SONIC*mole*: A DEBUGGING ENVIRONMENT FOR THE UltraSONIC RECONFIGURABLE COMPUTER

T. Wiangtong[1], *C. T. Ewe*[2], *P. Y. K. Cheung*[2]

[1]Electronic Department, Mahanakorn University of Technology, Bangkok, Thailand
[2]Department of Electrical & Electronic Engineering, Imperial College, London, England

ABSTRACT

Reconfigurable Computers based on a combination of conventional microprocessors and Field Programmable Gate Arrays (FPGAs) presents new challenges to designers. Debugging on such hardware/software cohabiting systems can be a nightmare. This paper presents SONIC*mole*, a debugging environment designed for the UltraSONIC reconfigurable computer, which is designed specifically for real-time video applications. The window-based integrated debugging environment includes a hardware debug module (the "*mole*") that performs the function of a logic analyzer, embedded within the FPGA design, and an easy-to-use software package that facilitates such a hardware/software system. The resource overhead of the hardware module is only 4% of a Virtex XVC1000 FPGA. The approach reported here is not limited to the UltraSONIC architecture, and can easily be modified for other reconfigurable computers.

1. INTRODUCTION

The development of Field Programmable Gate Arrays (FPGAs) has given rise to a new breed of programmable systems known as reconfigurable computers [1]. Such a system usually contains a mixture of conventional microprocessors executing software and FPGA-based computing engines implementing hardware algorithms. Application development and debugging on such reconfigurable systems presents new challenges. While the software portion of the design is relatively easy to handle, helped by visual-based integrated development environment (IDE) such as Microsoft's Visual Studio©, debugging the FPGA-based hardware remains difficult. The high-speed specification of the modern FPGA devices and the high pin count packaging technology prevent the use of conventional debugging tools such as a logic analyzer. Connecting an analyzer to a FPGA computing engine is often not only physically impossible, such action may also alter the timing of the design significantly. Furthermore, signals internal to the FPGA remain invisible.

This paper describes a debugging environment known as SONIC*mole*, which is specifically designed for the UltraSONIC reconfigurable computer [2]. SONIC*mole* provides a visual-based exploration and debugging environment for the FPGA-based hardware. It consists of a flexible, yet area-efficient, debug module which serves as the "*mole*" which resides inside the FPGA. The *mole* is dormant, only invoked during debugging, when it functions as an embedded multi-channel logic analyzer. The SONIC*mole* software provides a seamless interface to the *mole* and an easy-to-use graphical user interface running on the host computer.

The contributions of this paper are: 1) A novel debugging concept for reconfigurable computer is proposed; 2) An efficient but flexible hardware debugging module suitable for embedding invisibly within the hardware of a reconfigurable computer is designed; 3) A software environment that integrates into the reconfigurable computing environment is reported.

The paper is organized into 6 sections. Section 2 provides a brief introduction to the UltraSONIC reconfigurable computer system. In section 3, the problem addressed by this paper is described along with some related work and alternative approaches. Section 4 describes the design of the *SONICmole* in details. Section 5 provides evaluation and conclusion to the work.

2. THE SONIC ARCHITECTURE

The SONIC platform [2] [3] is a reconfigurable computing system designed to cope with the computational power and the high data throughput demanded by real-time video applications (figure 1). UltraSONIC is the latest implementation of the SONIC architecture. The design consists of **Plug-In Processing Elements** (PIPEs) interconnected by two types of buses: global bus, called **PIPE bus** (PB), and local bus, called **PIPEflow bus** (PF). The SONIC architecture exploits the spatial and the temporal parallelism in video processing algorithms. It also facilitates design reuse and supports the software plug-in methodology.

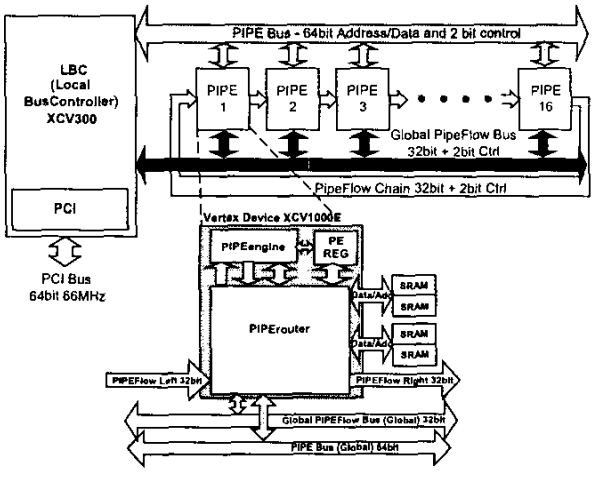

Figure 1. The SONIC architecture [2]

0-7803-7761-3/03/$17.00 ©2003 IEEE

By using the global bus, we can transfer information between the host system (PC) and the PIPEs such as image data, configuration data and controls of the routing of data on the PIPEs. Two local buses are used to transfer information between PIPEs. The global **PIPE**flow bus (PFG) enables data broadcasting, and the **PIPE**flow chain (PFC) connects adjacent PIPEs to support pipelined processing. The system uses a predefined raster-scan protocol to send data over these buses.

The basic PIPE consists of three parts (figure 2): PIPE engine (PE), PIPE router (PR) and PIPE memory (PM). The PE handles computation while the PR is responsible for image data movement and formatting. The PM provides local buffering of video data therefore reducing bus traffic.

The principle of separating the computation engine PE from the routing engine PR has many advantages. It allows computational functions to be implemented independently from dataflow, thus facilitating design reuse. A particular user-application configures only the PIPE engine and not the PIPE router. The unique design of the PIPE router provides a flexible and scalable solution to routing and formatting video data.

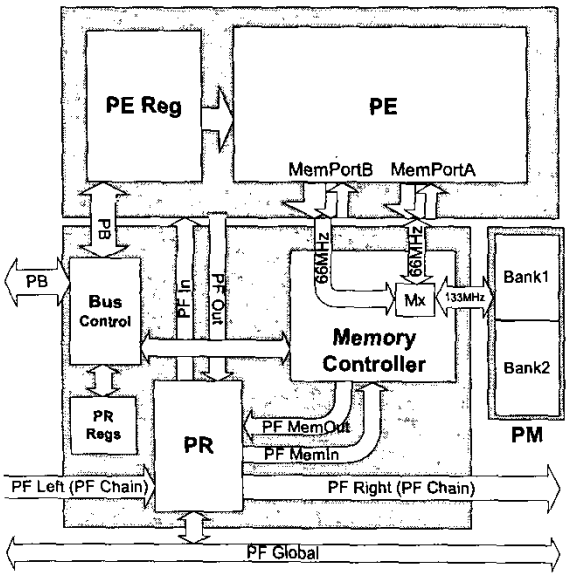

Figure 2. The UltraSONIC PIPE structure

The user application on the host system controls operations in the SONIC platform–such as setting registers in PR and PE, configuring selected PIPEs, loading or retrieving images from PM–through a set of well-defined SONIC Application Programming Interface (API) routines.

UltraSONIC, is designed with a Virtex XCV1000E device implementing both the PE and the PR [2]. The PIPE memory is implemented with synchronous SRAM, which allows fast and easy memory access. The available memory is sufficiently large to buffer two video frames at HDTV resolution. The communication between the host system and the UltraSONIC board is via a 64-bit PCI bus running at 66MHz.

3. RELATED WORK

Different types of logic analyzers, including PC software based logic analyzers and stand-alone units [4][5], have been built to help digital design engineers testing and debugging their work. The main jobs of a logic analyzer are to capture and store data from input channels, and to display into the captured data in an easily interpreted format. However, with the arrival of multi-million gate FPGA devices in reconfigurable computers, something more sophisticated is needed such as internal access to buses and signals, and proper integration of the analyzer engine to the entire system.

The Tagalyzer [6] can be used to debug JTAG chains made up of any manufacturers parts and provide information to the board level design engineers. At the circuit level, we need more than that to probe signals inside the FPGA. ChipScope™ Pro system [7] from Xilinx offers this through a number of standard black boxes embedded within individual hardware modules inside a FPGA device. Results are sent to the software program running on the PC through the JTAG interface to the display.

SONIC*mole* shares a number of common features with ChipScope: the ability for setting up trigger conditions, the capture of history list of values etc.. But unlike ChipScope, which is designed for debugging ASIC type of hardware, SONIC*mole* is designed specifically for a reconfigurable computer. Instead of using the slow JTAG interface and a software environment that is not easily adaptable, SONIC*mole* uses the fast PCI bus of the reconfigurable engine and a well-defined set of Application Programming Interface (API) functions common to the reconfigurable computing system. As a result, SONIC*mole* is better integrated with the software environment of the reconfigurable computer.

4. SONIC*mole* DEBUGGING ENVIRONMENT

SONIC*mole* consists of two components: a hardware *mole* module and a visual debugger software known as SONIC*bug*.

The *mole* resides inside the FPGA. It contains all the necessary hardware for triggering and capturing the specified signals (or variables). To insert the *mole* into a hardware routine (designed for the FPGA in Verilog or VHDL), the predesigned *mole* (exists as an EDIF file), is merged with the user's design before place-and-route. Connections between the *mole* and the user hardware are defined in a separate probe-list file and the probes are restricted to top level signals in the design. A user can therefore easily monitor different sets of signals by modifying the probe-list and rerun place-and-route.

The SONIC*bug* visual debugger is a software program that performs two functions: it allows access and control of the PIPE hardware on the UltraSONIC, and it provides a visual interface to the *mole* module.

4.1 Detail Design of the *mole* Module

Each SONIC PIPE consists of a dual-ported, dual-bank PIPE Memory (PM) (*MemPortA, MemPortB*). The two PM banks are

rarely accessed at the same time. The *mole* exploits this feature and employs a portion of the memory in the PM bank not used by the user design to store the signal states captured by the *mole*. This has the advantage that it does not use any embedded memory resource inside the FPGA. However, some user applications cannot avoid using both memory ports simultaneously. Therefore the SONIC*mole* is designed to have two modes of operations: *embedded* mode, and *standalone* mode.

Figure 3 shows the *mole* operating in *embedded* mode inside a PIPE Engine (PE). 4 selectable ports, each with 32 signal channels, are available for monitor and capture. These 128 channels can be used to debug signals within the PE or bus signals connecting the PE to other modules such as the PE registers or the PM memory port. It also shows how the *mole* is interfaced to one of the PM bank via the PIPE's memory controller.

The restriction of the embedded mode is that the two memory ports are not used simultaneously. If this is not true, then SONIC*mole* can be configured to work in a *standalone* mode. In this mode, one entire PIPE is dedicated to the operation of the *mole*. While this frees all the resources on the other PIPEs for user application, it also limits monitoring and capturing to the interface signals external to the PIPEs (i.e. signals on the PIPE Flow and Global buses).

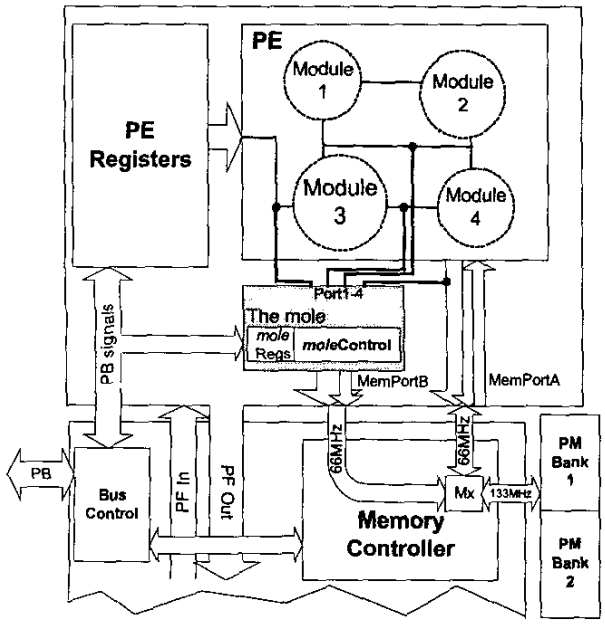

Figure 3. The mole operating in embedded mode

The internal architecture of the *mole* is shown in figure 4. There are three separate modules: *mole*Register, *mole*Controller and *mole*Multiplexers. Ports (P1-P4), each with 32 channels, are available for carrying signals either for trigger or for capture. 13 *mole*Registers are used to configure the operation of the *mole*. Overall control is done by the *mole*Controller which is designed as a finite state machine (FSM). The detail design of the FSM is depicted in figure 5.

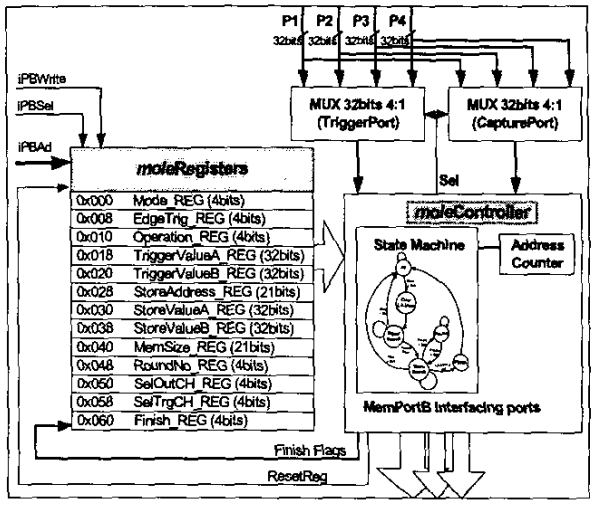

Figure 4. Detail architecture of the mole

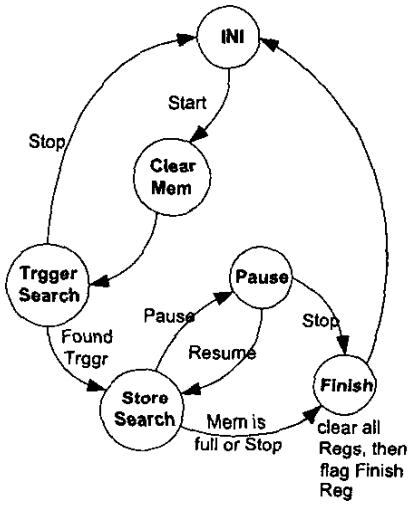

Figure 5. The *mole*Controller FSM

Our design of the *mole*Controller is capable of:

- Reporting the current operating mode.
- Specifying triggering on positive or negative edges of signals.
- Operating; start, stop, pause or resume, the search for trigger or capturing values anytime by sending commands to the *mole*Registers.
- Using any memory locations and sizes in PM (defined by users) to store the capture signal state. This makes the system more flexible because we can avoid the area occupied by normal applications.
- Changing trigger port and capture port by via software control without hardware recompilations.

4.2 The SONIC*bug* Program

The SONIC*bug* program, written in Microsoft Visual C++, has a user-friendly GUI. It provides an easy way to control and monitor the entire reconfigurable computer system by using API functions already existed for the UltraSONIC. For example, values of all the PIPE Registers and *mole*Registers can be dynamically monitored and altered during runtime without affecting the user logic or requiring recompilation of the user design. Figure 6 shows a screen shot of the SONIC*bug* while it is displaying captured signal states both as a history list and as a timing diagram. It also shows the dialog box for setting up the trigger and capture conditions.

5. EVALUATIONS & CONCLUSIONS

The UltraSONIC system is designed to operate at a maximum PIPE frequency of 66MHz. The SONIC*mole* design is shown to operate comfortably within this timing constraint. In addition, the overhead cost of the hardware *mole* is only 510 slices, representing around 4% of the XCV1000E device on a SONIC PIPE. As long as the user application can afford such overhead, embedding the *mole* in a design permanently should incur no penalties.

The entire SONIC*mole* debugging environment has been used for real applications. It can reside and operate independently on each PIPE. Monitoring and controlling to each PIPE from the SONIC*bug* program is changeable at runtime. It has proved to shorten verification and debugging time significantly, as well as has used in some specific jobs; for example, using with a counter, it can precisely measure the execution time of tasks implemented inside PIPE [8]. The transparency of the hardware *mole* and the ease of use of the SONIC*bug* program have made SONIC*mole* an invaluable tools for all developers on the UltraSONIC system.

Although SONIC*mole* has been designed specifically with UltraSONIC in mind, its basic concept and most of its implementation can easily be adapted for use on any reconfigurable computing system.

6. ACKNOWLEDGEMENTS

The authors would like to acknowledge the help and support of Simon Haynes, John Stone and their UltraSONIC team at Sony Broadcast Professional Laboratory, Basingstoke.

7. REFERENCES

[1] Vemuri R. R., Harr R. E.: Configurable Computing: Technology and Applications, *IEEE Computer*, April 2000, pp 39-40.

[2] Haynes S. D., and others: UltraSONIC: A Reconfigurable Architecture for Video Image Processing. *Field-Programmable Logic and Applications (FPL)*, 2002

[3] Haynes S. D., Stone J., Cheung P. Y. K., Luk W.: Video Image Processing with the Sonic Architecture. *IEEE Computer*, April 2000, Page(s) 50-57.

[4] Link Instruments - PC based Logic Analyzer: http://www.linkinstruments.com/logana.htm

[5] Tektronix - Logic Analyzers: http://www.tektronix.com/Measurement/logic_analyzers/

[6] Xilinx, The Tagalyzer - A JTAG Boundary Scan Debug Tool, *Application Note XAPP 103*, January 23, 1998.

[7] Xilinx, ChipScope ILA: Software and Cores, *User Manual*, March, 2002

[8] Wiangtong T., Cheung P. Y. K., Luk W.: Multitasking in Hardware-Software Codesign for Reconfigurable Computer, submitted to *International Symposium on Circuits and Systems*, 2003

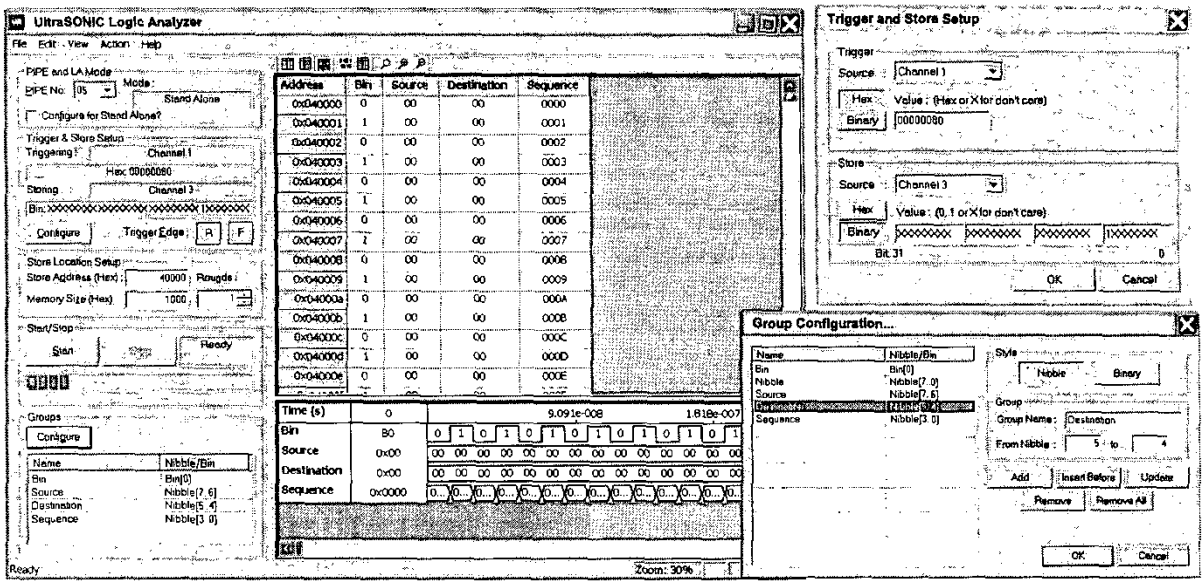

Figure 6. The SONIC*bug* user interface showing signal states as history lists and as timing diagrams