A HARDWARE LIBRARY REPRESENTATION FOR THE HYPER SYNTHESIS SYSTEM

by

Scarlett Zhijia Wu

Memorandum No. UCB/ERL M94/47

10 June 1994

A HARDWARE LIBRARY REPRESENTATION FOR THE HYPER SYNTHESIS SYSTEM

by

Scarlett Zhijia Wu

Memorandum No. UCB/ERL M94/47

10 June 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A HARDWARE LIBRARY REPRESENTATION FOR THE HYPER SYNTHESIS SYSTEM

by

Scarlett Zhijia Wu

Memorandum No. UCB/ERL M94/47

10 June 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Hardware Library Representation for the HYPER Synthesis System

by

Scarlett Z. Wu

ABSTRACT

The goal of the HYPER sysnthesis system is to build real-time systems optimized for any given set of design specifications. The effectiveness of the high-level optimization is often limited by the inaccurate estimation of the hardware being built. This project focuses its attention on a generic library representation used by the HYPER system and discusses the various considerations that are taken into account in selecting the set of hardware properties in the library. The design of a low power wavelet filter is used to illustrate the complexity of today's synthesis problem and through the design flow, it is shown how the different hardware properties are used in high-level architectural transformation to achieve optimization goal.

ACKNOWLEDGEMENTS

First of all, I would like to thank my research advisor, Professor Jan Rabaey, who has given me a great deal of inspiration and constant advice. It was a wonderful experience for me to work with the HYPER group. I sure will miss everyone.

I would also like to thank Anantha Chandrakasan, who has developed many of the low power design techniques discussed in this report. Some of his ideas have become the basis of this research project.

Last but not least, I would like to thank my parents, who have supported me all my life and encouraged me to achieve the best I can.

1

Introduction

1.1 Overview

The ultimate goal of any synthesis system is to find an optimized hardware solution for a given set of parameters. During the process of transforming a high-level design specifications to the final layout, information on the available hardware resources are often the basis of making many optimization decisions. As a result, the hardware database can influence the quality of an implementation in many ways.

In this report, a set of hardware properties are defined for the HYPER hardware libraries. These properties can be shared among a variety of hardware units in describing hardware performances, cost and behaviors. It has been shown through many benchmark designs that this hardware library gives the designer a great deal of freedom to make changes to existing hardware information, to add additional hardware unit to existing library, and to create new libraries. Lastly, a technology library is also created for each hardware library in which global properties that are applicable to all units are defined.

1.2 Organization

This report is composed of six chapters. Chapter 2 gives background knowledge on HYPER, the synthesis system where the hardware database is currently installed. It is also analyzed here how the hardware database interacts with several design stages within the HYPER system. Chapter 3 introduces the hardware library. And the technology library is defined in Chapter 4. In Chapter 5, the design and implementation of a wavelet filter is documented. Some of the low power design methodologies are discussed here as well. Finally, Chapter 6 summarizes the work done and proposes suggestions on future work. Examples of hardware database in text file format are provided in Appendix A. And Appendix B supplies a list of available library routines.

2

Background

HYPER is a synthesis environment for real-time DSP systems. A typical design flow starts with the intended system described in the data-flow language SILAGE. This description is then translated to an intermediate control data flowgraph (CDFG). During each synthesis step, individual transformations are performed on the flowgraph and certain hardware assignment information is annotated. At the final stage of design flow, the transformed flowgraph is mapped onto a specific hardware structure. In such a system, the hardware database is accessed frequently and it carries out the following four tasks: 1. to provide accurate cost and performance data on any hardware unit; 2. to select the best suited hardware unit with given constraints; 3. to describe the capabilities and limitations of a hardware unit; 4. to show the layout topology and connectivity of the hardware unit.

2.1 Flowgraph Representation vs. Cell Library

In the HYPER system, the internal CDFG representation is composed of nodes, data edges, and control edges. The nodes represent data operations, while edges represent data precedences between the nodes[Rab91]. When the CDFG is first

generated, certain generic parameters are automatically annotated onto nodes and edges. In most datapath cells, the bitwidth of an operation is a common parameter. It is also found that for some operations, extra generic parameters are needed to sufficiently describe their functionality and behavior. For example, a shift operation is not complete without specifying the shift range. Or, in the case of a multiply operation, three values of bitwidth are required, two for the input nodes and one for the output node. Naturally, this type of generic parameters depends upon the specific operation and is independent of the hardware unit used for implementation. Whenever the hardware database is accessed, these parameters become constraints for hardware selection. In other words, these parameters define the common link between the information at flowgraph level and that at cell library level. Table 2.1 lists the hardware operations available in the HYPER system along with their generic parameter(s).

Table 2.1 List of Operations and Corresponding Generic Parameters

Operation	Generic Parameter(s)	Description
adder, subtracter, counter, comparator, register, inverter, buffer	N	bitwidth
shifter	N, M	bitwidth and shift range
multiplexer, logic cells	N, NrIN	bitwidth & number of inputs
multiplier	N, N1, N2	bitwidth of output & two inputs
register file	N, R, NCON	bitwidth, total number of registers & number of constant registers
ram, rom, fifo	N, R	bitwidth & number of memory locations

The definition of the generic parameters above enables the separation of high-level flowgraph representation from the details of hardware cell description. To introduce a new hardware cell to a library, one only needs to find the operation family

the cell belongs to and its hardware properties can then be easily represented in terms of the generic parameters defined for the operation family.

2.2 Interaction with other HYPER Tools

As stated earlier, the hardware database is accessed from various stages in the HYPER system. Since each of the stages acquires different information from the hardware database, it is worthwhile to focus on how the database interacts with other HYPER tools. Namely, these tools include *Module Selection/Estimation* stage, *Design Exploration* stage and *Hardware Mapper* stage.

2. 2. 1 Module Selection/Estimation

The goal of *Module Selection* is to find the best choice of suited execution units to implement a flowgraph, given a set of active area, speed or power constraints. A designer is free to choose clock period, supply voltage and cell library. It is also up to the designer to specify selection constraints. For example, instead of choosing the fastest or the smallest unit, the designer can ask for multi-functional units in order to achieve a higher degree of time sharing. It is clear that the hardware database has to be able to differentiate between operation families while being capable of access to individual hardware units in each family.

When a node contains a constraint that exceeds the capability of any available hardware cell in the library, the node has to be replaced with a cluster of nodes which are implementable. For instance, if a hardware library only has shifter cells with shift range of 7, a node with shift range of 14 should then be transformed to two cascaded shift nodes, each with shift range of 7. Of course, doing such transformation in *Module Selection* requires the knowledge of hardware capability. It is, therefore, reasonable to place constraints on each individual hardware unit to describe its limitations.

Based on the result of *Module Selection*, the *Estimation* stage estimates the cost and performance of the implementation. At this point, values of the generic parameters have been annotated to the flowgraph. Using these parameter values, it is sufficient to generate the specific estimation data. Since only rough estimation is required at this stage, it is unnecessary and sometimes impossible to generate precise data without the knowledge of allocation and scheduling. For example, to estimate the active switching capacitance of a hardware unit, it is more appropriate to use the simulation result of a white noise input. However, once the scheduling of the system is determined, more accurate power analysis can be made with simulation results of expected input patterns. In many cases, default values should be used at early stage of the design flow and this information useful to be included for individual hardware unit.

2. 2. 2 Design Exploration

The *Design Exploration* stage embodies the capabilities of both *Module Selection* and *Estimation* by allowing the designer to explore a three-dimensional design space defined by clock period, sample period and supply voltage. The improvement or degradation of different design qualities, such as area, can be shown by varying these parameters. Since different time criteria imposes different timing constraints on *Module Selection* and as a result, *Estimation* produces different cost and performance. The reason for the variations can be explained as follows. It is found that for datapath cells, there is a fixed correlation between supply voltage and its propagation delay. Figure 2.1 shows how delay varies in function of supply voltage for a given technology[Cha92]. It is clear that as supply voltage decreases, the delay increases and this effect can become quite large at very low voltage supply. As a result, the critical path increases and at a given clock period and sample period, more hardware are required to implement the same design. So the implementation could become more costly. Therefore, in order to estimates accurate delay time under

different voltage supply, the correlation should be stored in the hardware database for different technology.

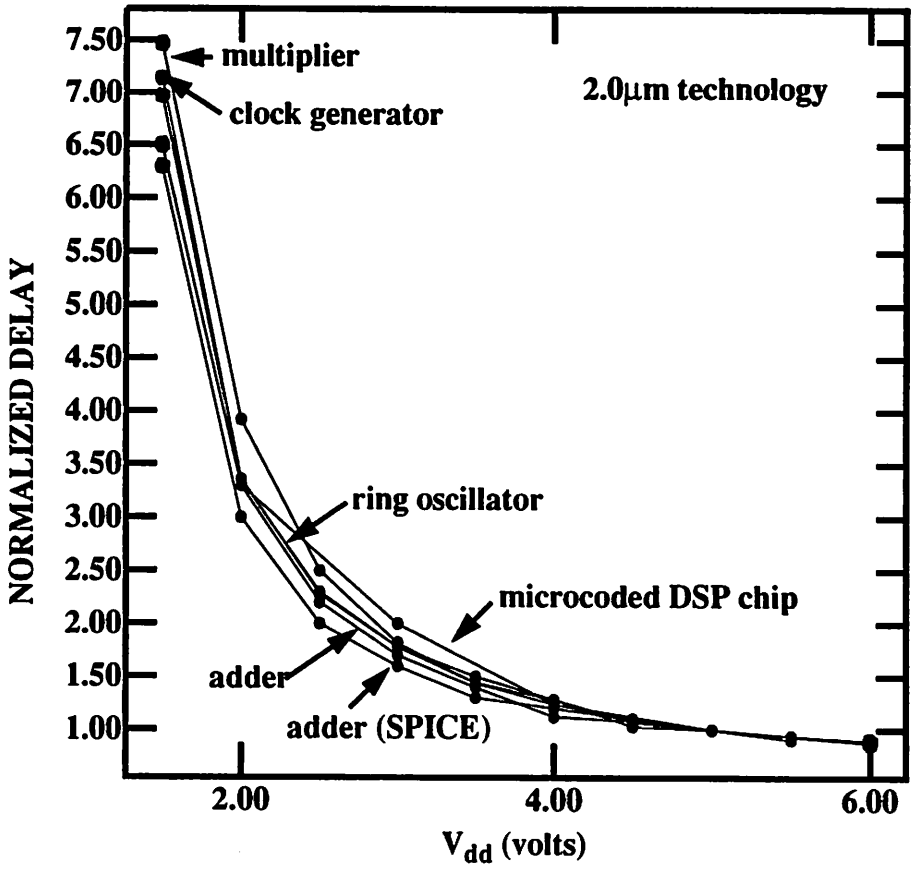

Figure 2.1 Correlation Between Time Delay and Supply Voltage

Since each hardware unit is characterized under a given process, it is also necessary to define scaling factors to reflect the variation of process and supply voltage.

2. 2. 3 Hardware Mapper

During the *Hardware Mapper* stage, the high-level flowgraph is mapped to a target hardware structure before it is handed over to the silicon compiler to generate layout. The target hardware structure can be described in the form of either VHDL or SDL language[Ben94]. In general, the three steps involved in the *Hardware Mapper* are -- adding peripheral hardware units, i.e. registers, multiplexers and tri-state buffers; generating appropriate control blocks; and creating layout floorplan for post processor.

It is extremely important during this stage for hardware database to provide information regarding data terminals, control terminals and hardware behaviors for different configurations. To set up the control table, the *Hardware Mapper* has to find out what input control terminals exist for each unit and how they are asserted as a function of time. It can become quite difficult in some cases to describe these behaviors in a hardware library. To begin with, the names of these control signals are not known at high level. Different operation has different control signals. And for some configurable units, control signals can even change the functionality of the unit. To assist the interactions between *Hardware Mapper* and the hardware library, a set of dynamic parameters are introduced for each operation family. These dynamic parameters are very different from the generic parameters defined in Section 2.1 because dynamic parameters are used by *Hardware Mapper* to set up the control table and they are never annotated to the high-level flowgraph. For example, the shifter operations have a dynamic parameter called "SHIFT" which denotes the dynamic shift value. For a given value of "SHIFT", the hardware library provides the control signal setups of the shifter. In the following chapter, it will be shown how this can be achieved for some complex control configurations. Table 2.2 lists the dynamic parameters valid in *Hardware Mapper.*

Table 2.2 List of Operations and Corresponding Dynamic Parameters

Dynamic Parameter	Operations	Description
Vdd	all	chip supply
GND	all	chip ground
CK1	all	phase 1 of global clock
CK2	all	phase 2 of global clock
WRITE	register, counter, memory	write control
READ	register, memory	read control
COUNT	counter	count enable

Table 2.2 List of Operations and Corresponding Dynamic Parameters

Dynamic Parameter	Operations	Description
RESET	counter	reset
OEN	register, counter, memory, tri-state buffer	output enable
SELECT	multiplexer	select configuration
SHIFT	shifter	shift configuration

In the last phase of *Hardware Mapper* -- layout floorplanning, it is important to find out the geometry of the unit as well as the location of input/output terminals. With the right place and route tool, this could effectively help to increase the compactness of a layout.

3

Hardware Library

In a high-level synthesis tool such as HYPER, the hardware library is the only hardware resource.

This chapter presents the complete hardware library in details. Section 3.1 describes the overall database structure. Section 3.2 introduces the set of list functions used for library expressions. Section 3.3 defines each hardware property in the hardware library and Section 3.4 makes some suggestions on future improvements.

3.1 Database Structure

The hardware library is written in text file format because this is the easiest form for any designer to make modification or to create new entries.

Figure 3.2 shows the overall structure of the hardware library. All primitive operators are placed at top level hierarchy. The second level hierarchy, placed under each operator family, consists of hardware cells that can implement that operator. The hardware descriptions are then listed under each hardware unit[Chu92]. In this data

structure, the only way to identify a hardware unit is by specifying an operator name and a cell name.

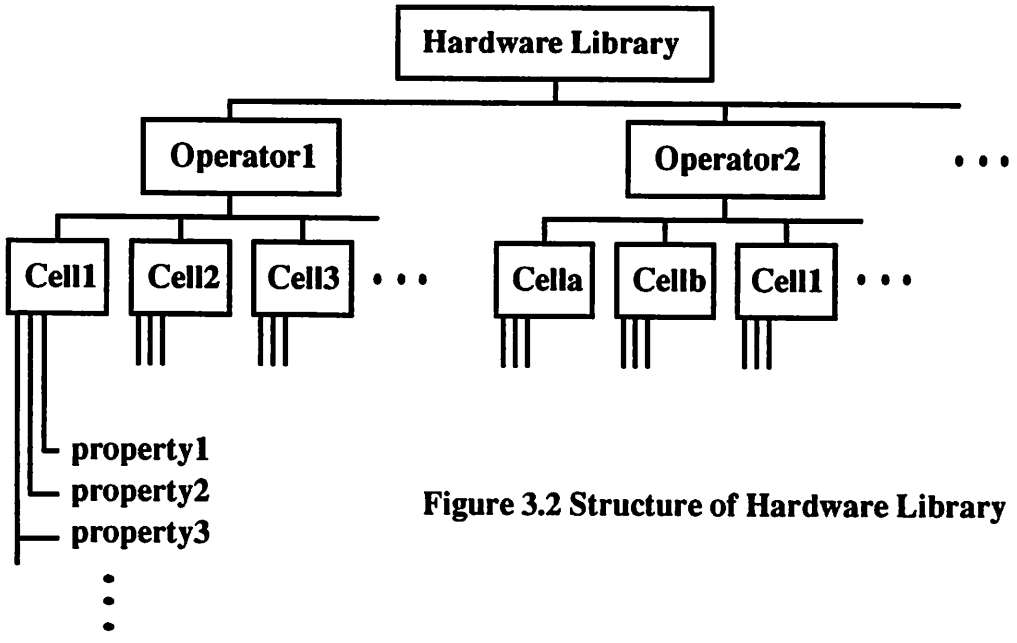

Figure 3.2 Structure of Hardware Library

To represent the above structure, the *Common Lisp* associate list format has been adopted in the library. Thus, in a text format, Figure 3.1 can be written as:

```
(
    (Operator1
        (Cell1
            (property1 ...)
            (property2 ...)
            (property3 ...)
            ...
        )
        (Cell2
            (property1 ...)
            ...
        )
        ...
    )
    (Operator2
        (Cella
            ...
        )
        ...
    )
    ...
)
```

3.2 List Functions

Each hardware property included in the hardware library is always associated with a specific data type. The numerical data type is widely feasible for information such as delay time, area, power, etc. Another commonly used data type is a list of character strings. For example, the input signals of a 3-input NAND gate are "A", "B" and "C". It has been observed that in general, a hardware property can always be expressed as either a numerical value or a list of objects.

It turned out that in most cases, the standard *Lisp* functions are often sufficient enough to interpret common numerical expression. The general form of a *Lisp* function looks as follows:

(operator operand1 operand2 ...)

Here, each *operand* following the *operator* can also simply be replaced with another *Lisp* function. However, the standard *Lisp* functions doesn't provide a convenient way for manipulating complex list structures, which is seen quite often in a hardware library.

The expanded *list functions* defined for the HYPER hardware library are strongly based on the standard *Lisp* functions and it has the following format:

(keyword list_structure)

For each *keyword*, there are two ways to express a *list structure* -- primitive or advanced. A primitive *list structure* is one that consists of a real value, an integer value or a string. In the case of a string, it can either be a character string or one enclosed by double quotes. Advanced *list structure*, on the other hand, can be a hierarchical list of primitive *list structures*. Since the result of any *list function* is a *list structure*, *list functions* can usually be hierarchically used by themselves. In other words, all *list*

functions are also *list structures*. And if the first item of a *list structure* matches to a valid *keyword*, it can be called a *list function*.

To get a better understanding, Table 3.3 lists all the valid *list functions*. In the second column, *int* denotes an integer value, *real* denotes a real value, *str* denotes a string and *strux* denotes a *list structure*.

Table 3.3 List of list functions in Hardware Library

Keywords	Operand(s) w/ Corresponding Data Type	Description	Result Data Type
+	$real_1$ $real_2$ ...	$real_1 + real_2 + ...$	real
-	$real_1$ $real_2$ ...	$real_1 - real_2 - ...$	real
*	$real_1$ $real_2$ ...	$real_1 * real_2 * ...$	real
/	$real_1$ $real_2$ ...	$(real_1 / real_2) / ...$	real
max	$real_1$ $real_2$ ...	maximum value	real
min	$real_1$ $real_2$ ...	minimum value	real
sqrt	*real*	square root of *real*	real
exp	*real*	e^{real}	real
expt	$real_1$ $real_2$ ...	$(real_1)^{real_2}$	real
ceiling	*real*	$\lceil real \rceil$	integer
==	int_1 int_2 ...	$(int_1 == int_2)\ ?\ 1 : 0$	integer
!=	int_1 int_2 ...	$(int_1 != int_2)\ ?\ 1 : 0$	integer
>=	int_1 int_2 ...	$(int_1 >= int_2)\ ?\ 1 : 0$	integer
<=	int_1 int_2 ...	$(int_1 <= int_2)\ ?\ 1 : 0$	integer
>	int_1 int_2 ...	$(int_1 > int_2)\ ?\ 1 : 0$	integer
<	int_1 int_2 ...	$(int_1 < int_2)\ ?\ 1 : 0$	integer
!	*int*	$(int == 0)\ ?\ 1 : 0$	integer
&&	int_1 int_2 ...	$(int_1 \&\& int_2 \&\& ...)\ ?\ 1 : 0$	integer
||	int_1 int_2 ...	$(int_1 \| \| int_2 \| \| ...)\ ?\ 1 : 0$	integer
strcmp	str_1 str_2	strcmp(str_1, str_2) ? 1 : 0	integer

Table 3.3 List of list functions in Hardware Library

Keywords	Operand(s) w/ Corresponding Data Type	Description	Result Data Type
!strcmp	str_1 str_2	!strcmp(str_1, str_2) ? 1 : 0	integer
strcasecmp	str_1 str_2	strcasecmp($str1$, str_2) ? 1 : 0	integer
!strcasecmp	str_1 str_2	!strcasecmp(str_1, str_2) ? 1 : 0	integer
lookup	str_0 ((str_1, $strux_1$) (str_2, $strux_2$) ...)	if !strcmp(str_0, "$strux_i$") return $strux_i$	list structure
strcat	str_1 str_2 ...	concatenation of strings and/or integer(integer allowed)	string
d2b	*strux*	convert any decimal number to its binary form	list structure
expand	*str* int_1 int_2 int_3	return a list of strings each transformed from *str* by replacing "%d" by integer values from int_1 to int_2 with step int_3	list structure
include	*/path/filename*	include lisp structure in "*/path/filename*" (physical path only)	list structure

Some examples of using the above *list functions* can be found in Section 3.3 as well as the complete hardware library file in the Appendix A. When referring to the hardware library, one thing to notice is that the parser in the HYPER system does not support the use of real numbers in a text file. However, for those *list functions* that require real operands, it is important to remember that the C routines developed for them do use real numbers in every calculation. For example, *list structure* (* N (/ 5 4)) is equivalent to calculating the expression N * 1.25.

3.3 Hardware Properties

The hardware information of a unit can be divided into three groups. The first group includes its performance and cost. The second group defines its structural model. While the last group describes its behavior. In the following three sections, hardware properties in each of these groups are defined.

3.3.1 Cost/Performance

The cost and performance data of a hardware unit is generated by characterizing the unit. It is often the case that these properties are expressed as a function of its generic parameters. In most examples of this section, the generic parameter "N" is used to specify the bitwidth. While reading these examples, it is helpful to refer to Chapter 2 and the previous sections for definitions of generic parameters and *list functions*.

Here is a property that are often used to estimate overall datapath area.

AREA

This property specifies the cell area in λ^2. Returns an integer.

```
Example:    (AREA (* 207 (+ 20 (* 64 N))))
```

It has been found through recent research that power consumption of a hardware unit is largely influenced by the choice of its input patterns. To accurately reflect such variations, the HYPER system has developed its own power estimation model[Lan94]. Characterized information for this model is stored in the following property.

CAP-COEFFS

This property specifies the coefficients for calculation of effective capacitance with different input patterns. Returns a list of integers.

```
Example:    (CAP-COEFFS (LOOKUP KEY
                        ((UU/XX (20))
                         (US/XX (20 18))
                         (SU/XX (21 21))
                         (SS/XX (0 39 41 0)))))
```

The time delay information of a hardware unit is quite important to the high-level transformation because it directly affects the critical path of an implementation. In general, the following property *DELAY* gives the overall delay time regardless the type of hardware unit. To obtain accurate delay values, it is advised to use fractions for intermediate terms as it is done in the example.

DELAY

This property specifies the worst case propagation delay. Returns an integer.

Example:

```
(DELAY (+ (/ 5 2) (/ 52 10) (* N (/ 3 4))))
```

For some hardware units such as adders and multipliers, the worst case delay time can often be described with a ripple model. The model for the adder is illustrated in Figure 3.3[Rab94]. The major time delay in this model is mostly contributed by the rippling effect and this component is usually proportional to the bitwidth of the adder. In addition to the ripple delay, an extra one-bit delay has to account for the time delay in the first or the last full adder. Depending upon whether the carry-out path or the sum path is faster, an overlap time might be added to or subtracted from the total delay time. As a result, three delay components are used to describe the ripple model.

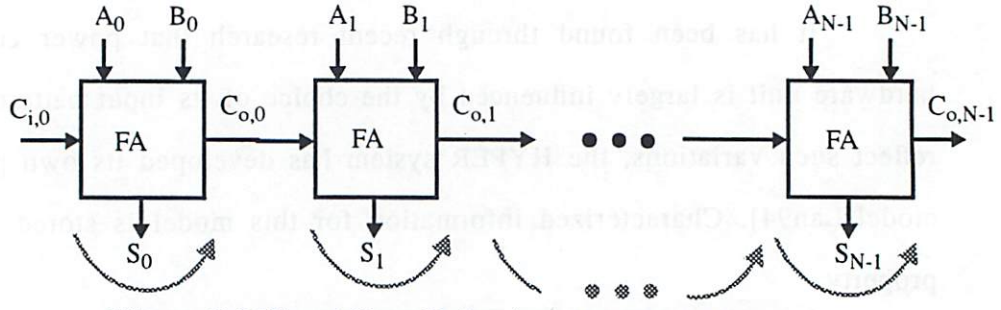

Figure 3.3 Worst Case Delay in a Ripple Adder

ONE-BIT-DELAY

This property specifies the one bit delay of a full adder. Returns an integer.

Example: `(ONE-BIT-DELAY (/ 52 10))`

RIPPLE-DELAY

This property specifies the delay caused by ripple effect. Returns an integer.

```
Example:     (RIPPLE-DELAY (* N (/ 3 4)))
```

RIPPLE-OFFSET

This property specifies the delay offset. Returns an integer.

```
Example:     (RIPPLE-OFFSET (/ 5 2))
```

For hardware units that drive large load, a set of cells are usually designed for various loading requirements. In this library, we classify the load to 6 groups -- "NO", "SMALL", "MEDIUM", "LARGE", "XLARGE" and "HUGE", each of which corresponds to a specific loading capacitance. This information can be proven effective during the *Hardware Mapper* stage for selecting bus drivers.

DRIVING-CAP

This property specifies the driving capability of a cell. Returns a character string.

```
Example:     (DRIVING-CAP XLARGE)
```

3.3.2 Structural Model

To generate a correct version of the hardware netlist, it is necessary for the *Hardware Mapper* to access terminal information from the library. In this section, our focus is on what signals goes into a unit, what signals comes out of a unit, and how signals should be asserted to set up a specific operation. As a reminder, the generic parameters and dynamic parameters defined in Chapter 2 are seen quite often in the examples provided.

While mapping a high-level operation node to its hardware implementation, the data input/output signals of the unit are equivalent to the input/output edges of the node.

DATA-IN-TERMINAL

This property specifies all input signals. Returns a list of character strings.

```
Example:
(DATA-IN-TERMINAL (A B))
(DATA-IN-TERMINAL (EXPAND IN%d 0 (- NrIN 1)))
```

DATA-OUT-TERMINAL

This property specifies all output signals. Returns of list of character strings.

```
Example:     (DATA-OUT-TERMINAL SUM)
```

Of course, to operate any hardware unit, we need a supply and a ground.

POWER-TERMINAL

This property specifies signal names for power and ground.

```
Example:     (POWER-TERMINAL (Vdd (GND))
```

Let us now turn our attention to control signals. Control signals are very different from the data signals described above because they don't directly relate to any high-level nodes or edges. After allocation and scheduling are carried out on the flowgraph, a schedule table can be derived as a function of time. As an example, Figure 3.4 shows a typical scheduling table.

Time	subtractor	adder0	adder1	i/o unit	shifter0	shifter1	transfer unit
0		X	X		X	X	X
1	X		X	X	X		
2	X	X				X	
3	X	X		X			
4	X	X					
5		X				X	X

Figure 3.4 Schedule Table of an IIR Filter Example

In order to generate the control logics for a specific schedule table, control signals on each hardware units should be set up for proper operation at every time point. For example, to find out how to set up a variable shifter so that it shifts right by 3, one has to find out what control signals are needed and how each signal should be

asserted. In this case, the dynamic parameter "SHIFT" can be set to 3 and when the hardware library is called, it returns the corresponding asserted shift controls. To achieve a direct mapping between the local control signals and the dynamic parameters, the following property *CTL-IN-TERMINAL is defined.* The general format of this property is to pair each local control signal on the hardware unit with its corresponding dynamic parameter. This is illustrated in the first example. The second example shows a case where two dynamic parameters are combined logically to produce one local control signal. In the last example, one dynamic parameter is mapped to two local control signals.

CTL-IN-TERMINAL

This property specifies the setups of input control signals. Returns a list structure.

```
Example:
(CTL-IN-TERMINAL ((CLK CK1) (CIN GND)
       (LD LOAD) (CNT COUNT) (RST RESET)))
(CTL-IN-TERMINAL ((CLK (and WRITE CK1))))
(CTL-IN-TERMINAL ((S1 S0) SELECT))
```

For certain hardware units, the output control signals are also important for setting up the control table. For instance, the output of a comparator can well be used as a branch indicator. These output control signals are defined in the following property. As shown in the second example, extra logics can be placed on any outcoming control signals to generate other desired output configurations.

CTL-OUT-TERMINAL

This property specifies the output control signals. Returns a list structure.

```
Example:
(CTL-OUT-TERMINAL AGTB)
(CTL-OUT-TERMINAL (not (or AGTB AEQB)))
```

As a special case of *CTL-OUT-TERMINAL*, the following property defines any existing complementary output control signals. This can be useful for reducing the number of inverter in the control logics.

COMPLEMENT-OUT

This property specifies complementary output signals.

```
Example:    (COMPLEMENT-OUT COUTINV)
```

To plan for a compact, routing efficient layout, it's essential to understand the geometry of each hardware block and their individual input/output terminal location. The following two properties do exactly that. For the he property *CTL-TERM-EDGE*, terminal edges an be defined in four different ways -- *TOP, BOTTOM, LEFT* or *RIGHT*.

HEIGHT/WIDTH

Cell height or width in λ. Each returns an integer.

```
Example:    (HEIGHT 207)
            (WIDTH (+ 20 (* 64 N)))
```

CTL-TERM-EDGE

This property specifies control edge of signals. Returns a list structure.

```
Example:
(CTL-TERM-EDGE ((CLK BOTTOM) (CIN BOTTOM)
    (LD BOTTOM) (CNT BOTTOM) (COUT TOP)))
```

Finally, for parameter driven silicon complier, such as the Lager tools[Lager91], hardware parameters are often required to be included in the structural description language. The following property maps the necessary hardware parameter(the first in the pair) to a generic parameter or a constant value(the second in the pair).

PARAMETERS

This property specifies the setups of hardware parameters. Returns a list structure.

```
Example:
(PARAMETERS (N N) (CLKINV 0) (FB 1))
```

3.3.3 Hardware Behavior

It is quite often the case that not all units in an operator family can implement its corresponding high-level operation because of hardware constraints. During *Module Selection*, hardware units with unsatisfied constraint requirements can never be selected. In the following example, a shifter's shift value is limit to be from 0 to 31.

CONSTRAINT

This property specifies the hardware constraints of the cell. Returns an integer.

```
Example:
(CONSTRAINT (&& (>= M 0) (<= M 31)))
```

As seen throughout Section 3, generic parameters and dynamic parameters are used in expressions often. It is usually true that a default value can be defined for these parameters to prevent any expression interpreter to crash. However, the most effective outcome of using a default value is that it can provide general hardware information without knowing specific details. In the example shown below, parameter "KEY" is set to default value of "UU/XX", which is the key corresponding to the white noise input pattern for capacitance calculation. This default value can be overwritten only when more specific input patterns are acquired and the key is set to something other key.

DEFAULT

This property specifies the default values for generic parameter or dynamic parameter. Returns a list structure.

```
Example:     (DEFAULTS (KEY UU/XX))
```

It has been observed that in many cases, we can combine *CONSTRAINT* and *DEFAULTS* for special selection purposes. For example, we can place a constraint on a certain parameter OUTTYPE to equal to "TRI", meaning the output of the unit is tri-stated. Then, at the *DEFAULTS* statement, OUTTYPE can be again set to "TRI".

```
(CONSTRINT (!strcmp OUTTYPE "TRI"))
(DEFAULTS (OUTTYPE "TRI"))
```

This generally does not change anything when the library is accessed as long as OUTTYPE is not specified otherwise. However, if OUTTYPE is specified as "NONTRI", meaning a unit that is not tri-stated, the default will be overwritten and the particular hardware unit is rejected from the selection.

3.4 Future Work

The hardware library described in this chapter has been developed to define a set of hardware properties for any generic cell library. Seen by many HYPER tool developers, the frame work it provides has proven to cover a wide variety of needs. However, future improvement should be looked into as well. As more new design features are being introduced into the HYPER system, new properties should be identified and added to the existing library.

4

Technology Library

Once a cell library is developed, each cell is characterized and its hardware properties can be extracted and compiled. The resulting product is a hardware library that can be used by the HYPER system. It is generally true that the development of such a cell library is based upon a certain process technology. Therefore, a set of technology properties are common to all hardware units in the library. These properties are now compied to the technology library. Properties introduced in this chapter are stored in a text file in the same directory where the hardware library resides. The *list functions* defined in the previous chapter are allowed here as well..

4.1 Scaling Factor

The cost and performance of a cell can only be comprehended with proper units attached. For example, the following three properties are used to annotate appropriate units to information on size, time and capacitance.

```
(size_units micron)
(time_units nsec)
(cap_units fF)
```

It is often the case that data indicated in the hardware library are not exactly scaled to the unite specified. For example, we usually like to use λ to describe size because such information can be applied to different process technology. For this reason, the scaling factors are also included in the technology file. The examples are shown as follows.

```
(size_scale (/6 10))
(time_scale 1)
(cap_scale 1)
```

4.2 Global Clock

One-phase clock scheme and two-phase clock scheme obviously have different impact of the global control generator. The *Hardware Mapper*, therefore, handles hardware units such as register files and memories quite differently for the two schemes. The property below defines the number of clocks needed for a specific library.

```
(nr_of_clocks 1)
```

Since a cell library is usually developed to meet certain timing specifications, it is reasonable to give a general ideal of the average delay in a library. Because the HYPER system is interactive, we define a default clock period based on the delay information. This property helps the HYPER users to start working on a problem with a appropriate clock frequency without doing trial-and-error.

```
(clock_period 50)
```

For an event-driven design, each clock period represents a state. The state transition point is therefore determined by either the rising edge or the falling edge of the clock. To illustrate this, the following property is defined.

```
(state_transition rising)
```

4.3 Supply Voltage

A specific supply voltage is used to compile any delay information. As illustrated in Figure 2.1, the delay time varies as a function of supply voltage. In the technology file, we define the following two properties to show this dependence. The *supply_voltage* specifies the voltage at which the library cells are characterized. And the *delay_scale* lists a group of points taken from an average delay versus supply curve similar to Figure 2.1. For a given voltage supply, the correct scaling factor for delay is can be obtained through interpolation between two adjacent points.

```
(supply_voltage 3)
(delay_scale ((10 280) (15 69) (20 35) (25 23)
     (30 18) (35 14) (40 12) (45 11) (50 10)))
```

Routines have been developed to read off the information provided by *delay_scale* and again, the numbers above appear to be ten times larger because the current parser does not parse real numbers.

4.4 Process Variation

The impact of using a different process technology lies in several aspects. First of all, the size scaling is different, which affects the total area estimation. This adjustment can be reflected in the property *size_scale* defined in Section 4.1.

As the device scales, the delay information should be re-calibrated as well. It is therefore necessary to generate a delay versus technology curve. In addition, the change of process also causes the delay versus supply curve to scale accordingly. These variation factors have not be described in the current technology library and are part of the future work.

5

Design and Implementation of A Low Power Wavelet Filter

It has been our experience that when the hardware properties are accurately modelled and represented in a hardware library, the HYPER system can be quite powerful in exploring various design spaces. The hardware models depicted in the individual library are the basis of optimizing designs in terms of time, area or/and power. As the design constraints become more complicated, high quality optimization can hardly be achieved by hand. However, powered by an advanced synthesis tool such as the HYPER system, multi-dimensional optimization is now feasible. The focus of this chapter is to investigate how different hardware properties in the library are used to achieve low power design.

5.1 Overview

With today's technology, high performance systems are widely begin developed while the demand for portable computing and communication devices with low power consumption has become a challenge for all designers. It has been found through research that in general, different techniques should be applied at different

levels of design processes in order to achieve low power. Some of these design approaches will be discussed in Section 5.2. One of the techniques we are particularly interested in is the architectural transformation. In a datapath intensive design, the trade-off between using more time shared units and using more dedicated units is a major factor in determining the total power consumption of the final implementation. To explore the effect of such trade-off, a case study is done on a 14th order wavelet filter. Two architectures are chosen to represent the two extreme cases of the time sharing trade-off. Two chips are designed and built in the HYPER system based on the selected architectures. A comparison of power consumption is then made and relevant simulation results are compiled.

5.2 Low Power Design Approaches

In a digital CMOS circuit, power consumption is contributed by three major factors -- dynamic switching, direct path current and leakage current. Generally speaking, the switching power is the most influential factor. As a rule of thumb, switching power is directly related to supply voltage, active switching capacitance and effective switching frequency, as illustrated in the following formula[Cha92]:

$$Power_{switching} = C_L * V_{DD}^2 * f_{effective}$$

It is apparent that power can be saved by reducing the values of the three variables above. However, power saving usually comes with other cost. For example, reducing the capacitance usually means reducing the transistor size, which can lead to slowing down the circuit. Same happens when the supply voltage is reduced. On the other hand, it is also possible to reduce power consumption with the least extra cost by optimizing circuit level designs and applying high-level transformations. For instance, to ensure that the number of glitch events is minimum, a tree structure is better than a chain structure in a sequence of additions[Cha93].

There are basically five levels of optimization at which power can be reduced. Namely, they are technology, circuit/logic, architecture, algorithm and system[Cha92]. While different low power design techniques are applicable at each level, optimization at the architectural level has by far the biggest impact on overall power reduction[Cha93].

Examples of architectural transformation include pipelining, parallelism, redundancy, etc. First, take a look at Figure 5.5. After applying pipelining, operation A is divided into N stages, separated by registers. Assuming the delay is evenly distributed among these N stages, the worst case clock frequency is increased by a factor of N. Therefore, to achieve the same throughput, the pipelined version can be run at a lower supply voltage. According to the previous power equation, the reduction of supply voltage reduces the total power consumption quadratically. Of course, the price we are paying here is the cost of extra registers, in terms of area as well as active capacitance. This is why optimal power consumption is not achieved with pipeline of infinite depth. Clearly, the optimal point in this case depends on the trade-off between delay scaling factor and register cost.

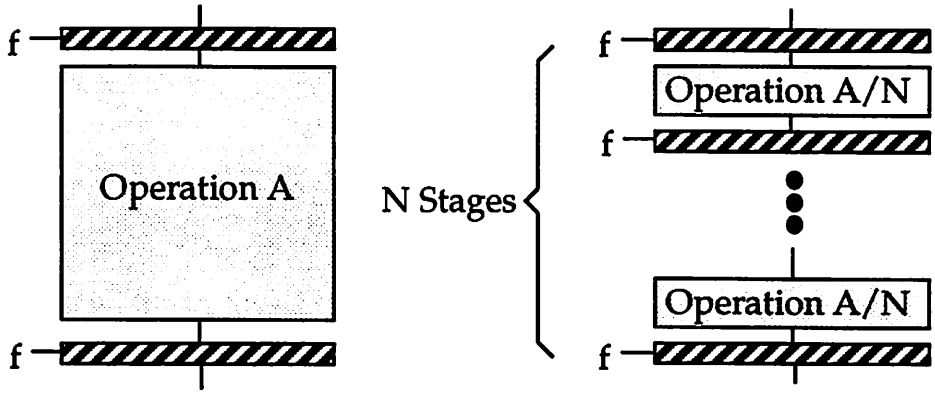

Figure 5.5 Effect of Pipelining

It has been observed that in many real-time applications, input data to a system are very often correlated. For instance, a sequence of speech samples does not consist many high frequency components and the value of each sample does not vary a lot

compared to its adjacent samples. The different behaviors between speech data input and random noise data input are illustrated in Figure 5.6. As expected, every bit in a random noise input is switching at about 50% probability. However, when interpreted in binary numbers, higher bits of the speech data exhibits much lower probability of switching. The most significant bit -- the sign bit, switches far less often than the least significant bit, which is acting almost like a random noise input.

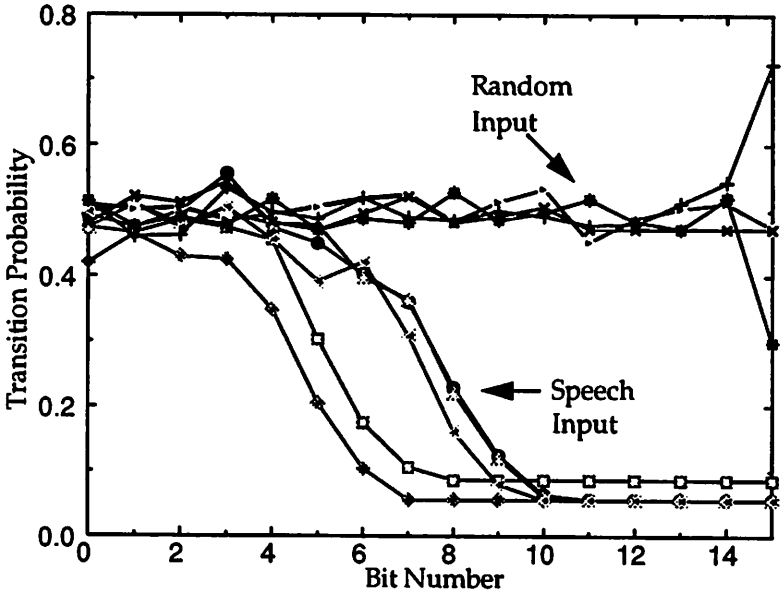

Figure 5.6 Transition Probability of Different Input Patterns

HYPER system has been used to synthesize systems that involve large number of computations. These computations must be carried out by a set of execution units. In the past, a minimum number of execution units is usually preferred to achieve minimum silicon cost. This results in high level of time sharing of execution units. In addition, data buses are shared among a number of units as well. In a system that handles correlated data, the time sharing tends to randomize the data being put on the bus. This effect is shown in the example on Figure 5.7. A single counter counts continuously and its output data exhibit the behavior shown in Figure 5.6 because of correlation. In the first case, when each counter has its dedicated output bus, we observe a low probability of transition on certain bits of the output, namely, the higher bits. In the second case on the right, when the two data buses are merged and two counters are putting uncorrelated

data onto the same bus during adjacent clock periods, the data on the bus become very random unless the two counters are not independent of each other. Although the total amount of capacitance on the bus does not vary too much, the effective switching frequency has been greatly increased. As a result, the power contribution from the bus puts a large burden on the total power dissipation.

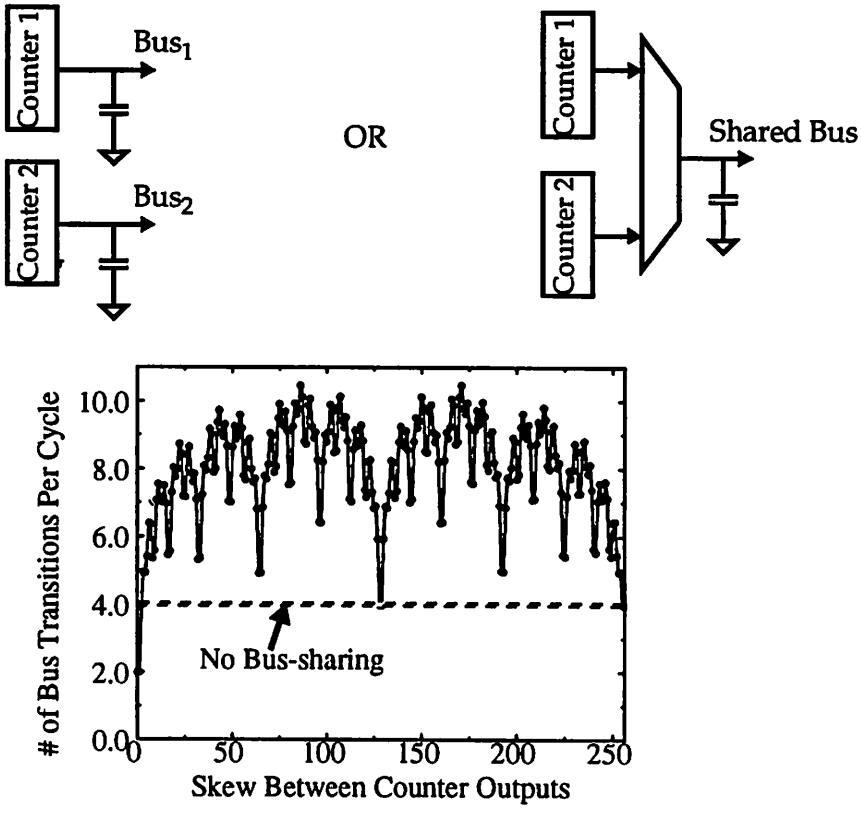

Figure 5.7 Example of Bus Sharing

On the other hand, without no time shared hardware unit, every single computation has to be carried out by a set of dedicated execution units. This implementation is called parallelism. From a traditional synthesis point of view, this scheme is not desirable because it always involves more silicon cost. However, as demonstrated in the previous paragraph, from a low power point of view, this approach may be a winner despite its overhead on silicon area. Also, as the process technology continues to make progress, the area cost will eventually become less significant than the power cost.

As stated earlier, the focus of this project is to investigate the trade-off between time sharing and parallelism. In theory, there exists a optimal point where the right amount of time sharing and parallelism can be combined to implement a system with the least power consumption. At the same time, area cost is also one of the issues considered for such a trade-off comparison.

5.3 Wavelet Filter

Wavelet theory has been used for many multi-resolution signal processing applications[Rio91]. The wavelet transform provides better frequency resolution at low frequencies by spreading the frequency response of encoding filters in a logarithmic scale. Figure 5.8 compares the analysis windows of the wavelet transform with the one using the short-time fourier transform. It is shown that the latter has fixed resolution at all frequencies.

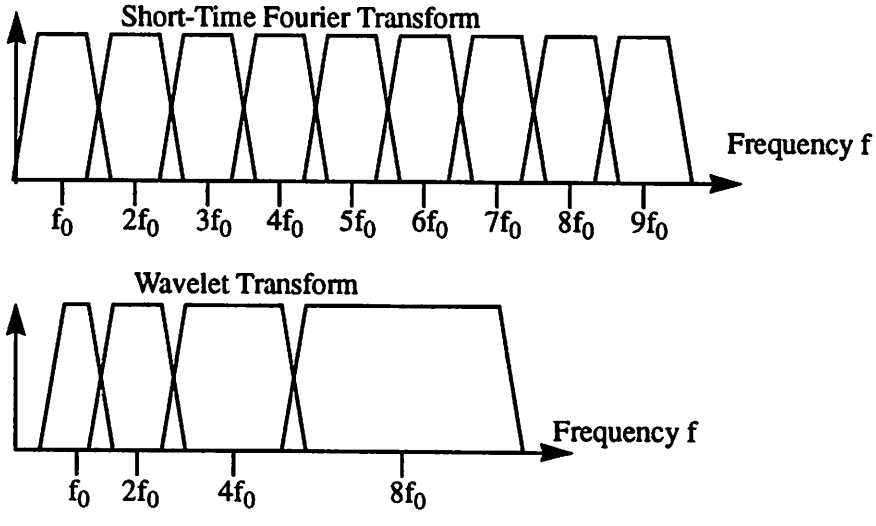

Figure 5.8 Division of the frequency domain

One way of implementing the above wavelet transform is shown in Figure 5.9. The basic block for this system is a discrete-time filter called wavelet filter. Each wavelet includes a halfband low-pass filter and a halfband high-pass filter. To achieve higher resolution, the low-pass filter output is cascaded to the next stage wavelet filter.

This scheme is called subband coding. The appropriate scaling between consecutive stages are made with sub-sampling by two.

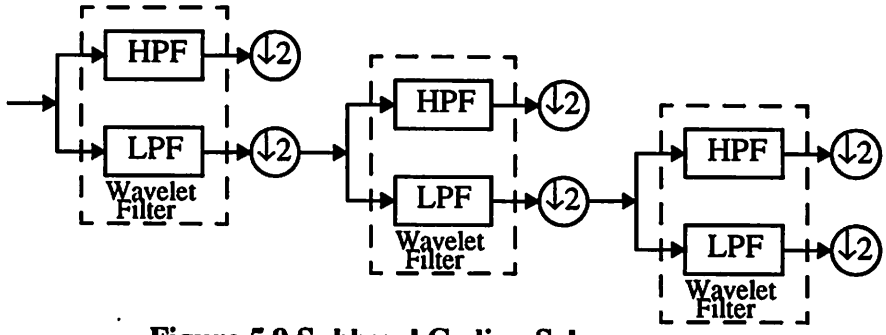

Figure 5.9 Subband Coding Scheme

5.4 Design Flow

5. 4. 1 Filter Specifications

In this project, we are designing a wavelet filter with 14-tap impulse response. The top level flowgraph is shown in Figure 5.10. As an input file to the HYPER system,

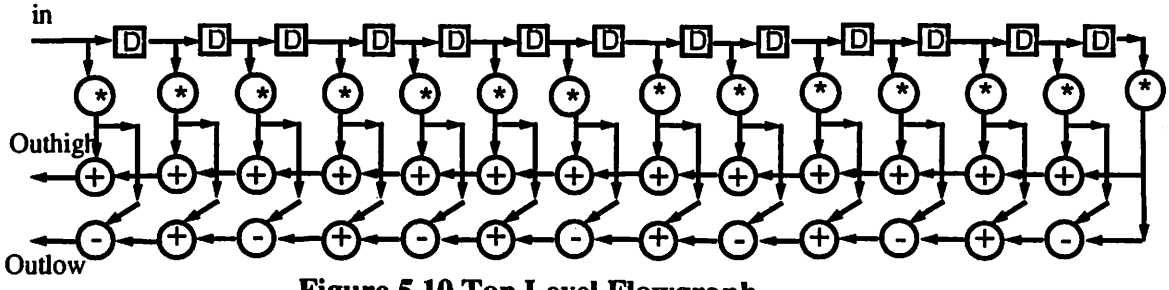

Figure 5.10 Top Level Flowgraph

the flowgraph can be described in the SILAGE language, shown below.

```
#define word fix<16,15>

/* 22 nonzero bits, high pass design */
#define a0    word(0.015625)
#define a1    word(0.015625)
#define a2   word(-0.046875)
#define a3   word(-0.031250)
#define a4    word(0.093750)
#define a5    word(0.109375)
#define a6   word(-0.468750)
```

```
#define a7    word(0.468750)
#define a8   word(-0.109375)
#define a9   word(-0.093750)
#define a10   word(0.031250 )
#define a11   word(0.046875)
#define a12  word(-0.015625)
#define a13  word(-0.015625)

func main(In : word) Outhigh, Outlow : word =
begin
    Acc13 = word(In@13 * a13);
    Acc12 = Acc13 + word(In@12 * a12);
    Acc11 = Acc12 + word(In@11 * a11);
    Acc10 = Acc11 + word(In@10 * a10);
    Acc9 = Acc10 + word(In@9 * a9);
    Acc8 = Acc9 + word(In@8 * a8);
    Acc7 = Acc8 + word(In@7 * a7);
    Acc6 = Acc7 + word(In@6 * a6);
    Acc5 = Acc6 + word(In@5 * a5);
    Acc4 = Acc5 + word(In@4 * a4);
    Acc3 = Acc4 + word(In@3 * a3);
    Acc2 = Acc3 + word(In@2 * a2);
    Acc1 = Acc2 + word(In@1 * a1);
    Outhigh = Acc1 + word(In * a0);

    Acclow13 = word(In@13 * a13);
    Acclow12 = Acclow13 - word(In@12 * a12);
    Acclow11 = Acclow12 + word(In@11 * a11);
    Acclow10 = Acclow11 - word(In@10 * a10);
    Acclow9 = Acclow10 + word(In@9 * a9);
    Acclow8 = Acclow9 - word(In@8 * a8);
    Acclow7 = Acclow8 + word(In@7 * a7);
    Acclow6 = Acclow7 - word(In@6 * a6);
    Acclow5 = Acclow6 + word(In@5 * a5);
    Acclow4 = Acclow5 - word(In@4 * a4);
    Acclow3 = Acclow4 + word(In@3 * a3);
    Acclow2 = Acclow3 - word(In@2 * a2);
    Acclow1 = Acclow2 + word(In@1 * a1);
    Outlow = Acclow1 - word(In * a0);
end;
```

In this example, the fourteen multiplication constant values are interpreted in 16-bit binary numbers.

5. 4. 2 Transformation

First, we start by mapping the SILAGE language to an initial flowgraph which resembles what is in Figure 5.10. Then, in the HYPER system, different transformations are performed at the flowgraph level, namely, they are *Constant Multiplication*, *Retiming* and *Partial Chaining*.

5. 4. 2. a Constant Multiplication

Because of the fact that each multiplication in this design has a constant operand, it is possible to transform these multiplication operations to add-shift operations. As illustrated in the example on Figure 5.11, multiplying by "00101000" can be transformed to an addition and two constant shift operations. In the wavelet filter design, expensive multipliers can now be replaced with cheaper adders and shifters. In addition to the area cost savings, simple blocks such as adders and shifters make it much more flexible to apply various high-level flowgraph transformations in the HYPER system.

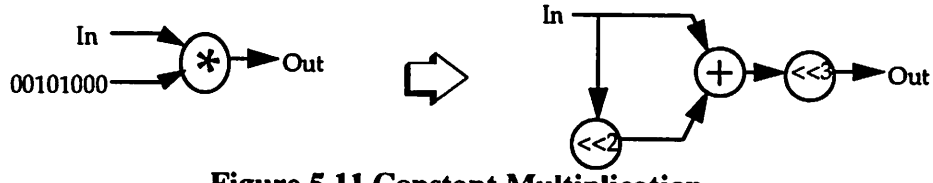

Figure 5.11 Constant Multiplication

5. 4. 2. b Retiming

After the constant multiplication, the high-level flowgraph only consists of adders, shifters and occasionally, delay elements. The delay elements define the stages of computation needed for the filter. In the wavelet filter case, a total number of 14 computation steps are required for each tap of output. With a certain sampling frequency, the delay elements can be mapped to registers which isolate subsequent computation results. Each of these registers is clocked by the global clock signal. However, through direct mapping, the nature of the FIR filter tends to create

unbalanced delay paths on the high-level flowgraph. This is not desired in getting the best performance because the critical path of this implementation is determined by the worst-case delay between subsequent registers. As a result, the retiming transformation is brought in to move the registers along the delay path in order to generate a flowgraph with the minimum register to register delay.

5. 4. 2. c Partial Chaining

From a hardware mapping point of view, the two target architectures for the final design has to be considered separately.

The time shared implementation calls for simple and regularly used hardware units to be shared among various arithmetic computations. Chained operators are not likely to be used by different computations as a regular basis.

As for the parallel implementation, each computation operator is mapped to a dedicated hardware unit. All operators between any two subsequent delay elements can be chained to form one datapath. An advantage of applying the parallel scheme is that the number of shifts is predefined for all shifters. Shifters with constant number of shifts can simply be mapped to metal wires, saving large amount of active silicon area. Moreover, partial chaining does not dramatically increase the register to register delay and hence keeps the critical path within reasonable range.

5. 4. 3 Hardware Allocation

The major objective of applying the time shared scheme is to utilize the minimum number of execution units, namely, the adders, the subtractors, the shifts and so on. Because of the limited number of execution units available, large number of registers are used for intermediate data storage. As mentioned earlier, the number of computation stages of a FIR filter is the same as the number of taps in the impulse response. With the time shared units, each computation stage is allocated to several

execution units and is carried out in more than one clock cycle, making the available time dependent on the amount of hardware resource provided. The scheduling of the system is optimized using HYPER's scheduler to ensure minimum possible available time. The architecture provided by current version of HYPER system uses a global state machine and local control logic blocks to manipulate the operations of each execution units at any time. Along with each execution unit, a big array of register file is generated to store intermediate data. Data transfers are conducted across execution units via data buses.

On the other hand, the fully parallel version of the wavelet filter is implemented with maximum number of execution units required, meaning dedicated hardware unit for each high-level operation. Compared to the fully time shared wavelet filter, the fully parallel one does not need any control logic blocks. This leaves us with datapath units only, which can be built with datapath compilers to utilize silicon area most efficiently. The constant shifters also contribute to area savings as well. As each computation stage is pipelined, the available time of the system is simply one clock cycle.

From a design point of view, the two implementations discussed above are chosen for comparison on the basis of area and power cost. Each has its advantages and disadvantages. First of all, the time shared filter tries to use the least amount of execution units, where complex calculation is often performed. It has been found that these hardware units cost more in area as well as in power. However, with less computation resources, the price being paid in this implementation is more silicon area and power dissipation for building storage register files, global and local control logics and randomized placement/routing. The other extreme, as discussed, tries to use more uniform hardware units, hoping the regularity of the data flow and the elimination of control logic, large routing and large register files will lead to a reduction of power

consumption. Obviously, there exists a trade-off that is dependent on the hardware allocation.

5. 4. 4 Place & Route

In the fully time shared wavelet filter, global and local logic blocks are synthesized with combinational logic optimizer and built with standard logic blocks. As a result, a large number of control signals need to be routed to their destinations. Because of the difference between standard cell compiler and datapath compiler, even with the best routing tools available, messy routing and large chunks of white space is observed in the final layout. In the fully parallel implementation, pipelined data follows a predictable computation flow, which is also reflected in the hardware routing. As all the units can be handled by datapath compiler, place and route become a much easier task to accomplish.

5. 4. 5 Other Considerations

It has been found that power consumed in the I/Os of the chip usually dominates the total chip power. In order to study the area and power trade-off between the two design approaches, separate supply and ground pins are designed to isolate I/O pad power and chip core power. The following simulation and test results are based on the core power only. It should also be noted that even though the core power is only an insignificant part of the total chip power, lowering this portion of the power consumption can be a crucial improvement factor with advanced MCM (Multi-Chip Module) technology.

5.5 Simulation/Test Results

5. 5. 1 Chip Area

As discussed in the previous section, the area cost could be one of the drawbacks in the fully parallel approach. Figure 5.12 compares the layout of the two fabricated chips in the same scale. It is obvious that the concern for extra area cost is not necessary. It turned out that the area overhead in the fully parallel wavelet is about 16.9% more. It is observed that because of placement regularity, there is less open

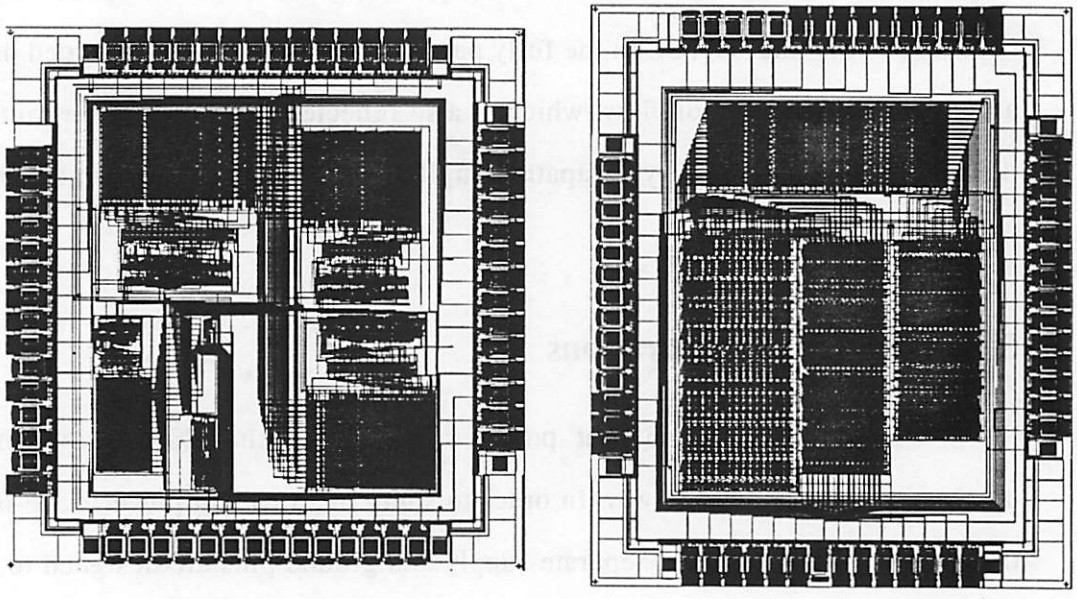

Figure 5.12 Chip layout of Wavelet Filter
Fully Time Shared (Left) & Fully Paralleled (Right)

space on the chip and silicon area has been efficiently utilized. If the active area is the concern, area ratio of the two designs are roughly 2:1, which is hardly unreasonable. On the fully time shared wavelet chip, each execution unit is densely placed and routed, while the overall chip placement and routing become quite messy, leaving a lot of white space.

5. 5. 2 Clock Frequency/Sample Rate

On the fully time shared wavelet chip, the critical path is the worst case delay in any of the execution units, plus the register set-up and hold time. However, it takes 22 clock cycles to compute a sample output. If the worse case delay is 17ns, the maximum clock frequency is limited to 58.8MHz and the maximum sample rate is limited to 2.7MHz.

The worst case register to register delay on the fully parallel wavelet chip, after retiming, is 18ns. Because every output of the filter is pipelined, the maximum achievable clock frequency and sample rate are the same, 55.6MHz.

5. 5. 3 Power Consumption

To compare the power consumption on each chip, a piece of digitized voice data is put onto the input of the filter. IRSIM, a switch-level simulator then simulates both chips and counts the number switching events on any non-zero capacitance nodes on the chip. The total power consumption for this voice data is then generated by:

$$TotalPower = \sum_i \frac{1}{2} C_i V^2 f_i$$

The simulation results are shown in Figure 5.13. It has been observed in the simulations that the average power consumption per sample stabilizes after some time.

It is also shown that the power ratio between the fully time shared wavelet and the fully paralleled one is about 1:8.

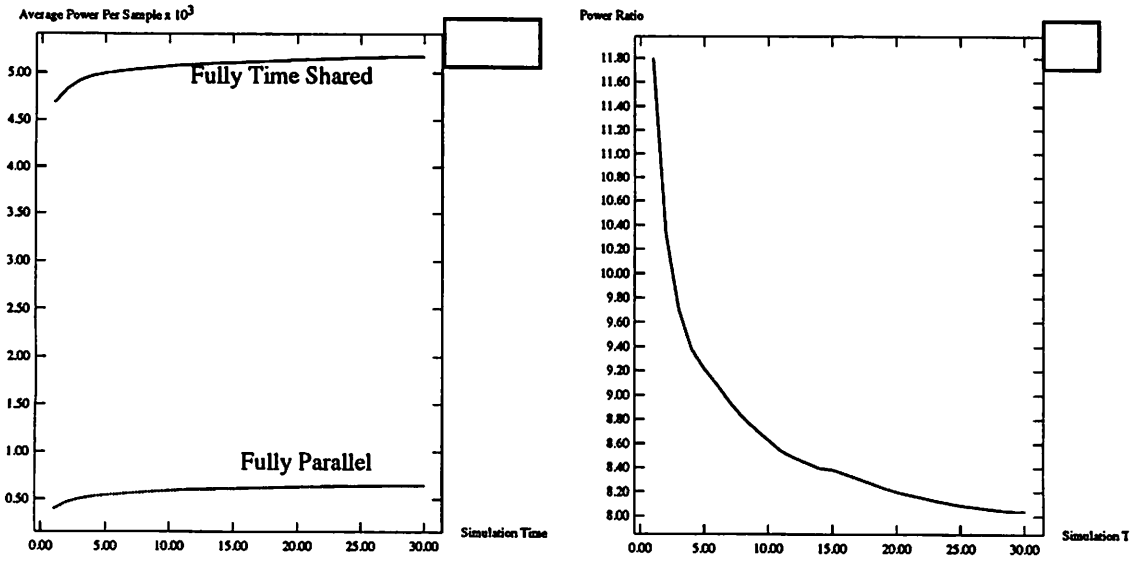

Figure 5.13 Power Consumption of Wavelet Filter

Surprisingly enough, a great amount of power can be saved using the fully paralleled wavelet filter. The following is a table showing power breakdown in the fully time shared chip.

Table 5.4 Power Breakdown of Fully Time Shared Implementation

Hardware Components	Power Consumption (%)
Tri-State Buffer & Mux	27.2
Local Control	18.9
Register File	11.7
Subtractor	10.5
Adder	8.4
Global Control	5.7
Shifter	5.0

It is obvious that more than half of the power is dissipated in the controller, register files and bus-related units. These units are eliminated in the fully parallel design, which only consists of the adders, the subtractors, the shifters and the

individual registers. Since shifters in the parallel design are simply metal wires, shifter itself does not consume any power. It, however, contributes to extra capacitance for the its driving stage. Even the adders and subtractors consume less power in the paralleled design than their counterparts in the time shared wavelet design because of correlated data.

The above analysis no doubt supports the power savings stated below. Table 5.5 lists the simulation results to compare the trade-offs between different aspects of the two implementations.The comparison is based on the same throughput at a sample frequency of 2.7MHz. The last column of Table 5.5 has taken into account the voltage supply difference for each of the two implementations.

Table 5.5 Comparison of IRSIM Simulation Results

	Area	V_{DD}	Eff. Cap.	Power
Time Shared	13.6 mm^2	5 V	5,171 pF	350 mW
Parallel	15.9 mm^2	1.2 V	643 pF	2.5 mW

5. 5. 4 Chip Testing

The two wavelet chips discussed in this chapter have been both fabricated using MOSIS 2.0-micron process. Some of the test results are given in this section.

Shown in Table 5.5, the fully parallel implementation can be run at a much lower supply voltage to achieve the same throughput as in the case of the fully time shared implementation. It is tested that the critical path of the fully parallel wavelet filter does follow the delay versus supply curve given in Figure 2.1. Figure 5.14 is

generated using chip testing results and it shows the predicted dependency of delay over supply.

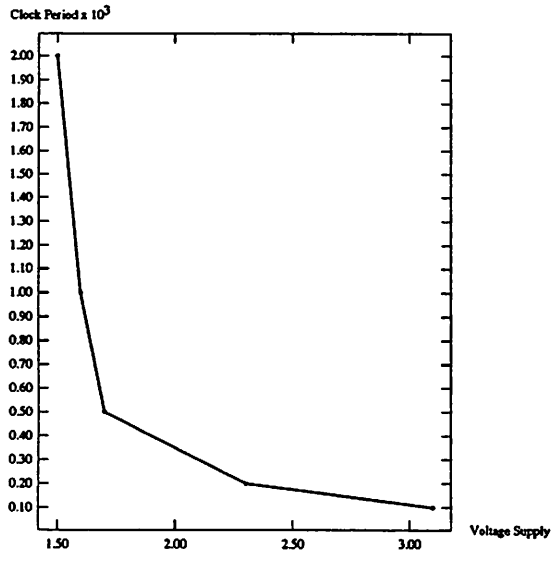

Figure 5.14 Tested Clock Frequency Versus Supply Dependency

To verify the functionality, an impulse input is put onto the input of the filter and corresponding output signals, both low-pass and high-pass, are collected. The transfer functions of the filter in the frequency domain are computed accordingly, shown in Figure 5.15.

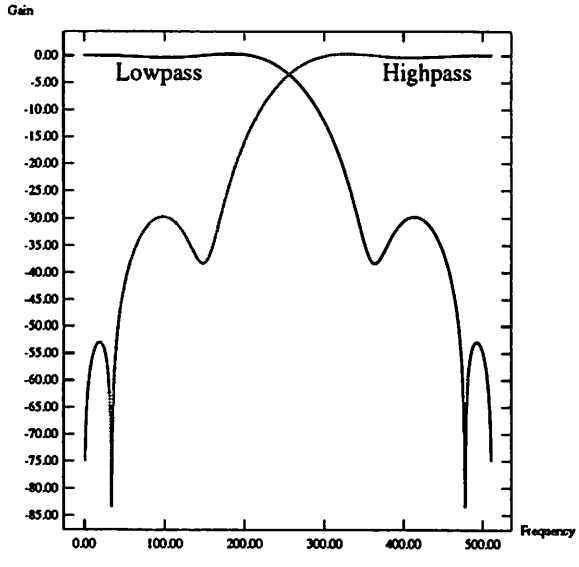

Figure 5.15 Transfer Function in Frequency Domain

5.6 Summary

Two extremely different architectures have been used to implement a wavelet filter. As expected, the parallel approach consumes much less power for the same throughput because of its simplified hardware structure and highly correlated data flow. The extra overhead cost is insignificant.

6

Conclusions

For any high-level synthesis tool, the hardware properties of available resources are compiled into libraries. This report describes such a library representation which can support a wide variety of hardware units. The syntax of the library and the different categories of hardware properties are described in detail. Efforts have been made to keep the high-level tools away from the properties of each individual hardware unit. It is expected that future improvement in the HYPER environment may require the library representation to include other properties. However, development of such can be easily achieved with very little change to the current library representation.

Also included in this report is the design and implementation of a wavelet filter. Through theoretical analysis, statistical analysis, simulation and testing, it is shown that the power consumption is greatly influenced by the architecture chosen for implementation. A good synthesis tool, therefore, should be able to find a optimal solution for least power consumption. Such a tool requires accurate estimation of hardware cost, in area as well as in power. The hardware library is essentially the backbone in every step of the synthesis step. In conclusion, the continuous refinement of the hardware library representation is a key factor in building a high-level synthesis system.

Appendix A

Hardware Database

```
        (

   (++
("counter"
   (PARAMETERS (N N))
   (CAP (* C0 N))
   (CAP-COEFFS (154))
   (AREA (* 160 (+ 75 (* 64 N) (* 31 (- (ceiling (/ (+ N 1) 16)) 1)))))
   (DELAY (+ 5 (* N (/ 3 4))))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 2)
   (RIPPLE-DELAY (* N (/ 3 4)))
        (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL
((CLK CK1) (CIN GND) (LD LOAD) (CNT COUNT) (RST RESET)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE
((CLK BOTTOM) (CIN BOTTOM) (LD BOTTOM) (CNT BOTTOM) (COUT TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)))
   (>=
("logcomp"
   (PARAMETERS (N N) (TWOS_COMP 1))
```

```
(DEFAULTS (KEY UU/UU/XX))
(CAP (* C0 N))
     (CAP-COEFFS (LOOKUP KEY
       ((UU/UU/XX ( 181))
        (UU/US/XX ( 160 202))
        (UU/SU/XX ( 215 153))
        (UU/SS/XX ( 112 312 206 101))
        (US/UU/XX ( 152 206))
        (SU/UU/XX ( 176 206))
        (US/US/XS ( 175 156 0 164 182 0 219 212))
        (SU/SU/SX ( 226 236 0 121 206 0 133 152))
        (US/SU/XX ( 232 90 188 195))


        (SU/US/XX ( 194 165 130 254))
        (US/SS/XS ( 189 134 0 285 152 177 0 59 92 0 315 262 220 0
              119 161))
        (SU/SS/SX ( 189 182 234 283 0 222 0 73 78 0 363 0 185 180
              85 126))
        (SS/UU/XX ( 84 261 254 129))
        (SS/US/XS ( 120 138 0 75 241 0 223 251 194 181 0 274 86 0
              231 171))
        (SS/SU/SX ( 111 143 0 62 296 293 0 238 299 0 105 157 115
              0 167 156))
        (SS/SS/SS ( 0 27 84 0 0 145 0 219 0 0 163 199 0 0 0 0 304
              0 251 0 216 262 311 212 0 0 245 0 0 0 180 223
              246 160 0 0 0 345 0 0 38 93 130 41 0 121 0 203
              0 0 0 0 316 235 0 0 212 0 161 0 0 27 62 0)))))
(AREA (* 64 (+ 187 (* 14 (ceiling (log N 2))))))
(HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
(WIDTH (* N 64))
(DELAY 0)
(RIPPLE-OFFSET 0)
(ONE-BIT-DELAY 0)
(RIPPLE-DELAY 0)
     (PIPE-DEPTH 1)
(DATA-IN-TERMINAL (A B))
(DATA-OUT-TERMINAL NIL)
(POWER-TERMINAL (Vdd GND))
(CTL-IN-TERMINAL NIL)
```

```
   (CTL-OUT-TERMINAL (or AEQB AGTB))
   (CTL-TERM-EDGE ((AGTB TOP) (AGTB TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)
)
   )
   (>
("logcomp"
   (PARAMETERS (N N) (TWOS_COMP 1))
   (DEFAULTS (KEY UU/UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 181))
           (UU/US/XX ( 160 202))
           (UU/SU/XX ( 215 153))
           (UU/SS/XX ( 112 312 206 101))
           (US/UU/XX ( 152 206))
           (SU/UU/XX ( 176 206))
           (US/US/XS ( 175 156 0 164 182 0 219 212))
           (SU/SU/SX ( 226 236 0 121 206 0 133 152))
           (US/SU/XX ( 232 90 188 195))
           (SU/US/XX ( 194 165 130 254))
           (US/SS/XS ( 189 134 0 285 152 177 0 59 92 0 315 262 220 0
                119 161))
           (SU/SS/SX ( 189 182 234 283 0 222 0 73 78 0 363 0 185 180
                85 126))
           (SS/UU/XX ( 84 261 254 129))
           (SS/US/XS ( 120 138 0 75 241 0 223 251 194 181 0 274 86 0
                231 171))
           (SS/SU/SX ( 111 143 0 62 296 293 0 238 299 0 105 157 115
                0 167 156))
           (SS/SS/SS ( 0 27 84 0 0 145 0 219 0 0 163 199 0 0 0 0 304
                0 251 0 216 262 311 212 0 0 245 0 0 0 180 223
                246 160 0 0 0 345 0 0 38 93 130 41 0 121 0 203
                0 0 0 0 316 235 0 0 212 0 161 0 0 27 62 0)))))
   (AREA (* 64 (+ 187 (* 14 (ceiling (log N 2))))))
   (HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
   (WIDTH (* N 64))
   (DELAY 0)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
        (PIPE-DEPTH 1)
```

```
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL NIL)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL AGTB)
   (CTL-TERM-EDGE ((AGTB TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)
)
   )
   (==
("logcomp"
   (PARAMETERS (N N) (TWOS_COMP 1))
   (DEFAULTS (KEY UU/UU/XX))
   (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 181))
           (UU/US/XX ( 160 202))
           (UU/SU/XX ( 215 153))
           (UU/SS/XX ( 112 312 206 101))
           (US/UU/XX ( 152 206))
           (SU/UU/XX ( 176 206))
           (US/US/XS ( 150 158 175 0 166 0 214 228))
           (SU/SU/SX ( 223 233 146 0 215 0 170 155))
           (US/SU/XX ( 253 103 198 199))
           (SU/US/XX ( 176 139 139 245))
           (US/SS/XS ( 166 161 283 0 161 160 60 0 95 0 303 293 221 0
                  123 146))
           (SU/SS/SX ( 177 182 258 257 225 0 80 0 94 0 324 0 195 196
                  117 101))
           (SS/UU/XX ( 82 249 252 121))
           (SS/US/XS ( 128 131 83 0 245 0 242 254 194 193 264 0 69 0
                  201 201))
           (SS/SU/SX ( 129 130 58 0 286 302 235 0 299 0 129 131 122
                  0 166 164))
           (SS/SS/SS ( 0 33 37 0 185 0 212 0 189 205 0 0 0 0 0 0 269
                  0 312 0 217 264 268 265 260 0 0 0 195 228 0 0
                  205 213 0 0 340 0 0 0 37 94 90 68 155 0 152
                  0 0 0 0 0 278 286 0 0 192 0 232 0 0 33 27 0)))))
   (AREA (* 64 (+ 187 (* 14 (ceiling (log N 2))))))
   (HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
   (WIDTH (* N 64))
   (DELAY 0)
```

```
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
        (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL NIL)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL AEQB)
   (CTL-TERM-EDGE ((AEQB TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)
)
   )
   (<=
("logcomp"
   (PARAMETERS (N N) (TWOS_COMP 1))
   (DEFAULTS (KEY UU/UU/XX))
   (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 181))
           (UU/US/XX ( 160 202))
           (UU/SU/XX ( 215 153))
           (UU/SS/XX ( 112 312 206 101))
           (US/UU/XX ( 152 206))
           (SU/UU/XX ( 176 206))
           (US/US/XS ( 170 170 164 0 0 182 210 211))
           (SU/SU/SX ( 241 218 121 0 0 206 154 143))
           (US/SU/XX ( 232 90 188 195))
           (SU/US/XX ( 194 165 130 254))
           (US/SS/XS ( 137 180 285 0 175 157 59 0 0 92 268 311 0 220
                148 131))
           (SU/SS/SX ( 172 195 278 230 222 0 73 0 0 78 0 363 167 195
                133 87))
           (SS/UU/XX ( 84 261 254 129))
           (SS/US/XS ( 137 119 75 0 0 241 242 232 167 211 274 0 0 86
                179 217))
           (SS/SU/SX ( 159 92 62 0 289 299 238 0 0 299 150 107 0 115
                145 176))
           (SS/SS/SS ( 0 62 36 0 218 0 145 0 204 159 0 0 0 0 0 0 0
                258 0 297 225 305 261 203 0 245 0 0 220 182 0
                0 0 0 167 235 0 0 345 0 38 126 90 41 193 0 128
                0 0 0 0 0 0 0 232 325 0 159 0 214 0 58 25 0)))))
```

```
   (AREA (* N 64 (+ 187 (* 14 (ceiling (log N 2))))))
   (HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
   (WIDTH (* N 64))
   (DELAY 0)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
        (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL NIL)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL (not AGTB))
   (CTL-TERM-EDGE ((AGTB TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)
)
   )
   (<
("logcomp"
   (PARAMETERS (N N) (TWOS_COMP 1))
   (DEFAULTS (KEY UU/UU/XX))
   (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 181))
           (UU/US/XX ( 160 202))
           (UU/SU/XX ( 215 153))
           (UU/SS/XX ( 112 312 206 101))
           (US/UU/XX ( 152 206))
           (SU/UU/XX ( 176 206))
           (US/US/XS ( 170 170 164 0 0 182 210 211))
           (SU/SU/SX ( 241 218 121 0 0 206 154 143))
           (US/SU/XX ( 232 90 188 195))
           (SU/US/XX ( 194 165 130 254))
           (US/SS/XS ( 137 180 285 0 175 157 59 0 0 92 268 311 0 220
                148 131))
           (SU/SS/SX ( 172 195 278 230 222 0 73 0 0 78 0 363 167 195
                133 87))
           (SS/UU/XX ( 84 261 254 129))
           (SS/US/XS ( 137 119 75 0 0 241 242 232 167 211 274 0 0 86
                179 217))
           (SS/SU/SX ( 159 92 62 0 289 299 238 0 0 299 150 107 0 115
                145 176))
```

```
            (SS/SS/SS ( 0 62 36 0 218 0 145 0 204 159 0 0 0 0 0 0 0
                        258 0 297 225 305 261 203 0 245 0 0 220 182 0
                        0 0 0 167 235 0 0 345 0 38 126 90 41 193 0 128
                        0 0 0 0 0 0 0 232 325 0 159 0 214 0 58 25 0)))))
   (AREA (* 64 (+ 187 (* 14 (ceiling (log N 2))))))
   (HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
   (WIDTH (* N 64))



   (AREA (* N 64 (+ 187 (* 14 (ceiling (log N 2))))))
   (HEIGHT (+ 187 (* 14 (ceiling (log N 2)))))
   (WIDTH (* N 64))
   (DELAY 0)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
        (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL NIL)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL (not (or AGTB AEQB)))
   (CTL-TERM-EDGE ((AGTB TOP) (AEQB TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)
)
   )
   (REG
("regfile"
   (CONSTRAINT (&& (== R 1) (|| (== NCON 0) (== NCON 1))
(!strcasecmp OUTTYPE TRI) (!strcasecmp CTRLTYPE HIGH)))
   (DEFAULTS (R 1) (NCON 0) (OUTTYPE TRI) (CTRLTYPE HIGH) (KEY UU/XX)
    (FUNC WRITE) (CS_OFFSET 44) (BUF_OFFSET 47))
   (PARAMETERS ((N N) (R R) (NC (- R NCON))
(REGPLANE (D2B N CONSTPLANE))))
   (CAP (* (/ N N_TOT) (+ C0 (* C1 R) (* C2 N_TOT) (* C3 R N_TOT))))
        (CAP-COEFFS (LOOKUP FUNC
((READ (LOOKUP KEY
                  ((UU/XX ( (87 51 35 8) ))
                   (US/XX ( (103 51 36 12) (98 43 34 3) ))
                   (SU/XX ( (78 43 34 3)   (68 33 39 14) ))
                   (SS/XX ( (88 43 17 3)   (88 43 51 3)
```

```
   (89 43 58 23)  (88 43 17 3) )))))
 (WRITE (LOOKUP KEY
                  ((UU/XX ( (99 30 65 10) ))
                   (US/XX ( (124 43 63 3) (120 52 65 17) ))
   (SU/XX ( (89 35 65 17) (78 43 67 3) ))
                   (SS/XX ( (104 43 38 3)  (104 43 92 32)
    (104 43 92 3)  (104 43 38 3) ))))))))
   (AREA (* (+ BUF_OFFSET (* 42 NCON) (* 62 (- R NCON))) (+ CS_OFFSET (* 64
N) (* 7 (- (ceiling (/ (+ N 1) 8)) 1)))))
   (HEIGHT (+ BUF_OFFSET (* 42 NCON) (* 62 (- R NCON ))))
   (WIDTH (+ CS_OFFSET (* 64 N) (* 7 (- (ceiling (/ (+ N 1) 8)) 1))))
   (DELAY (/ 9 2))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
   (POWER-TERMINAL (Vdd GND))
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (CTL-IN-TERMINAL
((CLK CK1)
((EXPAND RD[%d] 0 (- R 1) 1) (EXPAND READ[%d] 0 (- R 1) 1))
((EXPAND WR[%d] 0 (- (- R NCON) 1) 1)
   (EXPAND WRITE[%d] 0 (- (- R NCON) 1) 1))))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE
((CLK BOTTOM) ((EXPAND RD[%d] 0 R) BOTTOM)
((EXPAND WR[%d] 0 (- (- R NCON) 1)) BOTTOM)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)
)
   )
   (REGFILE
("regfile"
   (CONSTRAINT (&&
(!strcasecmp OUTTYPE TRI) (!strcasecmp CTRLTYPE HIGH)))
   (DEFAULTS (R 1) (NCON 0) (OUTTYPE TRI) (CTRLTYPE HIGH) (KEY UU/XX)
    (FUNC WRITE) (CS_OFFSET 44) (BUF_OFFSET 47))
   (PARAMETERS ((N N) (R R) (NC (- R NCON))
(REGPLANE (D2B N CONSTPLANE))))
   (CAP (* (/ N N_TOT) (+ C0 (* C1 R) (* C2 N_TOT) (* C3 R N_TOT))))
        (CAP-COEFFS (LOOKUP FUNC
((READ (LOOKUP KEY
                  ((UU/XX ( (87 51 35 8) ))
```

```
                (US/XX ( (103 51 36 12) (98 43 34 3) ))
                (SU/XX ( (78 43 34 3)  (68 33 39 14) ))
                (SS/XX ( (88 43 17 3)  (88 43 51 3)
  (89 43 58 23)  (88 43 17 3) )))))
 (WRITE (LOOKUP KEY
                ((UU/XX ( (99 30 65 10) ))
                (US/XX ( (124 43 63 3) (120 52 65 17) ))
  (SU/XX ( (89 35 65 17) (78 43 67 3) ))
                (SS/XX ( (104 43 38 3)  (104 43 92 32)
  (104 43 92 3)  (104 43 38 3) ))))))))
  (AREA (* (+ BUF_OFFSET (* 42 NCON) (* 62 (- R NCON))) (+ CS_OFFSET (* 64
N) (* 7 (- (ceiling (/ (+ N 1) 8)) 1)))))
  (HEIGHT (+ BUF_OFFSET (* 42 NCON) (* 62 (- R NCON ))))
  (WIDTH (+ CS_OFFSET (* 64 N) (* 7 (- (ceiling (/ (+ N 1) 8)) 1))))
  (DELAY (/ 9 2))
  (RIPPLE-OFFSET 0)
  (ONE-BIT-DELAY 0)
  (RIPPLE-DELAY 0)
  (POWER-TERMINAL (Vdd GND))
  (DATA-IN-TERMINAL IN)
  (DATA-OUT-TERMINAL OUT)
  (CTL-IN-TERMINAL
((CLK CK1)
((EXPAND RD[%d] 0 (- R 1) 1) (EXPAND READ[%d] 0 (- R 1) 1))
((EXPAND WR[%d] 0 (- (- R NCON) 1) 1)
  (EXPAND WRITE[%d] 0 (- (- R NCON) 1) 1))))
  (CTL-OUT-TERMINAL NIL)
  (CTL-TERM-EDGE
((CLK BOTTOM) ((EXPAND RD[%d] 0 R) BOTTOM)
((EXPAND WR[%d] 0 (- (- R NCON) 1)) BOTTOM)))
  (COMPLEMENT-OUT NIL)
  (DRIVING-CAP SMALL)))
  (mux
("mux2"
  (SDLNAME "mux")
  (VHDLNAME "mux")
  (CONSTRAINT (== NrIN 2))
  (DEFAULTS (SELECT 0) (KEY UU/XX))
  (PARAMETERS ((N N) (NUM_IN NrIN)))
      (CAP (* C0 N))
      (CAP-COEFFS (LOOKUP KEY
        ((UU/XX (20))
         (US/XX (20 18))
```

```
            (SU/XX (21 21))
            (SS/XX (0 39 41 0)))))
   (AREA (* 56 (+ (* 64 N) (* 25 (ceiling (/ (+ N 1) 8))))))
   (HEIGHT 56)
   (WIDTH (* N 64))
   (DELAY (/ 5 2))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 5 2))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (INA INB))
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL (((SA) SELECT)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE ((SA BOTTOM)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL))
("mux3"
   (SDLNAME "mux")
   (VHDLNAME "mux")
   (CONSTRAINT (== NrIN 3))
   (DEFAULTS (SELECT 0) (KEY UU/XX))
   (PARAMETERS ((N N) (NUM_IN NrIN)))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
           ((UU/XX (32))
            (US/XX (24 36))
            (SU/XX (41 26))
            (SS/XX (0 77 51 0)))))
   (AREA (* 93 (+ 30 (* 64 N) (* 20 (ceiling (/ (+ N 1) 8))))))
   (HEIGHT 93)
   (WIDTH (* N 64))
   (DELAY (/ 8 3))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 8 3))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (INA INB INC))
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL (((SB SA) SELECT)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE ((SA BOTTOM)(SB BOTTOM)))
   (COMPLEMENT-OUT NIL)
```

```
   (DRIVING-CAP SMALL))
("mux4"
   (SDLNAME "mux")
   (VHDLNAME "mux")
   (CONSTRAINT (== NrIN 4))
   (DEFAULTS (SELECT 0) (KEY UU/XX))
   (PARAMETERS ((N N) (NUM_IN NrIN)))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (39))
           (US/XX (30 44))
           (SU/XX (49 33))
           (SS/XX (0 93 63 0)))))
   (AREA (* 113 (+ 79 (* 64 N) (* 16 (- (ceiling (/ (+ N 1) 16)) 1)))))
   (HEIGHT 113)
   (WIDTH (+ 79 (* 64 N) (* 16 (- (ceiling (/ (+ N 1) 16)) 1))))
   (DELAY (/ 8 3))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 8 3))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (INA INB INC IND))
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL (((SB SA) SELECT)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE ((SA BOTTOM)(SB BOTTOM)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)))
   (nop
("add"
   (CONSTRAINT (== NrIN 2))
   (PARAMETERS ((N N) (CS_TYPE "z")))
   (DEFAULTS (KEY UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (76))
           (US/XX (57 86))
           (SU/XX (96 57))
           (SS/XX (10 182 123 0)))))
   (AREA (* 207 (+ 11 (* 64 N))))
   (DELAY (+ (/ 5 2) (/ 52 10) (* N (/ 3 4))))
   (RIPPLE-OFFSET (/ 5 2))
   (ONE-BIT-DELAY (/ 52 10))
```

```
   (RIPPLE-DELAY (* N (/ 3 4)))
        (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL A)
   (DATA-OUT-TERMINAL SUM)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE ((COUT TOP) (COUTINV TOP)))
   (COMPLEMENT-OUT COUTINV)
   (DRIVING-CAP NO)))
   (+
("add"
   (CONSTRAINT (== NrIN 2))
   (DEFAULTS (KEY UU/UU/XX))
   (PARAMETERS ((N N) (CS_TYPE "z")))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 268))
           (UU/US/XX ( 221 308))
           (UU/SU/XX ( 240 259))
           (UU/SS/XX ( 99 358 321 200))
           (US/UU/XX ( 214 283))
           (SU/UU/XX ( 232 265))
           (US/US/XS ( 208 0 350 233 308 281 0 314))
           (SU/SU/SX ( 253 0 228 208 278 214 0 295))
           (US/SU/XX ( 107 324 312 158))
           (SU/US/XX ( 136 354 295 270))
           (US/SS/XS ( 60 0 332 154 317 0 331 319 256 136 0 429 203
                298 0 136))
           (SU/SS/SX ( 108 0 373 0 286 133 164 302 222 81 336 348 0
                391 0 233))
           (SS/UU/XX ( 108 338 340 200))
           (SS/US/XS ( 82 0 289 114 264 183 0 401 317 0 415 323 208
                303 0 226))
           (SS/SU/SX ( 117 0 248 95 395 0 186 296 288 108 0 425 268
                277 0 147))
           (SS/SS/SS ( 3 0 0 0 341 165 0 0 307 0 121 0 0 223 269 15
                344 174 0 0 0 485 0 0 67 242 250 79 0 229 0
                377 300 0 101 0 31 284 342 80 0 0 384 0 0 0
                455 439 0 174 188 21 0 386 0 420 0 0 247 409 0
                0 0 3)))))
   (AREA (* 207 (+ 11 (* 64 N))))
   (HEIGHT 207)
```

```
(WIDTH (+ 11 (* 64 N)))
(DELAY (+ (/ 5 2) (/ 52 10) (* N (/ 3 4))))
(RIPPLE-OFFSET (/ 5 2))
(ONE-BIT-DELAY (/ 52 10))
(RIPPLE-DELAY (* N (/ 3 4)))
     (PIPE-DEPTH 1)
(DATA-IN-TERMINAL (A B))
(DATA-OUT-TERMINAL SUM)
(POWER-TERMINAL (Vdd GND))
(CTL-IN-TERMINAL NIL)
(CTL-OUT-TERMINAL NIL)
(CTL-TERM-EDGE ((COUT TOP) (COUTINV TOP)))
(COMPLEMENT-OUT COUTINV)
(DRIVING-CAP SMALL))
  ("csa"
     (CONSTRAINT (== NrIN 2))
(DEFAULTS (KEY UU/UU/XX))
     (PARAMETERS (N N) (CS_TYPE "z"))
     (CAP (* C0 N))
     (CAP-COEFFS (LOOKUP KEY
       ((UU/UU/XX ( 233))
        (UU/US/XX ( 241 247))
        (UU/SU/XX ( 243 230))
        (UU/SS/XX ( 199 305 299 172))
        (US/UU/XX ( 225 227))
        (SU/UU/XX ( 241 218))
        (US/US/XS ( 182 0 352 268 245 311 0 215))
        (SU/SU/SX ( 273 0 223 202 188 199 0 249))
        (US/SU/XX ( 129 286 313 145))
        (SU/US/XX ( 211 276 262 195))
        (US/SS/XS ( 82 0 265 158 297 0 353 295 292 381 0 327 170
              296 0 105))
        (SU/SS/SX ( 266 0 325 0 279 158 131 227 228 115 145 251 0
              320 0 210))
        (SS/UU/XX ( 96 333 299 102))
        (SS/US/XS ( 62 0 224 125 345 409 0 325 290 0 395 348 109
              231 0 87))
        (SS/SU/SX ( 101 0 214 102 389 0 212 283 258 156 0 349 120
              221 0 92))
        (SS/SS/SS ( 0 0 0 0 320 182 0 0 283 0 144 0 0 142 126 0
              390 476 0 0 0 368 0 0 204 324 310 191 0 207 0
              300 268 0 174 0 78 225 221 74 0 0 320 0 0 0
              499 422 4 141 136 0 0 173 0 292 0 0 174 330 0
```

```
                0 0 0)))))
        (AREA (* 232 (+ 40 (* 64 N))))
    (HEIGHT 232)
    (WIDTH (+ (* 64 N) 18 10))
        (DELAY (+ (/ 13 10) (* (/ 6 10) (sqrt N))))
        (RIPPLE-OFFSET (/ 13 10))
        (ONE-BIT-DELAY 2)
        (RIPPLE-DELAY (* (/ 6 10) (sqrt N)))
        (PIPE-DEPTH 1)
        (DATA-IN-TERMINAL (A B))
        (DATA-OUT-TERMINAL SUM)
        (POWER-TERMINAL (Vdd GND))
        (CTL-IN-TERMINAL NIL)
        (CTL-OUT-TERMINAL NIL)
        (CTL-TERM-EDGE ((CARRYOUT TOP) (COUTN TOP)))
        (COMPLEMENT-OUT NIL)
        (DRIVING-CAP SMALL)
      )
    )
    (-
("sub"
    (PARAMETERS ((N N) (CS_TYPE "z")))
    (DEFAULTS (KEY UU/UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 264))
           (UU/US/XX ( 291 232))
           (UU/SU/XX ( 263 232))
           (UU/SS/XX ( 203 351 342 115))
           (US/UU/XX ( 209 266))
           (SU/UU/XX ( 226 274))
           (US/US/XS ( 372 287 193 0 0 259 313 277))
           (SU/SU/SX ( 247 239 216 0 0 337 209 219))
           (US/SU/XX ( 298 105 210 286))
           (SU/US/XX ( 327 157 223 309))
           (US/SS/XS ( 348 300 309 0 334 199 76 0 0 130 310 372 0
                343 235 154))
           (SU/SS/SX ( 158 323 294 145 355 0 91 0 0 226 0 424 178
                335 236 86))
           (SS/UU/XX ( 118 316 315 199))
           (SS/US/XS ( 313 151 95 0 0 342 290 216 403 341 285 0 0
                133 269 340))
           (SS/SU/SX ( 283 109 105 0 206 314 339 0 0 369 266 129 0
```

```
                231 139 250))
          (SS/SS/SS ( 0 190 273 21 302 0 124 0 363 171 0 0 4 0 0 0
                0 236 0 347 107 254 242 107 0 375 0 0 348 184
                0 0 0 0 461 405 0 0 305 0 51 218 318 71 307 0
                129 0 0 0 0 16 0 0 366 486 0 230 0 319 5 169
                152 19)))))
  (AREA (* 207 (+ 20 (* 64 N))))
  (HEIGHT 207)
  (WIDTH (+ 20 (* 64 N)))
  (DELAY (+ (/ 5 2) (/ 52 10) (* N (/ 3 4))))
  (RIPPLE-OFFSET (/ 5 2))
  (ONE-BIT-DELAY (/ 52 10))
  (RIPPLE-DELAY (* N (/ 3 4)))
      (PIPE-DEPTH 1)
  (DATA-IN-TERMINAL (A B))
  (DATA-OUT-TERMINAL SUM)
  (POWER-TERMINAL (Vdd GND))
  (CTL-IN-TERMINAL NIL)
  (CTL-OUT-TERMINAL NIL)
  (CTL-TERM-EDGE ((CIN BOTTOM) (COUT TOP) (COUTINV TOP)))
  (COMPLEMENT-OUT OUTINV)
  (DRIVING-CAP SMALL)))
  (negate
("sub"
  (PARAMETERS ((N N) (CS_TYPE "z")))
  (DEFAULTS (KEY UU/XX))
  (CAP (* C0 N))
      (CAP-COEFFS (LOOKUP KEY
        ((UU/XX (104))
         (US/XX (113 99))
         (SU/XX (116 97))
         (SS/XX (67 189 197 27)))))
  (AREA (* 207 (+ 20 (* 64 N))))
  (DELAY (+ (/ 5 2) (/ 52 10) (* N (/ 3 4))))
  (RIPPLE-OFFSET (/ 5 2))
  (ONE-BIT-DELAY (/ 52 10))
  (RIPPLE-DELAY (* N (/ 3 4)))
      (PIPE-DEPTH 1)
  (DATA-IN-TERMINAL B)
  (DATA-OUT-TERMINAL SUM)
  (POWER-TERMINAL (Vdd GND))
  (CTL-IN-TERMINAL ((CARRYIN VDD)))
  (CTL-OUT-TERMINAL NIL)
```

```
   (CTL-TERM-EDGE ((CIN BOTTOM) (COUT TOP) (COUTINV TOP)))
   (COMPLEMENT-OUT OUTINV)
   (DRIVING-CAP SMALL)))
   (^
("xor"
   (SDLNAME "logics")
   (VHDLNAME "logics")
   (CONSTRAINT  (== NrIN 2))
   (PARAMETERS ((N N) (TYPE "xor") (NUM_IN NrIN)))
   (DEFAULTS (KEY UU/UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 45))
           (UU/US/XX ( 40 49))
           (UU/SU/XX ( 48 39))
           (UU/SS/XX ( 27 70 53 28))
           (US/UU/XX ( 36 50))
           (SU/UU/XX ( 52 38))
           (US/US/XS ( 40 0 0 35 0 40 61 0))
           (SU/SU/SX ( 56 0 0 48 0 44 30 0))
           (US/SU/XX ( 40 34 57 44))
           (SU/US/XX ( 44 60 35 40))
           (US/SS/XS ( 25 0 0 55 53 0 0 18 0 28 88 0 0 53 35 0))
           (SU/SS/SX ( 25 0 82 0 0 62 0 34 0 28 0 61 42 0 18 0))
           (SS/UU/XX ( 30 71 43 31))
           (SS/US/XS ( 33 0 0 31 0 56 86 0 46 0 0 41 0 24 38 0))
           (SS/SU/SX ( 31 0 0 35 74 0 0 66 0 50 36 0 0 37 22 0))
           (SS/SS/SS ( 0 0 0 0 0 63 0 0 0 0 67 0 0 0 0 0 0 54 0 0 98
                  0 0 0 0 0 0 57 0 0 74 0 0 0 54 0 0 0 0 46 37 0
                  0 0 0 35 0 0 0 0 0 0 0 0 75 0 0 47 0 0 0 0 0 0)))))
   (AREA (* N 56 64))
   (HEIGHT 64)
   (WIDTH (* N 56))
   (DELAY (/ 5 2))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 5 2))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL O)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
```

```
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)))
   (&
("and2"
   (SDLNAME "logics")
   (VHDLNAME "logics")
   (CONSTRAINT  (== NrIN 2))
   (PARAMETERS ((N N) (TYPE "nandand") (NUM_IN NrIN)))
   (DEFAULTS (KEY UU/UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 36))
           (UU/US/XX ( 22 48))
           (UU/SU/XX ( 39 35))
           (UU/SS/XX ( 9 65 35 31))
           (US/UU/XX ( 25 45))
           (SU/UU/XX ( 38 36))
           (US/US/XS ( 13 0 37 0 33 0 0 59))
           (SU/SU/SX ( 50 0 24 0 28 0 0 42))
           (US/SU/XX ( 22 29 53 39))
           (SU/US/XX ( 19 55 27 43))
           (US/SS/XS ( 0 0 45 0 26 0 29 0 20 0 0 89 46 0 0 30))
           (SU/SS/SX ( 18 0 78 0 18 0 29 0 0 0 57 0 0 56 0 30))
           (SS/UU/XX ( 10 62 38 31))
           (SS/US/XS ( 0 0 22 0 38 0 0 85 26 0 50 0 29 0 0 35))
           (SS/SU/SX ( 22 0 0 0 69 0 51 0 22 0 0 56 33 0 0 26))
           (SS/SS/SS ( 0 0 0 0 45 0 0 0 0 0 0 0 0 0 0 0 38 0 0 0 0
                       108 0 0 38 0 0 0 0 63 0 0 0 0 0 0 45 0 0 0 0 0
                       56 0 0 0 56 0 0 0 0 0 0 69 0 0 0 0 56 0 0 0 0
    0)))))
   (AREA (* N 51 64))
   (DELAY (/ 28 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 28 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL OB)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT OA)
   (DRIVING-CAP SMALL))
```

```
("and3"
   (SDLNAME "logics")
   (VHDLNAME "logics")
   (CONSTRAINT (== NrIN 3))
   (PARAMETERS ((N N) (TYPE "nandand") (NUM_IN NrIN)))
   (CAP (* C0 N))
   (CAP-COEFFS (50))
   (AREA (* N 59 64))
   (DELAY (/ 29 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 29 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (A B C))
   (DATA-OUT-TERMINAL OB)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT OA)
   (DRIVING-CAP SMALL)))
   (|
("or2"
   (SDLNAME "logics")
   (VHDLNAME "logics")
   (CONSTRAINT (== NrIN 2))
   (PARAMETERS ((N N) (TYPE "noror") (NUM_IN NrIN)))
   (DEFAULTS (KEY UU/UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/UU/XX ( 40))
           (UU/US/XX ( 35 44))
           (UU/SU/XX ( 54 25))
           (UU/SS/XX ( 31 76 38 13))
           (US/UU/XX ( 33 43))
           (SU/UU/XX ( 55 27))
           (US/US/XS ( 38 0 0 30 0 32 0 58))
           (SU/SU/SX ( 72 0 0 38 0 39 0 12))
           (US/SU/XX ( 44 25 63 25))
           (SU/US/XX ( 44 65 26 26))
           (US/SS/XS ( 24 0 0 64 50 0 0 0 0 40 0 92 0 28 0 25))
           (SU/SS/SX ( 36 0 103 0 0 51 0 24 0 26 0 52 0 25 0 0))
           (SS/UU/XX ( 29 78 37 12))
           (SS/US/XS ( 24 0 0 37 0 65 0 90 50 0 0 29 0 0 0 26))
```

```
            (SS/SU/SX ( 37 0 0 26 99 0 0 53 0 51 0 26 0 25 0 0))
            (SS/SS/SS ( 0 0 0 0 0 75 0 0 0 0 49 0 0 0 0 0 0 76 0 0 0
                  128 0 0 0 0 0 53 0 0 0 53 0 0 50 0 0 0 0 53 0
                  0 49 0 0 0 0 0 0 0 0 0 0 0 0 0 52 0 0 0 0 0 0 0
                  0)))))
   (AREA (* N 64 53))
   (DELAY (/ 29 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 29 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (A B))
   (DATA-OUT-TERMINAL OB)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT OA)
   (DRIVING-CAP SMALL))
("or3"
   (SDLNAME "logics")
   (VHDLNAME "logics")
   (CONSTRAINT (== NrIN 3))
   (PARAMETERS ((N N) (TYPE "noror") (NUM_IN NrIN)))
   (CAP (* C0 N))
   (CAP-COEFFS (50))
   (AREA (* N 64 70))
   (DELAY (/ 31 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 31 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL (A B C))
   (DATA-OUT-TERMINAL OB)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT OA)
   (DRIVING-CAP SMALL)))
   (!
("invs"
   (SDLNAME "inv")
   (VHDLNAME "inv")
   (PARAMETERS (N N) (SIZE "s"))
```

```
   (DEFAULTS (KEY UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (10))
           (US/XX (10 9))
           (SU/XX (10 11))
           (SS/XX (0 20 21 0)))))
   (AREA (* N 40 64))
   (DELAY (/ 17 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 17 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL))
("invm"
   (SDLNAME "inv")
   (VHDLNAME "inv")
   (PARAMETERS ((N N) (SIZE "m")))
   (DEFAULTS (KEY UU/XX))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (24))
           (US/XX (18 28))
           (SU/XX (32 20))
           (SS/XX (0 60 39 0)))))
   (AREA (* N 40 64))
   (DELAY (/ 17 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 17 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
```

```
   (DRIVING-CAP MEDIUM)))
   (buffer
("bufs"
   (SDLNAME "buf")
   (VHDLNAME "buf")
   (DEFAULTS (KEY UU/XX))
   (PARAMETERS ((N N) (SIZE "s")))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (21))
           (US/XX (18 21))
           (SU/XX (23 20))
           (SS/XX (0 44 39 0)))))
   (AREA (* N 64 40))
   (HEIGHT 40)
   (WIDTH (* N 64))
   (DELAY (/ 18 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 18 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL))
("bufm"
   (SDLNAME "buf")
   (VHDLNAME "buf")
   (DEFAULTS (KEY UU/XX))
   (PARAMETERS ((N N) (SIZE "m")))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (30))
           (US/XX (32 25))
           (SU/XX (28 35))
           (SS/XX (0 54 68 0)))))
        (AREA (* N 64 40))
   (AREA (* N 64 40))
   (HEIGHT 40)
   (WIDTH (* N 64))
```

```
   (DELAY (/ 22 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 22 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP MEDIUM))
("bufl"
   (SDLNAME "buf")
   (VHDLNAME "buf")
   (DEFAULTS (KEY UU/XX))
   (PARAMETERS ((N N) (SIZE "l")))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (47))
           (US/XX (54 35))
           (SU/XX (39 59))
           (SS/XX (0 74 113 0)))))
   (AREA (* N 64 40))
   (HEIGHT 40)
   (WIDTH (* N 64))
   (DELAY (/ 27 10))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY (/ 27 10))
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP LARGE))
("bufx"
   (SDLNAME "buf")
   (VHDLNAME "buf")
   (DEFAULTS (KEY UU/XX))
   (PARAMETERS ((N N) (SIZE "x")))
```

```
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (80))
           (US/XX (95 58))
           (SU/XX (64 104))
           (SS/XX (0 122 199 0)))))
    (AREA (* N 64 48))
    (HEIGHT 48)
    (WIDTH (* N 64))
    (DELAY (/ 29 10))
    (RIPPLE-OFFSET 0)
    (ONE-BIT-DELAY (/ 29 10))
    (RIPPLE-DELAY 0)
    (DATA-IN-TERMINAL IN)
    (DATA-OUT-TERMINAL OUT)
    (POWER-TERMINAL (Vdd GND))
    (CTL-IN-TERMINAL NIL)
    (CTL-OUT-TERMINAL NIL)
    (CTL-TERM-EDGE NIL)
    (COMPLEMENT-OUT NIL)
    (DRIVING-CAP XLARGE))
  ("bufh"
    (SDLNAME "buf")
    (VHDLNAME "buf")
    (DEFAULTS (KEY UU/XX))
    (PARAMETERS ((N N) (SIZE "h")))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (149))
           (US/XX (178 104))
           (SU/XX (116 194))
           (SS/XX (0 221 371 0)))))
    (AREA (* N 64 69))
    (HEIGHT 69)
    (WIDTH (* N 64))
    (DELAY (/ 33 10))
    (RIPPLE-OFFSET 0)
    (ONE-BIT-DELAY (/ 33 10))
    (RIPPLE-DELAY 0)
    (DATA-IN-TERMINAL IN)
    (DATA-OUT-TERMINAL OUT)
    (POWER-TERMINAL (Vdd GND))
    (CTL-IN-TERMINAL NIL)
```

```
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP HUGE)))
   (>>
("shr"
   (SDLNAME "sh")
   (VHDLNAME "sh")
   (CONSTRAINT (&& (>= M 0) (<= M 31)))
   (DEFAULTS (SHIFT 0) (MAX 31) (MIN 0) (KEY UU/XX))
   (PARAMETERS ((N N) (ENDTYPE "arithmetic") (STYPE "right")
(MAXSBY (- (expt 2 (ceiling (log (+ M 1) 2))) 1))))
         (CAP (* (/ N 1000)
   (+ C0
     (* (ceiling (log (+ M 1) 2))
 (+ (/ C1 N_TOT) C2 (* C3 N_TOT) (* C4 M) (* C5 SHIFT))
   ))))
         (CAP-COEFFS (LOOKUP KEY
            ((UU/XX ( (28722 43859 12296 62 244 -183) ))
             (US/XX ( (28916 47230 -3343 39 5 -73)
 (49760 -160179 38790 -182 1072 279) ))
 (SU/XX ( (29515 206333 19996 173 528 -279)
 (31079 -29660 3711 -43 -5 75) ))
 (SS/XX ( (0 0 0 0 0 0)
 (79266 46257 58788 -9 1600 0)
 (59791 24275 12 0 0 0)
 (0 0 0 0 0 0) )))))
   (AREA (* (+ 20 (* 64 N) (* 11 (ceiling (/ N 8)))) (- (* 109 (ceiling (log (+ M 1) 2)))
64)))
   (HEIGHT (- (* 109 (ceiling (log (+ M 1) 2))) 64))
   (WIDTH (+ 20 (* 64 N) (* 11 (ceiling (/ N 8)))))
   (DELAY (+ (/ 37 10) (* M (/ 3 10))))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
         (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL
(((EXPAND SHIFT[%d] (- (ceiling (log (+ M 1) 2)) 1) 0) SHIFT)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE
```

```
(((EXPAND SHIFT[%d] (- (ceiling (log (+ M 1) 2)) 1) 0) BOTTOM)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)))
   (<<
("shl"
   (SDLNAME "sh")
   (VHDLNAME "sh")
   (CONSTRAINT (&& (>= M 0) (<= M 31)))
   (DEFAULTS (SHIFT 0) (MAX 31) (MIN 0) (KEY UU/XX))
   (PARAMETERS ((N N) (ENDTYPE "arithmetic") (STYPE "left")
(MAXSBY (- (expt 2 (ceiling (log (+ M 1) 2))) 1))))
         (CAP (* (/ N 1000)
   (+ C0
     (* (ceiling (log (+ M 1) 2))
 (+ (/ C1 N_TOT) C2 (* C3 N_TOT) (* C4 M) (* C5 SHIFT))
   ))))
         (CAP-COEFFS (LOOKUP KEY
            ((UU/XX ( (29049 -113660 19883 -6 241 -200) ))
             (US/XX ( (34124 77815 -2916 -13 171 124)
 (35430 -62726 33235 36 622 -146) ))
 (SU/XX ( (32619 -69119 33361 -2 573 -113)
 (35075 139991 -8382 64 156 115) ))
 (SS/XX ( (5871 64108 -610 -45 166 204)
 (62518 -133309 62229 142 1046 -458)
 (63551 -64984 5783 -118 166 50)
 (4820 170171 -4412 -13 139 171) )))))
   (AREA (* (+ 20 (* 64 N) (* 11 (ceiling (/ N 8)))) (- (* 109 (ceiling (log (+ M 1) 2)))
64)))
   (HEIGHT (- (* 109 (ceiling (log (+ M 1) 2))) 64))
   (WIDTH (+ 20 (* 64 N) (* 11 (ceiling (/ N 8)))))
   (DELAY (+ (/ 37 10) (* M (/ 3 10))))
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
         (PIPE-DEPTH 1)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL
(((EXPAND SHIFT[%d] (- (ceiling (log (+ M 1) 2)) 1) 0) SHIFT)))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE
(((EXPAND SHIFT[%d] (- (ceiling (log (+ M 1) 2)) 1) 0) BOTTOM)))
```

```
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL)))
   (@
("transfer"
   (PARAMETERS (N N))
        (CAP 0)
        (CAP-COEFFS NIL)
   (AREA 0)
   (HEIGHT 0)
   (WIDTH 0)
   (DELAY 0)
   (RIPPLE-DIR NIL)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
   (TIMING-CONSTRAINT NIL)
   (DATA-IN-TERMINAL IN1)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)))
   ("#"
("transfer"
   (PARAMETERS (N N))
        (CAP 0)
        (CAP-COEFFS NIL)
   (AREA 0)
   (HEIGHT 0)
   (WIDTH 0)
   (DELAY 0)
   (RIPPLE-DIR NIL)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
   (TIMING-CONSTRAINT NIL)
   (DATA-IN-TERMINAL IN1)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
```

```
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)))
   (=
("transfer"
   (PARAMETERS (N N))
        (CAP 0)
        (CAP-COEFFS NIL)
   (AREA 0)
   (HEIGHT 0)
   (WIDTH 0)
   (DELAY 0)
   (RIPPLE-DIR NIL)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
   (TIMING-CONSTRAINT NIL)
   (DATA-IN-TERMINAL IN1)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)))
   (input
("ioUnit"
   (PARAMETERS (N N))
        (CAP 0)
        (CAP-COEFFS NIL)
   (AREA 0)
   (HEIGHT 0)
   (WIDTH 0)
   (DELAY 0)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 0)
   (RIPPLE-DELAY 0)
   (TIMING-CONSTRAINT NIL)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL NIL)
   (CTL-OUT-TERMINAL NIL)
```

```
   (CTL-TERM-EDGE NIL)
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP NO)))
   (trist-buf
("trist_buffer"
   (SDLNAME "trist_buffer_np")
   (DEFAULTS (KEY UU/XX))
   (PARAMETERS (N N))
        (CAP (* C0 N))
        (CAP-COEFFS (LOOKUP KEY
          ((UU/XX (27))
           (US/XX (26 26))
           (SU/XX (29 28))
           (SS/XX (0 55 54 0)))))
   (AREA (* N 38 50))
   (HEIGHT 50)
   (WIDTH (* N 38))
   (DELAY 1)
   (RIPPLE-OFFSET 0)
   (ONE-BIT-DELAY 1)
   (RIPPLE-DELAY 0)
   (DATA-IN-TERMINAL IN)
   (DATA-OUT-TERMINAL OUT)
   (POWER-TERMINAL (Vdd GND))
   (CTL-IN-TERMINAL ((CNTL OEN) (CNTLINV (not OEN))))
   (CTL-OUT-TERMINAL NIL)
   (CTL-TERM-EDGE ((CNTL TOP) (CNTLINV TOP)))
   (COMPLEMENT-OUT NIL)
   (DRIVING-CAP SMALL))))
```

Appendix B

Library Routines

Int
InputHardwareDatabase(DatabasePathName)
*char *DatabasePathName;*
Reads specified rb-dp into memory.

int
CleanUpDatabase()
Removes current rb-dp from memory.

ListPointer
HwExpandInclude(Entry)
ListPointer Entry;
Expands any included file in rb-dp.

char
**HwGetFunctionSource(Function)*
*char *Function;*
Returns hardware source type of a function, i.e., datapath, standard logic or array.

int
CheckHwProperty(Property)
*char *Property;*
Checks whether or not a hardware property is valid in the library representation.

int
HwIsFunctionInLibrary(Function)

*char *Function*
Checks whether or not a function exists in the library.

int
HwIsCellInLibrary(Cell)
*char *Cell;*
Checks whether or not a cell exists in the library.

ListPointer
HwGetParameterEntry(ParameterList, Name)
ListPointer ParameterList;
*char *Name;*
Returns list structure entry in *ParameterList* with label *Name.*

int
IsParameterValueSet(Label, ParameterList)
*char *Label;*
ListPointer ParameterList;
Checks whether or not a parameter is defined in the *parameterList.*

ListPointer
UpdateParameterList(CellPtr, ParameterList)
ListPointer ParameterList;
Updates the parameters with any default values specified by *CellPtr.*

int
HwConstraintMet(CellPtr, ParameterList)
ListPointer CellPtr;
ListPointer ParameterList;
Checks if the parameters specified by *ParameterList* satisfy the constraints specified by *CellPtr.*

ListPointer
HwGetCellList(Function)
*char *Function;*
Returns a list of cells included in *Function.*

ListPointer
HwGetHwParameters(Cell, ParameterList)
ListPointer Cell, ParameterList;

Returns the hardware parameter of *Cell* in list structure format.

*char **
HwGetCheapestCell(Function, ParameterList)
*char *Function;*
ListPointer ParameterList;
Returns the name of the cell with the least area cost in a specific *Function.*

pointer
HwGetCapSize(Cell)
ListPointer Cell;
Returns the loading capacity of the *Cell.*

ListPointer
HwGetCellByCapSize(Function, CapSize, ParameterList)
*char *Function;*
*char *CapSize;*
ListPointer ParameterList;
Returns the cell with a specific loading capacity in a *Function.*

int
IsCellShared(Function1, Function2, CellName)
*char *Function1;*
*char *Function2;*
*char **CellName;*
Checks if there exists a shared cell between *Function1* and *Function2.*

int
IsAluShared(Function1, Function2, CellName)
*char *Function1;*
*char *Function2;*
*char **CellName;*
Checks if there exists a shared ALU unit between *Function1* and *Function2.*

*char **
FindSharedCell(Function1, Function2, ParameterList)
*char *Function1;*
*char *Function2;*
ListPointer ParameterList;
Returns the cheapest shared cell in both *Function1* and *Function2.*

ListPointer
HwGetCheapestCellPointer(Function, ParameterList)

*char *Function;*
ListPointer ParameterList;
Returns the list structure pointer pointing to the cheapest cell in a *Function.*

ListPointer
HwGetFastestCellPointer(Function, ParameterList)
*char *Function;*
ListPointer ParameterList;
Returns the list structure pointer pointing to the fastest cell in a *Function.*

int
HwGetMinDelay(Function, ParameterList)
*char *Function;*
ListPointer ParameterList;
Returns the minimum delay time in a *Function.*

int
HwGetMaxDelay(Function, ParameterList)
*char *Function;*
ListPointer ParameterList;
Returns the maximum delay time in a *Function.*

*char **
HwGetCellWithMaxDelay(Function, ParameterList, MaxDelay, Delay)
*char *Function;*
ListPointer ParameterList;
int MaxDelay;
*int *Delay;*
Returns the cell name with the maximum delay time that is less than *Delay.*

int
HwGetRippleDelay(Function, Cell, ParameterList)
*char *Function;*
*char *Cell;*
ListPointer ParameterList;
Returns the ripple delay time.

int
HwGetBitDelay(Function, Cell, ParameterList)
*char *Function;*
*char *Cell;*
ListPointer ParameterList;
Returns the one-bit delay time.

*char **
HwGetCellWithMinDelay(Function, ParameterList, MinDelay, Delay)
*char *Function;*
ListPointer ParameterList;
*int *MinDelay;*
*int *Delay;*
Returns the name of the cell with the minimum delay time which is less then*Delay.*

int
HwGetDelay(Function, Cell, ParameterList)
*char *Function;*
*char *Cell;*
ListPointer ParameterList;
Returns the delay time of a specific *Cell.*

int
HwGetArea(Function, Cell, ParameterList)
*char *Function;*
*char *Cell;*
ListPointer ParameterList;
Returns the area cost of a specific *Cell.*

int
HwGetCap(Function, Cell, ParameterList)
*char *Function;*
*char *Cell;*
ListPointer ParameterList;
Returns the active capacitance of a specific *Cell.*

ListPointer
HwExpandList(List, ParameterList)
ListPointer List, ParameterList;
Expands any "EXPAND" list function in a *list.*

ListPointer
HwGetInData(Function, CellName, ParameterList)
*char *Function;*
*char *CellName;*
ListPointer ParameterList;
Returns the names of the input data signals.

ListPointer

HwGetOutData(Function, CellName, ParameterList)
*char *Function;*
*char *CellName;*
ListPointer ParameterList;
Returns the names of the output data signals.

char
**HwGetSupply(Function, Cell, SupplyType)*
*char *Function;*
*char *Cell;*
int SupplyType;
Returns the signal names for supply(*SupplyType = 1*) or ground(*SupplyType =0*).

ListPointer
HwGetInControl(Cell, ParameterList)
ListPointer Cell, ParameterList;
Returns the names of the input control signals.

ListPointer
HwGetOutControl(Cell, ParameterList)
ListPointer Cell, ParameterList;
Returns the names of the output control signals.

ListPointer
HwGetControlEdge(Cell, ParameterList)
ListPointer Cell, ParameterList;
Returns the edge sides of the control signals.

char
**HwGetSdlName(Function, Cell)*
*char *Function;*
*char *Cell;*
Returns the SDL file name to construct *Cell*.

char
**HwGetVHDLName(Function, Cell)*
*char *Function;*
*char *Cell;*
Returns the VHDL file name to construct *Cell*.

int
HwFindMaxShiftRange(Function, Max)
*char *Function;*

*int *Max;*
Returns the maximum shifter range.

int
InputTechnology(DatabasePathName)
*char *DatabasePathName;*
Reads the technology library file into memory.

int
CleanUpTechnology()
Removes the technology library from memory.

int
GetCharTechnologyParameter(Keyword, Value)
*char *Keyword;*
*char **Value;*
Returns character string value of parameter specified by *Keyword.*

int
GetDoubleTechnologyParameter(Keyword, Value)
*char *Keyword;*
*double *Value;*
Returns floating point value of parameter specified by *Keyword.*

int
GetIntegerTechnologyParameter(Keyword, Value)
*char *Keyword;*
*int *Value;*
Returns integer value of parameter specified by *Keyword.*

int
GetListTechnologyParameter(Keyword, List)
*char *Keyword;*
*ListPointer *List;*
Returns list structure value of parameter specified by *Keyword.*

*char **
GetSizeUnits()
Returns the unit of length.

double
GetSizeScaleFactor()
Returns the scaling factor of length.

double
GetDPAreaScaleFactor()
Returns the scaling factor of area cost.

*char **
GetTimeUnits()
Returns the unit of time.

double
GetTimeScaleFactor()
Returns the scaling factor of time.

*char **
GetCapUnits()
Returns the unit of capacitance.

double
GetCapScaleFactor()
Returns the scaling factor of capacitance.

int
TechGetNumberOfClocks()
Returns the number of clocks implemented in the library.

double
TechGetSupply()
Returns the default voltage supply value of the library.

ListPointer
TechGetDelayScale()
Returns the scaling factor of delay time.

int
TechGetStateTransitionEdge()
Returns the clock edge for state transition.

References

[Ben93] O. Bentz, *A Hardware Mapper for the HYPER High Level Synthesis System*, M.S. Report, EECS Department, U.C. Berkeley, 1993.

[Bur94] T. Burd, *Low Power CMOS Library Design Methodology*, M.S. Report, EECS Department, U.C. Berkeley, 1994.

[Cha92] A. Chandrakasan, S. Sheng, R. Brodersen, "Low-power CMOS Digital Design", *IEEE Journal of Solid State Circuits*, pp. 473-484, April 1992.

[Cha93] A. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen, "HYPER-LP: A System for Power Minimization Using Architectural Transormations", , , 1993.

[Chu92] C. Chu, *Hardware Mapping and Module Selection in the HYPER Synthesis System*, Ph. D. Report, EECS Department, U.C. Berkeley, 1992.

[Lan94] P. Landman, *SPA: A Stochastic Tool for Architectural Power Analysis*, EECS Department, U.C. Berkeley, 1994

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak: "Fast Prototyping of Data Path Intensive Architecture," *IEEE Design and Test*, vol. 8, no. 2, pp. 40-51, 1991.

[Rab94] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*, EECS241 Class Reader, EECS Department, U.C. Berkeley, 1994.

[Lager91] "*Volume 2: Lager Tool Set*", U.C. Berkeley, U.C. Los Angeles, Mississippi State University, Institute for Technology Development, June 1991.