設 計 自 動 化 60－8
（1991．12．12）

VLSI Circuit Synthesis Based on Algorithmic Description with the Constraints of Time and Area

Xing-jian XU　Mitsuru ISHIZUKA
Tokyo University

A high-level design synthesis of hardware based on algorithmic design specification has become an important issue in recent years. We have not found some efficient algorithms which are able to design a circuit which satisfies the constraints of time and area simultaneously. This paper presents a fast methodology which is able to allocate data-path under the constraints of processing time and the area of VLSI circuit. The core of the methodology is an efficient redesign starting from an initial configuration with the smallest circuit area.

アルゴリズム記述に基づく時間－面積制約下でのVSLI回路設計

徐　行倹　　石塚　満
東京大学

アルゴリズム記述に基づく上位レベルの回路構成の自動合成は一つの重要な設計手法になっております。特に回路は時間と面積の制約条件を同時に満たさなければならないの場合に、有効なアルゴリズムはまだ少ないです。本稿では、アルゴリズム記述に基づく時間と面積制約下でVSLI回路構成を生成について新しい手法を提案する。

1. INTRODUCTION

The goal of the data path synthesis step in a behavioral synthesis system is to produce a register-transfer (RT) level hardware design from an architectural description of a computer or to produce an RT design which implements a given behavioral specification in a high-level language [1]-[5]. The main task in such a synthesis system is the scheduling of operations to control steps, and the allocation of hardware to implement operations, storage and interconnections. A tutorial by Machael C .McParlend etc. [6] gives a good survey of such synthesis systems.

Given a fixed amount of hardware resources, the optimizations (or maximization) of the processing speed can be performed using the list scheduling techniques [7]. In the case of given a fixed amount of control steps, the force-directed scheduling is a very efficient approach to optimize hardware resources [8]. S. Devadas and A. R. Newton [9] present simulated-annealing-based algorithms which provide excellent solutions to the formulation of a two-dimensional placement problem of micro-instructions in space and time under a variety of user-specifiable constraints on hardware resources and costs.

In the case of designing a VLSI circuit which should satisfy the constraints of processing time and circuit area, there are many circuits which are different in structure at the Register-Transfer (RT) level for the same behavioral specification. Searching all the possible data-path for the best one is not difficult in theory. The problem, however, is that the searching algorithm should have a reasonable and acceptable processing speed to be executed, since there are a huge amount of possible candidate data-paths satisfying the same behavioral specification in a VLSI circuit design. The major purpose of this paper is to present a general redesigning methodology, which is able to automatically redesign the data-path of a circuit based mostly on data-path allocation according to the constraints of time and area. This methodology can be integrated into specialized or general-purpose high-level synthesis systems and it is able to optimize the circuit area and processing speed simultaneously under the constraints of time and area. In chapter 2, some of the basic concepts are introduced and chapter 3 describes the scheduling and the way to find all the redesigning methods. In chapter 4, the principle of our methodology is explained in details. An example of Kalman filter is given in chapter 5 and in the last chapter, a simple conclusion is given.

2. BASIC CONCEPTS

Our algorithmic behavioral specification is written in C. After the behavioral specification is compiled, the internal data representations of data flow graph and control flow graph are created. Essentially, the internal data representations consist of: (1) a control flow graph (O, P), where the nodes O represent a set of operations and the edges P represent the precedence relation; and (2) a data flow graph $(O \cup V, D)$, where O and V are the sets of operations and values respectively and the edges D represent the data dependances [10]. In the initial data-path design of our methodology, an initial data-path which implements the behavior is designed to obtain the most serialized initial circuit with as few hardware resources as possible. A circuit is called ***the most serialized circuit***, if the circuit has the lowest concurrence with the smallest area of the circuit but the longest processing time. Conversely, a circuit is called ***the most parallel circuit***, if the circuit has the highest concurrence with the highest processing speed but the largest area. The reasons we generate the most serialized circuit as initial result are: (1) since only the redesigning methods which can speed up the circuit are interested in, it makes the executoin of the algorithm more efficient; (2) since the number of candidate circuits increases, there are more chances to find the best one. The methodology to be introduced in this paper mainly consists of three sub-tasks:

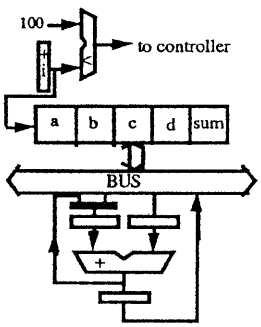
- Initial data-path allocation. The most serialized initial data-path is created by allocating as few hardware resources as possible;
- Scheduling. All operations are assigned to corresponding control steps. All the possible redesigning methods are found according to the relationship between operations; and then evaluations of these methods are calculated;
- Redesigning the initial data-path to obtain the best result according to the constraints of time and area.

3. SCHEDULING

In essence, the scheduler decides whether potentially parallel operations are scheduled in the same control step thus requiring more hardware, or if they are serialized, scheduling them in different control steps, thus allowing them to share the same hardware. The decisions

which schedule operations in the same control step make the processing speed of the circuit faster and the area of the circuit larger. Each decision has an evaluation of the difference of the number of control steps becomes faster divided by the difference of area becomes larger. The best combination of some of these decisions has to be made to obtain the best data-path according to the constraints of time and area. In this section, an example in Fig. 1 is used to explain our approach.

```
i = 0;
do {
   sum[i] = s[i]+b[i]+c[i]+d[i];
   i++;
} while ( i<10 );
```



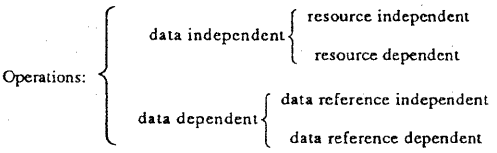
(a) behavioral specification (b) data-path
Fig. 1 Example

In the initial data-path, a memory is created for all the array variables. An adder is allocated to do the addition operations and a comparator is allocated to do the comparison of i<10, and some necessary registers are allocated to hold the values which have a life time more than one control step. To pass the values between the adder, comparator, memory and registers, a common bus and interconnections are also allocated as the shown in Fig. 1(b). Based on the data-path, an initial schedule table shown in Fig. 2 is created by 'As Soon As Possible' strategy.

In Fig. 2, there are three hatched rectangles with rounded corners. As described before, the initial data-path is designed to catch the smallest area of the circuit. Therefore, there is no more than one functional units for the operations with the same type. Although the two ADD operations in the hatched rectangle are data independent, they can not be executed in parallel since there is one ADDER only. It takes two control steps to execute the two ADD operations. Such operations is called ***resource dependent.*** If another new ADDER is allocated, these two ADD operations can be executed in parallel within one control step. As the same as ADD operations, the four MEMORY-READ operations are also resource dependent because they share a common data BUS to transfer data with other functional units and registers. If this memory is divided into two smaller memories with their own data BUSes. So that memories a[], b[] share a common data BUS and b[], d[] share another BUS, then these four MEMORY-READ operations can be executed within two control steps instead of four. If let memories a[], b[], c[] and d[] have their own data BUSes, it will take only one control step to finish these four MEMORY-READ operations. The ADD operation of a[i]+b[i] has to wait for the results of MEMORY-READ operations, i.e., a[i], b[i], until these two MEMORY-READ operations are finished. These three operations are called ***data dependent*** and these operations can not be processed in parallel. The operation of increasing variable ***i*** by 1 has to be executed at least one control step after the operation of memory writing of sum[i] because the ***i*** is used as the address of MEMORY-WRITE. Such kind of data dependent is named ***data reference dependent.*** There still is a methodology to save the control steps taken by the operations which are data reference dependent. In Fig. 2, the value of variable ***i*** is referenced at control step 9 by MEMORY-WRITE operation as the address of the memory. It seems that the increment operation of variable ***i*** has to follow the MEMORY-WRITE operation. However the specification can be rewritten into:

```
i = 0;      ii = 0;
do {
     ii++;
     sum[i] = a[i] + b[i] + c[i] + d[i];
     i = ii;
} while ( ii<10 );
```

Now, the variable of ***ii*** is not referenced and the operation of increasing ***ii*** by one and COMPARE operation can be executed in parallel with the MEMORY-READ and ADD operations. The variable ***ii*** is almost the same as variable ***i*** but is increased several control steps in advance, so that variable ***ii*** is called ***shadow variable*** of ***i***. In the circuit, there is a register ***i*** to save the value of variable ***i***, then the register ***ii*** to save the value of variable ***ii*** called ***shadow register*** of register ***i***. As a summary of scheduling, all the operations can be divided into as:



Only the operations which are resource dependent or data reference dependent can be scheduled into the same control step, if some new hardware resources are allocated. During the scheduling, each time the operations of resource dependent or data reference dependent are found, the information of redesigning methods can be obtained which will be used to redesign the circuit to satisfy the constraints of time and area in redesign procedure.

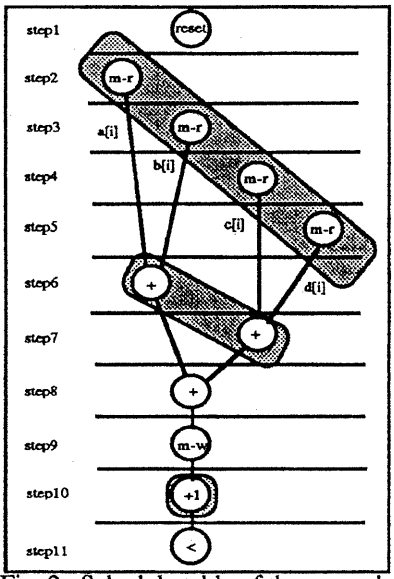


Fig. 2 Schedule table of the example

For the example of Fig. 1 and 2, there are following choices of allocating some new hardware resources to speed up the circuit:

m_1: divide the memory into two smaller memories with two separated common BUSes;
m_2: divide two memories into four smaller memories with their own data BUSes;
m_3: allocate a new ADDER;
m_4: allocate a shadow register *ii* for *i*;
m_5: allocate another shadow registers *iii* for *ii*;

These choices are called ***redesigning methods*** in this paper. If there is no limits on the total amount of the circuit area, the combination of these five redesigning methods will be selected to redesign the circuit to obtain the highest processing speed with the highest processing concurrence. In the case of no enough space to add all the necessary hardware resources, there is a problem of making the best combination of these redesigning methods according to the constraints of time and area, which will be introduced in details in chapter 4.

4. REDESIGN

4.1 Time-Area Plane

A Time-Area plane is used in our discussion, which is a two dimensional plane with two axes, i.e. a time axis in horizontal direction and an area axis in vertical direction as shown in Fig. 3.

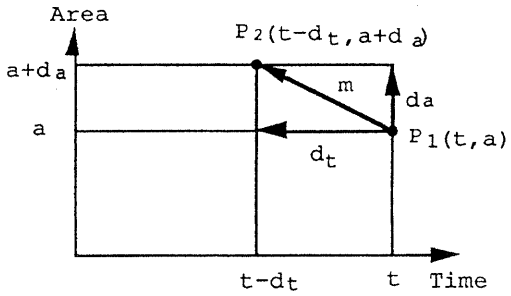


Fig. 3 Time-Area plane

Every circuit has its processing time and area size. It can be represented by a coordinate on Area-Time plane that is a point ***(t,a)***. Here, ***t*** stands for the processing time while ***a*** stands for the area size of the circuit chip. Only the points in the first quadrant have physical meanings because the processing time and area of a circuit would not be negative values. As an example, consider the circuit P_1 in Fig. 3. If we want to make the processing speed of P_1 faster, some extra components would be added to make some operations of circuit P_1 be processed in parallel with other operations. Figure 3 illustrates a situation where a redesigning method ***m*** reconstructes a circuit from P_1 to P_2 to make the processing time shorter, d_t say, and the area of circuit P_1 becomes larger, d_a say, because extra components are added to circuit P_1. The most characteristical parameter is so called ***Time-Area ratio*** of (d_t/d_a), which represents the tendency of line P_1P_2, since the angle between line P_1P_2 and the Time axis can be calculated by Cotangent $^{-1}$ (d_t/d_a). The physical meaning of Time-Area ratio indicates that how much time the circuit can become faster if the area becomes one unit larger. In general, the higher Time-Area ratio is, the better the redesigning method is.

The following one mapping function and two evaluation functions are used to describe the redesigning methods:

a. redesigning mapping function ***R(method,circuit)***;
b. time evaluation function ***T(method)***;
c. area evaluation function ***A(method)***.

Function ***R(method,circuit)*** converts a circuit indicated by ***circuit*** by a redesigning method indicated by ***method*** to create a new circuit which is returned as a result of the function ***R***. Function ***T(method)*** is a time difference becoming faster by redesigning the circuit using method indicated by ***method***. Function ***A(method)*** is the same as ***T(method)***, but returns a difference in area. In Fig. 3, these three functions can be written as:
$P_2 = R(m, P_1)$; $d_t = T(m)$; $d_a = A(m)$;
Since the data-path is going to be redesigned after the scheduling, the total amount of the control steps took by the data-path to do a complete process can be calculated. If an operation takes N_{op} control steps to be executed, and this operation is in a loop block with the loop number N_{lp}, then totally, it takes $N_{op}*N_{lp}$ control steps to execute this operation. If this operation can be processed in parallel with other operations instead of serially by a redesigning method ***m***, then the data-path can become $N_{op}*N_{lp}$ faster. That is,

$$T(m) = N_{op}*N_{lp};$$

The area function $A(m)$ can be calculated by the area of the new hardware resources to be added and the interconnections to be modified.

4.2 Candidate Circuit Region

To consider the distribution of all circuits satisfying the same behavioral specification, we suppose that:

a. P_0 is the most serialized circuit;
b. The number of all the redesigning methods from P_0 is n consisting of $m_1, m_2, ...m_n$;
c. $m_1, m_2, ... m_n$ is sorted in a order form the largest Time- Area ratio to the smallest one;
d. These n redesigning methods are independent to each others.

Two redesigning methods are called independent if

$$T(m_1+m_2) = T(m_1) + T(m_2) \quad (1)$$
$$A(m_1+m_2) = A(m_1) + A(m_2) \quad (2)$$

are satisfied.

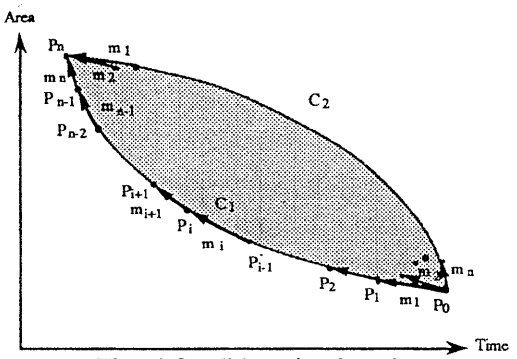


Fig. 4 Candidate circuit region

The situation without assumption d will be discussed in section 4.4. There are n different circuits redesigned from P_0 by one r-method of $m_1, m_2, ... m_n$ as illustrated in Fig.4, i.e. $R(m_1, P_0), R(m_2, P_0), ..., R(m_n, P_0)$. Since m_1 has the largest T-A Ratio, line $P_0R(m_1, P_0)$ is down-left among these n circuits. That is, the circuit $R(m_1, P_0)$ has the best evaluation of time and area. Let $P_1 = R(m_1, P_0)$, then there are n-1 methods of m_2, $m_3, ... m_n$ left to redesign the circuit from P_1. Since method m_2 has the largest $T(m)/A(m)$ among $m_2, m_3, ... m_n$, the method m_2 is used to redesign circuit P_1 to P_2, namely $P_2 = R(m_2, P_1)$. If continuing the redesign in the same way, that is, redesigning the circuit from P_0 by n steps, the r-method with the largest $T(m)/A(m)$ is used at each step to redesign the circuit. As a result, a sequence of n circuits of $P_1, P_2, ... P_n$ can be created and it corresponds to the down-left boundary C_1 of candidate circuit region shown in Fig. 5. If we redesign the circuit from circuit P_0 in the same way but the r-method with the smallest $T(m)/A(m)$ is selected at each step, we can find up-right boundary C_2. From Fig.4 and above discussion, it is clear that all the circuits redesigned from circuit P_0 resides in the hatched region bounded by boundaries C_1 and C_2 in Fig.4. If there are n redesign methods, the number of all possible circuits including P_0 and ones redesigned from circuit P_0 by all the combinations of $m_1, m_2, ... m_n$, is 2^n. Instead of these 2^n circuits, only the n+1 circuits on down-left boundary C_1 are searched for the best result. Consequently, the computational complexity is improved from $O(2^n)$ to $O(n)$.

Figure 5 illustrates the candidate circuit region and the boundaries of the example introduced in chapter 3:

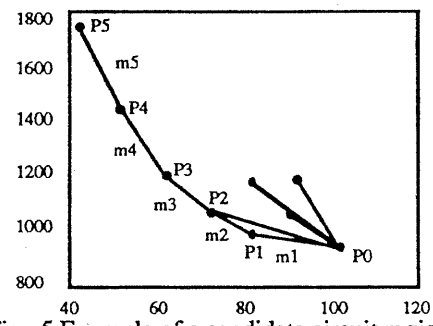


Fig. 5 Example of a candidate circuit region

4.3 Constraints and Result

Beside behavioral specification, a constraint specification is also needed as an input of the system in the form of: 1) Time ≤ C_{time}; 2) Area ≤ C_{area}; and 3) TAbalance = $C_{balance}$ ($0 \leq C_{balance} \leq 1$). Here, C_{time}, C_{area} and $C_{balance}$ are three constants. The balance between time and area is a trade-off problem. These two sides could not be achieved together. Equ. 3) indicates which of these two sides should be considered first and how much the balance is between time and area.

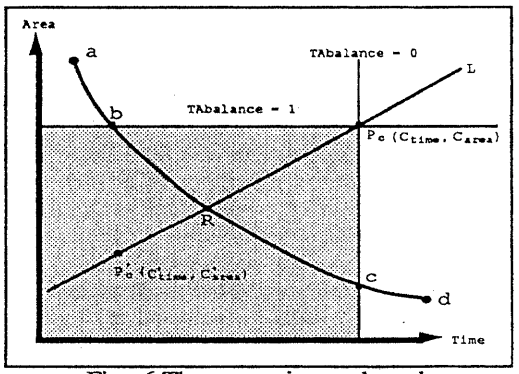


Fig. 6 The constraints and result

The principle of system depending on TAbalance is illustrated clearly in Fig. 6. On the Time-Area plane, curve a-b-c-d described as down-left boundary in chapter 3 stands for the circuits which satisfy a behavioral specification, while the hatched region stands for the circuits which satisfy the time and area constraints. The curve b-c which is the intersection of curve a-b-c-d and the hatched region, satisfies both the behavioral specification and time and area constraints. The point $\boldsymbol{P_c(C_{time}, C_{area})}$ indicates the given time and area constraints and is called ***constraints point***. Straight line L goes through the constraints point $P_c(C_{time}, C_{area})$ with a tendency of $\frac{(1\text{-TAbalance})*C_{area}}{TAbalance*C_{time}}$ indicates a balance relationship between time and area. It is called ***balance line*** and can be represented by,

$$y = \frac{(1\text{-TAbalance})*C_{area}}{TAbalance*C_{time}} *x - \frac{1\text{-TAbalance}}{TAbalance} *C_{area} + C_{area} \quad (3)$$

The intersection point R of the balance line and the curve b-c becomes a result of the design and is output to the designer.

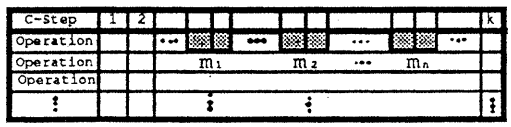
4.4 ResultRegion Without Assumptions

The assumption c guarantees that a circuit redesigned by i redesigning methods from circuit P_0 is determined only by i redesigning methods but the sequence of the redesigning methods. It can be proved easily by the calculations of the coordinate of the circuit redesigned by a combination of i redesigning methods from P_0 ($1 \leq i \leq n$) as,

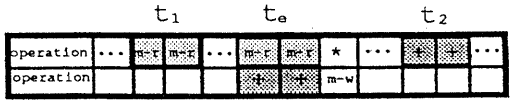
$$\text{Time} = \text{Time of } P_0 - T\left(\sum_{k=1}^{i} m_k\right) = \text{Time of } P_0 - \sum_{k=1}^{i} T(m_k); \quad (4)$$

$$\text{Area} = \text{Area of } P_0 - A\left(\sum_{k=1}^{i} m_k\right) = \text{Area of } P_0 - \sum_{k=1}^{i} A(m_k); \quad (5)$$

Here, $\sum_{k=1}^{i} m_k$ is any combinations of i redesigning methods from $m_1, m_2, \ldots m_n$ and $1 \leq i \leq n$. From eqs. (4) and (5), it is clear that $\sum_{k=1}^{i} T(m_k)$ and $\sum_{k=1}^{i} A(m_k)$ are values and independent on the sequence of these i redesigning methods. Whatever the sequences of these i redesigning methods are, the coordinates of the resultant circuits are the same point on the Time-Area plane which indicates the same circuit.



(a) Independet operations



(b) Dependet operations

Fig. 7 Physical interpretation of schedule table

The assumption of redesigning methods independent to each other can be represented in physical interpretation by Fig. 7(a). In schedule table, any two places where redesigning methods are found are not overlapped at all. Thus, when a redesigning method is used to reconstruct the circuit, it would not affect the calculations of the functions of time and area of any redesigning methods else because these operations are executed at different control steps. Figure 7(b) shows the situation that two redesign methods are dependent to each other, that is, they are overlapped in schedule table. These two cases can be described, for example, on Time-Area plane in Fig. 8.

There are a circuit $P_a(t,a)$, two redesigning methods, m_1 and m_2, and here

$T(m_1) = t_1$; $A(m_1) = a_1$;

$T(m_2) = t_2$; $A(m_2) = a_2$.

Circuit P_b and P_c are reconstructed from circuit P_a by redesigning methods m_1 and m_2. The coordinates on Time-Area plane of P_b and P_c are,

P_b: ($t-T(m_1)$, $a+A(m_1)$) = ($t-t_1$, $a+a_1$);

P_c: ($t-T(m_2)$, $a+A(m_2)$) = ($t-t_2$, $a+a_2$);

If these two redesigning methods are independent, the circuit redesigned by m_2 from P_b is:

($t-t_1-T(m_2)$, $a+a_1+A(m_2)$) = ($t-t_1-t_2$, $a+a_1+a_2$);

and the circuit redesigned by m_1 from P_c is:

($t-t_2-T(m_1)$, $a+a_2+A(m_1)$) = ($t-t_2-t_1$, $a+a_2+a_1$).

They have the same coordinate values so that they are the same point on Time-Area plane, which indicates the same circuit.

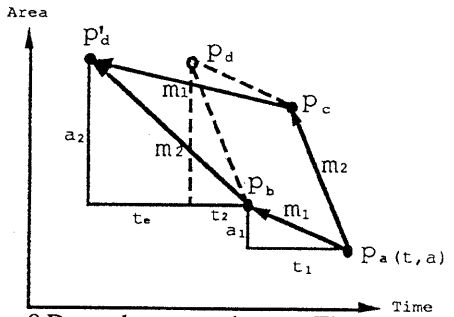


Fig. 8 Dependant operations on Time-Area plane

In the case of two redesigning methods are dependent to each other, the result will still be Pb or Pc, since the operations in control step1 and step2 can not be processed in parallel within only one control step, if only m_1 or m_2 is used to redesign the circuit. But if m_1 and m_2 are used together to redesign the circuit, two control steps, where redesigning methods m_1 and m_2 are overlapped, will become one control step. Let t_e be the control steps becomming faster by merge these two control steps into one control step. Then we have $T(m_1+m_2) = t_1 + t_2 + t_e$, instead of $T(m_1+m_2) = T(m_1) + T(m_2) = t_1 + t_2$. The redesigned circuit moves from P_d to P'_d by an extra displacement t_e in time axis only.

As a result of overlapped redesigning methods, some circuits are out of the candidate circuit region illustrated in Fig. 5. To guarantee that curve C1 is the down-left boundary, some refinements should be made as follows:

1. During the redesigning methods are found, for example in the case like Fig. 7(b) in schedule table, a ***combined redesigning method*** *is found which consists of several* ***primitive redesigning methods****;*

2. After all primitive redesigning methods are found, sort tall the methods including combined and primitive redesigning methods according to the time-area ratio from large to small;

3. Each time a combined method is selected to redesign the circuit, remove all the primitive redesigning methods which is involved in that combined method.

4.5 Redesign Algorithm

We are not able to determine result R of our design in Fig. 6 at the beginning of design, since we do not know curve a-b-c-d until the design is finished. But we know the straight line L by equ. (3) and the curve a-b-c-d can be created from initial circuit P_0 by redesigned initial circuit P_0 step by step. In this section, we present the convergency of our redesigning methodology, that is, the problem of whether the algorithm can be finished.

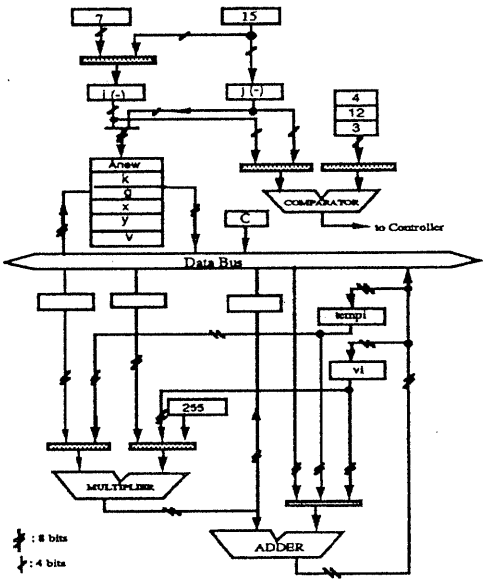
In Fig. 6, while we keep redesigning the circuit, the current circuit will move from P_0 to point R and then pass by point R. The distance from the current circuit to straight line can be calculated since we know the equation of straight L in the form of equ. (3) and the coordinate of current circuit, the processing time and area of the circuit. During the process of redesigning, this distance varies from large value to a minimum one and then starts to become larger. The circuit with a minimum distance to straight line L means that the circuit is nearest to the goal of the design and it is what we want. Then the redesign algorithm can be written as follows:

Redesign Algorithm

step1: current circuit = initial circuit P_0;

current distance = the distance from current circuit to straight line L;
next circuit = R(method$_1$, current circuit);
next distance = the distance from next circuit to line L;
if method$_1$ is a combined redesigning method, remove all primitive redesigning methods of method$_1$;
i = 2;
step2: while (next distance < current distance) do step3;
step3: current circuit = next circuit;
current distance = next distance;
next circuit = R(method$_i$, current circuit);
if method$_i$ is a combined redesigning method, remove all primitive redesigning methods of it;
next distance = the distance from next circuit to line L;
i = i + 1;
step4: output the currant circuit as result;
step5: end.

5. EXAMPLE



(a) Initial data-path of kalman filter

	Area	Time	T/A(100*)
P0	6700	1666	
m1	140	120	85.7
m2	460	300	65.2
m3	140	32	22.8

(b) Evaluations of redesigning methods

Fig. 9 Kalman filter

The behavioral specification written in C of Kalman filter is illustrated as following, which is written in PASCAL in [1],

```
Kalman( )
 {
    i<3>;     j<3>;     vi<15>;  tempi<15>;
    Anew[255]<15>; k[255]<15>;  g[127]<15>;
    x[15]<15>;  y[12]<15>;  v[3]<15>;

     j = 15;
     do {
          x[j] = 0;
          j = j - 1;
     } while ( j >= 4 );
     i = 15;
     do {
          j = 15;
          tempi = 0;
          do {
               tempi = tempi + Anew[i@j]*x[j];
               if ( j <= 12 )
                    tempi = tempi + k[i@j]*y[j];
               j = j - 1;
          } while ( j >= 4 );
          x[i] = x[i] + 255*tempi;
          i = i - 1;
     } while ( i >= 4 );
     i = 7;
     do {
          vi = 0;
          j = 15;
          do {
               vi = vi + g[i@j]*x[j];
               j = j - 1;
          } while ( j >= 4 );
          v[i] = vi*y[i];
          i = i - 1;
     } while ( i > 3 );
 }
```

In this example, there are three redesigning methods:

- m1: allocate a shadow register for counter j;
- m2: divide memory into two memories;
- m3: allocate a shadow register for counter i.

The initial data-path and the evaluations of these three redesigning methods are shown in Fig. 9. Figure 10 illustrates the most parallel circuit.

6. CONCLUDING REMARKS

We have presented our efficient methodology of a data-path synthesis based on algorithmic description under the constraints of processing time and area of the circuit. It allows simultaneous data-path allocation while trading off hardware resources against execution time according to the user-specified constraints and the balance between time and area. As this

design system is still in developing, hopefully, some new features will be added to our program to create a more intelligent and powerful VLSI circuit design system.

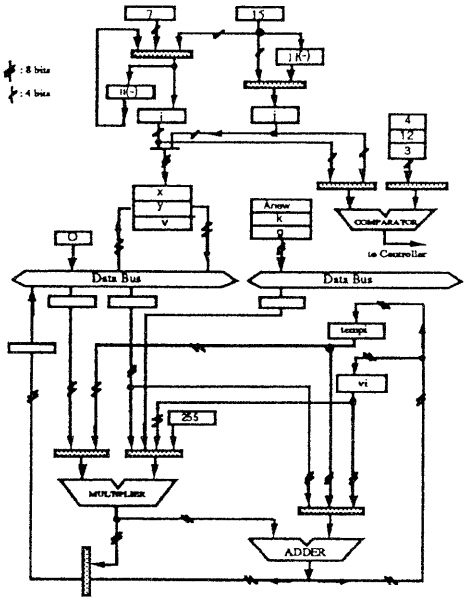


Fig. 10 The most parallel circuit of Kalman filter

REFERENCE

[1] D. E. Thomas, E. D. Langnese, R. A. Walker, J. A. Nestor, J. V. Rajan and R. L. Blackburn: Algorithmic and Register-Transter Level Synthesis: The System Architect's Workbench, Kluwer Academic Publishers, 1988.

[2] A. C. Parker, M. Mlinar and J. Pizarro: HAMA: A program for data-path synthesis. 23th ACM/IEEE Design Automation Conference. June 1986, pp. 482-485.

[3] P. Marwedel,: A new synthesis algorithm for the MIMOLA software system. 23th ACM/IEEE Design Automation Conference. June 1986, pp. 482-485.

[4] T. J. Kowalski et al.: The VLSI design automation assistant: From algorithms to silicon, IEEE Design & Test, Aug 1986, pp. 33-43

[5] H. Trickey: Flamel: A High-Level Hardware Compiler, IEEE Transactions on Computer Aided Design, vol. CAD-6, no. 2, March, 1987, pp. 259-269

[6] M. C. McParlend, A. C. Parker, R. Camposano: Toturial on High-level Synthesis, 25th ACM/IEEE Design Automation Conference. 1988, pp330-337.

[7] B. M. Pangrle and D. D. Gajski: Slicer: A state sybthesizer for intelligent silicon compilation, in Proc. IEEE Int. Conf. Computer Design, Oct. 1987.

[8] Pierre G. Paulin, John P. Knight: Force-Directed Scheduling for the Behavioral Synthesis of ASIC's, IEEE Tans. Computer-Aided Design, vol. 8, No. 6, June 1989, pp. 661-679

[9] S. Devadas, A. R. Newton: Algorithms for Hardware Allocation in Data-Path Synthesis, IEEE Tans. Computer-Aided Design, vol. 8, No. 7, July 1989, pp. 768-781

[10] R. Camposano, R. A. Bergamaschi: Aynthesis using path-based scheduling: Algorithms and exercises, 27th ACM/IEEE Design Automation Conference. 1990, pp 450-455