

## **DRIVE: An Interpretive Simulation and Visualization Environment for Dynamically Reconfigurable Systems\***

Kiran Bondalapati and Viktor K. Prasanna

Department of Electrical Engineering Systems  
University of Southern California  
Los Angeles, CA 90089-2562, USA  
{kiran, prasanna}@ceng.usc.edu  
<http://maarc.usc.edu/>

**Abstract.** Current simulation tools for reconfigurable systems are based on low level simulation of application designs developed in a High-level Description Language(HDL) on HDL models of architectures. This necessitates expertise on behalf of the user to generate the low level design before performance analysis can be accomplished. Most of the current simulation tools also are based on static designs and do not support analysis of dynamic reconfiguration.

We propose a novel interpretive simulation and visualization environment which alleviates these problems. The **D**ynamically **R**econfigurable systems **I**nterpretive simulation and **V**isualization **E**nvironment(**DRIVE**) framework can be utilized for performance evaluation and architecture and design space exploration. *Interpretive* simulation measures the performance of an application by executing an abstract application model on an abstract parameterized system architecture model. The simulation and visualization framework is being developed in *Java* language and supports modularity and extensibility. A prototype version of the **DRIVE** framework has been implemented and the complete framework will be available to the community.

### **1 Introduction**

Reconfigurable systems are evolving from rapid prototyping and emulation platforms to a general purpose computing platforms. The systems being designed using reconfigurable hardware range from FPGA boards attached to a microprocessor to systems-on-a-chip having programmable logic on the same die as the microprocessor. Reconfigurable systems have been utilized to demonstrate large speed-ups for various classes of applications. Architectures are being designed which support partial and dynamic reconfiguration. The reconfiguration overhead to change the functionality of the hardware is also being diminished by the utilization of configuration caches and multiple contexts on the same device.

---

\* This work was supported by the DARPA Adaptive Computing Systems Program under contract DABT63-96-C-0049 monitored by Fort Hauchuca.

Compilation of user level programs onto reconfigurable hardware is also being explored.

The general purpose computing area is the most promising to achieve significant performance improvement for a wide spectrum of applications using reconfigurable hardware. But, research in this area is hindered by the absence of appropriate techniques and tools. Current design tools are based on ASIC CAD software and have multiple layers of design abstractions which hinder high level optimizations based on reconfigurable system characteristics. Existing frameworks are either based on simulation of HDL based designs [1, 11, 13] or they are tightly coupled to specific architectures [5, 9, 14](See Section 1.1). It is also difficult to incorporate dynamic reconfiguration into the current CAD tools framework. Simulation tools provide a means to explore the architecture and the design space in real time at a very low resource and time cost. The absence of mature design tools also impacts the simulation environments that exist for studying reconfigurable systems and the benefits that they offer. System level tools which analyze and simulate the interactions between various components of the system such as memory and configurable logic are limited and are mostly tightly coupled to specific system architectures.

In this paper we present a novel interpretive simulation and visualization environment based on modeling and module level mapping approach. The **D**ynamically **R**econfigurable systems **I**nterpretive simulation and **V**isualization **E**nvironment(**DRIVE**) can be utilized as a vehicle to study the system and application design space and performance analysis. Reconfigurable hardware is characterized by using a high level parameterized model. Applications are analyzed to develop an abstract application task model. *Interpretive* simulation measures the performance of the abstract application tasks on the parameterized abstract system model. This is in contrast to simulating the exact behavior of the hardware by using HDL models of the hardware devices.

The **DRIVE** framework can be used to perform interactive analysis of the architecture and design parameter space. Performance characteristics such as total execution time, data access bandwidth characteristics and resource utilization can be studied using the **DRIVE** framework. The simulation effort and time are reduced and systems and designs can be explored without time consuming low level implementations. Our approach reduces the semantic gap between the application and the hardware and facilitates the performance analysis of reconfigurable hardware. Our approach also captures the simulation and visualization of dynamically reconfigurable architectures. We have developed the Hybrid System Architecture Model(HySAM) of reconfigurable architectures. This model is currently utilized by the framework to map applications to a system model.

An overview of our framework is given in Section 2. Various aspects of the simulation and visualization framework including our Hybrid System Architecture Model(HySAM) are described in detail in Section 3. Conclusions and future work are discussed in Section 4.

## 1.1 Related Work

Several simulation tools have been developed for reprogrammable FPGAs. Most tools are device based simulators and are not system level simulators. The most significant effort in this area has been the Dynamic Circuit Switching(DCS) based simulation tools by Lysaght et.al. [13]. Luk et.al. describe a visualization tool for reconfigurable libraries [11]. They developed tools to simulate behavior and illustrate design structure. CHASTE [5] was a toolkit designed to experiment with the XC6200 at a low level. There are other software environments such as CoDe-X [9], JHDL [1], HOTWorks [7], Riley-2 [14], etc.

These tools study the dynamically reconfigurable behavior of FPGAs and are integrated into the CAD framework. Though the simulation tools can analyze the dynamic circuit behavior of FPGAs, the tools are still low level. The simulation is based on CAD tools and requires the input design of the application to be specified in VHDL. The parameters for the design are obtained only after processing by the device specific tools. Most of the software frameworks do not support system level analysis and are utilized for for low level hardware design and evaluation.

## 2 DRIVE Overview

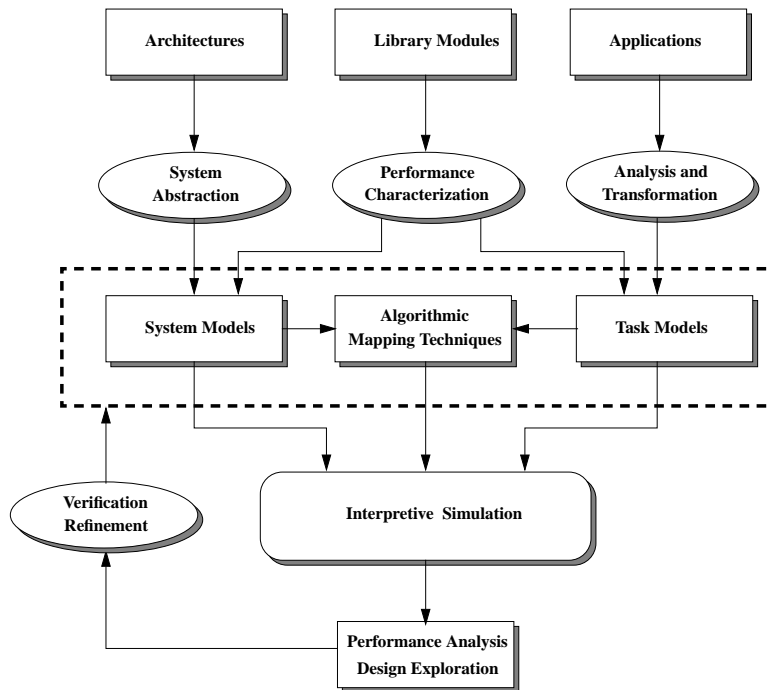


Fig. 1. DRIVE framework

Figure 1 shows an overview of our framework. The system architecture can be characterized to capture the parameter space which affects the performance. The implementations of various optimized modules can be encapsulated by characterizing the performance of the module with respect to the architecture. This characterization is partitioned into the *capabilities* of the system and the actual *implementations* of these *capabilities*. The application is not mapped onto a low level design but is analyzed to develop an application task model. The application model can exploit the knowledge available in the form of the system *capabilities* provided by the module characterization. Algorithmic techniques are utilized to map the application task model to the system models, to perform interpretive simulation and obtain performance results for a given set of parameter values.

Interpretive simulation is performed on the system model which permits a higher level abstract simulation. The application does not need to be actually executed by using device level simulators like HDL models of the architectures. The performance measures can be obtained in terms of the application and model parameters and system characteristics. An interpretive simulation framework will permit design exploration in terms of the architectural choices, application algorithm options, various mapping techniques and possible problem decomposition onto the system components. Development of all the full blown designs which exercise these options is a non-realizable engineering task. Simulation, estimation and visualization tools can be designed to automate this exploration and obtain tangible results in reasonable time.

The abstractions and the techniques that are developed are enclosed in the dashed box in Figure 1. Verification of the models, mapping techniques and simulation framework can be performed by mapping some designs onto actual architectures. This verification process can be utilized to expand on the abstraction knowledge and refine the various models and techniques that are developed. The verification and refinement process completes the feedback loop of the design cycle to result in final accurate models and efficient techniques for optimal designs.

### 3 Simulation Framework

The simulation framework consists of abstractions and algorithmic techniques as discussed in Section 2(Fig. 1). A high level model of reconfigurable hardware is needed to abstract the low level details. Existing models supplied by the CAD tools have either multiple abstraction layers or are very device specific. We have developed a parameterized model of configurable computing system, which consists of configurable logic attached to a traditional microprocessor. Our model cleanly partitions the *capabilities* of the hardware from the *implementations* and presents a very clean interface to the user. The algorithmic techniques for mapping are not the focus of this paper. Some algorithms for mapping based on the HySAM model are described in our prior work [3, 4].

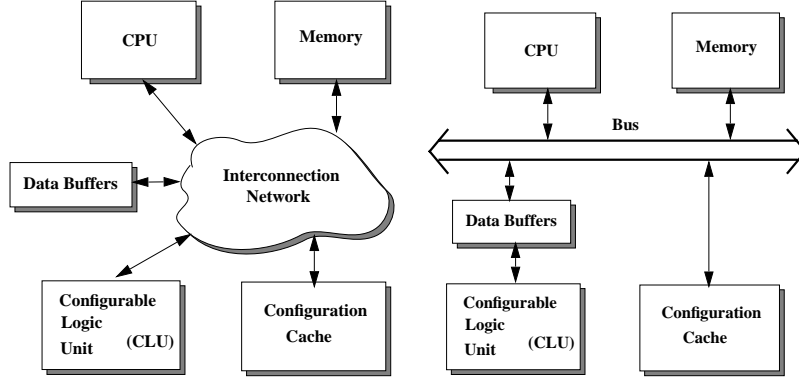


Fig. 2. Hybrid System Architecture and an example architecture

### 3.1 Hybrid System Architecture Model(HySAM)

The *Hybrid System Architecture Model* is a general model consisting of a traditional microprocessor with additional Configurable Logic Unit(CLU). Figure 2 shows the architecture of the HySAM model and an example of an actual architecture. The architecture consists of a traditional microprocessor, standard memory, configurable logic, configuration memory and data buffers communicating through an interconnection network.

We outline some of the parameters of the Hybrid System Architecture Model(HySAM) below.

$F$  : Set of functions  $F_1 \dots F_n$  which can be performed on configurable logic. (*capabilities*)

$C$  : Set of possible configurations  $C_1 \dots C_m$  of the Configurable Logic Unit. (*implementations*)

$A_{ij}$  : Set of attributes for implementation of function  $F_i$  using configuration  $C_j$ .

$R_{ij}$  : Reconfiguration cost in changing configuration from  $C_i$  to  $C_j$ .

$G$  : Set of generators which abstract the composition of configurations to generate more configurations.

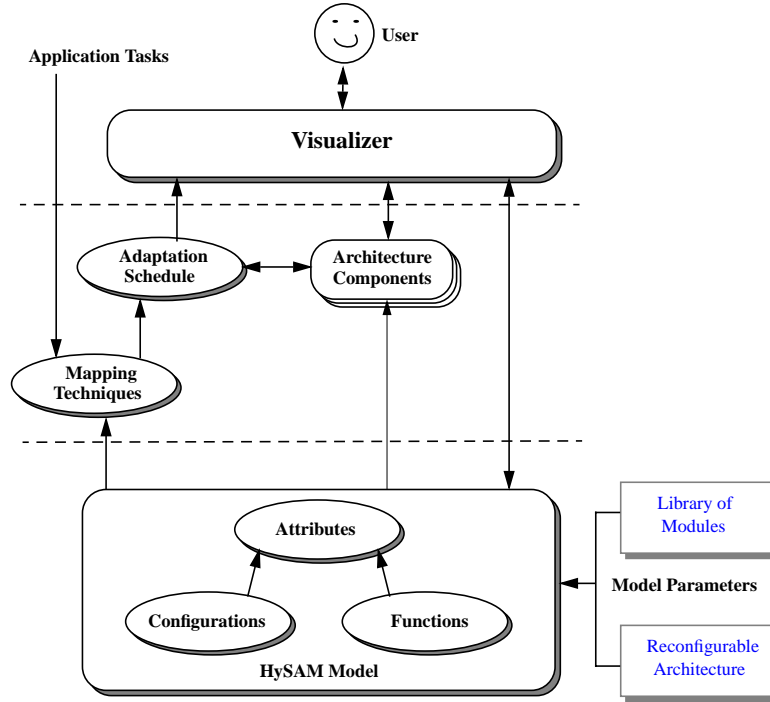
$B$  : Bandwidth of the interconnection network (bytes/cycle).

$N_c$  : The number of configuration contexts which can be stored in the configuration cache.

$k_c, K_c$  : The cost of accessing configuration data from the cache and external memory respectively (cycles/byte).

$k_d, K_d$  : The cost of accessing data from the cache and external memory respectively (cycles/byte).

The functions  $F$  and configurations  $C$  have a *many-to-many* relationship. Each configuration  $C_i$ , can potentially contain more than one function  $F_j$ . In the HySAM model, only function can be active in a configuration at any given time. Each function  $F_i$  can be executed by using any one configuration from a subset of the configurations. The different configurations might be generated by



**Fig. 3.** Major components in the **DRIVE** framework and the information flow

different tools, libraries or algorithms. These configurations might have different area, time, reconfiguration cost, precision, power, etc. characteristics.

The attributes  $A$  define the relationship between the functions and the configurations. The attributes define values such as the execution time and the data accessed during execution of a function in a configuration etc. For example, the different execution times and the different data input patterns when a multiplier is implemented as a bit parallel versus a bit serial multiplier are defined by the attributes. The reconfiguration costs  $R$  define the costs involved in changing the configuration of the CLU between two configurations. This cost can be statically evaluated based on the configuration information for different configurations. The cost can also be computed dynamically when the configurations are constructed dynamically.

### 3.2 DRIVE Framework Implementation

An overview of the major components in the **DRIVE** framework and their interactions is given in Figure 3. The framework utilizes high level models of reconfigurable hardware. The current prototype uses the HySAM model described in Section 3.1.

The main input requirements to the **DRIVE** framework are the model parameters and the application tasks. The model parameters supply information about the Functions, Configurations, Attributes and the Reconfiguration costs.

The user can visualize and update any of the instantiated parameters to explore the design space. For a given model parameters, performance results can be obtained for any set of application tasks with various algorithmic mapping techniques.

The high level model partitions the description of the hardware into two components: the Functions(*capabilities*) of the hardware and the Configurations(*implementations*). For example, ability of the hardware to perform multiplication is a capability. The implementations are the different multiplier designs available with varying characteristics such as area, time, precision, structure, etc. Components from a library or modules form the *implementations* in the model and can be determined for different architectures. Vendors and researchers have developed parameterized libraries and modules optimized for a specific architectures. The proposed framework can exploit the various efforts in design of efficient and portable modules [6, 12, 15]. The framework can incorporate such knowledge as the parameters for the HySAM model.

The user only needs to have a knowledge of the *capabilities*. The application task model consists of specification of the application in terms of the Functions(*capabilities*). The input to the framework consists of a directed acyclic graph of the application tasks specified with the Functions as the nodes of the graph. The edges denote the dependencies between the tasks. This technique reduces the effort and expertise needed on the part of the user. The application need not be implemented as an HDL design by the user to study the performance on various reconfigurable architectures. Automatic compilation efforts [2] can be leveraged to generate the Functions from high level language application programs.

Algorithmic mapping techniques are then utilized to map the application specification to actual implementations. These techniques map the *capabilities* to the *implementations* and generate a sequence of configuration, execution, and reconfiguration steps. This is the *adaptation schedule* which specifies how the hardware is adapted during the execution of the application. The schedule contains a sequence of configurations( $C_1 \dots C_q$ ) where each configuration  $C_i \in C$ . This *adaptation schedule* can be computed statically for some applications by using algorithmic techniques. Also, the simulation framework can interact with the model and the mapping algorithms to determine the *adaptation schedule* at run-time.

The interpretive simulation framework is based on module level parameterization of the hardware. The user can analyze the performance of the architecture for a given application by supplying the parameters of the model and the application task. Typically the architectural parameters for the model are supplied by the architecture designer and the library designer. But, the user can modify the model parameters and explore the architecture design space. This provides the ability to study design alternatives without the need for actual hardware. The simulation and the performance analysis are presented to the user through a Graphical User Interface. The framework supports incorporation of additional information in the configurations( $C$ ) which can be utilized for actual execution

or simulation. It can contain configuration bitstreams or class descriptions which can be utilized to perform actual configuration of hardware or simulation using low level models. Using this information, it is possible to link the abstract definitions to actual implementations to verify and refine the abstract models.

The parameters and attributes of the model can also be evaluated and adapted at run-time to compute the required information for scheduling and visualization. For example, reconfiguration costs can be determined by computing the difference in the configuration information and configurations can even be generated dynamically by future integration of tools like JBits [10]. It is assumed currently that the attributes for configurations are available a priori. It is easy to integrate simulation tools which evaluate the attributes such as execution time by performing simulations as in various module generators [1, 6, 15]. These simulations are based on module generators which do not require mapping using time consuming CAD tools. Once the attribute information for low level modules are obtained by initial simulations and implementations, the attributes for higher level modules can be simulated or computed without the intervention of CAD tools.

The **DRIVE** framework has been designed using object-oriented methodology to support modification and addition to the existing components. The framework facilitates addition of new architectural models, algorithmic mapping techniques, performance analysis tools, etc. in a seamless manner. The framework can also be interfaced to existing tools such as parameterized libraries(Xilinx XBLOX, Luk et. al. [12]), module generators(PAM-Blox [15], Berkeley Object Oriented Modules [6], JHDL [1]), configuration generators(JBits [10]), module interfaces(FLexible API for Module-based Environments [8]), etc. The components of the framework will be made available to the community to facilitate application mapping and modular extensions.

### 3.3 Visualization

The visualizer for the framework has been developed using the *Java* language AWT toolkit. A previous version of the visualizer was developed using Tcl/Tk. The C programming language was utilized for implementing the simulation engine. The current prototype has been developed in *Java* to utilize the object oriented framework and make the framework modular and easily extensible. Implementing the visualizer and the interpretive simulation in the same language provides for a clearer interface between the components. *Java* is becoming the language of choice for several research and implementation efforts in hardware design and development [1, 6, 10]. Incorporating the results and abstractions from other research efforts is simplified using the current version.

The visualizer acts as a graphical user interface to support the full functionality of the framework. It is implemented as a separate *Java* class communicating with the remaining classes. Any component of the simulation or visualizer framework can be completely replaced with a different component supporting the same interface. The visualizer is oblivious of the algorithmic techniques and implementation details. It accesses information from the different components

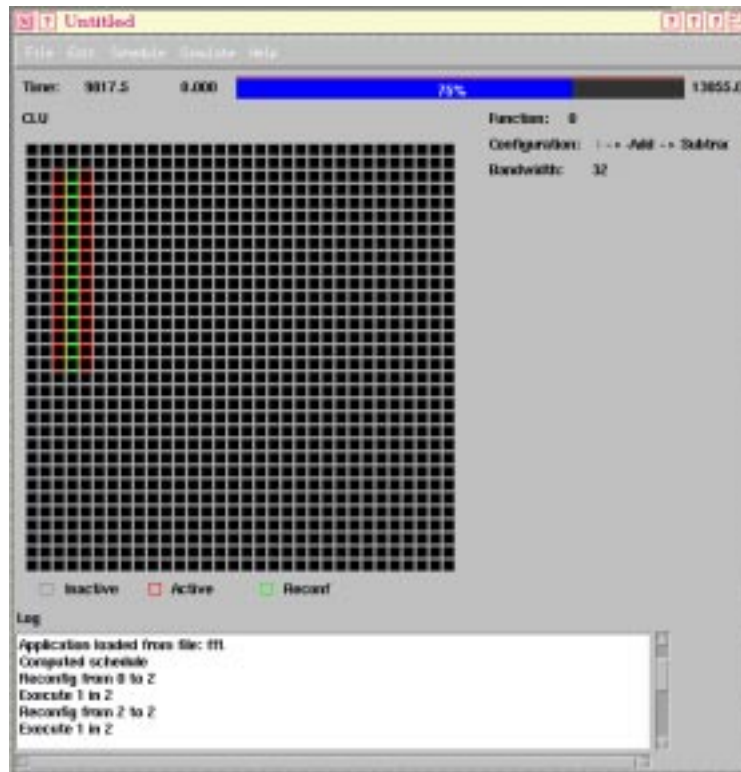


Fig. 4. Sample **DRIVE** visualization

in the simulation framework on an event by event basis and displays the state of the various architecture components and the performance characteristics. A sample view of the visualizer is shown in Figure 4.

## 4 Conclusions

Software tools are an important component of reconfigurable hardware development platforms. Simulation tools which permit performance analysis and design space exploration are needed. The utility of current tools for reconfigurable hardware design is limited by the required user expertise in multiple domains. We have proposed a novel *interpretive* simulation and visualization environment which supports system level analysis. The **DRIVE** framework supports a parameterized system architecture model. Algorithmic mapping techniques have been incorporated into the framework and can be extended easily. The framework can be utilized for performance analysis, design space exploration and visualization. It is implemented in the *Java* language and supports flexible extensions and modifications. A prototype version has been implemented and is currently available. The USC Models, Algorithms and Architectures project is developing algorithmic techniques for realizing scalable and portable applications using con-

figurable computing devices and architectures. Details on **DRIVE** and related research results can be found at <http://maarc.usc.edu>.

## References

1. P. Bellows and B. Hutchings. JHDL - An HDL for Reconfigurable Systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1998.
2. K. Bondalapati, P. Diniz, P. Duncan, J. Granacki, M. Hall, R. Jain, and H. Ziegler. DEFACTO: A Design Environment for Adaptive Computing Technology. In *Reconfigurable Architectures Workshop, RAW'99*, April 1999.
3. K. Bondalapati and V.K. Prasanna. Mapping Loops onto Reconfigurable Architectures. In *8th International Workshop on Field-Programmable Logic and Applications*, September 1998.
4. K. Bondalapati and V.K. Prasanna. Dynamic Precision Management for Loop Computations on Reconfigurable Architectures. In *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1999.
5. G. Brebner. CHASTE: a Hardware/Software Co-design Testbed for the Xilinx XC6200. In *Reconfigurable Architectures Workshop, RAW'97*, April 1997.
6. M. Chu, N. Weaver, K. Sulimma, A. DeHon, and J. Wawrzynek. Object Oriented Circuit-Generators in Java. In *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1998.
7. Virtual Computer Corporation. Reconfigurable Computing Products, <http://www.vcc.com/>.
8. A. Koch. Unified access to heterogeneous module generators. In *ACM International Symposium on Field Programmable Gate Arrays*, February 1999.
9. R. Kress, R.W. Hartenstein, and U. Nageldinger. An Operating System for Custom Computing Machines based on the Xputer Paradigm. In *7th International Workshop on Field-Programmable Logic and Applications*, pages 304–313, Sept 1997.
10. D. Levi and S. Guccione. Run-Time Parameterizable Cores. In *ACM International Symposium on Field Programmable Gate Arrays*, February 1999.
11. W. Luk and S. Guo. Visualising reconfigurable libraries for FPGAs. In *Asilomar Conference on Signals, Systems, and Computers*, 1998.
12. W. Luk, S. Guo, N. Shirazi, and N. Zhuang. A framework for developing parametrised FPGA libraries. In *Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, 1996.
13. P. Lysaght and J. Stockwood. A Simulation Tool for Dynamically Reconfigurable FPGAs. *IEEE Transactions on VLSI Systems*, Sept 1996.
14. P.I. Mackinlay, P.Y.K. Cheung, W. Luk, and R. Sandiford. Riley-2: A Flexible Platform for Codesign and Dynamic Reconfigurable Computing Research. In *7th International Workshop on Field-Programmable Logic and Applications*, September 1997.
15. O. Mencer, M. Morf, and M.J. Flynn. PAM-Blox: High Performance FPGA Design for Adaptive Computing. In *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1998.