# An Intelligent Component Database For Behavioral Synthesis

Gwo-Dong Chen *
Electrical Engineering Department
National Taiwan University
Taipei, Taiwan, R.O.C.

Daniel D. Gajski
Information and Computer Science Department
University of California, Irvine, CA 92727

## Abstract

This paper describes an intelligent component database system that delivers components for given set of attributes and constraints. Requirements of a component server are defined and an implementation is described. Our experiments demonstrate that such a component sever can replace component catalogs with hundreds of pages.

## 1 Introduction

Behavioral-synthesis tools generate a microarchitecture design from behavioral or register-transfer descriptions. The generated microarchitecture consists of register transfer components such as ALUs, multipliers, counters, decoders, register files and similar multipliers. These register-transfer components are usually composed of MSI components such as 4-bit ALUs, 4-bit counters, or 3-to-8 decoders. This introduces another level of synthesis between logic and behavioral synthesis. Unlike basic logic components, register-transfer components have many options. One of the parameters is the component size in number of bits. Other paramenters are related to functionality, electrical and geometrical properties of the component. For example, counters may have increment and decrement options, load, set, and reset functions. Also, each component may have different delays and drive different loads on each of its output pins. Each component in addition may have several different options of aspect ratios for layout as well as position of I/O ports on the boundary of the module.

Thus, a component database for behavioral synthesis should generate components that fit specific design requirements and provide information about a component's electrical and layout characteristics for possible architectural tradeoffs. Since behavioral synthesis is in its infancy, very little attention has been given to component generation. Some preliminary work has been reported in [RDVG88], [JKMP89], [ThDW88] and [Wayn86].

In this paper a component server, called Intelligent Component DataBase (ICDB), is described. ICDB can dynamically generate components for given set of constraints and attributes. Further, it provides delay, area and shape estimates that support design tradeoffs on the microarchitecture level. The ICDB is designed to be used with tools for behavioral, logic, and layout synthesis.

The rest of this paper is organized as follows. Section 2 provides an overview of the database. The language used to describe a component implementation and the user interface language are explained in Section 3. Section 4 describes the component management while tool management is described in section 5. Finally, several examples are given in Section 6 to demonstrate the capabilities of this component server.

## 2 System Overview

### 2.1 The role of ICDB in behavioral synthesis

The component database system, ICDB, is used as a component server for behavioral synthesis, microarchitecture optimization, partitioning, logic optimization, floor planning and layout generation and assembly (Fig. 1).

During transformation of a behavioral description into a microarchitecture level structure, ICDB provides the delay, minimum clock width, area, and minimum setup and hold time for each component. For example, during operator scheduling, a synthesis tool can use the component delay time to determine the proper clock width. A behavioral synthesis tool can also use this information to decide whether to chain two operations together in a single clock, or whether to place an operation in a multiple clock step. When doing resource allocation, ICDB supplies to the synthesis tool a list of components that perform the requested function. This way the synthesis tool can select appropriate components according to the delay requirements. In microarchitecture optimization, tools can replace selected components with other components that better meet additional considerations, such as area shape for floorplanning or transistor sizing for different loading. In the microarchitecture optimization phase, ICDB is also queried to determine if components can be merged and whether merging can produce a better design. For example, a register and a incrementer can be merged into a counter. To

*On leave from National Central University, Taiwan under suppoort of Ministry of Education, Republic of China

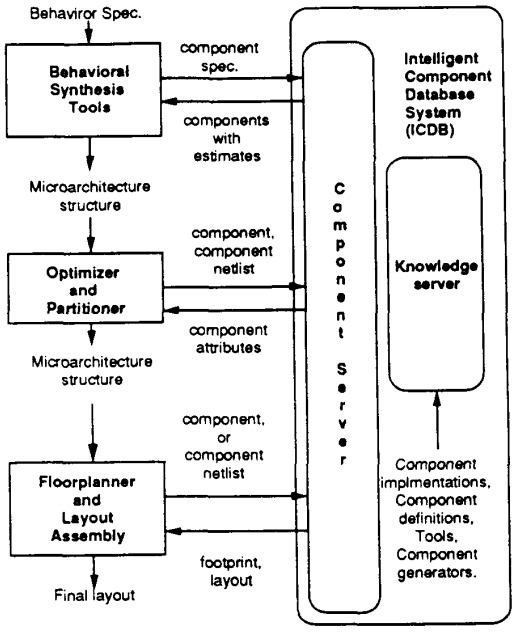


Figure 1: Functions of ICDB

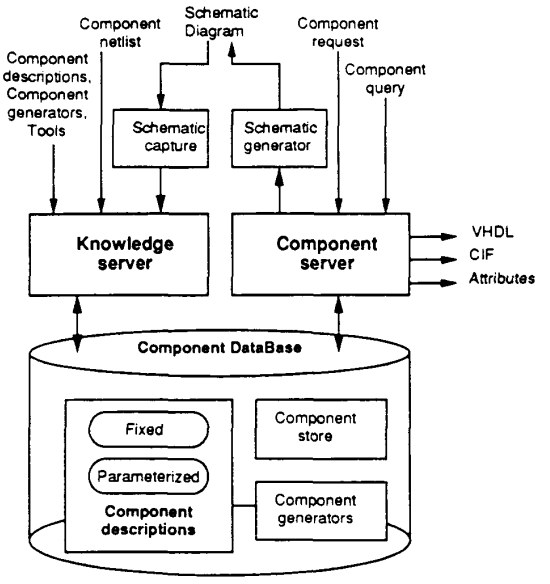


Figure 2: User's View of ICDB

achieve a good floor plan, the partitioner may try different ways of clustering components and retrieve their shape function from ICDB. It can also assign the pin positions of the combined object and ask ICDB to generate the layout according to this new plan.

## 2.2 User's view of ICDB

ICDB (Fig. 2) is composed of two subsystems: (1) a knowledge acquisition support system and (2) a component server. The component server provides two types of facilities. It (1) generates components from a given component specification and (2) answers queries about generated components. A generated component is represented in a VHDL netlist for logic level structure or CIF for layout. ICDB provides the area, delay, and other attributes of all generated components.

Users can insert component definitions, component generators, tools, and component implementations to ICDB through the knowledge acquisition support mechanism.

There are two types of component definitions: 1) fixed, and 2) parametrized. Component generators are scripts that call different tools to convert component definitions into component instances that satisfy user constraints.

During the synthesis process many components are generated. They are stored in a component store which is used by different users for passing components and their attributes.

# 3 ICDB User Interface

ICDB provides two user interface languages: (1) an intermediate format (IIF) to describe the low-level behavior of components and (2) a Component Query Language (CQL) to query what components are available and to query component characteristics such as area, delay, shape function, and port requirements.

## 3.1 Component implementation description

In order to describe microarchitecture level components (such as counters, shift registers, etc.), we need a format capable of describing sequential, asynchronous behavior, and I/O conversion. We use the Irvine Intermediate Format (IIF) [ChGa89] which extends the Berkeley EQN format by adding clocking, asynchronous and interface constructs. Besides providing the basic boolean operations of AND, OR, NOT, XOR, and XNOR, IIF contains operators for specifing clocked elements, asynchronous set and reset, tri-state, delay, schmitt trigger, and wire-or. In addition, IIF provides programming constructs for describing parameterized objects. The programming constructs include an IF statement and a loop statement in order to specify components with replicable structure. The function call is used for hierarchical descriptions. IIF has the same block structure and the similar syntax as the C programming language.

As an example of IIF description, an up-down counter (similar to the 74191 counter) is shown in Figure 3. The description has 3 parameters: size, type, and load. The

```
C[0] = 1;
for(i=0;i¡size;i++)
/* up counter only */
if (up_or_down == 1) C[i+1]= C[i] * Q[i];
/* down counter only */
elseif (up_or_down ==2) C[i+1] = C[i] * !Q[i];
/* updown counter */
else C[i+1] = C[i] * (Q[i] (+)DWUP);
/* asynchronous parallel load */
if ( load )
Q[i]=(Q[i](+)C[i])@(rCLK)
a(0/(!LOAD*!D[i]),1/(!LOAD*D[i]));
else
Q[i] = (Q[i](+)C[i]) @( rCLK);
```

Figure 3: Up-Down Counter Description

size determines the number of bits. The type determines whether counter counts only up (type = 1) or only down (type = 2) or both (type = 3). When the counter counts up and down, the direction is determined by control line DWUP. Another option can be activated by setting the parameter load = 1. In this case, the counter will be able to asynchronously load data on inputs D[i], 1¡i¡size, when input line LOAD is low. Thus, the description in Figure 3 is a generator for all possible options of size, load and type parameters.

## 3.2 Component Query Language

The Component Query Language (CQL) is the language for tool interface to ICDB [ChGa89]. CQL consists of four kinds of commands: (1) component queries, (2) component requests, (3) component instance queries, and (4) component list management.

### 3.2.1 Component query

The component queries are used to obtain names of components with required functionality or the list of functions available in a particular component. These queries are used in high-level synthesis during module selections, scheduling, and operation binding. During module selection we need the list of components that can perform operations specified in the behavioral description while during binding we need to know all the operations performed by a component. For example, the following query of type component-query leaves a list of 5-bit up counters in the variable counters.

```
ICDB("command: component_query;
component:counter;
function:(INC);
attribute:(size:5);
ICDBcomponents:?s[]",&counters);
```

### 3.2.2 Component request

Once a component type and functionality is determined, the user or a synthesis tool may request a generation of a particular instance of such a component. In addition to type and functionality, the user will supply electrical and geometric constraints. Electrical constraints include clock period delay from any input to any output, set-up time, hold time, and the load on any output. Geometric constraints include aspect ratio, position of ports on the boundary of the bounding box, and the layer of each port.

Alternatively, a user can use words *max* or *min* instead of integer values for any constraints. If *min* is used for I/O delays the database will generate components with the smallest delay.

The following example shows a query for a five bit up-counter with clock width of 30 ns, set-up time of 30ns, and the list of I/O delays given in the variable c_delay. The name of the component will be in the variable counter_ins.

```
ICDB("command:request_component;
component_name:counter;
attribute:(size:5);
function:(INC);
clock_width:30;
comb_delay:set_up_time:30;
generated_component:?s",
c_delay,&counter_ins)
```

### 3.2.3 Component instance query

The component instance query is used to get information about generated components. ICDB provides information for delay, area, shape function, port equivalency for commutative operators, function control and connection information. This information is useful for operator scheduling, resource allocation and binding. For example, the scheduler needs the delay time of components to determine the clock width. A microarchitecture technology mapper needs the connection information in order to substitute components. For instance, the technology mapper needs to know which input controls the function of an up-down counter. It also needs the control code for the control port that will invoke the up-count operation. The microarchitecture optimizer also needs all equivalent ports, inverted ports, and delay information to optimize the design. For example, if the output of a component is connected to a component which requires an active low input, the optimizer can use the inverted output if available to avoid inverter insertion.

The user can use the following component instance query to retrieve information about the component *counter_ins*. This query retrieves the delay and shape function:

```
ICDB(" command:instance_query;
generated_component:%s;
delay:?s;
shape_function:?s",
counter_ins,&delay_s,&shape_function_s);
```

The following delay estimation is stored in the string variable delay_s for the up-down counter with parallel load described previously.

```
CW 29.0
WD Q[4] 8.5
WD Q[3] 8.5
WD Q[2] 8.5
WD Q[1] 9.7
WD Q[0] 8.7
SD DWUP 26.7
```

The CW is the minimum clock width. WD is the delay from rising clock edge to indicated output while SD defines minimum setup time for the input port DWUP.

Similarly, the following shape function is generated by the ICDB area estimator and stored in the **shape_function_s** variable.

```
Alternative=1 width=12000 height=48000
Alternative=2 width=90350 height=61100
Alternative=3 width=72700 height=73500
etc.
```

The layout can be generated by the following query.

```
ICDB(" command:request_component;
instance:%s;
alternative:3;
port_position:%s;
CIF_layout:?s",
counter_ins,pin_locs,&counter_layout);
```

An example of port position assignment is shown as follows:

```
CLK left 10
D[0] top 10
D[1] top 20
D[2] top 30
D[3] top 40
D[4] top 50
LOAD left 20
DWUP left 30
Q[0] bottom 10
Q[1] bottom 20
Q[2] bottom 30
Q[3] bottom 40
Q[4] bottom 50
```

where left, right, top and bottom indicate layout sides while integer values indicate the relative positions of input and output pins.

Using the previous three queries (component query, component request, and component instance query) we can obtain a set of components that can be used as up-counters. Also, an area/time and height/width tradeoff graph for the up-counter can be generated by the tools. Those graphs are shown in Figure 4 and Figure 5.

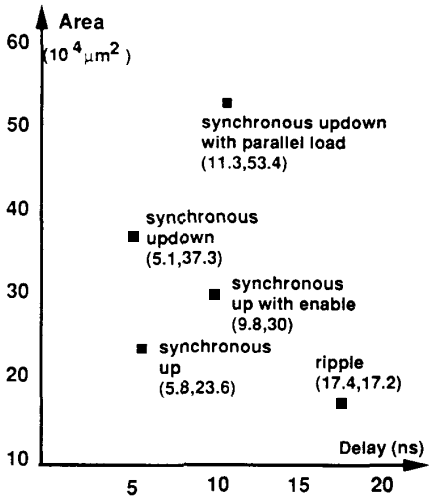


Figure 4: Area/time Tradeoff for the Up-counter

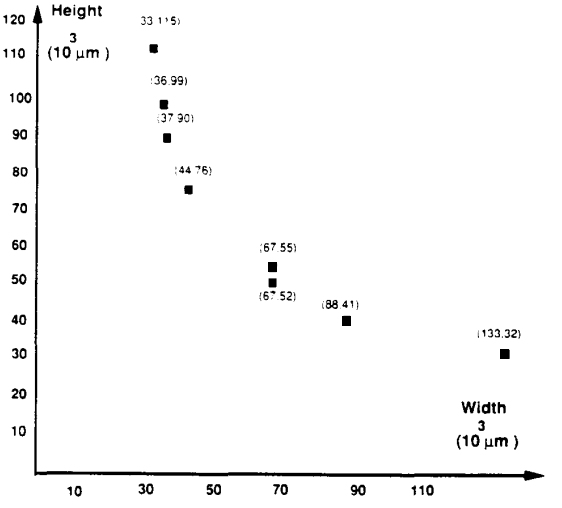


Figure 5: Shape Tradeoff for the Up-down Counter

# 4 Component Management

ICDB contains two types of component descriptions: (1) fixed and (2) parameterized descriptions. Component generators generate the logic or layout view for any set of parameter values.

An ICDB component description includes two types of descriptions: (1) design data such as IIF VHDL and CIF descriptions, and (2) ICDB data. The ICDB data includes (1) the component type, (2) the function that the component performs, (3) connection information, (4) I/O port descriptions, (5) parameter descriptions, (6) attributes, and (7) file names of the component design data. The component design data is stored in UNIX files, while the ICDB data is stored in the INGRES database. Tools communicate directly with the UNIX file system. They retrieve the UNIX file name from ICDB, then perform their own I/O. Thus, ICDB does not degrade the performance of tools. When a component is inserted into ICDB, the ICDB data is inserted into the INGRES database.

ICDB components are classified and retrieved by either a component type or the functions they perform. A component type is the name of a microarchitecture component such as counter, register, and adder. A function is an operation that a microarchitecture component may perform such as ADD, and SUBTRACT.

If a component performs multiple functions, the connection information will be retrieved. This information tells the synthesis tool how to connect a component and how to invoke its functions.

# 5 Tool Management

ICDB contains also a set of synthesis and optimization programs called tools. A subset of tools needed to generate a component view is called a component generator. For any requested set of constraints the ICDB will retrieve the proper component description from the description library and the proper generator to produce the desired implementation.

When a component generator is invoked, a series of shellscripts are executed. All the tools retrieve data from ICDB and place the resulting data back into ICDB.

The ICDB presently includes (1) an IIF expander, (2) a logic optimizer, (3) a technology mapper, (4) a transistor sizer, (5) a layout generator, (6) a timing estimator, and (7) an area estimator.

The logic synthesis and technology mapping [VaGa88] takes expanded IIF and delay constraints and generates a VHDL netlist. First, the sequential constructs are removed, creating a set of boolean equations. These equations are minimized and then factored by a logic optimizer. In the second phase, the number of levels along the longest paths are reduced by performing factoring. Technology mapping is performed in the third phase by combining gates into complex gates. Sequential logic is then reinserted. The fourth phase sizes the transistors according to the input delay constraints. A VHDL netlist with assigned transistor sizes is generated in the final phase.

The layout synthesizer [LiGa87] takes a sized netlist and generates a two-dimensional layout in which transistors are placed into a number of strips. Each strip has a pair of Vdd/Vss lines on its boundaries. Two neighboring strips share a common power line. The generated layout is stored in a CIF file.

In order to allow microarchitectural tradeoffs, ICDB uses two estimators for the delay and layout area.

ICDB stores three types of delay information for each basic component: (1) the delay increase for each additional unit of transistor load, (2) the delay from an input port to an output port, and (3) the delay increase for each additional fanout. The delay of a component is the sum of all the estimated delays of components along the path.

The area estimation is based on known transistor sizes and the number of tracks used for routing between two diffusion lines. The width of the layout is obtained by averaging transistor widths over the number of strips. Estimation of the layout height is based on (1) transistor height, and (2) the number of tracks. The height of transistors is estimated by the average height of all the transistors while the number of tracks needed is estimated by analyzing the netlist and estimating the horizontal length of each net. The estimated number of tracks used is obtained by total horizontal wire length divided by an experimentally-obtained track-utilization constant.

# 6 Examples and Results

ICDB is written in the C language and runs on Sun workstations under the UNIX operating system. We have run a number of examples to demonstrate some of the capabilities of the ICDB. The examples are based on the IIF description for the counter described in section 3.

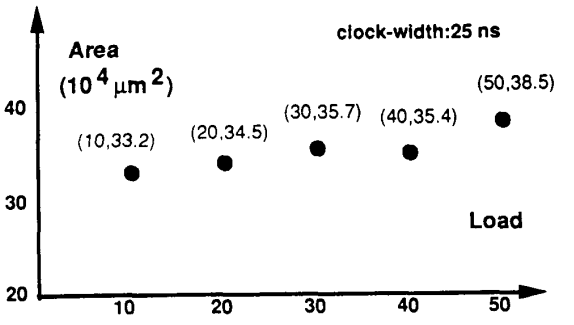


Figure 7: Area/Clock-width Tradeoff for the Up-down Counter

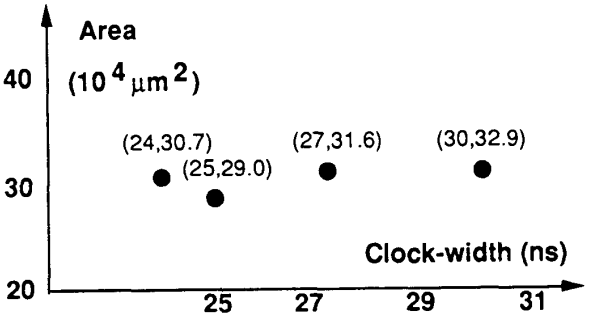


Figure 8: Counter layouts for Figure 5

Figure 4 shows the time/area tradeoff graph for different implementations of the 5 bit up-counter. The counters were generated by providing different parameter values. Layouts of these components are shown in Figure 6.
In further refinements of a design, the component output loads and delays constraints may be changed as well.

Figure 7 shows the area/output load tradeoff curve for the synchronous updown counter in Figure 5.
The required minimum clock width of this counter was set at 25 nanoseconds. Then different output load requirements were requested ranging from 10 to 50. ICDB sized the transistors of the counter to make it achieve the same minimum clock width. The results show that the area increased only 6 percent when the output load constraint was changed from 10 to 40.

The area/clock-width tradeoff curve of the counter is shown in Figure 8. In this figure, the output loads of the counter were held constant at 10 units. The minimum clock width was varied from 24 nanoseconds to 30 nanoseconds.

Figure 8 shows that the area range is within 6 percent. Tightening the delay constraints does not increase the area in all cases. ICDB will generate many components from the same netlist structure with different transistor sizes. With different transistor sizes, the layout system will perform a different placement and routing. Thus, it may produce a larger layout even though the total transistor size is smaller.

Different aspect ratios of a component are provided by

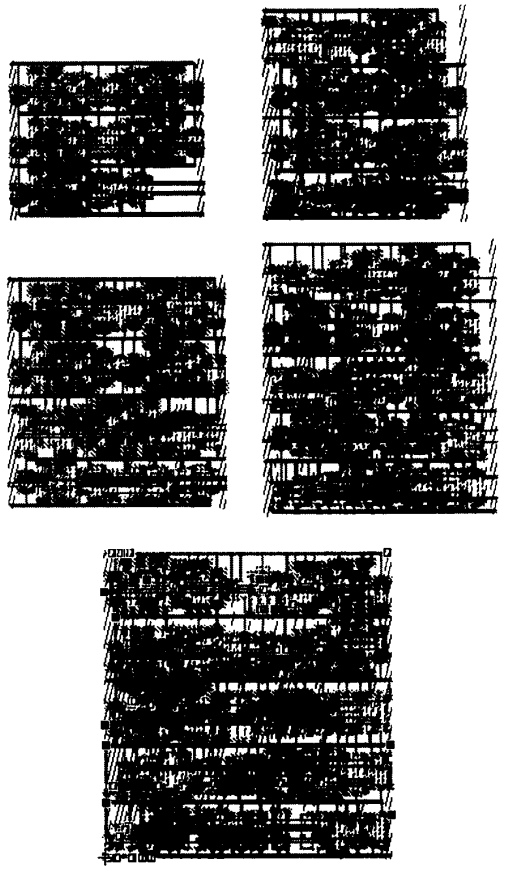

Figure 6: Area/load Function for Up-down Counter

laying it out with a different number of strips. Figure 5 shows the shape function for the updown counter.
The port positions can be assigned when a user requests generation of the layout.

# 7 Conclusion

We introduced in this paper the concept of a tool-transparent component server which allows high-level synthesis tools to ignore tradeoffs below the component level. We demonstrated a language for storing parameterizable components as well as a new query language for use by high-level synthesis tools. Additional contributions are a new management scheme for components, a set of generators and tools, the definition of exploratory strategies for high-level synthesis, and the development of estimators for the logic and layout levels. However, the main contribution of this work is the demonstration that component libraries and component catalogs with hundreds and hundreds of pages are no longer necessary.

# Acknowledgements

We would like to thank Allen Wu for helping with layout generation. The unparameterized IIF was defined originally by Nels Vander Zanden. We also want to thank Nels for his helpful comments during the writing of this paper.

Gwo-Dong Chen is thankful for the financial support provided by the government of the Republic of China.

# References

[ChGa89] G-D. Chen, D. Gajski, "An Intelligent Component Database for Behavioral Synthesis", Tech. Report No. 89-39, 1989, ICS Dept., Univ.of California, Irvine.

[LiGa87] Y-L Lin, D. Gajski, "LES: A Layout Expert System", IEEE Trans. on CAD, Vol. CAD-7, No.8, pp. 868-876, August 1988.

[RDVG88] J. Rabaey, H. De Man, J. Vanboof, G. Goosens, and R.H.J.M. Otten, "CATHEDRAL II: A Synthesis System for Multiprocessor DSP Systems" in Daniel D. Gajski, ed., Silicon Compilation, pp. 311-360, Addison-Wesley, 1988.

[JKMP89] R.Jain, K. Kuckukcakar, M.J. Milnar, and A.C. Parker, "Experience with the ADAM Synthesis System", Proc. of 26$^{th}$ Design Automation Conference, pp. 56-61.

[VaGa88] N. Vander Zanden, A. Wu, D. Gajski, "Performance Optimization in Layout Driven Synthesis", Tech. Report No. 89-21, ICS Dept., Univ. of California, Irvine.

[ThDW88] D.E. Thomas, E.M. Dirkes, R.A. Walker, J.V. Rajan, J.A. Nester, R.L. Blackburn, "The System Architect's Workbench", Proc. of 25$^{th}$ Design Automation Conference, pp. 337-343.

[Wayn86] W. Wolf, "An Object Oriented Procedural Database for VLSI Chip Planning", Proc. of 23$^{rd}$ Design Automation Conference.