Title
IP characterization and reuse in behavioral synthesis

Authors
Fan, Nong
Chaiyakul, Viraphol
Gajski, Daniel D.

Publication Date
2000-01-06

Peer reviewed

ICS

TECHNICAL REPORT

IP Characterization and Reuse in Behavioral Synthesis

Nong Fan†
Viraphol Chaiyakul‡
Daniel D. Gajski†

Technical Report #00-01
January 6, 2000

†Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697
(949) 824-8059

‡Y Explorations, Inc.
20902 Bake Parkway, Suite 100
Lake Forest, CA 92630
(949) 457-0294

nfan@ics.uci.edu
viraphol@yxi.com
gajski@ics.uci.edu

Abstract

This report presents a usage-based component characterization to facilitate reuse of presynthesized complex component in behavioral synthesis. It identifies necessary attributes needed to be obtained for component reuse, and demonstrates the importance of these attributes in behavioral synthesis tasks, such as allocation, scheduling and binding.

Information and Computer Science
University of California, Irvine

Contents

List of Figures

1 Introduction

Recent advances in semiconductor technology allows companies to build complex designs containing millions of gates on a chip. The need for design automation on higher abstraction levels where functionality and design tradeoff are easier to understand. is unavoidable. Meanwhile, the time-to-market requirement is becoming increasingly aggressive. A quantum jump in design productivity is necessary and that is obtainable through the reuse of pre-designed and pre-verified IP blocks or cores, such as memories, FIR filters, or DSP cores that perform MPEG encoding and decoding algorithms. Thus, design reuse in behavioral synthesis is becoming the methodology of choice for improving design quality, productivity and predictability.

To make design reuse in behavioral synthesis a reality, synthesis tools need cutting-edge libraries containing various components, which can be as simple as gates or as complex as DCT and MPEG cores to support a variety of applications. Synthesis system can not use a component until it is properly characterized. Component characterization is a process in which the component attributes, which are necessary for it to be reused, are extracted and stored in design libraries. If behavioral synthesis algorithms are going to support reuse, components in the library have to be characterized in such a way that the algorithms can handle those components.

Figure 1(a) shows an adder which are designed by using gates as building blocks in a design library. Once the design is finished, we can put the adder back to the library by extracting such attributes as bitwidth, delay, area and power consumptions and

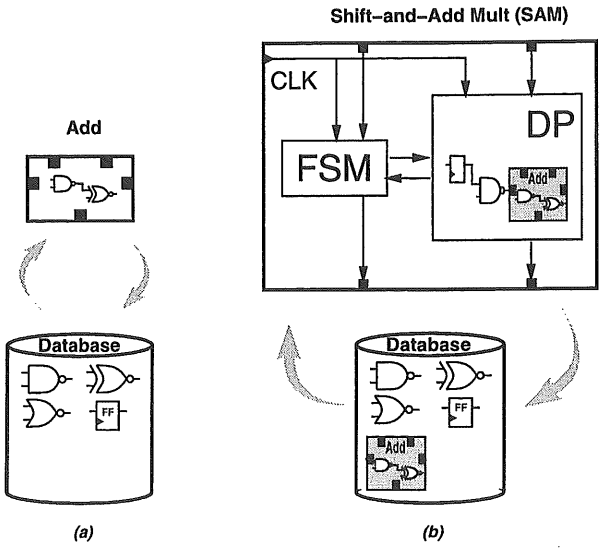

Figure 1: (a) Use gate to design adder and put it back to the database (b) use the adder to design SAM and put it back to the database.

storing them into the library. We have already known that with these informations, this adder can be reused in behavioral synthesis systems.

Figure 1(b) shows an shift-add-multiplier (SAM) which are designed by using gates and the adder which are just designed in Figure 1(a) as the building blocks. Let us assume that this SAM has only one input port to get operands. Thus, two operands are fed into it at different clock cycles. Once the computation is done, it sets done signal to one, notifying that the result is ready. After finishing this design, we still want to put it back into the library for future reuse. Now the question is: what attributes about the SAM need to be extracted and stored in the design library such that behavioral synthesis systems are able to reuse it? Obviously, the attributes used to characterize the adder are not enough for the SAM to be reusable in behavioral synthesis systems. The reason is that, compared to the adder, the SAM is a sequential circuit, has its own Finite State Machine

(FSM), and has a much more complex I/O interface. So there is a need to investigate how to characterize such complex components such that behavioral synthesis algorithms can handle them.

In this paper, we will present a novel usage-based component characterization approach for design reuse in behavioral synthesis. Section 2 will present the previous work on component characterization. Section 3 will present the problem statement and target architecture. Section 4 will present our usage based component characterization approach. Section 5 will briefly present the importance of the attributes in support of reuse in behavioral synthesis. Section 6 will show some experiment results and conclude the paper.

2 Previous Work

Traditionally, component characterization is component based, that is, for each component, information regarding timing, area and power consumption is extracted and stored in the design library. The timing information may include clock constraints, input setup/hold timing constraints, input to output delays, and clock to output delays. The area information may include gate counts or height/width of the layout. For more complex components, e.g., sequential components with handshaking protocols, signal relationships are specified using timing diagrams in data book. With the help of detailed documentation, designers would be able to reuse the components through manually transforming original algorithmic descriptions into RTL descriptions with selected components instantiated. However, every time an alternative component is selected, the RTL code needs to be modified to reflect the change of components. Thus, the traditional component characterization approach does not provide enough component information for behavioral synthesis algorithms to be able to reuse them.

Recently, several works have been published on how to abstract presynthesized components in design libraries such that they can be reused in behavioral synthesis. In AMICAL[1], each component are abstracted at four different levels: the conceptual view, the behavioral view, the implementation view and the high-level synthesis view. The forth view is used to link behavioral and implementation view. Though their approach supports characterization of multi-functional components, it does not support components which perform the same operation on different data types since in the synthesis view, components are abstracted based on operation name only. In OOCADSyn[2], an object oriented scheme is used to model presynthesized components, where each component is modeled as a class which has a structure and behavior. The behavior of a class is captured by messages and methods, while the structure of a class is used to describe the constituent components of a circuit, and its various attributes such as delay, area, and power etc. For every message there will be an interface graph which captures the I/O behavior of the message. Operations are performed by sending appropriate messages to components. However, sending an appropriate message implies that component selection needs to be decided manually before writing descriptions.

On the other hand, our usage based component characterization can support reuse in component se-

lection, scheduling and binding.

3 Problem Statement and Target Architecture

Given a presynthesized component, component characterization is to identify necessary attributes about the component and store them into design libraries such that synthesis tools are able to reuse it.

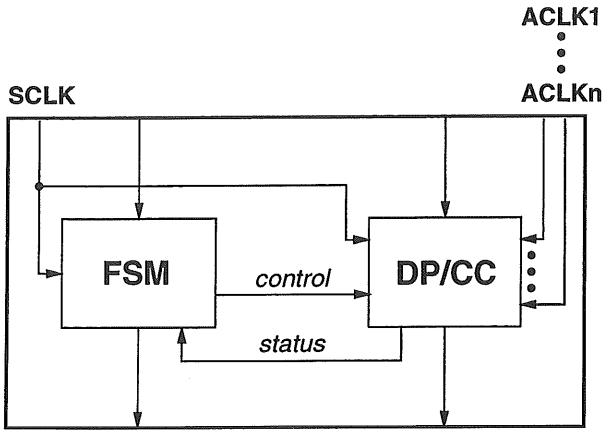

Figure 2: Target Architecture.

Figure 2 shows the target architecture which allows us to design a circuit using components with various complexities. It is basically a Finite State Machine with Datapath (FSMD) [4]. The FSM controls components in the datapath. Besides traditional RTL components, the datapath may contain complex components(CC). These complex components may run at their own clocks and communicate with the FSM through hand-shakings. Here we call the clock going to the FSM as system clock (SCLK) and the clocks going to datapath components as auxiliary clocks (ACLKs).

4 Usage Based Component Characterization

The usage based component characterization is to characterize a component based on its usage, i.e., the functionality of operations it can perform and the conditions under which it performs the operations. It includes 4 tasks: 1)specifying component usage, 2)describing component structural attributes, 3)describing interface template for each usage and 4) describing mapping protocols.

4.1 Specifying component usage

The first step in usage based characterization is to specify usages of a component. The usage of a component is specified by the functionality of operations it is to perform and the conditions under which it performs the operations.

The functionality of an operation is specified by giving the name of the operation and the data types of operands. In an algorithmic description, operations are performed through subroutine calls with appropriate parameters passed into them. The subroutines can be predefined functions or procedures in description languages, or can be user defined. In some Hardware Description Languages (HDLs), such as VHDL, where subroutine overloading is allowed, both the subroutine name and the parameter data types need to be specified in order to uniquely identify the functionality of the operation. For example, the functionality of multiplying two signed numbers is different from that of multiplying two unsigned numbers.

Conditions under which a component performs

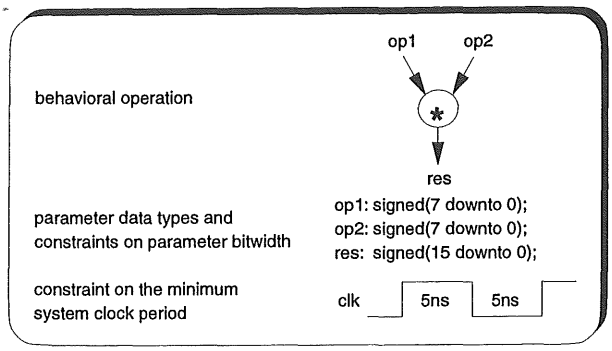

Figure 3: One usage for Shift-and-Add Multiplier (SAM).

the required operation include the constraint on the system clock, $SCLK$, used for synthesis and the constraints on the bitwidths of operands. The constraint on $SCLK$ specifies the requirement of the system clock for using the component. Suppose that the minimum period of $20ns$ is the constraint of a usage on the system clock. If the period of system clock used for synthesis is $10ns$, then the component can not be used to perform the required operation. The constraints on the bitwidths of operands specify the size of the operands the component can take. For example, if the constraints on operand bitwidth is 8, then the component can not be used to perform a 16-bit operation.

One component may have multiple usages if it can perform different operations or perform the same operation under different conditions. On the other hand, different components may have the same usage since within one library, there may exist more than one component which can perform the same operation under the same conditions. A component can be selected to perform the required operation **only if** the functionality of the operation is exactly matched, and the constraints on the system clock and on operand bitwidths are satisfied.

Figure 3 shows one usage for the Shift-and-Add Multiplier, SAM. From the example, we know that the SAM can be used to perform multiplication of two 8-bit signed numbers if the system clock period is not less than $10ns$.

4.2 Describing structural attributes

The structural attributes need to describe port direction and bitwidth, input port setup/hold timing constraints, input to output delays, clock to output delays and component clock constraints.

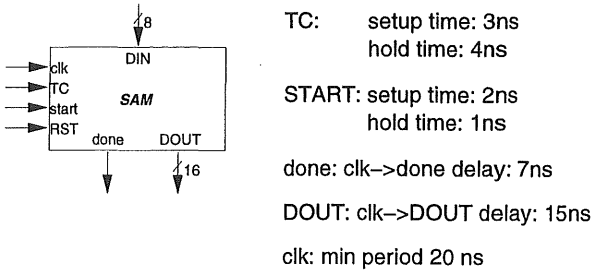

Figure 4: Structural attributes for SAM.

Figure 4 shows the structural attributes of the SAM. It has four 1-bit control input ports, one 8-bit data input port DIN, one 1-bit output port *done* and one 16-bit data output port $DOUT$. It also lists setup/hold timing constraints on input ports, clock to output delays and constraint on clock.

4.3 Describing interface template for each usage

For each usage of a component, there is one interface template. The purpose of the interface template is to hide the implementation details of a component, while providing enough information about how

to perform the required operation under the given conditions specified in the usage. It bridges the gap between the behavioral operation and the component. The usage shown in Figure 5(a) specifies that the SAM can perform multiplication of two signed numbers. The operation takes two operands and generates one result. The constraints to perform the operation are that (1) the bitwidth of both input operands is 8, and the bitwidth of result is 16, and (2) the minimum system clock period is $10ns$. Figure (c) shows the structural component. The component has one circuit reset signal rst, one clock signal clk and one start signal $START$. The signal tc determines whether the input and output data is interpreted as unsigned ($tc = 0$) or signed ($tc = 1$) numbers. Both operands are fed into the SAM through input port DIN at different cycles. Due to the data dependent execution time, it uses the signal $done$ to denote the validity of the result. From Figure (a) and (c) we can obtain the following observations: first, the behavioral operation and the SAM have different number of ports. Secondly, the component has more complicated mechanism to receive operands and send results than the behavioral operation. To bridge the gap between the behavioral operation and the component, an interface template is needed.

The interface template is modeled as an design entity with two sets of ports, left ports L_i and right ports, R_j. Left ports are used by the interface template to get operands and to send out results. For every parameter of the operation specified in the usage, there must exist a corresponding left port. Right ports are used by the interface template to communicate with the component. They will be connected to the ports of the component. A right port can be connected to more than one component port.

The behavior of the interface template is specified using a state transition diagram, where the nodes represent I/O operations of the component, and the edges represent state transitions, which are controlled by the system clock.

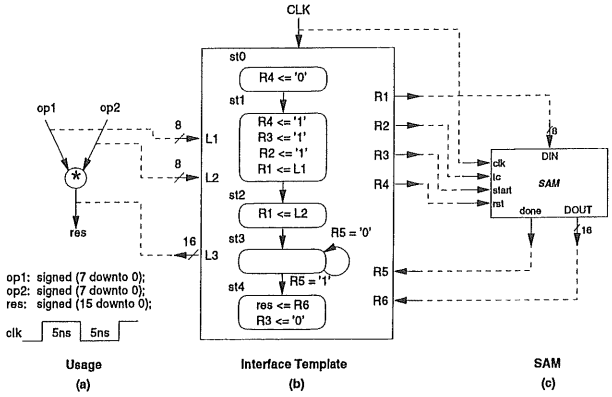

Figure 5: (a) One usage of the SAM (b) Interface template of the SAM (c) Component SAM

Figure 5(b) shows the interface template of the SAM for the usage specified in Figure 5(a). It has 3 left ports and 6 right ports. The state transition diagram inside the template shows how to control the SAM to perform the required operation, i.e., multiplying two signed numbers. In the initial state ST_0, it resets the SAM by sending '0' to it through the right port R_4. In the state ST_1, it gets the first operand through the left port L_1 and sends it to the SAM through R_1. It also sends '1' to the port TC of the SAM to notify that operands should be treated as signed numbers. After sending the second operand in the state ST_2, it waits in the state ST_3 until the signal from the port R_5 becomes '1', indicating that the result is ready. In the next state, ST_4, it gets the result from the port R_6 and sends it out through the port L_3.

4.4 Describing mapping protocols

The mapping protocols consist of two parts, left mapping protocol and right mapping protocol. The left mapping protocol specifies the one-to-one correspondence between operation parameters and left ports of the interface template. For each parameter or result of the operation, there must exist one and only one corresponding left port, and vise versa. The right mapping protocol specifies the connections between the right ports of the interface template and the ports of component. As shown in Figure 5, the dashed lines between the operation in the usage and the interface template represent the left mapping protocol. For instance, the parameter "op1" of the operation "*" corresponds to the left port L_1. The dashed lines between the interface template and the SAM represent the right mapping protocol. For instance, the port R_1 is connected to the port DIN of the SAM.

5 Use of Attributes in Behavioral Synthesis

In this section we present the use of the attributes described in the previous section in behavioral synthesis. Behavioral synthesis is a process of synthesizing a design from a given behavioral description into a structural implementation. It consists of three major tasks, *allocation*, *scheduling* and *binding*. Thus, the use of the attributes in these three tasks will be discussed here.

5.1 Use of component usage in allocation

The task of allocation is to define the number and type of resources used in the design. The resources include functional units, storage units and interconnection units. The component usage can be used to select functional units (components) from design libraries to perform the operations specified in a behavioral description. Allocation algorithms can use as keys operation name, parameter data types, together with synthesis constraints to search design libraries. The design library search engine compares operation name, parameter data types and the given constraints with usages of each component, and returns the matched components. In this way, synthesis systems can automatically find appropriate component candidates which can guarantee to perform all the operations specified in the behavioral description. When multiple components are identified as candidates, other attributes, such as performance, size and power, can be used in cost functions to evaluate design alternatives and explore design spaces.

The design shown in the top of Figure 6 is described in VHDL hardware description language. The constraint on the minimum system clock period is $20ns$. Since the operations needed to be performed are two additions and one multiplication of signed numbers under the given system clock constraint, three search keys are formed as shown under the description. Suppose that the design library contains three components, a full adder $ADDER$, an array multiplier AM, and a shift-and-add multiplier SAM. On the right side of each component are the usage(s) of the components. The constraint on the clock in each usage is the minimum clock period. For instance, a clock with the minimum period of $8ns$ is required to use the SAM. It is obvious that the $ADDER$ is able to perform the specified addition operations, and both AM and SAM are able to perform

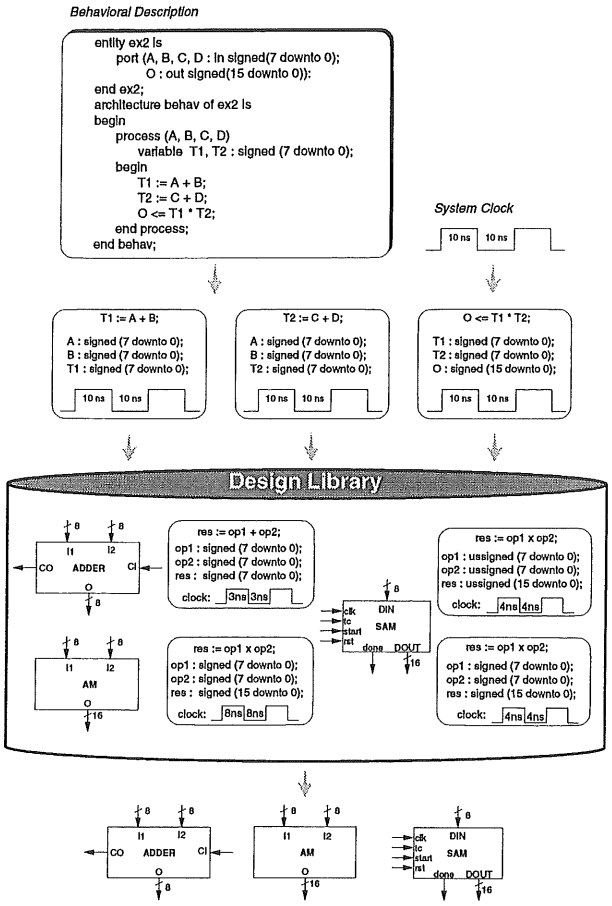

Figure 6: Use of component usage in allocation.

the specified multiplication operation in the description. Allocation algorithm could further select one *ADDER* and one *SAM* to implement the specified design.

5.2 Use of left mapping protocol and interface template in scheduling

The task of scheduling is to sequence executions of operations in a behavioral description by assigning each operation to control steps, where each control step corresponds a clock cycle in the synthesized design. Since execution of an operation on a component may take several clock cycles with complicated I/O protocols involved, it is important to ensure that I/O operations of the component be sequenced properly as well. I/O operations of a component include sending control signals and data to the component, and fetching status signals and results back from the component. Since the interface template describes the sequence of the I/O operations with cycle based accuracy, it can be embedded into the schedule to control the execution of the operation on the component. This concept is illustrated using the example shown in Figure 7.

Figure 7(a) shows the Data Flow Graph (DFG) of the design described in VHDL in Figure 6. Suppose that the *SAM* is allocated to perform the multiplication operation. From the interface template shown in Figure 5 we can see that, with the help of the left mapping protocol, the multiplication operation in the DFG can be substituted by the *SAM* and its interface template as shown in Figure 7(b) and (c). In Figure 7(b) we use the dashed box to represent the interface template since it will be implemented into the controller, instead of as an individual entity.

In Figure 7(c) we use the dashed box to represent the SAM since the SAM is not a real component, and needs to be mapped to a real component during binding. Therefore, every time we substitute an operation with an interface template and a component, we call the component as **virtual component** and it is subject to be bound to a real component. The left mapping protocol, shown as the dashed lines between the DFG and the interface template, represents the intercepted data flows. From the Figure we can see that data $T1$ and $T2$ flow into the left port L_1 and L_2 of the interface template, respectively. The interface template shows that $T1$ is fed into SAM's DIN in st_1 and $T2$ is fed into SAM's DIN in st_2. Thus, operation $T1 := A + B$ needs to be scheduled before st_1, and $T2 := C + D$ needs to be scheduled before st_2. Figure 7(d) shows the transformed design with the final schedule in the left box and the allocated SAM. Note that the multiplication operation $O <= T1 \times T2$ in the original description has been replaced by a set of assignment statements, which assign data and control signals to SAM and fetch status and data from it, and the allocated component. A scheduling algorithm, which can handle the interface templates during scheduling, is discussed in [5].

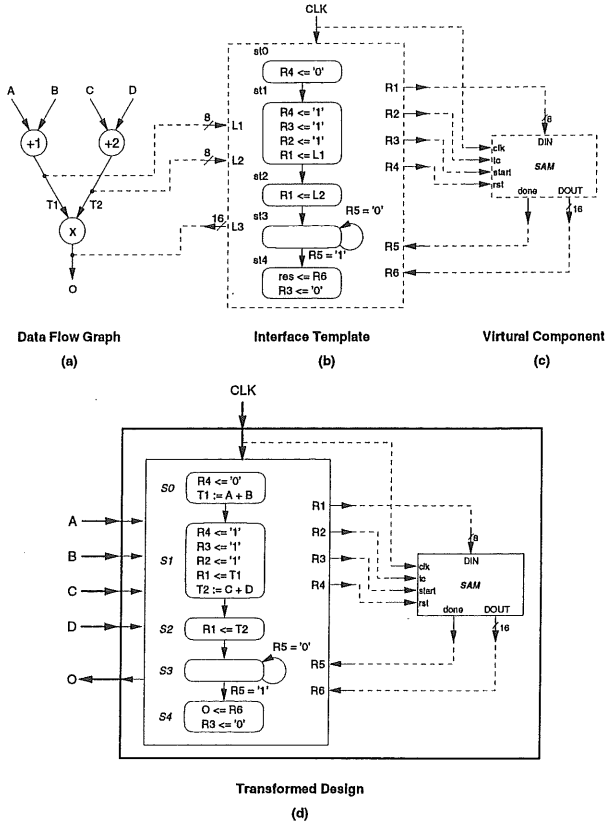

Figure 7: Use of interface template in scheduling.

5.3 Use of right mapping protocol in component binding

The right mapping protocol is used during virtual component bindings. As mentioned in the above, a virtual component is introduced during scheduling when an operation is replaced by a component and its interface template. Typically, the binding tasks include Functional Unit (FU) binding, storage binding and interconnection binding. FU binding maps

each operation in a behavioral description onto a FU. Storage binding maps data carriers (e.g., constants, variables and data structures like arrays) in the description to storage elements (e.g., ROMs, registers and memory units). Interconnection binding maps each data transfer (i.e., a read or write) to a interconnection path between its source and its sink. The only difference between traditional binding and binding for virtual components is in FU binding. The traditional FU binding maps operations onto functional units, while binding with virtual components maps virtual components onto real components.

Due to the existence of the right mapping protocol in our usage based component characterization, the problem of binding virtual components to real components can be solved using the same binding algorithms as traditional FU binding.

First, component sharing conditions in FU binding and virtual component binding are the same. In FU binding, one operation may be mapped onto a FU only if the FU is capable of executing the operation and is idle during the control steps in which the operation is scheduled to execute. The same is true with virtual component binding: a virtual component may be mapped onto a real component only if the real component exactly matches the virtual component and is idle during the control steps in which the virtual component is scheduled to be active.

Secondly, the optimization goal for FU binding and virtual component binding is the same, i.e., to minimize the interconnection cost. In FU binding, the interconnections are between storage units and FUs or between FUs. In virtual component binding, the right mapping protocol specifies the connections between an interface template and a virtual component. Since the sources/sinks of the connections in the interface templates will eventually be mapped to either storage units or other components, interconnections are still between storage units and components or between components.

Since component sharing condition and cost functions are the same, the same binding algorithms for FU binding can be used directly with virtual component binding.

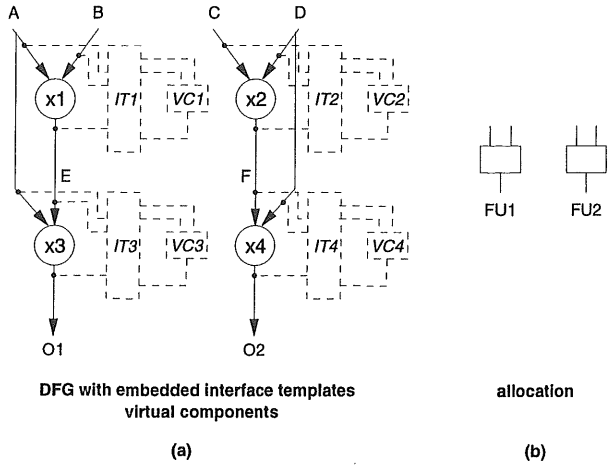

Figure 8: Use of right mapping protocol in binding.

Figure 8 shows a DFG consisting of four multiplication operations, $\times_1$, $\times_2$, $\times_3$ and $\times_4$, which are to be executed on the two functional units of the same type, FU_1 and FU_2. Assume that $\times_1$ and $\times_2$ can not be mapped to the same FU since their execution times are overlapped, and neither $\times_3$ and $\times_4$ for the same reason. Traditional FU binding algorithms may need to determine, for example, whether to map operations $\times_1$ and $\times_3$ or $\times_1$ and $\times_4$ onto the same functional unit. If virtual components are used to perform the operations, the operations will be replaced with the interface templates and the vir-

tual components, i.e., the operation $\times_1$ replaced with the interface template IT_1 and the virtual component VC_1, $\times_2$ with IT_2 and VC_2, and so on. Virtual component binding may need to determine, for example, whether to map virtual component VC_1 and VC_3 or VC_1 and VC_4 onto the same real component.

6 Experiments and Conclusions

Usage based component characterization has been applied to characterize components with various complexities, such as combinational RLT components, shift-and-add multipliers, AMD 2901 ALUs, DCT, and memory components. These components have been inserted into component database of Interactive Synthesis Environment (ISE) such that they can be reused during behavioral synthesis.

7 References

[1] P.Kission, H.Ding, A. Jerraya, "VHDL Based Design Methodology for Hierarchy and Component Re-Use," *Proc. EURO-VHDL*, 1995

[2] S. Sarkar, A. Basu, and A.K.Majumdar, "Representation and Synthesis of Interface of a Circuit for its Reuse," *Proc. Euro DAC*, 1994

[3] E. Girczyc and S. Carlson "Increasing design quality and engineering productivity through design reuse", *Proc. 30th DAC*, 1993.

[4] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design,* Kluwer Academic Publishers, 1992.

[5] H. Juan, *Design Methodology and Algorithms for Interactive Behavioral Synthesis,* Ph.D. Dissertation, University of California, Irvine, 1997.