorithm and FIFO ring buffer data write-back FSM algorithm, the tracer data compression ratio of variable-length fine compression should be higher than that of fixed-length fine compression. The main drawback is that we can not complete the data compression and storage within the tracer sampling clock.

Fig. 13 Simulation result of variable-length Fine Compression Process

6. Conclusions

We have proposed a real-time on chip universal debugger with tracer data compression processor. The compression ratio can be up to 100 times with very small FPGA resource consumption.

At counter-based coarse level compression stage, an 8-bit compressing counter can achieve a compression ratio of 255. For full speed real-time fine compression processor process, the fixed-length memory access architecture can achieve higher compression ratio when the two neighboring data MSB have the same bits with higher probability. When the variable-length memory data access algorithm adopts lower tracer data sampling rate, we can save tracer memory usage and greatly increase data compression ratio. With the data structure and dynamic data pointer FSM algorithm, we have achieved a novel result that the compression processing clock only needs two times quicker than the system clock of the SoC application system under debugging.

7. Acknowledgement

This work was supported in part by the Taiwan National Science Council under Grant NSC-94-2215-E-168-005.

8. References

[1]. Guo-Ruey Tsai, Min-Chuan Lin, Wen-Zong Tung, Kai-Chu Chuang, Sung-Yu Chan, (2003), “An Universal Debugger for Embedded SoPC Development”, 2003 International conference on Informatics, Cybernetics and Systems, I-Shou, ROC., pp84-89.

[2]. Farzin Karimi, Zainalabedin Navabi, Waleed M. Meleis, Fabrizio Lombardi, (2004),“Using Data Compression in Automatic Test Equipment for System-on-Chip Testing”, IEEE Trans. on Instrumentation & Measurement, Vol. 53, No. 2, pp308-317.

[3]. Luca Benini, Davide Bruni, Alberto Macii, Enrico Macii, (2004), “Memory Energy Minimization by Data Compression: Algorithm, Architectures and Implementation”, IEEE Trans. on VLSI System, Vol. 12, No. 3, pp255-268.

[4]. Eric E. Johnson, Jiheng Ha, M. Baqar Zaidi, (2001), “Lossless Trace Compression”, IEEE Trans. on Computers, Vol.50, No.2, pp158-173.

[5]. D. A. Huffmann, (1952), “A Method for construction of minimum redundancy codes”, Proc. IRE, Vol.40, No.9, pp1098-1101.

[6]. J. Ziv, A. Lempel, (1978), “Compression of individual sequences via variable coding”, IEEE Trans. Inform. Theory, Vol. 24, pp530-536

SoC晶片內部除錯器的即時資料壓縮技術

蔡國瑞　　林明權

崑山科技大學電子工程系

摘要

晶片內建型的萬用除錯器(On Chip Debugger)是暫時為了 SoC 系統除錯而共同合成於同一顆 FPGA 晶片上，一旦完成除錯任務後，就不再需要此萬用除錯器，而必須由應用系統中移除，因此這個萬用除錯器不可以耗損太多的 FPGA 晶片資源。然而為了能夠加深萬用除錯器的追蹤資料的儲存容量，就需要一個能夠即時的追蹤資料之壓縮技術，來幫助 FPGA-based 內嵌 CPU 之 SoC 數位邏輯系統的除錯功能。

本文提出一種具管線結構之雙階層追蹤資料壓縮技術，可用作 FPGA 系統晶片之內建除錯器所需之即時資料序列壓縮。第一層之計數器型式粗壓縮程序被用來處理追蹤資料序列之連續重覆特性，第二層之精細壓縮程序有兩種：固定資料長度即時壓縮與可變資料長度即時壓縮，被用來進一步處理兩組相鄰追蹤資料序列之相似位元。本壓縮技術不僅與追蹤器取樣頻率同步，而且需要較少的晶片合成面積。此種追蹤資料壓縮器的壓縮比與追蹤資料頻道之安排有關，可高達 100 倍以上。透過參數化硬體描述語言模組之設計,我們可以彈性重組追蹤資料頻道與資料儲存結構，以匹配不同之系統需求。

關鍵詞：系統晶片、壓縮、晶片內建除錯器、邏輯狀態追蹤