THE INFLUENCE OF HARDWARE MAPPING ON HIGH-LEVEL SYNTHESIS

by

David P. Schultz

Memorandum No. UCB/ERL M92/54

28 May 1992

THE INFLUENCE OF HARDWARE MAPPING ON HIGH-LEVEL SYNTHESIS

by

David P. Schultz

Memorandum No. UCB/ERL M92/54

28 May 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE INFLUENCE OF HARDWARE MAPPING ON HIGH-LEVEL SYNTHESIS

by

David P. Schultz

Memorandum No. UCB/ERL M92/54

28 May 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

1 Introduction

1.1 Motivation

High-level synthesis in IC design is a process by which a behavioral or functional description of a system is used to generate a lower level description such as an architectural description or netlist. The low-level description can in turn be used as the input to a silicon compiler to produce layout. When the high-level synthesis process is not influenced by implementation details at the low-level inefficient use of resources such as area and time may result. By using low-level information to influence high-level transformations it is possible to improve resource usage. Possibly more important is the opportunity to see the influence of high-level decisions on the final result. Since area is a major factor influencing the cost of an integrated circuit it is important that the designer have a good idea of the implementation area of a given design. For this reason it is desirable to have an accurate area estimate early in the design process.

The Hyper system [1] is a high-level synthesis tool targeted for high-performance computation-intensive systems such as are often found in digital signal processing, speech processing, or video processing. Hyper takes a description of an algorithm in the control/data-flowgraph language Silage [2] and ultimately produces a structural description which can then be used as input for the LagerIV silicon compiler to produce library-based custom layout.

This report describes the characterization of the hardware mapper, introduces a new estimation tool to be integrated into the Hyper system, and

discusses a new hardware model which may be used to improve the efficiency of area usage in future Hyper designs.

1.2 Organization of the report

An overview of the Hyper system is given in chapter 2. Chapter 3 discusses the motivations for using multiple hardware libraries and the extension of the hardware mapper to use them. Chapter 4 describes the collection of data from layouts generated by Hyper, the analysis of this data, and the area estimation formula and its basis. In chapter 5 future directions and a new hardware model are discussed.

1 Introduction

1.1 Motivation

High-level synthesis in IC design is a process by which a behavioral or functional description of a system is used to generate a lower level description such as an architectural description or netlist. The low-level description can in turn be used as the input to a silicon compiler to produce layout. When the high-level synthesis process is not influenced by implementation details at the low-level inefficient use of resources such as area and time may result. By using low-level information to influence high-level transformations it is possible to improve resource usage. Possibly more important is the opportunity to see the influence of high-level decisions on the final result. Since area is a major factor influencing the cost of an integrated circuit it is important that the designer have a good idea of the implementation area of a given design. For this reason it is desirable to have an accurate area estimate early in the design process.

The Hyper system [1] is a high-level synthesis tool targeted for high-performance computation-intensive systems such as are often found in digital signal processing, speech processing, or video processing. Hyper takes a description of an algorithm in the control/data-flowgraph language Silage [2] and ultimately produces a structural description which can then be used as input for the LagerIV silicon compiler to produce library-based custom layout.

This report describes the characterization of the hardware mapper, introduces a new estimation tool to be integrated into the Hyper system, and

discusses a new hardware model which may be used to improve the efficiency of area usage in future Hyper designs.

1.2 Organization of the report

An overview of the Hyper system is given in chapter 2. Chapter 3 discusses the motivations for using multiple hardware libraries and the extension of the hardware mapper to use them. Chapter 4 describes the collection of data from layouts generated by Hyper, the analysis of this data, and the area estimation formula and its basis. In chapter 5 future directions and a new hardware model are discussed.

The Hyper System 2

2.1 Overview

Hyper is an interactive menu-driven design environment for producing layout for high-performance digital systems. It consists of a number of software modules linked under a common X-windows based management tool called Xhyper. Figure 1 shows the Xhyper window while figure 2 illustrates how the various modules are interlinked.

The input to Hyper is a Silage description of a digital system. Silage is an applicative signal-flowgraph language that is well suited for describing digital systems that have little explicit control. Figure 3 shows an example of a Silage system description, in this case an IIR bandpass filter.

The Silage description is parsed and translated into an ascii control/ flowgraph description. The flowgraph description is a textual representation of the nodes and edges of the system graph. This flowgraph can then be manipulated or transformed by Hyper to achieve the desired results. The final output of Hyper is a structural description of a digital system consisting of netlists which interconnect blocks of library layout cells.

The Hyper modules are: parsing, selection, estimation, transformation, scheduling, and hardware mapping. Essentially each module is a step in the design process. During parsing the Silage description is translated into a control/ data flowgraph which consists of control and data edges connecting operator nodes.

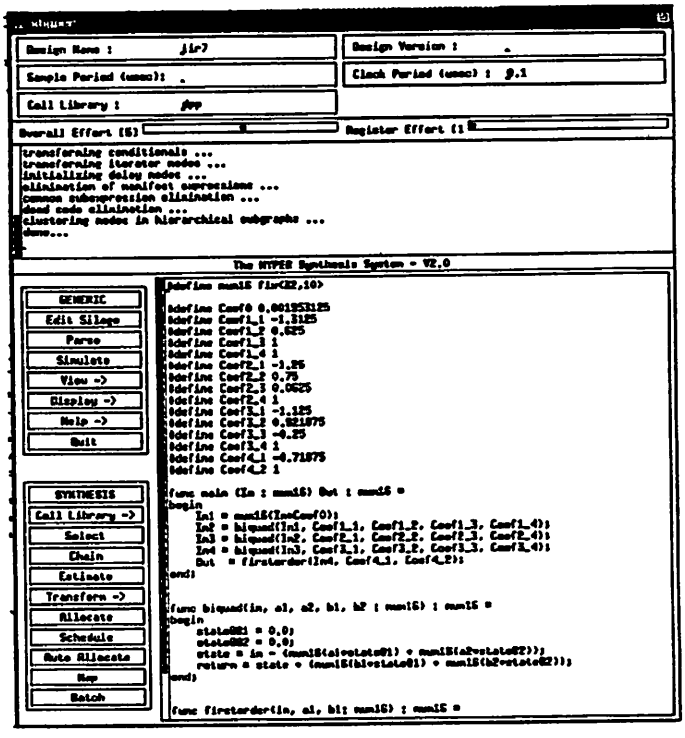

FIGURE 1. The Xhyper window.

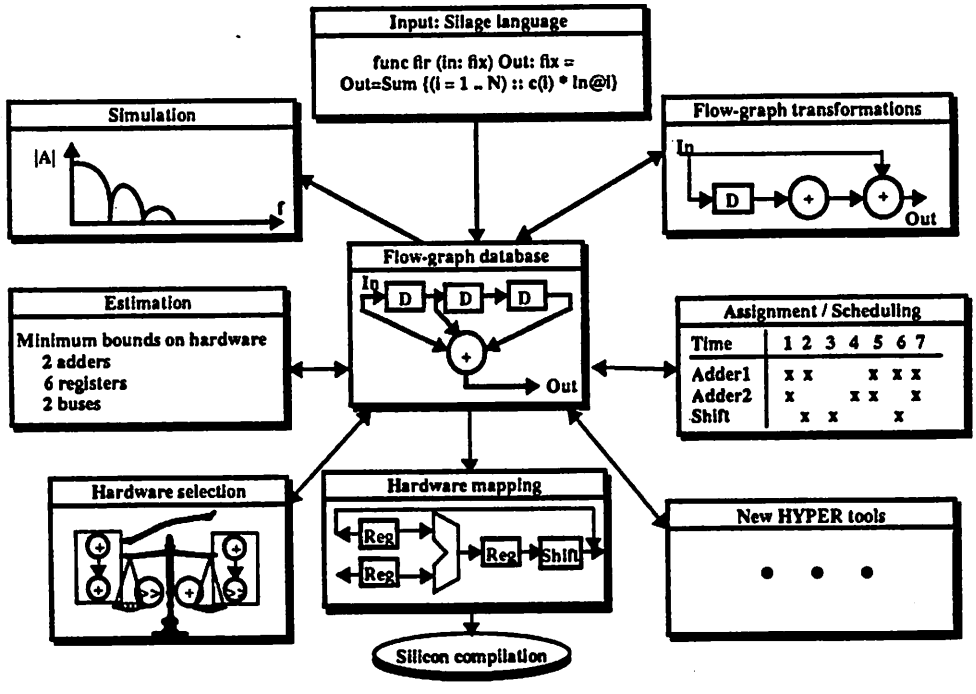

FIGURE 2. The Hyper modules.

The selection step assigns a unit from the hardware library to each type of node in the flowgraph. The hardware library contains a number of different implementations of certain blocks in order to trade off performance and area.

The transformation step consists of algorithmic transformations which can be performed on the flowgraph in order to achieve some specific result. Some possible transformations include expansion of fixed-coefficient multiplications into a series of adds and shifts, retiming for either critical path reduction or hardware reduction, loop unrolling, or pipelining.

The estimation step provides the designer with estimated resource requirements. The resources estimated are: the number of cycles in the critical path, the number and type of execution units (EXUs), the number of registers associated with each EXU, and the number of buses connecting the EXUs to each other.

The assignment/scheduling step assigns each flowgraph node to a specific hardware unit at a specific time. After this step the number and type of each EXU and the number of registers is known. Prior to scheduling the clock and sampling rates must be specified in order to define the number of cycles available to the system.

The hardware mapping step maps the flowgraph onto the selected hardware units by generating the interconnect information for the execution units, placing multiplexers or tri-state buffers where needed, defining the finite state machine and control block, and then partitioning the datapath. After this step the system is completely defined. The output of the hardware mapper is a group of hardware description files suitable for the LagerIV silicon compiler. These files consist of datapath descriptions in the SDL (Structure Description Language) format, and control logic specifications in BDS format. An SDL file contains a series of instance declarations which instantiate library cells followed by net declarations which describe the connections between the cells. A BDS file consists of logical equations defining states and outputs.

```
/* bpiir3.sil - 6th order IIR bandpass filter */

#define num16 fix<8,2>
#define Coef0 0.015625

#define Coefa1_1 -1.9377627
#define Coefa1_2 1
#define Coefb1_1 -1.9085335
#define Coefb1_2 0.93137014

#define Coefa2_1 -1.9910688
#define Coefa2_2 1
#define Coefb2_1 -1.9320831
#define Coefb2_2 0.96581414

#define Coefa3_2 -1
#define Coefb3_1 -1.9601845
#define Coefb3_2 0.97627349


func main (In : num16) Out : num16 =
begin
 In0 = num16(In*Coef0);
 In1 = biquad(In0, Coefa1_1, Coefa1_2, Coefb1_1, Coefb1_2);
 In2 = biquad(In1, Coefa2_1, Coefa2_2, Coefb2_1, Coefb2_2);
 Out = biquad0(In2, Coefa3_2, Coefb3_1, Coefb3_2);
end;

func biquad(in, a1, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a1*state@1) + num16(a2*state@2);
end;

func biquad0(in, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a2*state@2);
end;
```

FIGURE 3. Silage description of an IIR bandpass filter.

2.2 Hardware Model

The hardware model or architecture used in the Hyper system maps each operation onto an execution block consisting of an EXU with its input(s) connected to a register file or group of register files. An EXU is a functional unit such as an adder, subtracter, or shifter and a register file is simply a block of one or more register with a common input bus. In the case of a fully pipelined design the register file would consist of a single register while in a more multiplexed design the register files could be of arbitrary size. The output of each EXU is connected to destination register files through dedicated data buses controlled by tri-state buffers, one per fanout. Registers can also broadcast their contents to other register files through tri-state buffers. Control is provided using a mixture of global and local control units. Global control is provided by an FSM implemented as a PLA. Local control blocks consisting of address

decoders for the register files and inverters to provide complementary signals are made up of standard cells. Each local control block is associated with a datapath partition and is tiled together to approximately pitch-match with the datapath partition. Figure 4 illustrates this model.

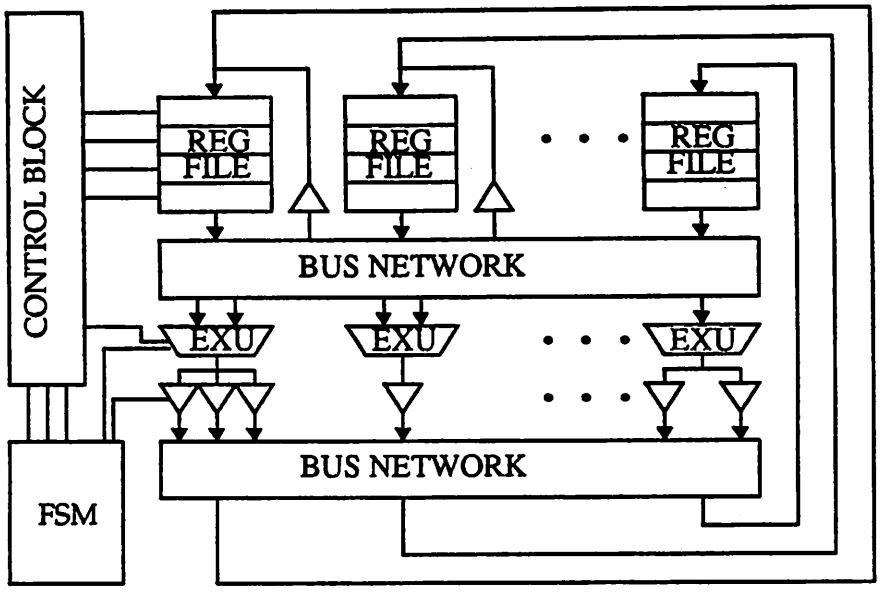

FIGURE 4. Datapath hardware model.

2.3 Layout Style

The Hyper system uses a datapath oriented layout style, i.e., all the functional blocks are tiled together to form a single row. In most cases this would result in a layout with a large aspect ratio, a "ruler" shape. Since this is not an efficient use of area the datapath and the local control block are partitioned into smaller pieces. The number of partitions can be chosen by the user or by the hardware mapper.

Once the mapper has generated the sdl files Hyper is finished with the project and the LagerIV design manager DMoct is used to call the necessary layout generation tools. The final step is to use the interactive floorplanner Flint to place and route the blocks that make up the chip core. Flint allows the user to place each block in any position and orientation desired, define wiring channels, and route buses through the defined channels. The detailed routing is then performed by Flint and the layout stored in OCT and/or Magic formats. Figure 5 shows the Flint window with a placed and routed example. This example is the

IIR bandpass filter described earlier. The two largest blocks are datapath partitions and the two smaller blocks adjacent to them are local control blocks. The remaining two blocks are the FSM and the clock buffers. Figure 6 shows the finished layout.

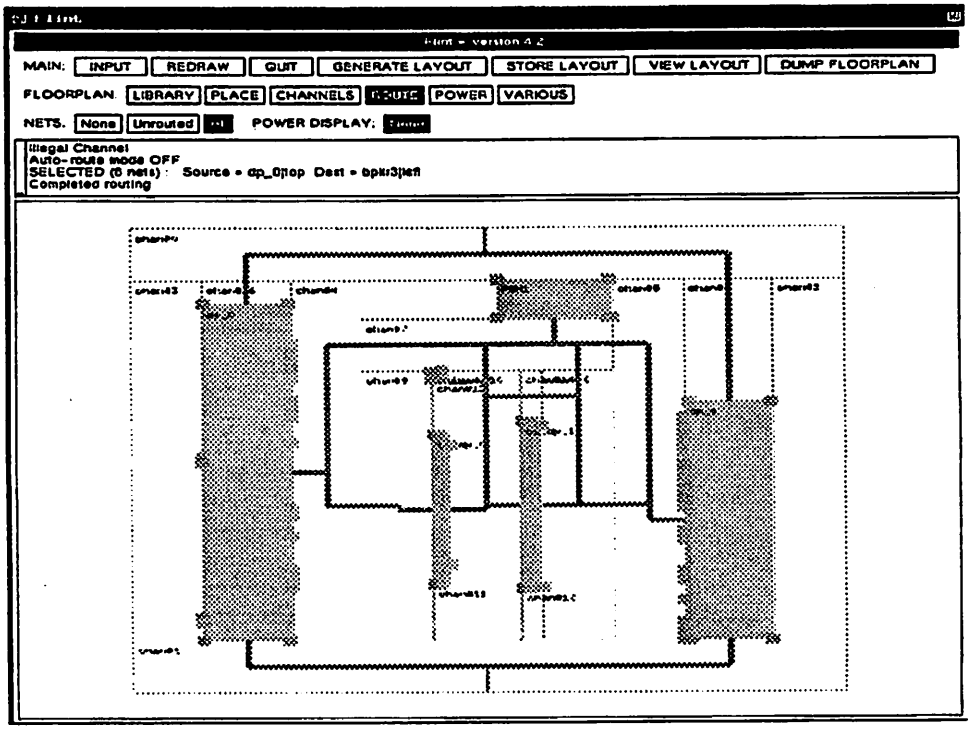

FIGURE 5. Flint window showing a placed and routed example.

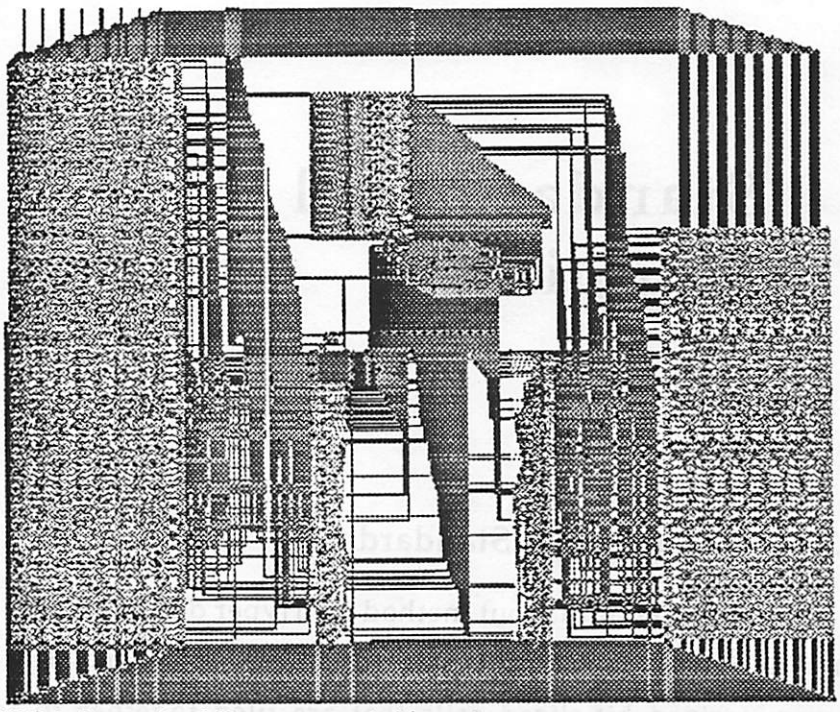

FIGURE 6. Example of Hyper generated layout.

Standard Cell Mapping

3

3.1 Datapath vs. Standard Cell

The default layout method for Hyper designs uses a library of cells which are designed for a datapath oriented floorplan. The library consists of parameterized bit-sliced cells that are tiled together to form datapath blocks. These blocks are then stacked together into a datapath. The datapath is usually partitioned into two or more smaller datapaths in order to avoid chip cores with large aspect ratios, i.e., the "ruler" effect. The datapath blocks are connected together using feedthroughs in the cells or, if there are not enough feedthroughs, routing cells are inserted between the bit-slices of the blocks. Figure 7 shows the floorplan of a general datapath. The datapaths are connected together using long data buses in two metal layers. In designs where resources are shared there is often a great deal of interconnection between blocks in the datapath and between datapath partitions. This results in a large percentage of area devoted to routing. In fact, as will be discussed in chapter 5, the area devoted to routing and white space accounts for 80% or more of the total area of the benchmarks studied.

3.2 Motivation

It was thought that the area consumed by routing might be reduced if the rigid datapath structure was abandoned in favor of a more flexible cell placement and routing style. A method using standard cells rather than parameterized macrocells might offer this kind of flexibility. The methodology for a standard cell design would be the same as for a datapath design except that instead of

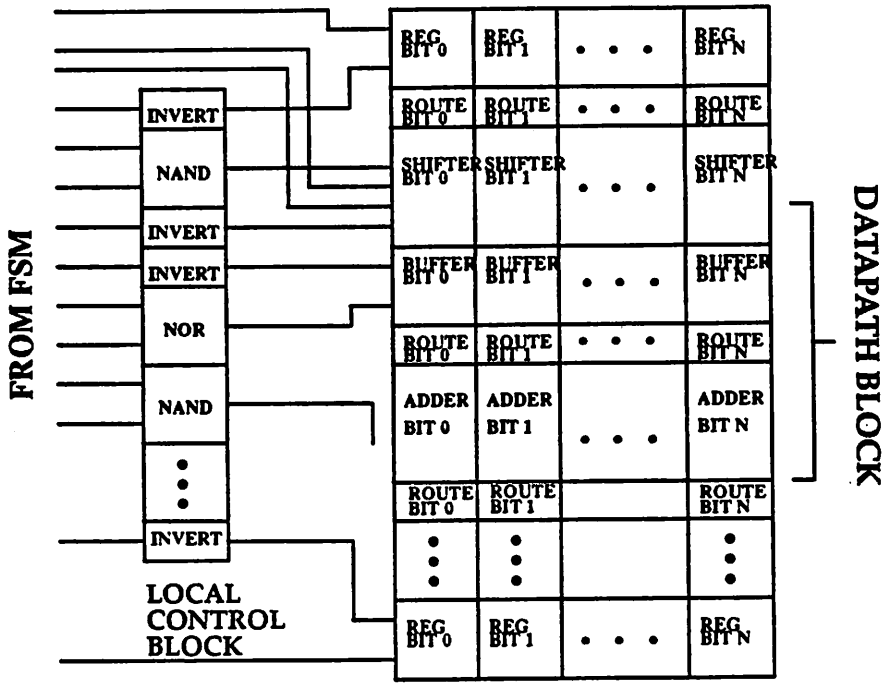

FIGURE 7. General datapath floorplan.

partitioning the datapath into smaller pieces it would be left as one large chunk of standard cells which could then be placed and routed in way that minimized net length. TimberWolf is a standard cell placement tool which uses simulated annealing to find nearly optimal cell placement. Using this tool it was possible to generate some standard-cell examples to compare with datapath versions.

3.3 Implementation

The standard-cell implementation was accomplished in the following manner: standard-cell macros were written with an almost one-to-one correspondence to the macrocell functional blocks, e.g., adder, subtractor, shifter, etc. In this way when hardware selection is performed the actual library being used is irrelevant since equivalent blocks are available. The macro for each block was defined hierarchically, consisting of a parameterized block macro which instantiates bit-sliced macros of standard cells. The standard cell macro versions differ in one respect from their datapath counterparts: the control signals. The control for each block usually requires both true and complementary signals. When using the datapath cells it is necessary to generate both signals external to

the datapath block and then route the signals from the control block to the datapath. Conversely, it was possible to include the inverters required to generate complementary signals inside the standard cell macro blocks and thus eliminate the extra control signal routing. Since this reduction applies to almost every control signal a significant amount of routing area is eliminated.

Since the control for each block is generated based on the information in the hardware database it was necessary to write a standard cell hardware database. The database contains information about each cell in the hardware library. The information stored in the database includes cell area, terminal names for both data and control, drive capability, and delay information based on Spice simulations. One drawback to the standard cell layout style is that the timing information provided can only be of limited accuracy. This is because the length of the interconnect cannot be determined prior to placement. The delay information in the library is for heavily loaded outputs. An example of a database entry is shown in figure 8, this entry is for a standard cell adder. The first entry associates a symbol with the operation, in this case "+", this is associated with a symbol attached to a node in the flowgraph during hardware selection. The second entry indicates the direction of delay ripple, while the third and fourth entries specify the one-bit and overall delays for the block. The

```
(+ ("SC_add" (PARAMETERS (N)) (AREA (* N 76 136)) (DELAY (+ 6 (* N 2)))
(RIPPLE-DIR LSB2MSB)
(ONE-BIT-DELAY 6)
(RIPPLE-DELAY (* N 2))
(DATA-TERMINAL (OUT (IN1 IN2)))
(POWER-TERMINAL (Vdd GND))
(CTL-IN-TERMINAL ((CIN GND)))
(CTL-OUT-TERMINAL COUT)
(CTL-TERM-EDGE ((CIN BOTTOM) (COUT TOP)))
(COMPLEMENT-OUT OUTINV) (DRIVING-CAP NO)))
```

FIGURE 8. Example hardware database entry.

remaining entries are simply terminal declarations with the exception of the "driving cap" entry which indicates whether or not the block output needs to be buffered.

At this time only two libraries are provided, the datapath library and the standard cell library, however, with the hardware database now a parameter of hardware selection it is possible to have any number of hardware libraries. In particular it might be useful to have specific hardware libraries for high-performance or low-power implementations.

3.4 Comparison

Table 1 shows a comparison of core areas for standard cell and datapath style implementations of identical designs. As can be seen, the standard cell versions are much larger in every case. The reason for this result may be that the standard cell macros are not the most efficient implementations of the required functions. A comparison of the area of some the standard cell macro blocks to the comparable datapath versions indicates that the standard cell implementations are twice as large in some cases. Since the desired result of this experiment was to take adavantage of the standard cell place and route functions it may be desirable to design a special set of standard cells for use with Hyper which would be more area efficient than using the current standard cell macros. Figure 9 shows a standard cell layout of the IIR bandpass filter example. Compare this with the layout in figure 6.

TABLE 1. Area comparison: datapath stvle s. standard cell.

Benchmark Name	Type	Purpose	Order or # of Taps	Data Bit Width	Datapath Style Total Area (mm^2)	Standard Cell Total Area (mm^2)
IIR7	IIR	Low pass	7	32	61.0	155.0
BPIIR3	IIR	Band pass	6	16	26.3	44.7
HAMMLP	FIR	Low pass	21	16	30.2	70.8
BMANBP	FIR	Band pass	21	16	45.8	91.3
BPIIR5	IIR	Band pass	6	16	24.1	42.7

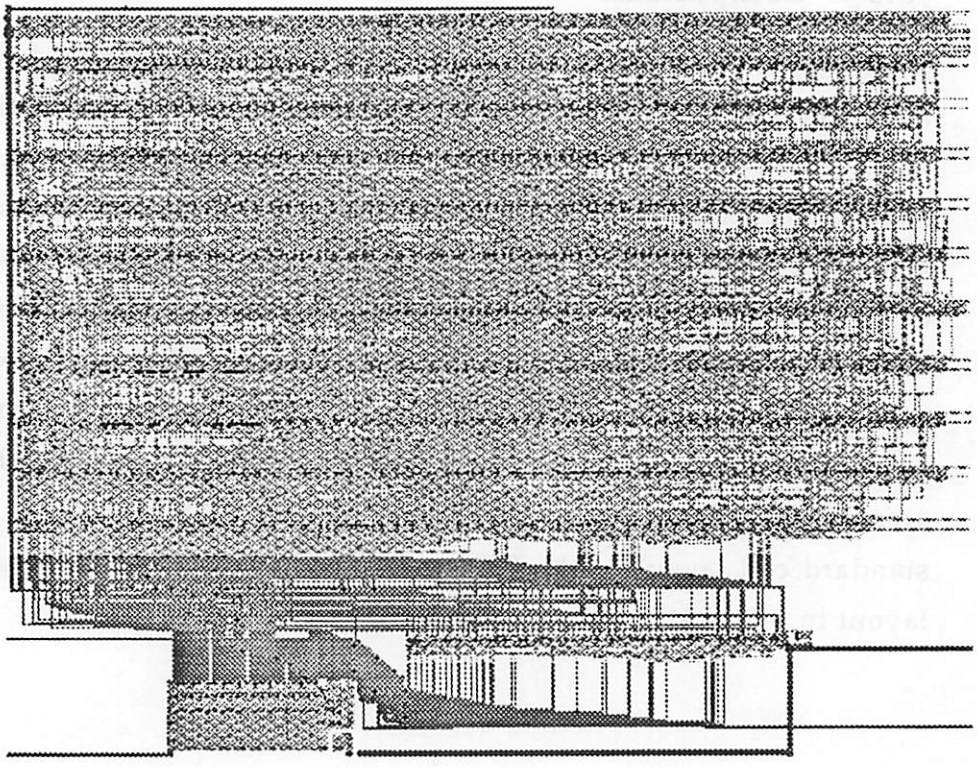

FIGURE 9. Standard cell layout of IIR bandpass filter.

4

Area Estimation

4.1 Goal

The main goal of this project was to generate an algorithm to estimate layout area, ideally using factors at the highest level possible. By using high-level information it would be possible to provide the designer with area estimates at an early stage in the design process.

4.2 Methodology

There have been many efforts to predict the area of VLSI layouts, [3][5][7] are a few examples. Some have been targeted for synthesis systems [4][6]. Most of these seek to derive an estimation formula from layout principles. The approach taken in this project is based on statistical methods rather than underlying principles, although it may certainly be possible to relate the findings to those principles.

In seeking to generate a formula or algorithm to estimate layout area from statistics there were a two possible avenues to take. One method would be to perform a multivariate analysis of the parameters found in the studied systems. However, this method could not be employed due to the high degree of correlation between the relevant parameters. Another method would be to empirically generate a formula through careful observation of the data and then verify it by predicting the areas of the benchmark systems. The formula could

then be validated by predicting the areas of some designs outside the data set. This is the approach adopted in this project.

4.3 Benchmarks

The design benchmarks chosen for this project are all digital filters of one kind or other. They all fit into one of three filter categories: finite impulse response or FIR, infinite impulse response or IIR, or wave-digital. Although this is only a single class of digital systems which can be implemented using Hyper, these designs offer enough generality to provide useful information. Each IIR example was described in Silage as a cascade of direct form II biquad blocks and first-order sections to form the required filter. The FIR examples were based on the transpose network. The following table summarizes some pertinent information about each benchmark, a more detailed set of tables and a Silage description of each benchmark can be found in the appendix.

TABLE 2. Benchmarks.

Name	Type	Purpose	Order or # of Taps	Critical Path (Cycles)	Nodes/ Edges	Sampling Rate
IIR7	IIR	Low pass	7	10	43/43	770kHz
BPIIR3	IIR	Band pass	6	10	55/55	500kHz
HAMMLP	FIR	Low pass	21	6	50/50	500kHz
BMANBP	FIR	Band pass	21	7	73/73	360kHz
BPIIR5	IIR	Band pass	6	12	65/65	420kHz
WAVE1	Wave-Digital	Noise Shaping	12	35	128/128	290kHz

4.4 Data Collection

In order to generate a large set of data each example was run through Hyper a number of times while varying certain parameters. For example, IIR7 was run 12 times with bitwidth varying from 8 to 32 bits in multiples of 2 and the number of datapath partitions varying from 2 to 4. Each example was floorplanned at least 10 times and some as many as 20 times in order to find a final area as close to optimal as possible.

4.5 Analysis

The determination of the parameters that have the strongest correlation with total area was performed in the following manner: (1) Plots were made of the total area versus each data point recorded, (2) The area data was corrected to remove the influence of other parameters, (3) Any observed correlation was noted. The derivation of the area estimation formulas was accomplished by fitting curves to the corrected data.

The factors determined to have the strongest influence on final area are the instance area, the number of global buses connecting the datapath partitions, and the number of control nets distributed to the datapath partitions. For the purposes of this report instance area is defined as library cell area, this being the lowest level at which it was possible to differentiate easily between device area and routing area.

The data indicates that a strong relationship exists between instance area and total area. This relationship is roughly linear as can be seen clearly in figure 10. However, it is obvious that there are other factors influencing this relationship. By successively correcting the area data for other factors it was determined that the global bus count has a strong effect on the way total area varies with instance area. This influence is apparent in figure 11 which shows total area vs. instance area where the area for each benchmark has been multiplied by a correction factor. This factor is equal to the average number of global buses divided by the number of global buses in that benchmark. This results in a more obviously linear relationship than is seen in figure 10 (Note that the y-scale has changed).

Another factor which shows strong correlation with total area is the control bus width, i.e., the number of control signals in the system. This relationship seems to be quadratic in nature as can be seen in figure 12. In this plot area has been corrected to minimize the influence of instance area as a factor. This correction is similar to the correction for global buss count and is a multiplication of the area for each benchmark by a factor equal to the average instance area for all the benchmarks divided by the instance area for that benchmark.

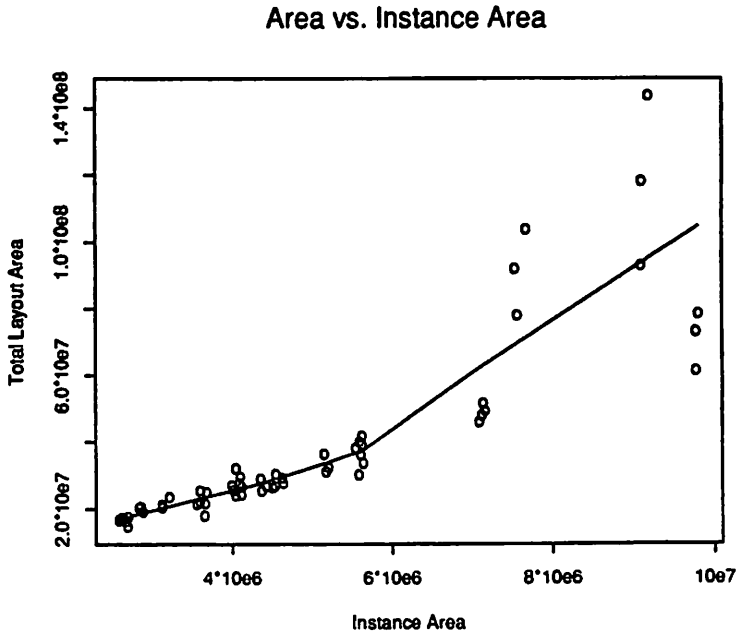

FIGURE 10. Total area vs. instance area.

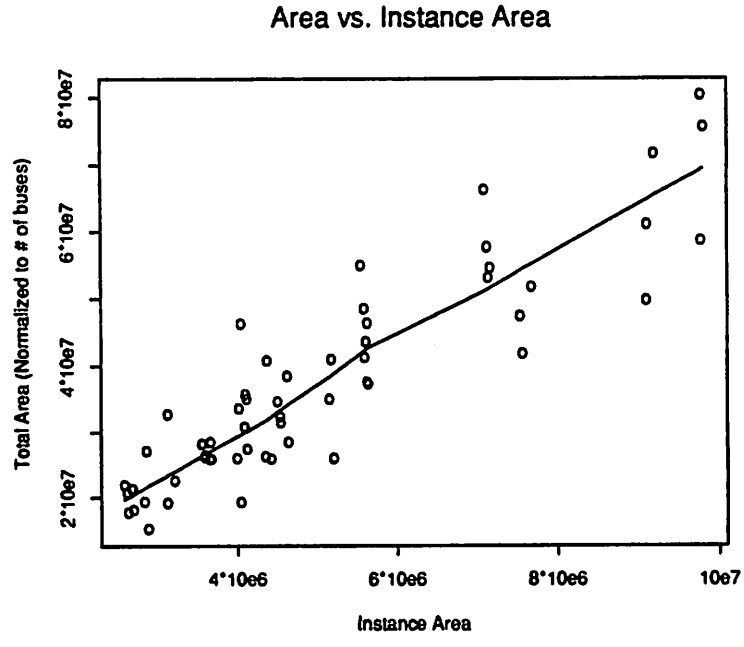

FIGURE 11. Total area vs. instance area - total area has been corrected to account for the influence of global bus count.

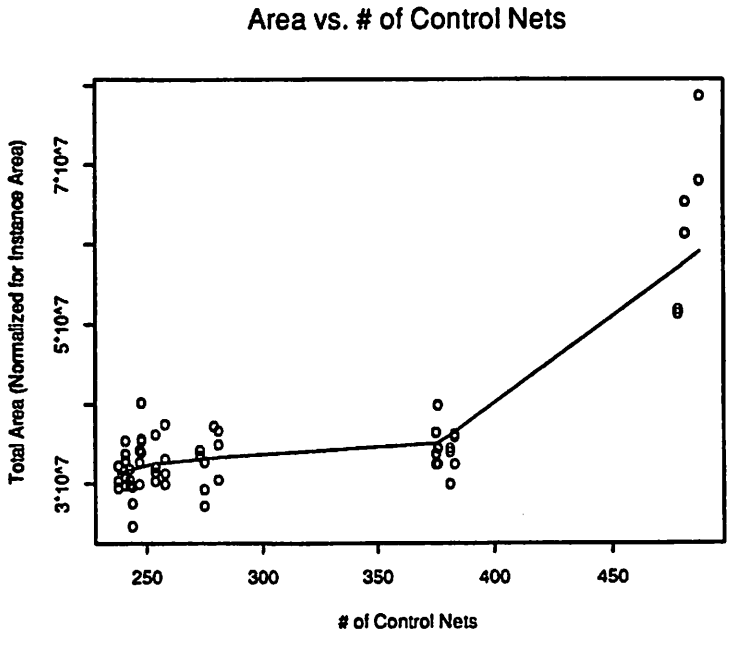

FIGURE 12. Total area vs. control bus width.

Instance area, global buss count, and control bus width are factors which are not known with certainty until after the hardware mapping step. For this reason the estimation cannot be performed until after this step. In fact, instance area must stll be estimated to some degree.

4.5.1 Instance Area Estimation

In the benchmark systems instance area is largely a function of the number and type of EXUs and the number of registers. However, multiplexers, tri-state buffers, and control also contribute significantly. The estimation of datapath instance area can be very accurate once a system has been scheduled since at this point we know how many of each unit there will be and the area of each unit. However control is not so easily estimated, and since we can't use deterministic methods we must resort to statistical methods. Fortunately, strong correlations are found between control area and both the control bus width and the number of states in the FSM. These relationships are illustrated in figures 13 and 14.

The dependence on control bits is linear and can be approximated by a simple linear equation. The dependence on number of states is polynomial in

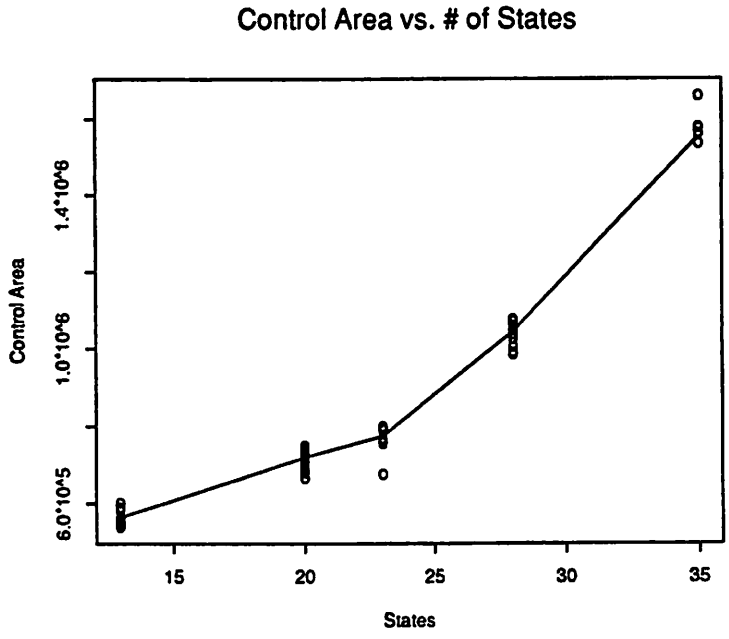

FIGURE 13. Control instance area vs. number of states in the FSM.

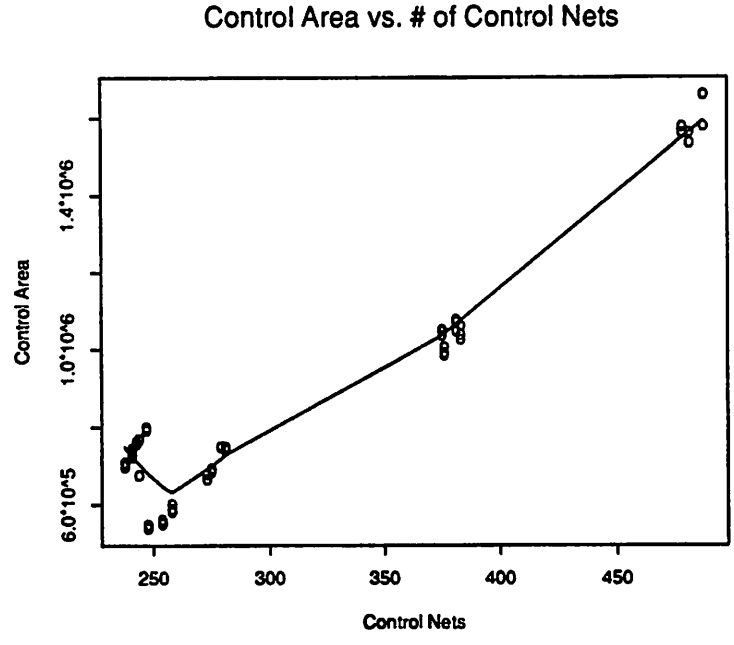

FIGURE 14. Control instance area vs. control bus width.

nature and a quadratic equation is a good approximation. There is certainly correlation between the number of states and the control bus width but the relationship is not well defined. Estimations based on either factor alone gave fair results with error around 25%. However, a linear combination of the two equations results in a highly acccurate prediction. The following is the formula derived for estimation of control area:

$$EContArea = 0.69f(S) + 0.31f(C)$$

$$f(S) = 404S^2 + 20278S + 176510$$

$$f(C) = 3910(C - 99)$$

Where *EContArea* is the estimated control area, S is the number of states in the FSM, C is the number of control signals distributed to the datapaths, and the constants are derived from graphs of the data. This is obviously an oversimplification since there are certainly other factors influencing control area, however it does predict control instance area very accurately for the studied examples. Figure 15 is a bar chart showing the error between predicted and actual control instance area. The average error is 4.7% and the maximum error is 14.3%.

The estimated control area becomes part of the overall equation for instance area given by:

$$EInstArea = Econtrol + \sum_i AB_i \times BW_i$$

Where *EInstArea* is the estimated instance area, AB_i is the area of a single bit-slice of a hardware unit such as an adder, register, etc., and BW_i is the bit width of that unit. In other words, the estimated instance area is the sum of the known areas of the library cells multiplied by the number of bits per block. plus the estimated control instance area. The estimated instance area is then used in the formula for overall area.

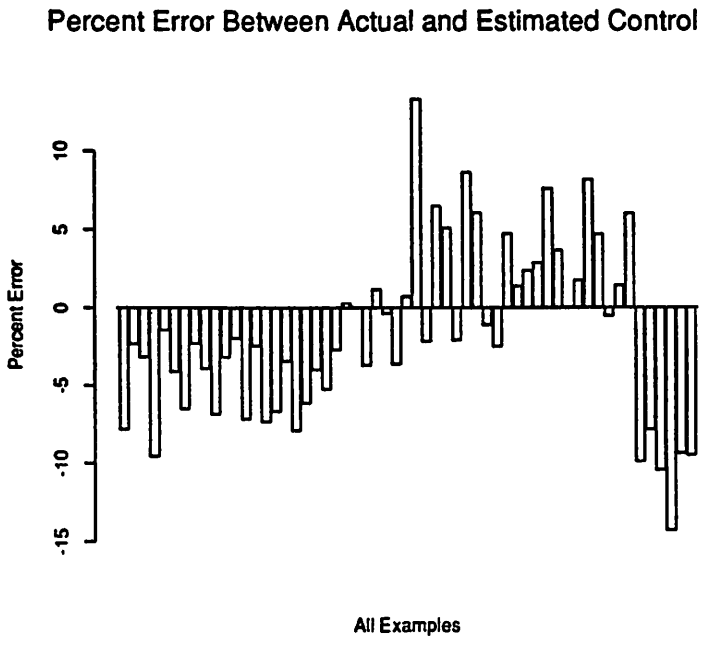

FIGURE 15. Error between actual and predicted control instance area.

4.5.2 Total Area Estimation

Now that we can estimate the instance area fairly accurately it should be simple to estimate the total area using the results. However, experiment shows that estimations based on instance area alone are highly inaccurate. In order to improve the accuracy we must model the effects of other factors.

Two factors which have an obvious impact on routing area are the number of global buses and the number of control signals. Since a large percentage of the total area is routing, the total area should be a function of these parameters. This is confirmed by the relationships shown in figures 11 and 12. The effect of the global bus count can be modeled as a simple scaling factor. The influence of the control bus width is not as easily characterized. As discussed earlier this seems to have a quadratic correlation with total area. This correlation appears only after the data has been corrected for the influence of instance area which indicates that the area function of control bus width is dependent on the instance area. Experiments attempting to establish correlation between total area and control bus width with the global buss count as a scaling factor were not

successful. This seems to indicate that global bus count and control bus width are uncorrelated and that their influences must be combined.

The method of combining the two uses a linear combination to model the effect of each factor. An equation showing the relationship of total area with instance area, global bus count, and control bus width is of the form:

$$Area = K_1 B\,(InstArea + K_2) + K_3\,(InstArea)\;(aC^2 + bC + c)$$

In this equation *Area* is the actual total area, *InstArea* is the actual instance area, K_1, K_2, and K_3 are constants to be determined, B is the global bus count, and C is the control bus width. By extending this to use the estimated instance area and determining the constants we arrive at a total area estimation formula. This formula can be shown as a weighted sum of two functions. One a function of instance area and global bus count and the other a function of instance area and control bus width. This formula is given below with constants derived from the benchmark data:

$$EArea = 0.47f(EInstArea, B) + 0.53f(EInstArea, C)$$

Where *f(EInstArea, B)* and *f(EInstArea, C)* are given by:

$$f(EInstArea, C) = \left(\frac{EInstArea}{5.033 \times 10^6}\right) \times (156C^2 - 1790C + 1.828 \times 10^7)$$

$$f(EInstArea, B) = \left(\frac{7B}{14.5}\right)\,(EInstArea + 3.64 \times 10^5)$$

4.6 Results

The overall measure of the accuracy of the estimation formula is the average absolute error between the estimated area and the actual area of the layout examples. This is 8.9% with a maximum error of 24.7% and a minimum of 0.07%. It should be noted that only seven of the fifty-seven layout examples have an absolute error over 15%. Figure 16 shows a bar chart of the error between the actual and estimated area for all examples.

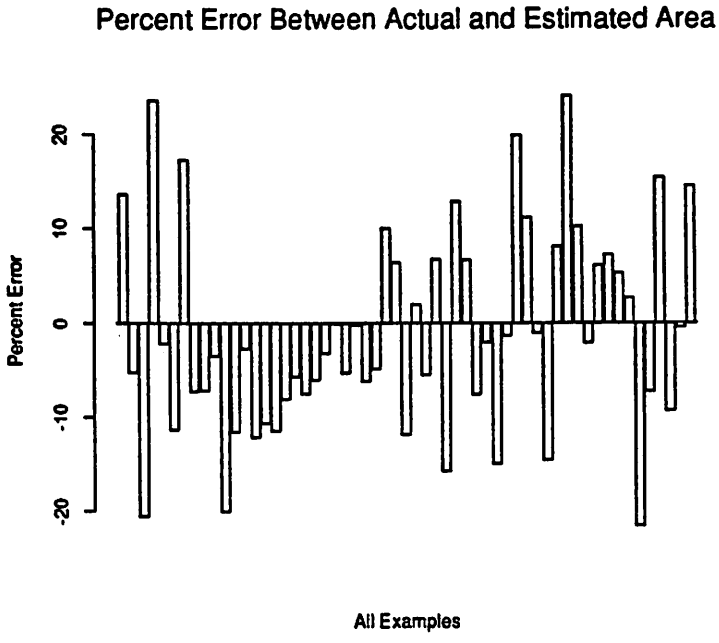

FIGURE 16. Error between actual and estimated total area.

The true test of the validity of a statistically derived estimation formula is to estimate an example outside the data set. This has been done for one example, a seventh-order IIR high-pass filter, with the encouraging result of 4.6% error between actual and estimated area. Table 3 shows the results of each part of the estimation. The Silage description and statistics for this system are included in the appendix.

TABLE 3. Results of estimation of non-benchmark system.

	Control Area (μm^2)	Instance Area (μm^2)	Total Area (μm^2)
Estimated	983003.4	7433307.4	66900731.8
Actual	1168158.0	7618462.0	70111536.0
% Error	15.9%	2.4%	4.6%

4.7 Caveat

The interactive floorplanning using Flint introduces a user-dependent variation in the final area of the layout. Obviously there are many ways to floorplan and route a selection of blocks, some of them efficient, others not. Unfortunately it is difficult to quantify the influence of the user on the area of a particular floorplan. It seems likely that the user would introduce a constant scaling factor which may affect the absolute numbers but not the overall factors governing area. However, since the intent of this project was to show a classification of the variables influencing area rather than absolute numbers, the goal has been met.

The data used in this project was derived from a restricted set of benchmarks. The benchmarks were limited in the amount of resource-sharing and required only small controllers. This may or may not affect the validity of the results if applied to other classes of systems.

Future Directions 5

5.1 Hardware Model

It has become apparent that the existing Hyper architecture has some flaws. The percentage of total chip area devoted to routing is generally over 80% which is an unacceptably high figure. The amount of area devoted to routing is the result of two factors. The first of these is the interconnect architecture. There is a great deal of superfluous interconnect arising from the dedicated data buses that connect each EXU to each of its destination register files as well as the data buses connecting some register files to several other register files. Another problem which also increases the interconnect overhead is the partitioning strategy. The partitioning is accomplished using the min-cut algorithm to divide the datapath into smaller blocks in order to improve the aspect ratio of the final core. Unfortunately the algorithm does not take into consideration the communication between the register files and their associated execution unit. This often results in the register file(s) being put into partitions other than the one where the execution unit is, and as a consequence, additional global busing is required.

Given this result with the current hardware model some new models have been suggested. One proposed model takes advantage of the fact that the global data buses are not efficiently used, i.e., many buses are idle during the majority of system cycles. To take advantage of this we can collapse several dedicated buses into one shared bus. This does require some additional hardware, namely some multiplexers must be placed at the inputs to the register

files in order to select between multiple buses. The partitioning can also be improved. Instead of partitioning the datapath as in the original model each individual hardware block can be treated as a partition. Using this model each partition would consist of a single execution unit, one or two register files, a tri-state buffer for each shared bus that the execution unit needs to drive, and any multiplexers that might be necessary due to multiple input buses. Figure 17 shows the proposed architecture while figure 18 illustrates a general floorplan for this model. It is anticipated that this model will significantly reduce the amount

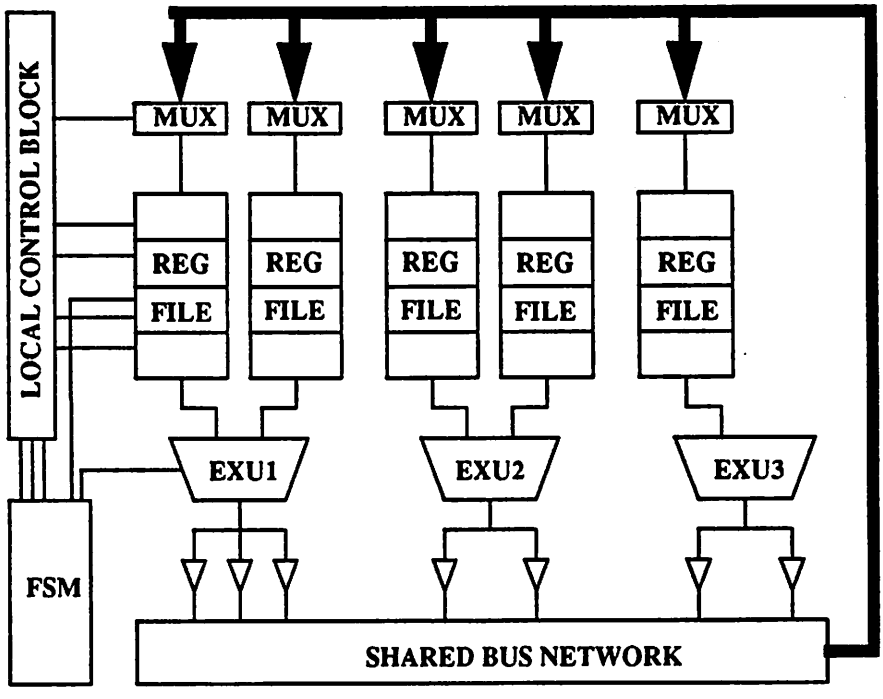

FIGURE 17. New hardware model.

of routing overhead at the slight expense of a few additional multiplexers. Preliminary estimates indicate a decrease in area of 25% may be possible in some cases.

5.2 Estimation

While the results of the area estimation are good the information which we use to make our estimations is fairly low-level. This means that the factors which we need are all generated at a very late stage in the Hyper design process.

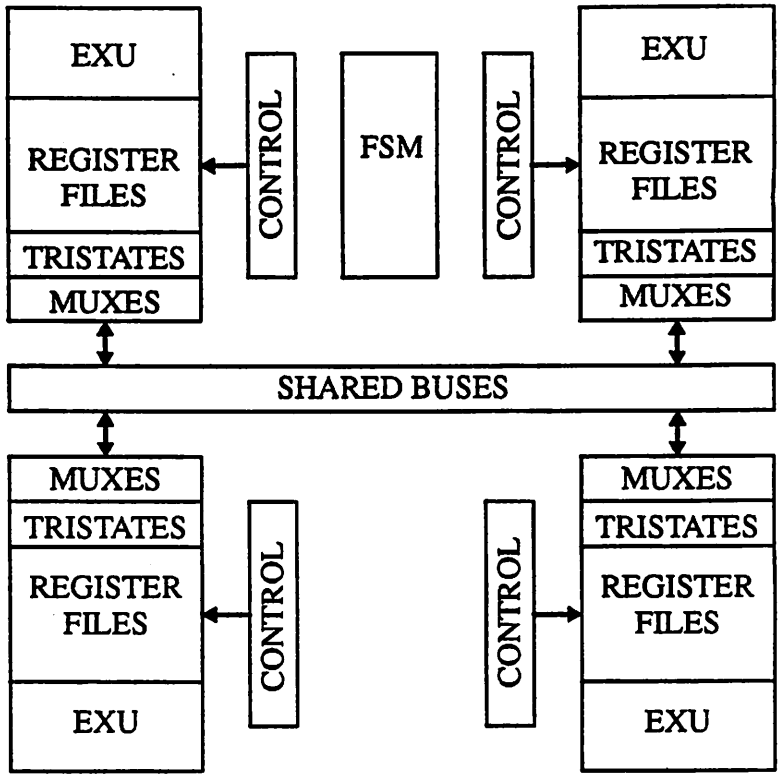

FIGURE 18. Floorplan using new hardware model.

Of the factors used in the estimation formulas only instance area can be partially estimated prior to scheduling. This can only be partially estimated because the control instance area cannot be estimated until after the control nets are generated in the hardware mapping step. The number of global busses also cannot be estimated earlier than after the scheduling step since this depends on the amount of bus sharing which may be possible and the number of partitions and these in turn depend upon the schedule.

We would like to make estimations at an earlier stage, the question is, can accurate area estimation be performed at an earlier stage? Of course the answer is yes, but the estimations will not be as accurate as those which can be made later. This topic requires more research to determine the fundamental relationships between the parameters used to describe a system behaviorally and the actual implementation cost.

Appendix

This appendix contains tables of data collected during the project and the Silage descriptions of each of the benchmark designs. Also include is the Silage description and statistics of a system that was used to validate the estimation formulas.

TABLE 1. System Data - Independent of Bitwidth and Number of Datapath Partitions

			Number of Execution Units			Number of Registers for each EXU Type				
Name	Critical Path (cycles)	Cycles	Adder	Subtracter	Shifter	Adder	Subtracter	Shifter	Transfer Registers	Tri-State Buffers
IIR7	10	13	2	2	2	14	8	8	4	28
BPIIR3	10	20	1	1	1	9	9	6	5	26
HAMMLP	6	20	4	1	1	35	6	2	1	12
BMANBP	7	28	2	1	2	35	7	10	4	25
BPIIR5	12	24	1	1	2	9	10	6	5	26
WAVE1	35	35	2	4	2	9	37	2	2	86

TABLE 2. System Data with Dependency on Bitwidth and Number of Datapath Partitions - IIR7

IIR7									
Bitwidth	**Datapath Partitions**	**Data buses**	**Control Nets**	**FSM Outputs**	**X Dimension (μm)**	**Y Dimension (μm)**	**Total Area (mm^2)**	**% Instance Area**	**% Control Area**
8	2	11	254	73	5263	3915	20.6	13.9	2.7
8	3	15	248	69	4535	4443	20.1	14.1	2.7
8	4	18	258	80	5561	3420	19.0	15.2	3.1
12	2	11	254	73	5592	4567	25.5	15.7	2.2
12	3	15	248	69	5330	5056	26.9	14.8	2.0
12	4	18	258	80	5927	4056	24.0	16.8	2.5
16	2	11	254	73	5937	5246	31.1	16.6	1.8
16	3	15	248	69	6419	5647	36.2	14.2	1.5
16	4	18	258	80	6475	4988	32.3	16.1	1.8
32	2	11	254	73	8207	7438	61.0	16.0	0.9
32	3	15	248	69	9856	7956	78.4	12.5	0.7
32	4	18	258	80	8606	8474	72.9	13.4	0.8

```
/*
iir7.sil - Seventh order IIR lowpass filter
*/

#define num16 fix<32,10>

#define Coef0 0.001953125
#define Coef1_1 -1.3125
#define Coef1_2 0.625
#define Coef1_3 1
#define Coef1_4 1
#define Coef2_1 -1.25
#define Coef2_2 0.75
#define Coef2_3 0.0625
#define Coef2_4 1
#define Coef3_1 -1.125
#define Coef3_2 0.921875
#define Coef3_3 -0.25
#define Coef3_4 1
#define Coef4_1 -0.71875
#define Coef4_2 1

func main (In : num16) Out : num16 =
begin
 In1 = num16(In*Coef0);
 In2 = biquad(In1, Coef1_1, Coef1_2, Coef1_3, Coef1_4);
 In3 = biquad(In2, Coef2_1, Coef2_2, Coef2_3, Coef2_4);
 In4 = biquad(In3, Coef3_1, Coef3_2, Coef3_3, Coef3_4);
 Out = firstorder(In4, Coef4_1, Coef4_2);
end;


func biquad(in, a1, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(a1*state@1) + num16(a2*state@2));
 return = state + (num16(b1*state@1) + num16(b2*state@2));
end;


func firstorder(in, a1, b1: num16) : num16 =
begin
 state@@1 = 0.0;
 state = in - num16(a1*state@1);
 return = state + num16(b1*state@1);
end;
```

TABLE 3. BPIIR3 - Data Dependent on Bitwidth and Number of Datapath Partitions

BPIIR3									
Bitwidth	**Datapath Partitions**	**Data buses**	**Control Nets**	**FSM Outputs**	**X Dimension (μm)**	**Y Dimension (μm)**	**Total Area (mm^2)**	**% Instance Area**	**% Control Area**
8	2	12	241	76	4678	3676	17.2	15.3	4.3
8	3	11	238	74	4536	3667	16.6	15.6	4.2
8	4	14	241	76	5982	2870	17.2	15.4	4.3
12	2	12	241	76	5009	4370	21.9	16.4	3.4
12	3	11	238	74	5716	3742	21.4	16.6	3.3
12	4	14	241	76	6295	4022	25.3	14.2	2.9
16	2	12	241	76	5361	4996	26.8	16.9	2.7
16	3	11	238	74	6612	3977	26.3	17.1	2.7
16	4	14	241	76	5841	5208	30.4	14.9	2.4

```
/*
bpiir3.sil - IIR bandpass filter
*/

#define num16 fix<8,2>

/* #define Coef0 0.0057643052 */
#define Coef0 0.015625

#define Coefa1_1 -1.9377627
#define Coefa1_2 1
#define Coefb1_1 -1.9085335
#define Coefb1_2 0.93137014

#define Coefa2_1 -1.9910688
#define Coefa2_2 1
#define Coefb2_1 -1.9320831
#define Coefb2_2 0.96581414

#define Coefa3_2 -1
#define Coefb3_1 -1.9601845
#define Coefb3_2 0.97627349


func main (In : num16) Out : num16 =
begin
 In0 = num16(In*Coef0);
 In1 = biquad(In0, Coefa1_1, Coefa1_2, Coefb1_1, Coefb1_2);
 In2 = biquad(In1, Coefa2_1, Coefa2_2, Coefb2_1, Coefb2_2);
 Out = biquad0(In2, Coefa3_2, Coefb3_1, Coefb3_2);
end;

func biquad(in, a1, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a1*state@1) + num16(a2*state@2);
end;

func biquad0(in, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a2*state@2);
end;
```

TABLE 4. HAMMLP - Data Dependent on Bitwidth and Number of Datapath Partitions

HAMMLP									
Bitwidth	**Datapath Partitions**	**Data buses**	**Control Nets**	**FSM Outputs**	**X Dimension (μm)**	**Y Dimension (μm)**	**Total Area (mm^2)**	**% Instance Area**	**% Control Area**
8	2	9	275	63	5331	3812	20.3	15.4	3.4
8	3	13	279	67	5160	4117	21.2	14.7	3.2
8	4	15	281	69	6098	3843	23.4	13.7	3.2
12	2	9	275	63	5584	4542	25.3	17.2	2.7
12	3	13	279	67	6314	4589	29.0	15.0	2.3
12	4	15	281	69	6249	4282	26.8	16.5	2.8
16	2	9	275	63	5871	5136	30.2	18.5	2.3
16	3	13	279	67	7598	5483	41.7	13.5	1.8
16	4	15	281	69	7053	5535	39.0	14.4	1.9

```
/*
hammlp.sil - Hamming window lowpass filter
*/


#define num16 num<16,5>

#define a0 0.078125
#define a1 0.109375
#define a2 0.171875
#define a3 0.265625
#define a4 0.390625
#define a5 0.546875
#define a6 0.687500
#define a7 0.812500
#define a8 0.906250
#define a9 0.984375
#define a10 1.000000

func main(In : num16) Out : num16 =
begin
 Acc1 = num16(In * a0);
 Acc2 = num16(In * a1) + Acc1@1;
 Acc3 = num16(In * a2) + Acc2@1;
 Acc4 = num16(In * a3) + Acc3@1;
 Acc5 = num16(In * a4) + Acc4@1;
 Acc6 = num16(In * a5) + Acc5@1;
 Acc7 = num16(In * a6) + Acc6@1;
 Acc8 = num16(In * a7) + Acc7@1;
 Acc9 = num16(In * a8) + Acc8@1;
 Acc10 = num16(In * a9) + Acc9@1;
 Acc11 = num16(In * a10) + Acc10@1;
 Acc12 = num16(In * a9) + Acc11@1;
 Acc13 = num16(In * a8) + Acc12@1;
 Acc14 = num16(In * a7) + Acc13@1;
 Acc15 = num16(In * a6) + Acc14@1;
 Acc16 = num16(In * a5) + Acc15@1;
 Acc17 = num16(In * a4) + Acc16@1;
 Acc18 = num16(In * a3) + Acc17@1;
 Acc19 = num16(In * a2) + Acc18@1;
 Acc20 = num16(In * a1) + Acc19@1;
 Out = num16(In * a0) + Acc20@1;
end;
```

TABLE 5. BMANBP - Data Dependent on Bitwidth and Number of Datapath Partitions

BMANBP									
Bitwidth	**Datapath Partitions**	**Data buses**	**Control Nets**	**FSM Outputs**	**X Dimension (μm)**	**Y Dimension (μm)**	**Total Area (mm^2)**	**% Instance Area**	**% Control Area**
8	2	10	376	92	7123	4494	32.0	12.6	3.1
8	3	14	380	95	5451	5435	29.6	13.8	3.5
8	5	13	381	98	6101	4525	27.6	14.8	3.8
8	6	14	383	90	5289	5018	26.5	15.5	4.0
12	2	10	376	92	7310	5194	38.0	14.6	2.6
12	3	14	380	95	6170	5847	36.0	15.5	2.9
12	5	13	381	98	6273	5347	33.5	16.8	3.3
12	6	14	383	90	6333	6305	39.9	14.0	2.6
16	2	10	376	92	7551	6062	45.8	15.5	2.2
16	3	14	380	95	7398	6461	47.8	14.9	2.2
16	5	13	381	98	7200	6815	49.1	14.6	2.2
16	6	14	383	90	7283	7057	51.4	13.9	2.0

```
/*
 bmanbp.sil - Blackman window bandpass filter
*/


#define num16 num<16,6>

#define a1 0.010066148
#define a2 0.049648163
#define a3 0.14402198
#define a4 0.31796218
#define a5 0.58364007
#define a6 0.92843705
#define a7 1.3097011
#define a8 1.6610868
#define a9 1.9099674
#define a10 1.9999999

func main(In : num16) Out : num16 =
begin
 Acc2 = num16(In * a1);
 Acc3 = num16(In * a2) + Acc2@1;
 Acc4 = num16(In * a3) + Acc3@1;
 Acc5 = num16(In * a4) + Acc4@1;
 Acc6 = num16(In * a5) + Acc5@1;
 Acc7 = num16(In * a6) + Acc6@1;
 Acc8 = num16(In * a7) + Acc7@1;
 Acc9 = num16(In * a8) + Acc8@1;
 Acc10 = num16(In * a9) + Acc9@1;
 Acc11 = num16(In * a10) + Acc10@1;
 Acc12 = num16(In * a9) + Acc11@1;
 Acc13 = num16(In * a8) + Acc12@1;
 Acc14 = num16(In * a7) + Acc13@1;
 Acc15 = num16(In * a6) + Acc14@1;
 Acc16 = num16(In * a5) + Acc15@1;
 Acc17 = num16(In * a4) + Acc16@1;
 Acc18 = num16(In * a3) + Acc17@1;
 Acc19 = num16(In * a2) + Acc18@1;
 Out = num16(In * a1) + Acc19@1;
end;
```

TABLE 6. BPIIR5 - Data Dependent on Bitwidth and Number of Datapath Partitions

BPIIR5									
Bitwidth	Datapath Partitions	Data buses	Control Nets	FSM Outputs	X Dimension (μm)	Y Dimension (μm)	Total Area (mm^2)	% Instance Area	% Control Area
8	2	10	244	79	4228	3486	14.7	18.3	5.2
8	3	11	243	79	4859	3342	16.2	16.5	4.7
8	4	14	247	80	5416	3248	17.6	15.4	4.5
12	2	10	244	79	4480	3994	17.9	20.4	4.3
12	3	11	243	79	5789	3734	21.6	16.9	3.5
12	4	14	247	80	5930	4216	25.0	14.7	3.2
16	2	10	244	79	4956	4878	24.1	17.0	2.8
16	3	11	243	79	6167	4739	29.2	15.8	2.6
16	4	14	247	80	6041	4564	27.6	16.8	2.9

```
/*
bpiir5.sil - IIR bandpass filter
*/

#define num16 fix<16,6>

#define Coef0 0.015625

#define Coefa1_1 -1.2283669
#define Coefa1_2 1
#define Coefb1_1 -1.5145303
#define Coefb1_2 0.96173355

#define Coefa2_1 -1.7409752
#define Coefa2_2 1
#define Coefb2_1 -1.4522301
#define Coefb2_2 0.98113835

#define Coefa3_2 -1
#define Coefb3_1 -1.5987214
#define Coefb3_2 0.98356567


func main (In : num16) Out : num16 =
begin
 In0 = num16(In*Coef0);
 In1 = biquad(In0, Coefa1_1, Coefa1_2, Coefb1_1, Coefb1_2);
 In2 = biquad(In1, Coefa2_1, Coefa2_2, Coefb2_1, Coefb2_2);
 Out = biquad0(In2, Coefa3_2, Coefb3_1, Coefb3_2);
end;

func biquad(in, a1, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a1*state@1) + num16(a2*state@2);
end;

func biquad0(in, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a2*state@2);
end;
```

TABLE 7. WAVE1 - Data Dependent on Bitwidth and Number of Datapath Partitions

WAVE1									
Bitwidth	**Datapath Partitions**	**Data buses**	**Control Nets**	**FSM Outputs**	**X Dimension (μm)**	**Y Dimension (μm)**	**Total Area (mm^2)**	**% Instance Area**	**% Control Area**
12	3	27	478	147	8909	8760	78.0	9.7	2.0
12	4	28	481	151	11232	8182	91.9	8.2	1.7
12	5	29	487	152	11702	8867	103.8	7.4	1.6
15	3	27	478	147	9772	9495	92.8	9.8	1.7
15	4	28	481	151	12908	9158	118.2	7.7	1.3
15	5	29	487	152	13504	10638	143.7	6.4	1.1

```
/*
wave1.sil - A wave digital noise-shaping filter.
*/

#define num16 fix<12,3>

#define alpha1_1 0.2728263481983
#define alpha1_2 0.30760606507159
#define alpha1_3 0.018981781960232

#define alpha2_1 0.16783644485455
#define alpha2_2 0.4122356119527
#define alpha2_3 0.057365199668124

#define alpha3_1 0.091633222582463
#define alpha3_2 0.47386174108547
#define alpha3_3 0.11728595722224

#define alpha4_1 0.034583693538301
#define alpha4_2 0.37137396664558
#define alpha4_3 0.20900698482719


func main (In : num16) Out : num16 =
begin
A1 = Adaptor1(In, alpha1_1);
A2 = Adaptor2(A1, alpha1_2);
Out1 = Adaptor2(A2, alpha1_3);

B1 = Adaptor1(In, alpha2_1);
B2 = Adaptor2(B1, alpha2_2);
Out2 = Adaptor2(B2, alpha2_3);

C1 = Adaptor1(In, alpha3_1);
C2 = Adaptor1(C1, alpha3_2);
Out3 = Adaptor2(C2, alpha3_3);

D1 = Adaptor1(In, alpha4_1);
D2 = Adaptor1(D1, alpha4_2);
Out4 = Adaptor2(D2, alpha4_3);

Out = Out1 + Out2 + Out3 + Out4;
end;

func Adaptor1 (In, alpha : num16) : num16 =
begin
state@@1 = 0.0;
state = num16(alpha * (state@1 - In)) - state@1;
return = state - state@1 + In;
end;

func Adaptor2 (In, alpha : num16) : num16 =
begin
state@@1 = 0.0;
state = num16(alpha * (In - state@1)) - In;
return = num16(alpha * (In - state@1)) - state@1;
end;
```

TABLE 8. HPIIR1- Data.

HPIIR1								
Bitwidth	Datapath Partitions	Data buses	Control Nets	X Dimension (μm)	Y Dimension (μm)	Total Area (mm^2)	% Instance Area	% Control Area
16	4	19	400	10914	6424	70.1	10.9	1.7

		Number of Execution Units			Number of Registers for each EXU Type				
Critical Path (cycles)	Cycles	Adder	Subtracter	Shifter	Adder	Subtracter	Shifter	Transfer Registers	Tri-State Buffers
16	24	2	2	2	11	19	8	5	65

```
/*
hpiir1.sil - IIR High-pass filter.
*/

#define num16 fix<16,7>

#define Coef0 0.6875

#define Coefa1_1 -1.9663138
#define Coefa1_2 1
#define Coefb1_1 -1.7096749
#define Coefb1_2 0.73967969

#define Coefa2_1 -1.9932827
#define Coefa2_2 1
#define Coefb2_1 -1.7682765
#define Coefb2_2 0.81296479

#define Coefa3_1 -1.9782710
#define Coefa3_2 1
#define Coefb3_1 -1.8702533
#define Coefb3_2 0.92918093

#define Coefa4_1 -1
#define Coefb4_1 -0.84541923


func main (In : num16) Out : num16 =
begin
 In0 = num16(In*Coef0);
 In1 = biquad(In0, Coefa1_1, Coefa1_2, Coefb1_1, Coefb1_2);
 In2 = biquad(In1, Coefa2_1, Coefa2_2, Coefb2_1, Coefb2_2);
 In3 = biquad(In2, Coefa3_1, Coefa3_2, Coefb3_1, Coefb3_2);
 Out = first(In3, Coefa4_1, Coefb4_1);
end;

func biquad(in, a1, a2, b1, b2 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - (num16(b1*state@1) + num16(b2*state@2));
 return = state + num16(a1*state@1) + num16(a2*state@2);
end;

func first(in, a1, b1 : num16) : num16 =
begin
 state@@1 = 0.0;
 state@@2 = 0.0;
 state = in - num16(b1*state@1);
 return = state + num16(a1*state@1);
end;
```

Bibliography

[1] C. Chu, et all., "Hyper: An Interactive Synthesis Environment for High Performance Real Time Applications", Proc. Int'l Conf. Computer Design, IEEE Computer Society Press, Los Alamitos, Calif., 1989, pp.432-435.

[2] P. Hilfinger, "A High-level Language and Silicon Compiler for Digital Signal Processing", *Proc. Custom Integrated Circuits Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1985, pp. 213-216.

[3] F. Kurdahi and A. Parker, "Technique for Area Estimation of VLSI Layouts", IEEE Trans. on CAD of IC, Vol. 9, No 9, pp. 938-950, 1990.

[4] R. Jain, "High-Level Area-Delay Prediction with Application to Behavioral Synthesis", Technical Report 89-23, University of Southern California, 1989.

[5] F. J. Kurdahi, "Area Estimation of VLSI Circuits", PhD Thesis, University of Southern California, 1987.

[6] K. Kucukcakar and A.C. Parker, "BAD: Behavioral Area-Delay Predictor", Tech. Report 90-31, University of Southern California.

[7] M. Pedram and B. Preas, "Accurate Prediction of Physical Design Characteristics for Random Logic", 1989 IEEE ICCD Conf., Boston, pp. 100-108, 1989.