

# A Behavioral Synthesis Estimation Interface for Configurable Computing

## *Extended Abstract*

**Pedro Diniz and Ashok Venkatachar**

University of Southern California / Information Sciences Institute  
4676 Admiralty Way, Suite 1001  
Marina del Rey, California 90292  
{pedro, ashok}@isi.edu

### 1. Introduction

The dramatic increase of the number of available transistors on a die have enabled the use of Field-Programmable-Gate-Arrays (FPGAs) as custom computing elements. The mapping of ever larger computations to larger FPGAs, however, has exacerbated the problems of current design flows. Mapping steps such as place-and-routing can now last in the order of hours to accomplish; as the substantial increase in configurable elements and routing possibilities has undoubtedly stressed the algorithmic limitations of current synthesis tools.

In this abstract we describe an estimation interface for behavioral synthesis that attempts to represent in a common format the parameters, resources constrains, and estimation results for existing synthesis tools. This interface includes functions to generate the scripts automatically and invoke the synthesis tool as well as parse the estimation results. The internal data structures allow a designer to retain constraint values and observe the results of the estimation without ever knowing or invoking the synthesis tool.

We have implemented our estimation interface for two popular behavioral synthesis tools - the Synopsis Behavioral Compiler™ (BC) [1] and the Mentor Graphics Monet™ (MT) [2] synthesis tool. **Using this estimation interface we have coded in C several design exploration algorithms and applied them successfully in the quick generation and selection of alternative designs for a small set of computations for the two behavioral synthesis tools mentioned above.**

This experience reveals that it is possible to integrate, in a seamless way, compilation and synthesis tools. With the growing number of available transistors on a die it will be increasingly desirable to use program analysis, available in the compilation community, in the mapping of computations to configurable computing devices such as

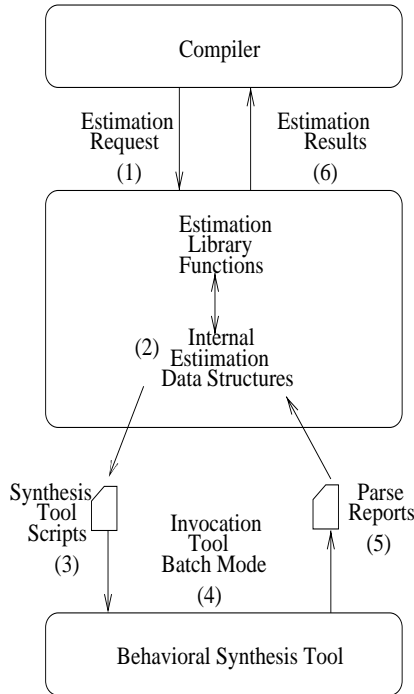
FPGAs. The estimation interface described here is a first step to bridge the interface barriers between the two domains and enable the development of more sophisticated design flows with feedback between the compilation and synthesis domains. The experience described here reveals the importance of the proposed interface in allowing access to estimation features of existing tools to a wide range of users and programmers.

### 2. Behavioral Synthesis Estimation Interface

Given a specific design in behavioral VHDL, the interface allows the programmer to *request* a set of estimates from the synthesis tool. Each *request* is defined by a set of parameters and a set of resource constraints and throughout the life of the design cycle it is assigned a unique identifier. The interface uses a set of internal functions to format the request using the syntax specific to the synthesis tool at hand. The interface then invokes the synthesis tool in batch mode and parses the results the tool has generated in the form of report files into data structures. The interface then allows the programmer to inspect the data structures to extract data about the *estimates* the synthesis tool has generated. The designer is expected to use this data in the selection of the best implementation that matches its needs. In addition we have developed initialization and ending procedure that interface with the tools execution environment for the setup and maintenance of the directories and files as well as environment variables so that the programmers need not worry about running these scripts manually.

Figure 1 below illustrates the control flow inside the interface for a given estimation request. In step 1 the programmer loads the design file references into internal data structures along with a list of specification for the constraints to be used in behavioral synthesis. Internally the interface creates a data structure that uses in step 2 to generate a tool specific execution script. In step 3 the interface invokes the synthesis tool in batch mode. In step 4 the interface extracts the results from the tool generated report files and loads these results into a data structure associated with the request submitted in step 1. Finally in

step 5 the programmer can access the information stored in step 4 and refine the request for subsequent designs. For this purpose we have included functions that create, destroy and manipulate estimate request so that programmers and/or compiler designer can easily add/remove design constraints.



**Figure 1. Estimation Interface Control Flow.**

Part of the difficulty of interfacing multiple synthesis tools is related to the fact that a given synthesis tool may not allow to constrain its design implementation and estimation using metrics another allows. For example, while Monet™ allows the design to be constrained by specifying the number of individual components to be used in the design, in BC™ this option of resource constraining can only be applied not to the whole design but rather the programmer must specify a set of particular operators in the source VHDL to which these components are to be bound. The BC™ tool also allows the designer to specify the maximum number of clock cycles to be used in the execution of a particular loop, option that is not directly available in Monet™. Because of the distinct approach for its internal solution implementation strategy we support in the interface the information required for each different paradigm. When using one tool, part of the requested data is unused whereas that same information is used for the other tool.

We currently support the following set of constraint specifications:

- Clock period for each hardware process along with the maximum allowed clock overhead (MT and BC)

- Set of component libraries to use (MT and BC).
- Component selection and maximum number of instances (MT only)
- Number of maximum clock cycles per iteration for a given loop (BC only).

Notice that the information specified above does not include any tool-specific transformations such as loop unrolling, pipelining pragmas or variable mapping to library RAM modules. We view the specification of these pragmas as orthogonal design definition specifications and are therefore left for the programmer to directly control as part of higher-level source code transformations.

As output of the design estimation the interface parses the following data into its data structures.

- Obtained clock period, clock cycle delay and the maximum delay path (critical path).
- Number of operations per hardware process that are to be bound to which library components.
- Number of control cycles (c-steps in Monet™) and iterations per loop of the design.
- Estimated area and number of each of the used components. Total area consumed by the design.

The interface provides several utility functions to copy and manipulate the resulting results so that the programmer can easily refine a given set of constraints without explicitly stating all of them from scratch. In addition we also provide trivial functions to compute total area and global clock cycle counts for a given design as there may be several loop constructs associated with the design.

We have developed and implemented the interface and API functions outlined above in C (approximately 1,000 lines of source code) under Solaris™ and compiled with gcc as well as SunPro Compiler. We currently support all scripting and initialization for the Synopsys Behavioral Compiler™ (v2000.05) and Mentor Graphics Monet™ (R43 or 8.7-1.1).

### 3. Summary

We have briefly described a uniform estimation interface to two commercially available behavioral synthesis tools. Using this interface we have developed a simple design exploration strategy and applied it to a set of kernels computations. This experience reveals the importance of the proposed interface in allowing access to estimation features of existing tools to a wide range of users and programmers.

### References

[1] Behavioral Compiler™ User's Manual, Synopsys Inc., 2000.  
 [2] Monet™ User's Manual, Mentor Graphics Inc., 2000.