Real-Time Software Profiler for Embedded Systems

Medhat H. A. Awadalla, Kareem Ezz El-Deen

Abstract*— Embedded systems are a mixture of software running on a microprocessor and application-specific hardware. There are many co-design methodologies that are used to design embedded systems. One of them is Hardware/Software co-design methodology which requires an appropriate profiler to detect the software portions that contribute to a large percentage of program execution and cause performance bottleneck. Detecting these software portions improves the system efficiency where these portions are either reprogrammed to eliminate the performance bottleneck or moved to the hardware domain gaining the advantages of this domain. There are profiling tools used to profile software programs such as GNU Gprof profiler. GNU Gprof integrates an extra code with the software program to be profiled causing inaccurate results and a significant execution time overhead. To address these issues, this paper proposes a software profiler called AddressTracer that is accurately able to evaluate performance matrices of any specific software portion. A set of benchmarks, Dijkstra, Secure Hash Algorithm, and Bitcount are profiled using AddressTracer, Airwolf and GNU software profiling tool (Gprof), for a quantitative comparison. The achieved results show that AddressTracer gives accurate profiling results compared to Gprof and Airwolf profilers. AddressTracer provides up to 50.15% improvement in accuracy of profiling software compared to Gprof and 6.89% compared to Airwolf. Furthermore, AddressTracer is a non-intrusive profiler which does not cause any performance overhead.

Index Terms*— Embedded Systems, FPGA, profiling tools, Hardware/Software co-design.

I. INTRODUCTION

Nowadays embedded systems, involved in most of life aspects, have the capability of reacting in real-time with sensory inputs, and designed to perform one or more dedicated applications. The embedded systems include hardware and software components operational together to execute specific computation, control, and communication tasks. In other words, embedded systems usually contain application-specific hardware accelerator circuits, general input/output interfaces and processor core which execute a software program exists in memory storage. Designing these systems is commonly known as hardware/software co-design. The advantage of the application specific hardware is that it is faster and more power efficient than software, but it is expensive. On the other hand, the software is cheaper than hardware but it is slow and consumes much power when executed on a general purpose processor.

Therefore hardware based system is more suitable in fast realization or power critical situations, whereas noncritical modules of embedded systems are realized in software [1]. Therefore, embedded system designers must decide which part/components of the system should be implemented on hardware domain and which one in software to get satisfactory behavior, in terms of cost, power and performance. In any embedded system, separating the design into software and hardware domains is a remarkable issue that should be addressed which is referred to hardware/software partitioning problem in embedded systems design.

There are verities of existing profiling tools which provide different profiling capabilities, and different measuring techniques. Profiling tools have classified based on its implementation, hardware-based, software-based and FPGA-based tools [2, 3]. This variety helps the embedded system designers to find the appropriate profilers that enable them to optimize their software code. However, there are many profiling tools probably provide inaccurate results. Some of profiling tools, such as those are software-based implementation, require the embedded system designers to compile their software programs with special compiling options that insert extra instrumentation code at the binary level. This intrusive to the software program incurs significant performance overhead during the run-time. There are also other profiling tools depend on sampling technique, which generate a number of interrupts on the target processor to collect information about the running code. This can cause a very significant disruption of runtime system behavior leading to inaccurate profiled results. There are other tools that profile the software code by simulating the behavior of microprocessor. This approach is extremely slow, especially when simulating a system on-a-chip. It is clear that, to make designers able to effectively partition their embedded system designs, the profiling tools should accurately collect the desired performance information with without disturb the running systems as shown in fig. 1.

In this design flow, the complete software is first programmed using a high level language and then verified for correct operation. An appropriate profiler is used to collect information about runtime performance of the program. The embedded system designers use this information to specify the functions that impact the performance of the system. The designers repartition the system by implementing these functions in hardware, and then redesign the system to comprise the hardware and software components. The flow design steps are repeated until the system performance is satisfactory.

Manuscript received on March 2013.

Medhat Hussein Ahmed Awadalla, Electrical and Computer Engineering Department, SQU, Muscat, Oman.

Kareem Ezz El-Deen, Department of Computer Science, Faculty of Computers and Information / Organization Name, University of Fayoum, Egypt.

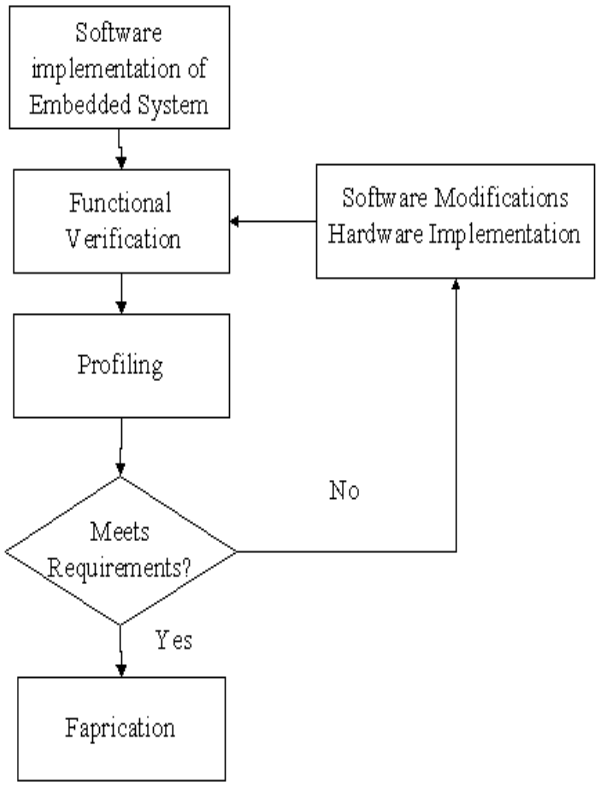

Figure 1: Software Profiling Methodology [4]

This paper proposes AddressTracer as a function-level, non-intrusive, software profiler for software applications running on soft-core processor instantiated on an FPGA. The AddressTracer counts the number of times the functions are called, and calculates the exact clock-cycle execution time of these functions.

The rest of this paper is organized as follows: Section 2 reviews the related work of software profiling tools. Section 3 presents the AddressTracer architecture. Section 4 demonstrates and discusses the results obtained from profiling three software benchmarks using Airwolf, gprof, and AddressTracer. Section 5 gives conclusion and suggestions for future work.

II. RELATED WORK

The most related work to theme of this paper includes that of GNU's gprof [4], which is software profiling tool used to profile C/C++ application codes. Gprof is a statistical profiler used to count the number of times the functions have been called and calculate the execution time for each function. The application must be first compiled and linked with the enabled profiling options. These options will insert additional instrumentation codes into the binary executable file. Instrumentation codes generate an appropriate number of interrupts to sample the program counter (PC) of the processor. The results obtained by gprof are based on the sampling process, so they are subjected to statistical inaccuracy. Gprof can calculate an accurate number of times the functions have been called. However the execution time of each function is not accurate. Frequent Loop Analysis Tool (FLAT) is on chip non-intrusive, small, and low power profiler. It specifies functions in software that heavily use loops [5]. FLAT architecture contains the Frequent Loop Cache Controller, which is a cache controller used to keep the data updated with the latest values. A Frequent Loop Cache (FLC) stores the execution frequency of each loop function at the index memory location that is based on the Short Backward Branch (SBB) value. Usually a loop in an application is typically denoted by the last instruction being SBB that jumps back to the first instruction of the loop. The SBB instruction is not a special instruction; rather, it is a jump instruction with a small negative offset. SBB is required from the microprocessor. If the SBB signal is not one of the architecture control signals, the cache controller could substitute it by replicating a small portion of the instruction decode logic. Watches Over Data Streaming On Computing element links (WODSTOCK) is a real-time system profiler that runs on the reconfigurable design platform [6]. It is used to monitor the data flow between each Computing Elements (CE). It is able to detect and remove bottlenecks in a system to improve the overall system performance. The CEs are connected by the system data links which are implemented using Xilinx Fast Simplex Links (FSL) [7] to allow steaming and buffering of data. The FSL are FIFOs that support slave read and master write protocols by Xilinx MicroBlaze soft-core processor [8]. WODSTOCK watches the data streaming between CEs on FSLs and measures the number of run-time execution clock cycles to determine which CE is stalled or starved for data. SnoopP [6] is an on-chip, function-level, nonintrusive profiler for software application running on a soft-core processor. SnoopP contains a user-specified variable number of segment counters. These counters are used to measure the number of clock cycles spent in executing of contiguous regions of memory. Each segment contains two comparators to check the value of the program counter between the specified low and high address. If the value of the PC is presently accessing an address within these bounds, then the counter value is incremented. The design is implemented on Xilinx Multimedia Board with Virtex-II 2000 based on MicroBlaze soft-core processor [8]. Two benchmarks, Dhrystone and AES, are profiled using SnoopP and gprof. The results show that the SnoopP is more accurate than gprof. SnoopP does not provide any performance overhead because it is a non- intrusive profiler.

Airwolf [9], is an on-chip, function-level, real time profiler for software program running on soft-core processor. It is developed on NiosII processor [10]. It is used to calculate the run-time of each function by counting the number of system clock cycles. Airwolf does not require any instrumentation code added to the binary file while it adds an extra code to each function; "A pair of software drivers needs to be placed in between a software function block in the source code in order to activate and deactivate a particular profiling counter contained in Airwolf" [9]. Airwolf contains 20 profiling counters which allow for up to 20 functions to be profiled at a time. Each profiling counter consists of two counters; 32-bit hit counter and a 64-bit time counter. Four software benchmarks are profiled using Nois2-gprof and Airwolf profiler. The results show that the Airwolf provides an improvement in accuracy and reduces the run time performance overhead.

III. ADDRESSTRACER PROFILER

This section introduces an FPGA-Based profiling tool, the AddressTracer profiler. AddressTracer is an on-chip FPGA-based nonintrusive profiler for software running on a processor. This profiler is used to determine the run time spent in the execution of any software portion, such as loop block, function, or an entire program, by accurately counting the number of the clock cycles taken in the execution of this portion. AddressTracer does not require inserting any instrumentation code like Gprof [4] or adding any extra code like Airwolf [9]. The profiling technique that is used in AddressTracer profiler depends on tracing addresses via the system buses. Therefore, this approach does not disturb the program or the system behavior during the execution compared to the other techniques explained in pervious section. The objective of AddressTracer is to provide accurate and quick results to embedded system designers without modifying the software code. AddressTracer contains a set of dedicated hardware segments that are used to profile any software code. This section begins by explaining the architecture of a segment and then introduces the simulation to confirm the Hardware implementation. Finally, the address tracing methodology that the AddressTracer follows to profile software is discussed.

3.1. Segment Architecture

AddressTracer profiler contains a variable number of segments. These segments are used to profile software running on a processor. Each segment contains two hardware counters which are used to profile one software portion such as a function. Therefore, the number of segments in this profiler is determined by the number of portions the designers want to profile. Fig. 2 illustrates the general architecture of a segment.

As shown in Fig. 2, the segment contains a Tracer block which is used to decide if count_en signal is high or low depending on the Add_bus. Add_bus is a 32-bit input to the Tracer block, representing the addresses values that AddressTracer monitors. Count_en is an output signal from Tracer block used to enable or disable the counters. If the Add_bus holds the starting address of a specific portion, the count_en signal will be active high, and when the Add_bus equals the ending address of the portion, the count_en signal will be active low. Hence, the count_en signal is getting high as long as the desired portion is being executed.

AddressTracer segment contains two counters. The first counter is a 32-bit Hit counter used to count the number of positive edges of the count_en signal. Therefore, it can count the number of times a portion has been called. The second counter is a 64-bit Time counter that is fed by two signals; count_en and system clock. The count_en signal is used to mask the system clock to make the Time counter able to count the number of clock cycles of a specific portion's execution. The 64-bit Time counter is able to measure over 100 million hours of profiling time when using a 50 MHz system clock.

3.2 Simulation

Mentor Graphic Modelsim simulator [11] was used to perform simulation for AddressTracer to confirm the behavior of this profiler. An Experimental environment was built using Xilinx Embedded Development Kits [12 - 15] to profile a software function. More details about the experimental environment are presented in the section 4. This environment simulates MicroBlaze soft-core processor executing a software function and AddressTracer profiling this function. The starting and ending addresses of this function are 0x00000168 and 0x00000198 respectively. As shown in the fig. 3, Trace_PC bus is used to trace the addresses on the system bus which mirrors the program counter register of MicroBlaze soft-core processor. Trace_valid_instr is a signal used to mask the Trace_PC bus which if it equal 1, this means the Trace_PC value is valid and vice versa [8].

The count_en signal is converted to 1 when the processor starts to execute the function, which the Trace_PC bus is equal to 0x00000168. In additions, fig. 4 shows that the count_en goes to 0 when the Trace_PC hold the ending address of the function. Clk_Count_LSB represents the Least Significant 32-Bits of the Time counter which is incremented by 1 at every clock cycle during the count_en signal is active high. Therefore, Clk_Count_LSB counts the number of clock cycles of the function execution (0X304 clock cycles). The Hit counter will be incremented by 1 when the count_en signal is active high, and does not change any more. It will be incremented again if the count_en signal changes from low to high, it is a positive edge counter.

Simulation results show that the hardware implementation of the AddressTracer is validated as described before. Count_en signal goes high and low at the starting and ending execution of the function. Furthermore, Time counter is enabled and disabled with count_en transactions. Finally, Hit counter counts the number of count_en changes from low to high calculating the number of function calls.

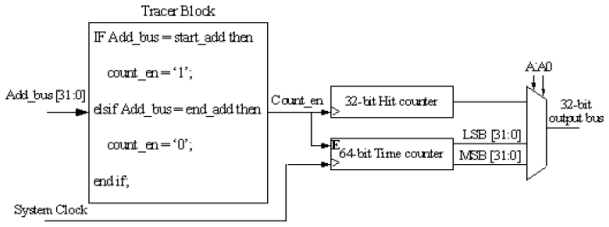

Figure 2: Segment Architecture of AddressTracer

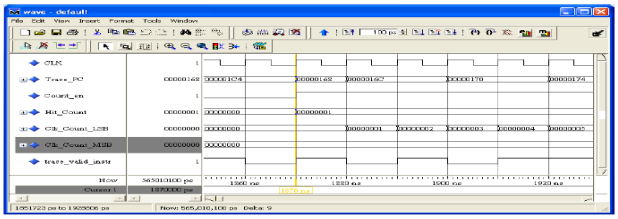

Figure 3: Simulation of AddressTracer (Starting the execution of the function)

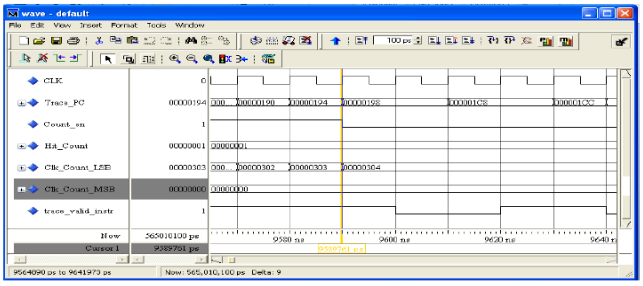

Figure 4: Simulation of AddressTracer (Ending the execution of the function)

3.3. Address Tracing Methodology

AddressTracer is developed to profile the programs running on Xilinx MicroBlaze soft-core processor. At the early stages of AddressTracer development, AddressTracer is connected to MicroBlaze to profile the program by tracing its addresses as they appear on the On-chip peripheral bus (OPB). OPB is a bus designed for easy connection of on-chip peripheral devices [12, 13]. Suppose a memory controller connected to the MicroBlaze using OPB as shown in fig. 5. When the MicroBlaze reads the program from the memory, the addresses will pass via the OPB. Therefore, the AddressTracer can trace the function execution by feeding the Add_bus of the AddressTracer with the Address bus of OPB (OPB_Addr). This scenario is applied, and a set of benchmarks software are profiled using AddressTracer, Airwolf, and gprof. The results show that the AddressTracer is more accurate and does not cause any performance overhead compared to Airwolf, and gprof.

As shown in fig. 6, when the cache is used, the MicroBlaze is connected to the memory controller using XCLs (Xilinx Cache Links). XCLs are P2P links used in the caching processing [8]. DXCL and IXCL are used to cache Data and Instruction from the external memory respectively. When the MicroBlaze caches data or instructions, the addresses are passed through XCL buses and not through the OPB. To make the AddressTracer able to trace these addresses, the Add_bus of the AddressTracer is connected to Trace_PC bus. The MicroBlaze core exports a number of internal signals for tracing purposes such as Trace_PC. Trace_PC is a 32-bit bus used to mirror the executing PC of the MicroBlaze. Therefore, AddressTracer is able now to trace any addresses through the PC of the processor not through address bus of OPB. Furthermore, as illustrated in Fig. 6, The AddressTracer is connected to Trace_Valid_Inst signal which is used as an indication to the PC value [8]. When the PC value is valid the Trace_Valid_Inst will be high and vice versa.

From tracing via OPB and via Trace_PC, it is concluded that the implementation of AddressTracer does not depend on the MicroBlaze's architecture and AddressTracer can be connected to any bus that provides the addresses of the programs. Therefore, AddressTracer can profile programs running on any processor whether this processor is MicroBlaze or any other one.

It is clear that, AddressTracer does not require inserting any extra code at the program source code level or at binary level. Furthermore, AddressTracer profiles the software programs in parallel with the running system. Therefore, the profiling technique used by AddressTracer does not disturb the software program or the system behavior during the software running, so it does not cause any performance overhead on the program execution.

The AddressTracer counters are memory mapped and the AddressTracer is connected to OPB bus. This allows the MicroBlaze processor to read the profiling results and then pass them to the host computer through the UART.

3.4. AddressTracer, Airwolf, and SnoopP Architectures discriminations

Based on the simulation results, Airwolf results are erroneous because of its drivers. For example, the counters of Airwolf will not be disabled unless the drivers of the Airwolf are added before the Return statement of any function. This causes that the Airwolf will not count the cycles that the processor spent in executing this statement. Moreover, the header of a function may contain some bytes as arguments. The drivers of Airwolf are added after the header of any function. Therefore, Airwolf will not also profile these bytes. Hence, if the header size is higher, the error in the results of Airwolf will be higher and vice versa. Fig. 7 shows a simulation for Airwolf verses AddressTracer to confirm the previous conclusions. Airwolf and AddressTracer are both used to profile the same function which takes int x as an argument and contains Return statement at the end of it. As shown in the fig. 7, AT_count_en, the enable signal of AddressTracer, goes to high with the starting address of the function. On other hand, Airwolf does not detect that the function has already started. After some clocks are spent, Airwolf started counting which AW_count_en is active. After Airwolf counted 9 clocks, AW_count_en goes down and dis-activated announcing the end of the function's execution. The AddressTracer reports that the function's execution consumes 30 clock cycles.

When a function profiled by Airwolf does not have a Return statement or arguments in the header, the Airwolf will count more cycles due to its drivers. Therefore, the results obtained by Airwolf depend on the implementation of the functions and the position of its drivers [16]. However, the experimental results show that Airwolf has a good accuracy compared with Gprof software profiling. AddressTracer does not need any extra code to be added to a function, so it is more accurate than Airwolf. It profiles any software portion from its starting address to ending address.

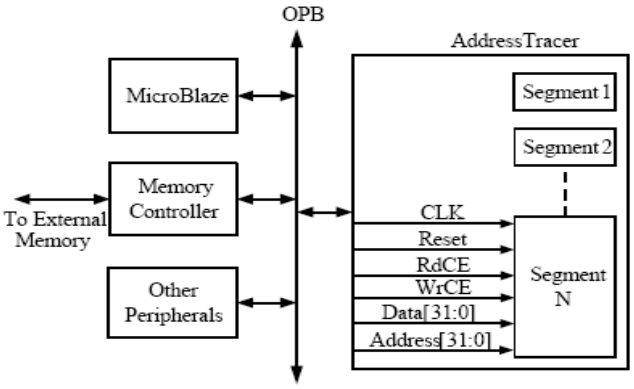

Figure 5: Tracing addresses via OPB

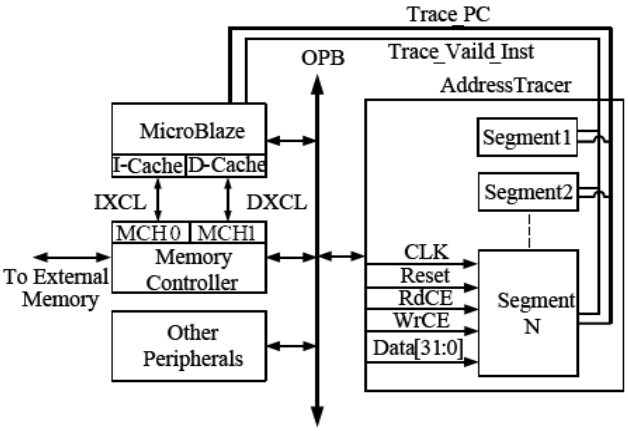

Figure 6: Tracing addresses using Trace_PC bus

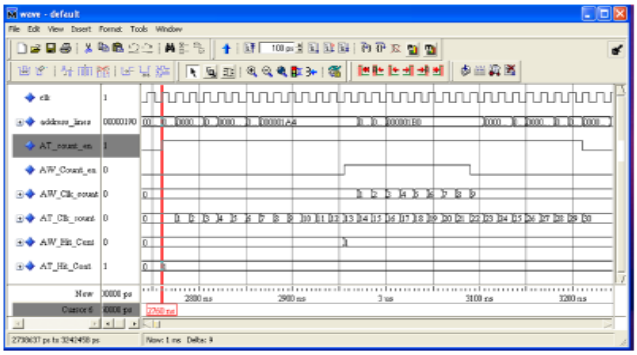

Figure 7: Airwolf versus AddressTracer Simulation

SnoopP profiler does not count the number of times a function is called, and only counts instructions executed in contiguous regions of memory [6]. Therefore, SnoopP does not give accurate results in some situations. For example, if function A calls a sub-function C and SnoopP is used to profile function A. SnoopP will accurately count the instruction clock cycles that is required by A and C. The total time spent by the function A will equal the summation of the execution time of both function A and function C. However, if function A and function B call sub-function C, and SnoopP is used to profile A and B. It may not be possible to distinguish which portion of the sub-function C's execution time is due to function A versus function B. Hence it cannot calculate the accurate execution time spent by function A or function B.

IV. EXPERIMENTAL RESULTS

Xilinx Embedded Development Kit (EDK) is a suite of tools and intellectual properties (IP) that enables to design a complete embedded system for implementation in a Xilinx FPGA device [15]. EDK has several components like Xilinx Platform Studio (XPS), Software Development Kit (SDK), and GNU Compiler. XPS tool suite is the development environment or GUI used for designing the hardware portion of the embedded processor system. XPS contains a processing IP library, software drivers, documentation, and reference designs. SDK is an integrated development environment used for C/C++ embedded software application creation and verification (Xilinx Incorporated, 2007b).

Xilinx Microprocessor Debug (XMD) engine for MicroBlaze provides a user debugging interface using command line tools. All software applications will be compiled using GNU Compiler for C/C++ software development targeting the MicroBlaze soft-core processor. Gprof requires C/C++ program to be integrated with instrumentation code. Therefore, the program must be compiled with the –pg option [4]. The size of the function and its starting and ending addresses can be obtained by viewing the assembly code using the mb-objdump utility or using GNU Debugger (GDB) interface [7].

The system, shown in fig. 8, is implemented using Spartan-3E Starter Kit. Spartan-3E development board is a low-cost solution for evaluating the Xilinx Spartan-3 XC3S500 FPGA. Its features are a 500K gates, on-board I/O devices, and 32MX16 DDR SDRAM. The board also contains a Platform Flash JTAG-programmable ROM, so designs can easily be made non-volatile [7].

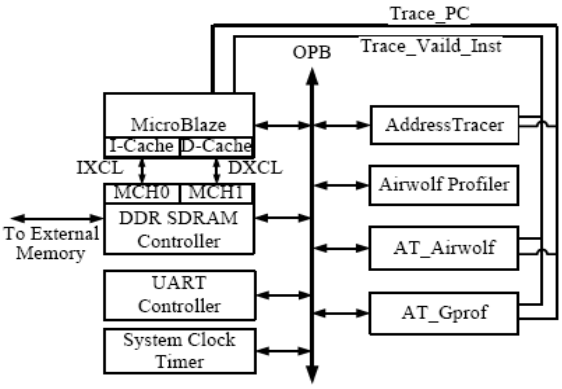

Figure 8: MicroBlaze Profiling Environment

The system consists of MicroBlaze soft-core processor, and DDR SDRAM off-chip memory. This memory stores the program to be profiled and OPB bus is used to connect all components. Universal Asynchronous Receiver Transmitter (UART) controller is used to transfer streaming message back to the host computer. System Clock Timer is used by gprof in order to profile the software functions. Airwolf was developed to run on Altera Nios II soft-core processor at which Airwolf profiler was instantiated onto Nios Development Board, Stratix Professional Edition, featuring a Stratix EP1S40F780C5 FPGA chip. The results that published in [4] were obtained from experiments running using SRAM off-chip memory, while Spartan 3E starter kit does not have SRAM. Therefore, to be able to compare the results of Airwolf with that of AddressTracer, Airwolf was re-implemented to run on Xilinx MicroBlaze soft-core processor on Spartan 3E starter kit. AddressTracer and Airwolf Profiler blocks are used to profile the software functions. Finally, AT_Airwolf and AT_Gprof blocks are used to measure the performance overhead caused by Airwolf and gprof using AddressTracer technique.

4.1. Profiling Benchmarks and comparison of results

The results obtained from profiling a set of software benchmarks (Dijkstra, Secure Hash Algorithm (SHA), and Bitcount) are compared. Each software benchmark is profiled using AddressTracer, Airwolf, and Gprof with respect to their software compilation settings.

4.1.1. Dijkstra

The Dijkstra benchmark constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of Dijkstra's algorithm [17]. In this experiment, Dijkstra program is implemented to compute the shortest path between 100 distinct nodes. Table 1 shows the profiled results of Dijkstra program. The first column lists the names of functions. Second column, has two sub-columns, shows the results that obtained by AddressTracer. The first sub-column shows the execution time of each function in seconds. The second sub-column lists the number of calls of each function that are calculated. Column 3 and 4 show the results that obtained by Airwolf and Gprof respectively.The execution time of Enqueue function via AddressTracer is 2.97 second, 2.91 second by Airwolf, and 0.82 second by Gprof. As noticed, AddressTracer and Airwolf results are very close to each other. The difference in accuracy between AddressTracer and Airwolf can be calculated as follows:

[(TAddressTracer -TAirwolf)*100/ (TAddressTracer)]. AddressTracer provides up to 0.33% accuracy improvement in profiled results of this program compared to Airwolf. This implies that Airwolf reports results with comparable accuracy to those of the AddressTracer profiler for smaller, less computationally intensive benchmarks. However, AddressTracer provides up to 13.11% accuracy improvement compared to Gprof. Gprof profiler provides inaccurate calculations of the execution time due to its sampling technique. To confirm that the results obtained by Gprof depend on the value of sampling frequency, the same program with the same environment was executed many times with different sampling frequency. Gprof results were varied according to the sampling frequency as shown in table 2. Table 2 illustrates the results reported by Gprof with different sampling frequencies. It is clear that the sampling techniques used by Gprof provide inaccurate results. As also obviously shown in Table 1, the profilers have a similar ranking of the time consuming functions. Table 1 also outlines that the number of times the functions have been called. AddressTracer, Airwolf, and Gprof have a similar number of functions call.

1) Secure Hash Algorithm (SHA)

Secure hash algorithms are typically used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, or in the generation of random numbers (bits) using massage digests. When a message is applied to SHA algorithm, the output result is called a message digest. Message digests range in length from 160 to 512 bits, depending on the algorithm [9]. Gprof needs numerous loops of the main algorithm to obtain its percentage of the execution time per function, so the main algorithm is looped 10000 iterations.

Table 3 shows the execution time of the functions of SHA program. AddressTracer and Airwolf have a similar rank of the time consuming functions and there is a very small difference in their execution times of all functions. In both of them, the update and transform functions consume approximately 50.3% and 44% of the total execution time of the program respectively. However in Gprof, the update function takes 4.64% and transforms function requires 88.51% of the total execution time of the program. This inaccuracy of the Gprof is due to the sampling techniques used. There is 0.47% improvement in accuracy when using the AddressTracer Profiler instead of Airwolf. Furthermore, the clock-cycle counting method that AddressTracer utilizes shows that a 50.15% accuracy improvement in the reported time for this program compared to the insertion of instrumentation code method that Gprof utilizes. However, AddressTracer, Airwolf, and Gprof have the same number of functions call.

4.1.3. Bitcount

The bit count algorithm tests the bit manipulation abilities of a processor by counting the number of bits in an array of integers. It performs it using five methods including an optimized 1-bit per loop counter, recursive bit count by nibbles, non-recursive bit count by nibbles using a table look-up, non-recursive bit count by bytes using a table look-up, and shift and count bits. The input data is an array of integers with equal numbers of 1's and 0's [9].

Table 4 shows the execution time of the functions of Bitcount program. AddressTracer and Airwolf have a similar time consuming functions. Gprof reported that the btbl_bitcnt and Flipbit functions consumed 0.42, 1.19 seconds during the execution of this benchmark. However, the AddressTracer and Airwolf Profilers did not detect calls to those functions. The insertion of instrumentation code not only generate additional functions calls, interrupts and performance overhead, but also cause reporting of unpredictable profiling results.

AddressTracer and Airwolf reported that the ntbl_bitcnt function, a recursive function, was running for 79.978 and 63.509 seconds respectively. This shows that Airwolf presents inaccurate execution times when profiling recursive functions. On the other hand, Gprof reports that ntbl_bitcnt consumes 197.92 sec, which is completely inaccurate due to the recursive functions are not supported by EDK Gprof version [12, 13]. AddressTracer Profiler provided up to 6.89% improvement in accuracy in Bitcount program compared to Airwolf.

Table 1: Profiled result of Dijkstra

	AddressTracer		Airwolf		Gprof	
Function name	Time (sec)	# of Calls	Time (sec)	# of Calls	Time (sec)	# of Calls
Dijkstra	16.18	100	16.21	100	15.19	100
Enqueue	2.97	72448	2.91	72448	0.82	72448
Dequeue	0.53	72448	0.50	72448	0.28	72448
Qcount	0.023	72548	0.017	72548	0.83	72548

Table 2: Profiling Dijkstra using Gprof with different sampling frequency

	10000Hz	20000Hz	30000Hz	35000Hz	40000Hz
Function name	Time (sec)	Time (sec)	Time (sec)	Time (sec)	Time (sec)
Dijkstra	10.57	12.34	14.07	15.19	11.87
Enqueue	3.23	3.83	4.37	0.82	1.56
Dequeue	0.34	0.35	0.35	0.28	0.37
Qcount	0.84	1.21	1.23	0.83	1.42

Table 3: Profiled results of SHA

	Address Tracer		Airwolf		Gprof	
Function name	Time (sec)	# of Calls	Time (sec)	# of Calls	Time (sec)	# of Calls
Byte_reverse	9.36	1290000	9.50	1290000	6.11	1290000
Transform	80.89	1290000	79.83	1290000	80.63	1290000
Init	0.014	10000	0.0124	10000	0.12	10000
Update	91.70	10000	91.77	10000	4.23	10000
Final	0.79	10000	0.79	10000	0.01	10000

Table 4: Profiled result of Bitcount

	Address Tracer		Airwolf		Gprof	
FCN Name	Time (sec)	# of Calls	Time (sec)	# of Calls	Time (sec)	# of Calls
Bit_Shifter	191.258	1000000	188.95	1000000	201.11	1000000
Flipbit	0	0	0	0	0.42	0
Bit_count	59.657	1000000	57.484	1000000	74.42	1000000
Bitcount	23.667	1000000	21.600	1000000	39.88	1000000
Ar_btbl_bBitcount	17.620	1000000	15.391	1000000	31.73	1000000
Bw_btbl_ bitcount	10.641	1000000	8.459	1000000	28.36	1000000
Ntbl_Bitcount	47.780	1000000	45.535	1000000	70.75	1000000
btbl_Bitcnt	0	0	0	0	1.19	0
Ntbl_Bitcnt	79.978	8000000	63.509	8000000	197.92	1000000

Table 5: Size of Dijkstra functions

	AddressTracer	**Airwolf**	**Gprof**
Function name	Size (Byte)	Size (Byte)	Size (Byte)
Dijkstra	568	588	576
Enqueue	188	208	196
Dequeue	108	128	116
Qcount	16	36	36

4.2. Performance overhead Analysis

As explained earlier the AddressTracer does not insert any extra code to programs. It profiles any program in parallel with the processor (MicroBlaze) while it runs the program. Therefore, the AddressTracer does not disturb the running system and does not cause any execution time overhead in the system.

Two blocks, AT_Airwolf and AT_Gprof, are built to determine if Airwolf and Gprof cause any performance overhead. As shown in fig. 7, AT_Airwolf and AT_Gprof were implemented using AddressTracer technique to profile the execution time of the functions that have an extra code added by Airwolf and Gprof respectively. The starting and ending addresses of each function are calculated taking into account the extra code added by either Airwolf or Gprof. The execution time overhead is determined by calculating the difference between the execution time of a function without (result obtained using AddressTracer) and with (results obtained using AT_Airwolf and AT_Gprof) extra code.

Then the percentage of this difference is calculated from the total execution time [(T with extra code –T without extra code)*100/T with extra code].

4.2.1. Dijkstra Overhead Analysis

Table 5 shows the size of extra code added to each function. The first column lists the name of each function. Second column shows the size of each function with no extra codes. Column 3 shows the size of each function plus the size of the extra code added by Airwolf (20 bytes at each function). Column 4 shows the size of each function plus the size of the instrumentation code added by Gprof (8~20 bytes at each function).

Table 6 shows the execution time overhead incurred by Airwolf. Column 1 lists the function's name. Columns 2 and 3 show the execution time for the program without and with the extra code respectively. The last column shows the time difference between the two execution runs. As shown in Tables 6 and 7, Dijkstra function has 0.03 and 12.18 seconds as additional seconds in execution time caused by Airwolf and Gprof respectively. Therefore, Airwolf caused a total execution time overhead (about 0.283%) that can be ignored compared to the overhead imposed by Gprof (49.80%).

4.2.2. SHA Overhead Analysis

As shown in Table 8, Airwolf adds 20 bytes at each function, while Gprof adds 8~20 bytes at each function. Table 9 shows that Airwolf causes additional fractions of second in execution time. For example, Byte_reverse function has 0.29 second as an additional execution time. Table 10 shows that Gprof added 18.15 and 23.68 seconds in execution time to transform and update functions respectively. The results show that a negligible execution time overhead caused by Airwolf (about 0.23%) compared to that caused by Gprof (about 20.37%).

Table 6: Performance overhead in Dijkstra program caused by Airwolf

	AddressTracer	AT_Airwolf	
FCN Name	Time (sec)	Time (sec)	Difference (sec)
Dijkstra	16.18	16.21	0.03
Enqueue	2.97	2.975	0.005
Dequeue	0.53	0.534	0.004
Qcount	0.023	0.04	0.017

Execution time overhead: 0.056 / 19.759 = 0.283%

Table 7: Performance overhead in Dijkstra program caused By Gprof

	AddressTracer	*AT_Gprof*	
FCN Name	*Time (sec)*	*Time (sec)*	*Difference (sec)*
Byte_reverse	*9.36*	*13.98*	*4.62*
Transform	*80.89*	*99.05*	*18.15*
Init	*0.014*	*0.042*	*0.028*
Update	*91.7*	*115.38*	*23.68*
Final	*0.79*	*1.07*	*0.28*
Execution time overhead: 46.758 /229.522 = 20.37%			

Table 8: Size of SHA functions

	AddressTracer	AT_Gprof	
FCN Name	Time (sec)	Time (sec)	Difference (sec)
Dijkstra	16.18	28.36	12.18
Enqueue	2.97	6.91	3.94
Dequeue	0.53	1.8	1.27
Qcount	0.023	2.18	2.157
Execution time overhead: 19.547 / 39.25 = 49.80%			

Table 9: Performance overhead in SHA program caused by Airwolf

	AddressTracer	Airwolf	Gprof
FCN Name	Size (Byte)	Size (Byte)	Size (Byte)
Byte_reverse	92	112	108
Transform	1196	1216	1208
Init	76	96	96
Update	532	552	540
Final	268	288	276

V. CONCLUSIONS AND FUTURE WORK

In this paper, the AddressTracer software profiler was proposed as an FPGA-Based, nonintrusive software profiler,used to profile a set of software benchmarks. A set of software benchmarks were profiled using software-based profiler (Gprof) and FPGA-Based profilers (AddressTracer and Airwolf).

Table 10: Performance overhead in SHA program caused by Gprof

	AddressTracer	AT_Airwolf	
FCN Name	Time (sec)	Time (sec)	Difference (sec)
Byte_reverse	9.36	9.65	0.29
Transform	80.89	80.91	0.02
Init	0.014	0.015	0.001
Update	91.7	91.79	0.09
Final	0.79	0.81	0.02
Execution time overhead: 0.421 / 183.175 = 0.23%			

The results obtained from profiling these benchmarks using the different profilers were compared and discussed. The experimental results showed that the AddressTracer provides up to 50.15% improvement in accuracy of profiling software programs compared to Gprof and up to 6.89% compared to Airwolf. This improvement in accuracy is helpful for embedded system designers to partition their designs and implement the appropriate software functions in the hardware domain.

The execution time overheads caused by Airwolf and Gprof were measured and analyzed to study the effects of inserting instrumentation code to the binary file of a program and effects of adding extra code to the source code of a program. In some of the software benchmarks, inserting extra code by Airwolf incurred overhead up to 5.65% of the total execution time while adding instrumentation code with Gprof caused a execution time overhead up to 49.80%.

For further work, the AddressTracer can be adapted to profile hardware components such as monitoring memory related events such as the number of off-chip memory accesses, cache misses and memory leaks. The area occupied by AddressTracer on an FPGA chip could be studied and minimized to meet cost constraint. AddressTracer can be adapted to interface with the personal computer to collect different profiling information about running programs on the computer.

This interface can be via PCI or PCI express and with the capability of FPGA, AddressTracer can profile different performance metrics.

REFERENCES

1. Sungpack H., Tayo O., Jared C., Nathan G., Kozyrakis C., Olukotun K. " A case of system-level hardware/software co-design and co-verification of a commodity multi-processor system with custom hardware. Proceedings of the 10th International Conference on Hardware/Software Co-design and System Synthesis, pp. 513-520, 2012.
2. Patrick Schaumont . A Practical Introduction to Hardware/Software Codesign. 2nd Edition., xxii+480p, ISBN:978-1-4614-3736-9, December 2012
3. Miller, F. Vahid, T. Givargis. Application-Specific Codesign Platform Generation for Digital Mockups in Cyber-Physical Systems IEEE Electronic System Level Synthesis Conf. (ESLsyn), pp 1-6, June 2011.
4. Patrick R. Schaumont. A Practical Introduction to Hardware/Software Codesign. ISBN: 978-1-4419-5999-7 (Print) 978-1-4419-6000-9, 2010.
5. A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham, "Hardware Software Partitioning Problem in Embedded System Design Using Particle Swarm Optimization Algorithm". Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems, 2008.
6. Jason G. Tong and Mohammed A. S., "Profiling Tools for FPGA-Based Embedded Systems: Survey and Quantitative Comparison", journal of computers, Vol. 3, No. 6, 2008.
7. R. Lysecky, S. Cotterell, and F. Vahid, "A Fast On-Chip Profiler Memory", in Proc. of the 39th Conference on Design Automation, pp. 28–33, June 2002.
8. Fenlason J. and Stallman R., GNU Gprof, accessed 2008. Available Online: http://gnu.huihoo.org/gprof-2.9.1/html_chapter/gprof_toc.html.
9. Gordon-Ross A. and Vahid F., "Frequent Loop Detection Using Efficient Non-Intrusive On-Chip Hardware", in Proc. of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, San Jose, California, USA, pp. 117–124, 2003.
10. Shannon L. and Chow P., "Maximizing System Performance: Using Reconfigurability to Monitor System Communications", in Proc. of the 2004 International Conference on Field Programmable Technology (ICFPT), the University of Queensland, Brisbane, Australia, pp. 231–238, 2004.
11. Xilinx Incorporated, "Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Channel (FSL) Link", 2004.
12. Xilinx Incorporated, "Embedded System Tools Reference Manual", v7.0 January 8, 2007.
13. Jason G. Tong and Mohammed A. S., "Profiling Tools for FPGA-Based Embedded Systems: Survey and Quantitative Comparison", journal of computers, Vol. 3, No. 6, 2008.
14. Xilinx Incorporated, "Spartan-3E Starter Kit Board User Guide", V1.0, 9, 2006.
15. Dhaval N. Vyas, "FPGA-Based Hardware Accelerator Design for Performance Improvement of a System-on-a-Chip Application", Master of Science in Electrical Engineering in the Thomas J. Watson School of Engineering and Applied Sciences Binghamton University State University of New York 2005.
16. Xilinx Incorporated, "Embedded System Tools Reference Manual", v7.0 January 8, 2007.
17. Altera Corporation, "Nios Development Board Reference Manual, Stratix Professional Edition", 2004.