Reconfigurable Real-Time Address Trace Compressor for Embedded Microprocessors

Shyh-Ming Huang, Ing-Jer Huang and Chung-Fu Kao

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
E-mail: smhuang@esl.cse.nsysu.edu.tw

Abstract

Address trace compression represents that the address data, which were generated from instruction fetch stage of microprocessor, can be retrieved for later observation and analysis. In this paper, we present how to design and implement this real-time address trace compressor (RTATC), which can be used to collect the address trace information of FPGA Prototyping system based on embedded microprocessors. Address trace compressor is able to monitor address bus of microprocessor, and to get operation workload under the analysis of profiling software. Address trace compressor is also allowed to perform accurate, successive trace collection in an unlimited length and can be used in different embedded microprocessors. At last, by the help of FPGA, this silicon intelligent property can own the abundant reconfigurable parameters, the removable modules and the reusable ability.

1. INTRODUCTION

In 1999, Motorola Corporation and a number of chief electronic factories established IEEE-ISTO Nexus 5001 Forum [1] to draw up the related standards of dynamic debugging. The so-called dynamic debugging means not stopping the clock pulses to microprocessor and providing system developer with a well observation into the internal statuses of system-on-chip. However, this standard is faced plenty of difficult design challenges. A primary one of these objectives is how to catch up the operation statuses with kept the original internal architecture of chip non-intrusively and not made the operation speed of microprocessor descended.

For supporting dynamic debugging, we need to develop the address trace compression technique. Address trace compression represents that the address data, which were generated from instruction fetch stage of microprocessor, can be retrieved for later observation and analysis. Because the address information generated from the instruction fetched cycle can represent the execution records of computer program, it not only can provide the system developer with the references of debugging program, but also be able to offer the analysis of computer architecture. For achieving the real-time requirement, the operation clock of microprocessor cannot be halted when running address trace. Hence, the address compression technique will play an important role in determining whether the address trace is successful.

The reason why the address information needs to be compressed is because the operation speed of microprocessor is very fast; it will generate a huge amount of address trace information. Owing to the massive data to be generated, the capacity of storage media must be large enough so as to store the longer execution records of program. In addition, the internal data transfer rate of storage systems also must be large enough in order to transfer these massive data. Hence, we may require widening the number of pin counts or extending the transmission cycles for the purpose of completing the transmission of a large number of data. Nevertheless, for integrated circuit, the pin counts are very expensive resource. Therefore, confronted the above difficult issues, we have to investigate how to filter the unwanted address information via the properties of computer architecture to perform an effective compression to let the system reach the objective of saving the transmission bandwidth and reducing the storage space of memory. This is a very important motivation to our research.

The rest of this paper is organized as follows. Section 2 provides the overview of address tracing technique and lossless data compression. Section 3 describes the principle and architecture of real-time address trace compressor (RTATC). Section 4 discusses its reconfigurable ability. Section 5 provides the verification strategy to reconfigurable ability. Section 6 shows experiment and results. Finally, Section 7 concludes the paper with a discussion on future work.

2. Related Work

Due to the address tracing technique been a very important research in computer architecture related field, the adjustments of various system architectures must reply on this analysis of the address tracing information. Hence, there are a great many of approaches to discuss this technique in the past. In the following, we will take arrangements to the past address tracing research.

The address trace technique has been developed for many years. As shown in figure 1, this taxonomy of address tracing technique was expanded from Craig [2]. It mainly can divide the address trace technique into two parts that are software trace technique [3][4][5][6] and hardware trace technique, respectively. Using software technique, its advantage is easy to design, but it will encounter the problem of time dilation so that it cannot match the real-time requirement. The so-called time dilation means that the overall execution time of original program is exceeding because the execution records of running program will be saved into the storage systems and this extra action will cause the overall execution time been expended. If using hardware microcode technique [7][8], the target machine must support this microcode architecture but embedded trace unit doesn't have this constraint. Accordingly, we will adopt the hardware technique of embedded trace unit [9] here to design our real-time trace compressor.

Address trace technique based upon the hardware implementation is to provide system-on-chip (SoC) with a well observation of internal bus. Seeing that current operation speed of microprocessor is becoming faster and faster but the address compression capability is not promoted obviously, so we will discuss whether there is any address trace technique been able to refine it. Besides, we hope that this can provide SoC developer with an advanced debugging tool.

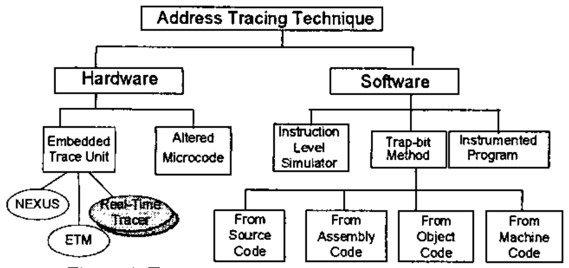

Figure 1. Taxonomy of Address Tracing Technique

3. Principle and Architecture of RTATC

3.1 Fundamental Principle of RTATC

Real-Time Address Trace Compressor is to exploit the basic characteristic of computer architecture to carry out compression. This characteristic is namely locality principle that was used in the design of cache memory. As locality principle [10] proposed, the execution of common computer program will include two kinds of localities that are temporal locality and spatial locality, respectively. The temporal locality denotes that if data items were fetched by program right now, the other ones near these data items will be fetched soon. Before discussion, we first define what "Run Block" is. During execution of program, there are some small code segments that were often executed repeatedly. These small code segments usually have continuous property that their address value is increased by offset. These code segments are called run block or basic block. A common program is usually composed of many various run blocks. Each oblique line shown in figure 2 is namely run block.

The characteristic of run block was beneficial for dealing with compression. As shown in figure 2, it is a simplified model that is a program running for 26 time units. While program is running from 13 to 15 in time unit, the address from 18 to 22 will be successively fetched by increment value. This situation comes from the property of spatial locality. Since these continuous fetched addresses were increased by a constant value, the trace compressor only records the discontinuous parts so as to reach the objective of address trace compression. At this position of the 15th time unit, there is a branch jump point that is so-called discontinuous point, so the trace compressor can record it and drop the other continuous parts. For example, while program is running from 13 to 17 in time unit, the addresses generated by microprocessor will be 6 pieces of data that are 18, 20, 22, 27, 28 and 29, separately. If recording the branch address, we only need to transfer 4 pieces of data that are 18, 22, 27 and 29, respectively. As for run block whose time unit is from 17 to 24, the address from 24 to 26 will be fetched repeatedly. This situation comes from the principle of temporal locality. Because some run blocks will be executed repeatedly, we are able to compress again those run blocks that were appearing repeatedly. For example, at the position of time units from 17 to 24, the addresses generated by microprocessor are totally 8 pieces of data that are 24, 26, 24, 26, 24, 26, 24, 26, respectively. After encoded, they become 3 pieces of data that are 24, 26, 4, separately. These encoded lists can represent that the run blocks composed of 24 and 26 and were repeated for 4 times. Therefore, the counts of data prepared to transmit will be reduced substantially. Due to the distance of branch jump address having two types that are far distance and near one, the branch instruction jumped to near address should be able to be compressed further.

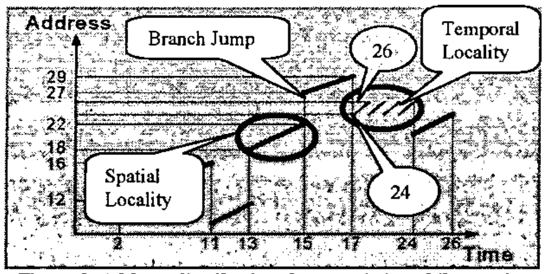

Figure 2. Address distribution characteristics while running program

3.2 Compression Procedures of RTATC

For achieving the effect of high compression ratio, this paper will divide the trace compressor into three phases to perform real-time address compression. At first, the address data generated from address bus of microprocessor will be sent to non-sequential PC filtering phase to reserve the non-sequential portions. Then, pass to PC pattern reduction phase to eliminate the needless parts from address pattern by means of the similarity of address pattern. At last, to use the lossless compressor perform further compression. The primary function of this program flow compressor is focused on offering the real-time compression on the address information generated from microprocessor. The primary aim to utilize these compression phases is to reduce the transmission bandwidth. We will give more descriptions on each compression phase in the below subsections to see whether this design is really work.

3.2.1 Non-Sequential PC Filtering Phase

In this phase, the primary manner is to record the non-sequential jumping address only. In virtual of the computer program usually having the property of spatial locality, each adjacent address pattern often is increased by a sequential offset. Hence, we can choose to only record the adjacent address pattern increased by a non-sequential offset, also called branch jump address, so as to eliminate the most address information. There are two kinds of approaches been able to satisfy this demand. One is directly to observe the instruction code and to determine whether or not the instruction is branch jump instruction. The other is to perform simple subtraction calculation via the offset of adjacent address. If offset of address is increased by a non-sequential increment, it represent that a branch jump is occurred. In the below, we will get more descriptions to these two methods.

Method 1. Instruction Decoding Approach

This method is to directly observe whether the instruction is branch jump instruction. To adopt this method, we have to setup an instruction decoder to recognize the branch jump instructions from the instruction bus of target microprocessor. The branch jump instruction is composed of both direct branch and indirect branch. The next jumping address of direct branch instruction was recorded at its instruction code. However, the jumping address of indirect branch instruction cannot be realized from its instruction code. The real next jumping address only can be obtained while microprocessor has performed to this instruction. We take instruction set of ARM7TDMI [11] for illustration. The mnemonic code called B is a direct branch instruction. Hence, for a simple code statement, B 8000h, we can map it to a micro operation, which is PC $\leftarrow$ 8000h. To observe this micro operation, we can know that the next jumping address is directly recorded at the instruction code. Instead, the mnemonic code called LDR may be an indirect branch instruction. For instance, the following is a code segment of ARM7TDMI assembly code. The left statements are some instructions ready for execution, and the right ones are the micro operations of these instructions.

```
ADD R1, R2, R3   ; R1 ← R2 + R3
STR R1, LABEL    ; M [LABEL] ← R1
LDR PC, LABEL    ; PC ● M [LABEL]
```

From these micro operations, we can know that the next address, after performed the LDR statement, comes from the sum of R2 and R3. However, this jumping address cannot be realized from the instruction code of LDR until the microprocessor has performed to LDR instruction completely. We call this kind of instruction "indirect branch". Hence, for indirect branch instruction, if decompression system cannot simulate the internal operations of microprocessor accurately, the decompression system cannot get the next jumping address while decompression. Accordingly, while performed compression, we have to keep track of both indirect branch instruction and next address so that the decompression system can get the correct next jumping address for restoring the original address trace information. For direct branch instruction, the next jumping address has been recorded in its instruction code, so we can get the next jumping address from the instruction code itself without recording the next jumping address while compression.

Method 2. Offset Detection Approach

In this method, we want this hardware to have reusable ability, so the compression procedure has to be irrelevant to instruction set architecture of any specific microprocessor. The basic reusable architecture is shown in figure 3. For owning the reusable ability, we have to adopt essential strategies to achieve this effect. At first, in this design, whether the current instruction is direct branch or indirect branch, this phase will select two kinds of address to be recorded. One is the branch jump address, and the other is the next address of this branch jump address. Besides, the current offset prepared to be sent into comparator is obtained from a subtractor rather than to look up the decode value from each instruction code. Also, the sequential offset register also can be set from outside regarding characteristics of various microprocessors.

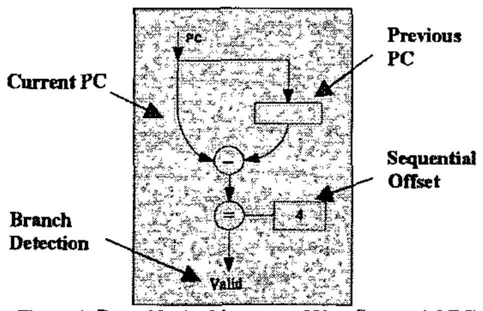

Figure 3. Reusable Architecture of Non-Sequential PC Filtering Phase

Evaluation

We will choose a suitable trace compression strategy for our design from the evaluating result of the above two kinds of methods. We find that the first method could get the higher compression rate, but its disadvantage is that this method has to design its hardware according to architecture of various microprocessors. Although the disadvantage of the second method is its lower compression ratio, it has the reusable characteristic instead. Here, we take the reusable ability as the first consideration to meet the advantage of FPGA prototyping system, so we choose the second method to be our compression strategy during non-sequential PC filtering phase. It deserves to be mentioned that this phase was often appeared in the design of current existing systems.

3.2.2 PC Pattern Reduction Phase

During PC Pattern Reduction phase, we want to perform the compression according to the address pattern characteristic and to observe whether there are any redundant parts been able to be eliminated during this phase. We have investigated two kinds of methods. One is to eliminate the similar part between last address and current address. The other is to obtain the differential value between last address and current address. We will get more descriptions to these approaches in the following.

Method 1. Slicing Approach

During PC pattern reduction phase, we want to perform compression regarding basic characteristics of address pattern and to observe which portion being able to be eliminated. The higher part between two adjacent address patterns is almost similar. Hence, if we can make the similarity between the current address and the previous address to be eliminated, the compression ratio may be rising dramatically. For example, there are two adjacent address patterns that are 0000255D8h and 000255ECh, respectively. The higher part of these two address patterns, 000255h, is redundancy. Thus, while completing the transmission of the former address, 000255D8h, we next just transfer this data, ECh. The original idea is to segment the address pattern to several equal-size slices, as shown in figure 4. Besides, for recognized whether these slices belong to the identical address, we have to apply the information of connection bit in this slice pattern. In other words, if most significant bit (MSB) of slice is logic 1, it represent that the current slice and the upcoming next slice belongs to the same address pattern.

If some slices between the adjacent addresses are the same, we can drop them and only transfer the slice with different bit streams to achieve the compression effect. The compression example is shown in table 1. The leftest part is two address trace patterns with 8 bits in address width. The middle part is the process to segment address pattern and eliminate the redundant parts. The pattern is becoming to four sets of slices in this case. In the most right of example in table 1, to remove the redundant address pattern, we indeed can make the compression ratio rise. Due to the limitation of real-time property, the number of address patterns to be sliced has to follow the rule of theorem 1.

Theorem 1. *For a microprocessor with number of pipeline being equal to n, if branch occurs and the number of instructions to be flushed is m, the maximum number of address patterns to be sliced during PC Pattern Reduction Phase will be m + 1, where n and m are belong to natural number and m is less than n.*

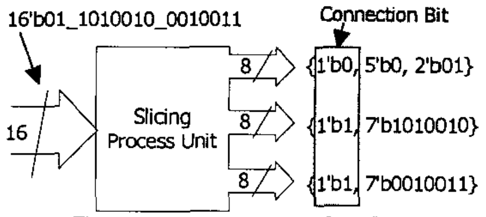

Figure 4. Segmentation of Address Pattern

Items	Address Value	Slicing	Compressed Slices
Previous PC	01101101	01 10 11 01	001 110 111 101
Current PC	01101111	~~01~~ ~~10~~ ~~11~~ 11	011

Table1. Example of Segmentation

Method 2. Differential Approach

The other method is to obtain the differential value between the last address and current address. Owing to two addresses being very close, the differential value between them is not very large. Hence, to transfer differential value will have better efficiency than to transfer the overall address pattern. In other words, this method can also raise the overall compression rate as slicing approach.

Evaluation

To evaluate the above two kinds of approaches, we can find that the compression rate of those two kinds method are both very similar. However, the advantage of this first method to eliminate the similar part of address pattern is able to save the output bus width. Because the bus widths of embedded microprocessor become larger and larger, this is a very serious overhead to integrated circuits. By utilizing the first method, we can solve this problem. The disadvantage of the first method is that it requires building trace buffer inside. This problem is arising that the slices become lot after segmentation, but some slices cannot be transferred at this clock. Hence, slicing approach requires an internal FIFO to store these slices for temporal storage. Although the second method doesn't need to build trace buffer inside, how to eliminate the overall output bus width will become a difficult problem. Therefore, we will adopt the first method here for the compression strategy during PC Pattern Reduction Phase.

3.2.3 Data Compression Phase

The last phase is to use the famous lossless data compressor for address trace compression. According to the characteristics of address trace compression, the selection principles of the lossless data compression are as following: Owing to the processing target been the real-time transmission bus, if the operation time of data compressor is so long that the overall execution time of program is extended, this is against our design requirement of real-time trace. Besides, the real-time trace compressor was implemented in hardware chip. If the area of compressor is too large, it will cause the fabrication cost rising to make the address trace compression been not able to apply in real design. The principle of this phase comes from the fact that there are the many loop statements in program, and the execution of these loops will form many repeatedly no-sequential address pattern after the above two compression phases. This situation is called the principle of temporal locality. For these addresses that occurred repeatedly, they have quite potential to make more compression further. Hence, we have to choose a suitable lossless compressor to utilize the compression of current phase according to the above basic requirement. For these existed lossless data compressors, Lempel-Ziv Compressor was the most suitable compressor for the above selection principles. Because the time complexity of Lempel-Ziv Compressor to perform compression is O(n) [13], it can obtain the effective compression rate near 1:10 left to the repeatedly sub-strings. Besides, there is one more important characteristic. This is that it can be implemented to hardware easily, and remain the cost effective in hardware area.

Hence, we intend using Lempel-Ziv compressor to perform lossless data compression. First of all, we describe the fundamental operation of Lempel-Ziv compressor. As shown in figure 5, the right side in this diagram is the source window, and the left side is the sliding dictionary. The source window is used to illustrate how the word entered into compressor one by one. The leftest of source window is a word being about to enter into compressor. We use the gray block to express it, and call it candidate word. The word at right side of candidate word will enter into gray block to become the next candidate word at next clock. This sliding dictionary is composed of several serial-in serial-out registers. These shifting registers will move one by one from right to left at each clock. Each register will include an additional tag bit to record whether the content of word in this register is the same as the candidate word. If being equal, this tag bit will be set to logic 1. In this figure, we express this concept to gray block in the sliding dictionary. If being different, we will set this tag bit to logic 0. While compressor cannot find any word that is the same as candidate word, LZ compressor will drive the output logic to generate a codeword, and clear all tag bits in these shifting registers in the same time. This candidate word will be transferred to output logic to generate the encoded codeword that is C=(Cp, Cl, Cn). The encoded codeword is composed of three parameters. From left to right are Cp, Cl and Cn, respectively. Cp indicates that the successful matching word in sliding dictionary is located at which position before outputted codeword. Cl is word number of successful matching. Cn is candidate word that fails to match at last clock.

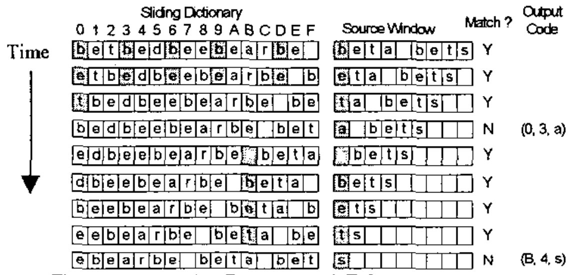

Figure 5. Operation Principle of LZ Compressor [12]

For understanding this operation procedure more clearly, we use figure 5 to explain it. At first, the left side of sliding dictionary will fills with 16 symbols, which are "betbedbeebearbe". At source window, there are 9 symbols which are "beta bets" preparing to enter into sliding dictionary from right to left. At beginning, the candidate word in source window is 'b'. Hence, if the value of the register in the sliding dictionary, which is 'b', its tag bit will be set to logic 1 and it will be labeled as gray block in this figure. At next clock, the register content of the sliding dictionary and the source window will be moved to left one by one simultaneously. Again, the candidate word in source window is 'e'. Hence, if the value of the register in the sliding window, which is 'e', its tag value will be set to logic 1 and it will be labeled as gray block in the figure. Once again, the candidate word in source window is 't'. Hence, if the value of the register of sliding window, which is 't', its tag value will be set to logic 1 and it will be labeled as gray block in the figure. At last, the candidate word of next clock in source window is 'a', and cannot be found in the sliding dictionary. Hence, clear all the tag bit to zero and output the encoding result. In the example, the codeword will be (0, 3, a). This accounts for one fact that the last successful matching word is located at the position zero in sliding dictionary and occupied for 3 time units with a last unmatched candidate word, named 'a'. In the same way, the next output codeword will be (b, 4, s). This accounts for one fact that the last successful matching word is located at the position b, and occupied for 4 time units with a last unmatched candidate word, named 's'.

4. Reconfigurable Ability

On the reconfigurable ability, we can provide four adjustable parameters, which are:

Parameter	Description
Offset	Indicate that the differential value of two adjacent address pattern during phase 1 and is used to decide if the branch jump is occurred. To various microprocessors, we just adjust this parameter and then the program flow compressor is able to have the reusable ability.
Cn	Indicate that the length of each slice during phase 2, or length of the last unmatched word at each iteration during phase3.
Cp	The position of successful matching words in the dictionary at each comparison iteration during phase3.
Cl	Indicate that the word number of successful matching at each comparison iteration during phase 3.

Table 2. Configurable Parameters

Otherwise, owing to this program flow compressor adopting the multi-phases architecture, each phase can be removable according to requirement. Each new added phase is used to provide higher compression ratio. However, this will also raise the hardware cost. Because our design is full in compliance with the design spirit of silicon intellectual property, the designer can be free to add or delete each phase according to production development.

5. Verification Strategy To Reconfigurable Ability

Through this FPGA prototyping system, we can easily verify whether this address trace compressor can work properly in various configurable parameters. As shown in figure 6, the microprocessor with address trace compressor can be changed in various configurable parameters. Besides, instruction code of target microprocessor located at ROM Emulator can be altered via download circuit of prototyping board. In virtual of the address offset being different from various operation modes, we can drag out this offset register outside, and provide user with the ability to change its value. Therefore, we can provide the reusable ability to various microprocessors by changing the value of offset register. After changed configurable parameters, if this address trace compressor still can perform the accurate address trace and decompress this address trace record to original address trace log via decompression program, we can declare that this address flow compressor is reconfigurable.

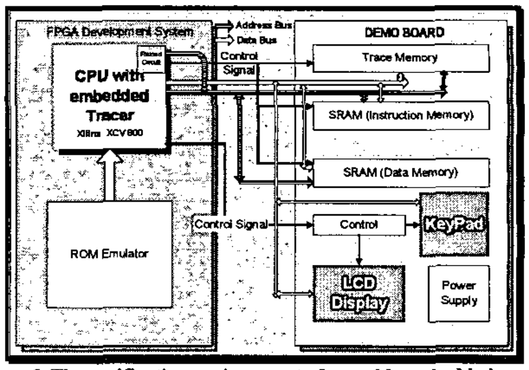

Figure 6. The verification environment of reusable embedded program flow compressor

The real circuit diagram was shown in figure 7. Label 1 is a FPGA Chip, which is used to load our reconfigurable trace hardware. Label 2 is a LCD, which is used to display the operation information. Label 4 is a Dipswitch, which is used to adjust the offset value. Label 5 is a reset button. Label 6 is a keypad, which is used to change the operation modes. At last, Label 7 is a ROM Emulator, which is used to load address patterns of various microprocessors.

We use bus model to provide system verification technician with an environment to update the peripheral model on line. The compressed address information will be transferred to system platform to perform decompression.

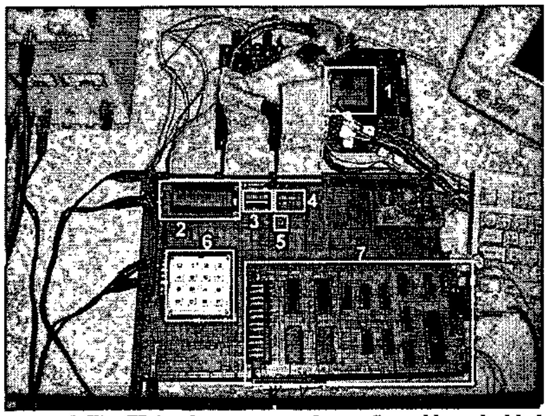

Figure 7. The FPGA demo system of reconfigurable embedded address trace compressor

6. Experiments And Results

By the creation of simulation model, we not only can get the compression ratio from various address compression strategies, but also can get the feasible analysis to this hardware. We exploit C++ language to construct this simulation model, and employ ARM7TDMI [11] for our target microprocessor/microcontroller.

Benchmark	DTMF	Dhrystone 2.1 (10000 runs)	JPEG Encoder (103*164 with 256 colors BMP)
Original Trace Data	32 bits * 1000000 lines	32 bits * 1000000 lines	32 bits *1000000 lines
Trace Data after Non-Sequential PC Filtering Phase	32 bits * 277884 lines	32 bits * 294751 lines	32 bits * 247722 lines
Trace Data after PC Pattern Reduction Phase	12 bits * 348272 lines	12 bits * 400472 lines	12 bits * 292983 lines
Compression Rate after Non-Sequential PC Filtering Phase	72.21%	70.52%	75.23%
Compression Rate after PC Pattern Reduction Phase	86.94%	84.98%	89.01%

Table 3. Compression ratio of the former two phases

Table 3 shows the compression results during non-sequential PC filtering phase and PC pattern reduction phase. The first row shows the capacities of original traced data of each benchmark. Because the address widths of instruction memory of ARM are 32 bits, the overall traced capacities are equal to 32 bits multiplied by the total line counts of traced file. However, after the compression by PC pattern reduction phase, it will divide the 32 bits address into three slices with equal sizes. Owing to widths of each slice being 12 bits, the output address widths will also be reduced from 32 bits to 12 bits. The overall address trace sizes are equal to 12 bits multiplied by the total line counts of traced file after PC pattern reduction phase. At last, the compression rate of non-sequential pc filtering phase is near 1:5 left and the compression rate of pc pattern reduction phase is near 1:10 left.

This compression rate is not sufficient for current address trace compression after the above two phases. If the traced data can save to the storage media with large enough capacities, it can record a longer traced execution log. For current embedded microprocessors, the address information generated by microprocessor usually is very large. For example, an embedded microprocessor working at frequencies of 120 MHz and its address width is at 32 bits, the output bandwidth of address trace will reach this value, which is 120 * 32 / 8 = 480M Bytes/Sec. However, the internal data transfer rate of current hard disk is between 35M Bytes/Sec and 45M Bytes/Sec. Hence, if we adopt the compression strategies from the above two compression phases to obtain the compression rate near 10:1 left, its output bandwidth, which is 480 / 10 = 48M Byte/Sec, is still higher than the internal transfer rate of current hard disk. For the other embedded microprocessor such as ARM, it can work at 200 MHz or higher. This will result in this fact that we cannot utilize the hard disk of large capacities to store the address trace log. Hence, we still need the last phase that is namely lossless compression for further address trace compression.

Benchmark	DTMF	Dhrystone 2.1 (10000 runs)	JPEG Encoder (103*164 with 256 colors BMP)
Original Trace Data	32 bits * 1000000 lines	32 bits * 1000000 lines	32 bits * 1000000 lines
L3 (25 bits) Cp: 7bits, Cl: 6 bits, Cn: 12 bits	25 bits * 136025 lines	25 bits * 233432 lines	25 bits * 6478 lines
L3 (27 bits) Cp: 8 bits, Cl: 7 bits Cn: 12 bits	27 bits * 17386 lines	27 bits * 3883 lines	27 bits * 4185 lines
Compression Rate after L3 in 15 bits	89.37%	81.76%	99.49%
Compression Rate after L3 in 20 bits	98.53%	99.67%	99.64%

Table 4. Compression ratio of the last phase

After compression of the lossless data compression phase, address trace compressor gets an excellent compression ratio than before. Because we use LZ-Compressor during this phase, we have three parameters been able to be adjusted. These three parameters are Cp, Cl and Cn, respectively. As shown in table 4, we use two sets of parameters of LZ-compressor to perform compression. The output widths of Phase3 from these two sets of parameters are 25 bits and 27 bits, respectively. From this table, we can know that the compression ratio of LZ-compressor in 25 bits is less than the former two phases. (Such as: Dhrystone). Hence, we try to use the LZ-compressor in 27 bits to perform compression. From this table, we can know that the compression rate of LZ-compressor in 27 bits is approximately to 100:1 left. This is a very ideal compression rate to address trace compression. The reason why the LZ-compressor has the excellent compression ratio is that it can compress the repeatedly sub-strings. These sub-strings were generated from the temporal principle and this principle is not used in the former two phases. Compared with related design in table 5, we found that RTATC that we proposed still has better compression rate than other designs. The TDM (Time Division Multiplexing) technique proposed by ARM ETM is used to reduce the bus width to half by double clocking rate rather than to compress address trace data. In table 6, we show the synthesized result of RTATC (Real-Time Address Trace Compressor) by using XILINX Virtex 800 and its output bandwidth is limited to 20 bits by the configurable parameters from Lempel-Ziv compressor. The maximum frequency of this design implemented in XILINX Virtex 800 is at 42.159MHz. This frequency is limited to the physical constraint of Virtex 800 chip.

Features	MPC555 [9]	ARM ETM [9]	TriCore OCDS1 [9]	RTATC (Ours)
Trace Technique	Discontinuous Address References	Discontinuous Address References with Time Division Multiplexing	Discontinuous Address References	Multi-Phase Re-configurable Architecture
Compression Ratio	5:1	5:1	5:1	100:1 (Roughly) or Higher

Table 5. Compression ratio of last phase [9]

Component	Numbers	Utilization
Number of External GCLKIOBs	1 out of 4	25%
Number of External IOBs	49 out of 166	29%
Number of SLICEs	469 out of 9408	4%
Number of GCLKs	1 out of 4	25%
Equivalent Gate Counts	7333	4%

Table 6. Device utilization summary on XILINX Virtex800 Chip

7. Conclusion

To current in-circuit emulators, they usually have the function of program trace. However, the trace depth is rather limited. In general, the trace depth is at 2KB limited to the size of trace memory. Besides, they usually only support the microcontroller working at very slow speed (such as 12 MHz). In this address trace compressor, it not only can provide the more trace logs to finite memory space, but also can support the embedded microprocessor with higher speed (such as 200 MHz). Besides, by utilizing FPGA based prototyping system; this reconfigurable real-time address trace compressor can be used in various microprocessors with suitable compression ratio in very fast updated speed. Hence, it can provide the developer with the reusable ability and abundant parameters to meet the fast change in the market, and achieve the requirement of time to market.

8. References

[1] IEEE-ISTO Nexus 5001 Forum Web Site, *http://www.nexus5001.org/*

[2] Craig B. Stunkel, Bob Janssents and W. Kent Fuchs, "Collecting Address Traces from Parallel Computers", *Pro. of the 24 Annual Hawaii Int. conf. on system*, 1991.

[3] Mohammand I. Malkawi and Janak H. Patel, "Performance Measurement of Paging Behavior in Multiprogramming Systems", *Computer Architecture News*, 1986.

[4] Mani Azimi, Carl Erickson, "A Software Approach to Multiprocessor Address Trace Generation", *Annual International Computer Software and Application Conference*, 1990.

[5] C.B Stunkel and W. K. Fuchs, "TRAPEDS: Producing Traces for Multicomputers via Execution Driven Simulator", Proceedings. *1989 ACM SIGMETRICS International Conference Measurement Modeling Computer System*, Berkeley, CA, pp.70-78, May 1989.

[6] S. J. Eggers, D. R. Keppel, E. J. Koldinger, and H. M. Levy, "Technique for Efficient Inline Tracing on a Shared-Memory Multiprocessor," *ACM SIGMETRICS International Conference Measurement Modeling Computer System*, pp. 37-46, May 1990.

[7] Anant Agarwal, Richard L.Sites, and Mark Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode", *13th Annual International Symposium on Computer Architecture*, 1986.

[8] Richard L.Sites and Anant Agarwal, "Multiprocessor cache analysis using ATUM", 15th *Annual International Symposium on Computer Architecture*, June 1988.

[9] Ciaran MacNamee and Donal Heffernan, "Emerging on-chip debugging techniques for real-time embedded systems", *IEE Computing & Control Engineering Journal*, pp. 295-303, Dec. 2000.

[10] Denning, P.J. "The Working Set Model for Program Behavior", Communication ACM 11, 5, pp. 323-333, May 1968.

[11] ARM Corp. Web Site, *http://www.arm.com.*

[12] Wei-Je Huang, Nirmal Saxena, et.al., "A Reliable LZ Data Compressor on Reconfigurable Coprocessors", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000, pp. 249-259.

[13] Ziv, J. and Lempel, A,. "A Universal Algorithm for Sequential Data Compression", *IEEE Trans. On Information Theory*, Vol. IT-23, No.3, 1977, pp. 337-343