

# An Architecture Simulator for National Semiconductor's Adaptive Processing Architecture (NAPA)

Jeffrey M. Arnold  
*Consultant*  
*jmarnold@znet.com*

## 1 Introduction

Early simulation is a very important tool in the development any large scale system. Accuracy and flexibility are critical characteristics which allow the architect to explore the design tradeoff space. Moreover, in many systems, especially those for reconfigurable computing, a good simulation environment will continue to be used long after the architecture solidifies, serving a variety of roles including as a platform for the development of run time systems, programming tools, benchmarks, and even end applications[1]. Therefore, visibility, controllability and user interface are also important design considerations.

National Semiconductor's Adaptive Processing Architecture (NAPA) integrates a Fixed Instruction set Processor (FIP), an Adaptive Logic Processor (ALP), memory and other support circuitry into a single reconfigurable computing device[3]. The NAPA architecture simulator, NAPAsim, consists of a C language, cycle accurate model of the RISC core, peripherals and memories, coupled with an event driven logic simulator for modelling the user-defined contents of the reconfigurable logic and a Tcl/Tk[2] based GUI to provide source level symbolic debugging capabilities. NAPAsim was developed to serve as both a tool for architectural exploration and as a platform for system and application software development.

## 2 Configurability

To support the architectural exploration NAPAsim was developed as a "configurable" simulator. A run time configuration file optionally provided by the user determines such architectural parameters as the size, shape, number and performance of memories; the type of FIP processor; the presence and configuration of caches; and the presence of peripheral devices. Built in support exists for linking in user-provided simulation models of external devices. To model multiprocessing configurations NAPAsim is structured to allow the direct simulation of clusters of NAPA devices connected

via the ToggleBus.

The configurability of NAPAsim allowed the NAPA designers to explore a number of architectural tradeoffs, including the ratio of the area devoted to memory vs. reconfigurable logic; the performance of internal vs. external memory; and the efficacy of a hardware managed instruction cache vs. a software managed fast on-chip SRAM.

## 3 Debugging and Instrumentation

To fill the role of software development platform, NAPAsim was designed with extensive instrumentation to support debugging and performance tuning of the application. This instrumentation includes the ability to set a variety of breakpoints, the collection of run time statistics, and automatic, non-intrusive procedure level profiling.

Standard source code breakpoints allow the user to break execution at a given instruction or source line of the FIP program. Additionally, *data breakpoints* allow the user to break execution when particular events occur in memory, FIP registers, or ALP signals. The user can specify a symbol name or memory location to cause a break whenever the value of the named object changes. Alternatively, the user can specify a particular value (with an optional bit mask) to cause a break only when the named object changes to or from the specified value. This mechanism effectively allows the user to set breakpoints on events in the ALP logic. For example, a break can be set whenever a particular user defined signal or bus reaches a certain value.

A trace mechanism allows the user to record changes in value of a selected set of signals, registers and memory locations. The trace data may be sent to a file in a format of the user's specification, or it may be viewed in a logic analyzer like waveform display.

The collection of run time statistics provides details of the utilization of the various resources in the NAPA system. Statistics collected on the FIP execution include the average number of clock cycles per

instruction, the average instruction fetch and data access times, and the percentage of time spent executing LOAD and STORE instructions. For each memory, the average utilization is reported. The average core bus utilization and contention are also reported.

FIP source level profiling is provided to allow the user to dynamically analyze the performance of an application on a per procedure basis. Profiling within the simulator is *non-intrusive*, that is, no additional code is added to the application program. Profiling and run time statistics together provide the application programmer with all of the information needed to perform iterative refinement of an application.

Support also exists for architectural debugging. For example, by enabling certain switches the architect can watch the queuing behavior of the buses and memories to identify arbitration problems and resource contention.

## 4 Implementation

The NAPA simulator is written in ANSI C, C++ and Tcl and runs on both Linux and 32-bit Windows platforms. Figure 1 illustrates the structure of NAPAsim. The NAPAsim kernel is a standalone program that communicates with the user through a simple command line interpreter. For each NAPA component in the system there exists a set of models for the FIP, peripherals and memories, and a logic simulation database for the ALP. A simple command line interpreter communicates with the simulation engine through a well defined API. The user can directly control the simulator through the command interpreter, or can interact with the graphical user interface (GUI). The GUI, written in Tcl/Tk, runs as a separate process and communicates with the NAPAsim kernel through its own interface to the API.

The GUI provides the user with several different views of the application, including the C source code, the disassembled text, a “watch list” of program symbols and locations, the contents of memory, the performance profile, and the run time statistics. All of these views are updated dynamically as the application executes. The user can also display the contents of any of the memories, and can trace through the logic structure in the ALP. In addition, the GUI provides extensive access to the on-line User Manual.

The command interpreter of the standalone kernel can read from script files, making it ideal for batch mode execution. This mode of operation is useful for running regression tests and for performing architectural tradeoff experiments. These experiments are organized as a set of simulator configuration files, with

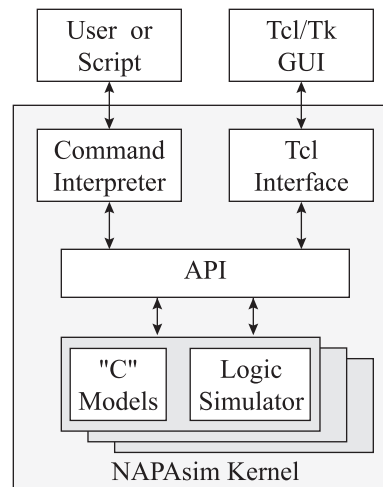


Figure 1: Implementation of NAPAsim

each file defining a system with a different set of architectural parameters. A set of benchmark programs are then executed on the simulator for each different architecture, the results collected and analyzed.

The cycle accurate nature of NAPAsim permits us to derive test vectors for the major blocks of the chip for use in the logic design. The trace facility is used to generate a record of the expected behavior of the top level buses of the chip. A separate program, cbstim, generates Verilog stimulus files from the trace data, which are then used to drive the block and chip level logic simulation.

## 5 Acknowledgements

This work was supported by National Semiconductor Corp., DARPA through Contract DAB63-94-C-0085 to NSC. Special thanks to the NAPA1000 team, including Tim Garverick, Charlé Rupp, Edson Gomersall, Mark Landguth, Harry Holt, Maya Gokhale, Jeff Hutchings and Stephen Bade.

## References

- [1] Buell, D.A., Arnold, J.M. and Kleinfelder, W.J., *Splash 2 FPGAs in a Custom Computing Machine*, IEEE CS Press, Los Alamitos, CA, 1996.
- [2] Ousterhout, J.K., *Tcl and the Tk Toolkit*, Addison Wesley, Reading, Mass., 1994.
- [3] Rupp, C. et al., “The NAPA Adaptive Processing Architecture,” Submitted to FCCM’98.