Hot Potato Techniques in High Level Synthesis

Miodrag Potkonjak

Sujit Dey

C&C Research Laboratories, NEC USA, 4 Independence Way, Princeton, NJ 08540

ABSTRACT

This paper introduces hot potato high level synthesis transformation techniques. These techniques add deflection operations in the behavioral description of a computation in such a way that the requirements for two important components of the final implementation cost, the number of registers and the number of interconnects, are significantly reduced. Moreover, it is shown how hot potato techniques can be effectively used during high level synthesis to minimize the partial scan overhead to make the synthesized design testable. The paper presents theoretical foundations, efficient algorithms and experimental results which indicate the wide application range and effectiveness of the new transformation technique for behavioral level synthesis.

1.0 Introduction

1.1 Transformations in High Level Synthesis

Computational transformations are alternations in the structure of control-data flow graph (CDFG) so that the functionality of the initial specification is maintained [Fis88, Wal91]. Transformations are standard optimization tool in many research and commercial domains, including compilers [Fis88], database systems [Ull89], artificial intelligence systems [Dar81], symbolic mathematics [Cha91], VLSI signal processing [Par89], logic synthesis [Bra84], high level synthesis [Wal91] and emerging system level synthesis [DeS94]. Interestingly, the exploration of transformations in each of the mentioned domains is dominated by a different set of specific optimization approaches.

For example, compiler transformations techniques are usually conceptually simple and run time efficient procedures capable of handling exceptionally large programs. The application of transformations in database query optimization systems can be divided in two groups. The first group uses strategies which are similar to the compilers strategy [Ull89], the second group relies on generic powerful optimization algorithms, such as simulated annealing, to extensively optimize queries applied on large databases [Ioa87]. Transformations are applied in the VLSI signal processing domain mainly manually on specific important and widely used instances of applications so that throughput [Bla85, Par89] are optimized, while the regularity and locality are preserved. Techniques which extensively use comprehensive mathematical knowledge is manually applied on both numerically intensive and conditional intensive application in design of algebraic and numerical mathematical algorithms [Gol89].

Logic synthesis treatment of boolean and algebraic transformations is most often based on use of elaborate script-based techniques [Bra84]. The scripts are most often results of extensive experimentation. While the initial application of transformations in high level synthesis was mainly based on the use of compiler like strategies [McF92], recently emphasis has been shifted to target statistically validated objective functions which model area [Pot92], fault tolerance overhead [Gue93] and power [Cha92].

Transformations used in all domains are mainly based on a very few common principles. All currently used syntax transformations can be classified into three groups: control flow transformations, transformations for alternation of the organization of computations and dataflow computations. Syntax transformations are those transformations which do not use semantic knowledge about the context in which the computation is conducted.

Control flow transformations itself can be classified into four groups: procedure transformations, loop transformations, conditional transformations, and external signals transformations. Among procedure transformations most widely used is in-lining [Fis88]. The most popular loop transformations include loop fusion and fission, loop permutation and software pipelining, and among conditional transformations are a variety of code motion techniques. Transformations which address external signals (e.g. interrupts, synchronization signals, and inputs and outputs) are mainly used in real time systems [Ott84, Wei84].

Among transformations which target changes in the organization of computations the most widely used in substitution of constant multiplications by shifts and additions [Rab91].

Dataflow transformations encompass two large groups: algebraic transformations [Fis88] and redundancy manipulation transformations [Fis88, Pot92]. Algebraic transformations is probably the most widely studied class of transformations and are based on exploration of algebraic axioms and theorems in a variety of algebraic structures. Typical examples include commutativity, associativity, distributivity, the inverse and the zero element laws. Redundancy manipulation transformations are mainly targeting elimination of redundant computations and include often used transformations such as common subexpression elimination, constant propagation and copy propagation [Fis88]. To the best of our knowledge, currently, only one transformation targets introduction of redundancy, as a mean for speed optimization.

1.2 Transformations in High Level Synthesis

In this paper, we introduce a new redundancy replication transformation. Interestingly, it does not improve speed of execution, but it often reduces the implementation and testability cost. While the new transformation is general and can be used in almost all mentioned computational domains, it is e3specially well suited for high level synthesis since it provides an effective means to manipulate the relationship between the behavioral and structural descriptions of the application specific system.

Many operations used in a computation structure has an identity element associated with it. For instance, addition and subtraction operations have an identity element zero, while multiplication and division operations have an identity element one. If one of the inputs of an operation is v, and the other input is the identity element of the operation, then the output of the operation remains v. Adding such an operation op between two operations op_1 and op_2 has the effect of deflecting the result of op_1 to op, before reaching op_2; hence, we term such an operation a **deflection operation**. A deflection operation can be added anywhere in a computation structure without changing the functionality of the computation. In particular, adding a deflection operation after a variable v has been computed, keeps the behavior invariant, because the output of the deflection operation is also v.

Techniques which deflect packets of data, called hot potato techniques, have been applied earlier in the computer networks domain [Bar65]. We introduce a new class of transformation techniques, termed **hot potato techniques**, based on adding deflection operations to the computational structure, while preserving the functionality of the computation.

In addition to demonstrating the effectiveness of the new transformation mechanism for improving resource utilization (i.e. reducing cost of design), we also apply the new transformation technique to reduce the partial scan cost needed to make the resultant design testable.

1.3 Motivational Examples: Why More at Behavioral Level Means Less at the Structural Level

This subsection presents five small, insightful motivational examples which are used to provide the intuition behind the hot potato techniques. The assumed computational model is synchronous data flow [Lee87] which is well suited for addressing numerous applications in video and image processing, DSP, control, graphics, communications and many other domains. In the remainder of the paper, if otherwise not stated, we will assume that the underlying hardware model is the dedicated register file model. This model assumes that all registers are grouped in certain number of register files (each register file contains one or more registers) and that each register file can send data to exactly one execution unit. At the same time each execution unit can send data to an arbitrary number of registers files. This model is used not only in a number of high level synthesis systems [Rab91], but also in many manual ASIC and general purpose datapaths. In all motivational examples, for the sake of simplicity, it is assumed that all operations take one control step for their execution.

1.3.1 Optimizing the Number of Interconnects

Figure 1a shows an example which illustrates how a deflection operation can be used to reduce interconnect requirements, while preserving the resource utilization of other components of the datapath, and maintaining the throughput requirements. Let's denote an addition by +, a

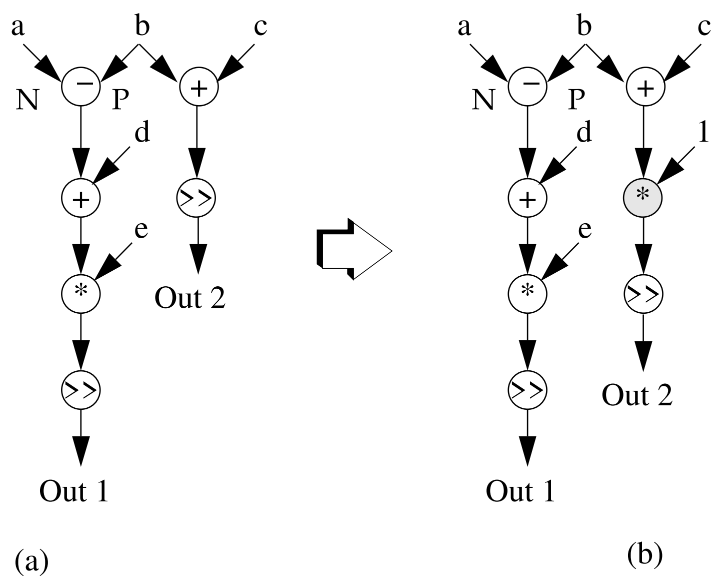


Figure 1: Reducing the number of interconnects by adding Deflection Operations

subtraction by -, a multiplication by *, a shift by >>, and input and output by IN and OUT. Also let A, S, M and SH denote respectively adder, subtracter, multiplier and shifter used in the datapath. The left and right inputs to operations and execution units are denoted by L and R respectively. One unit of each type is sufficient to implement the computation. Finally, the positive and negative input of a subtraction and subtracter are labeled by P and N respectively.

Inspection of the computation graph on Figure 1a indicates a total of 12 edges: IN -> P-, IN -> N, IN -> L+, IN -> R*, - -> R+, + -> L*, * -> >>, + -> >>, and two instances of both IN -> R+ and >> -> OUT. At the RT level there are ten corresponding interconnects, since two instances of IN -> R+ can share the interconnect IN -> RA and the instances of >> -> OUT can share the interconnect SH -> OUT.

Figure 1b shows a computation functionally isomorphic to the computation of Figure 1a, obtained by adding a deflection operation, multiplication by 1. An inspection of this computation indicates that the new structure has 14 edges. However, now only nine interconnects are sufficient at the RT level: IN -> PS, IN -> NS, IN -> RA, IN -> LA, IN -> RM, S -> LA, A -> LM, M -> SH and SH -> OUT, and therefore a better hardware sharing of interconnects is achieved. Note that neither the critical path nor the number of required execution units has been affected.

1.3.2 Reducing the Number of Registers

Figures 2a and 2b show two functionally identical computational structures. The structure on Figure 2b is derived from the structure on Figure 2a by adding a deflection operation on the edge V6. The critical path in both structures is identical 4 control steps. The same number of execution units:

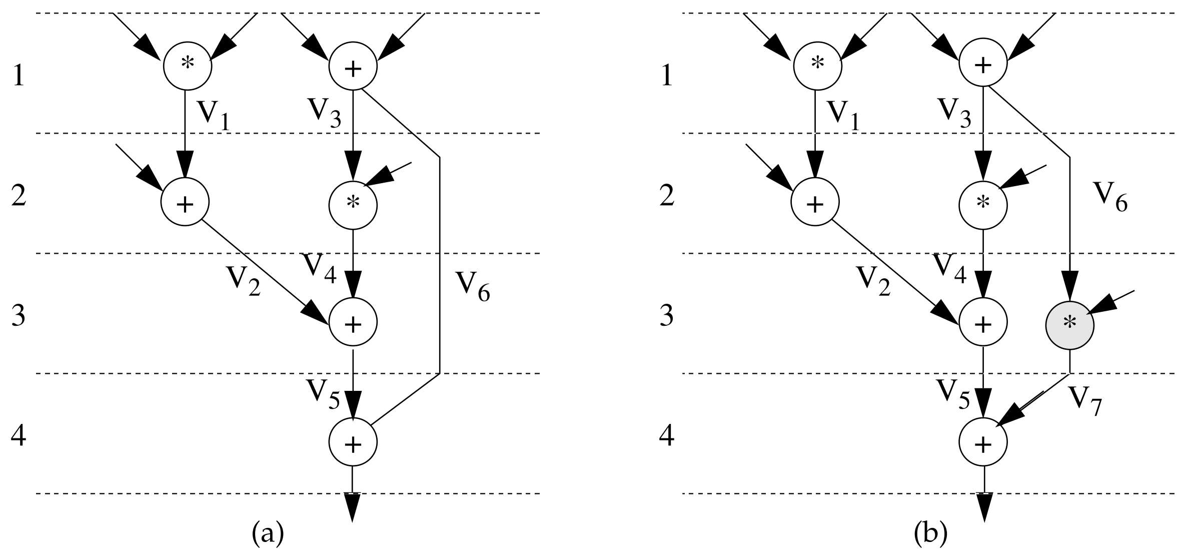
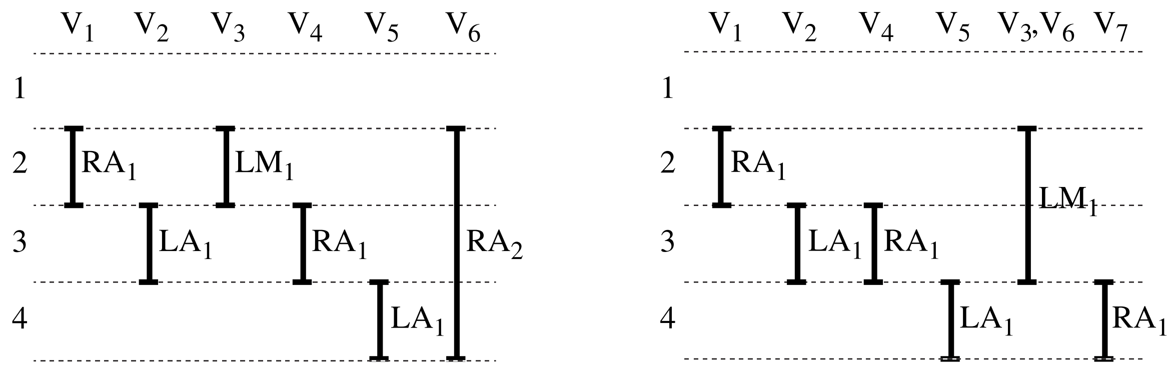


Figure 2: Reducing the number of Registers using Hot Potato Transformations

one multiplier and one adder are sufficient for both implementation. A simple analysis also indicates that the structure from Figure 2b needs one interconnect less, since the interconnect + -> R+, is replaced by interconnects which reuse already allocated resources. An analysis of the lifetimes of all intermediate variables (note that all input and output variables in both structures have identical lifetimes) is shown in Figures 3a and 3b respectively. The register assignment is also shown, where LA1, RA1, LM1, RM1, represent the left and right registers of adder A and multiplier M used in the datapath. Note that in Figure 2a, variable V6 has overlapping lifetime with variables V1 and V4, and hence cannot share the same register RA1 assigned to variables V1 and V4. In the transformed structure shown in Figure 2b, the introduction of the deflection multiplication operation reduces the lifetime of the variable V6 at the expense of introducing a new variable V7. However, V6 can now be assigned to register LM1, while the new variable V7 can share register RA1 with the variables V1 and V4. Consequently, the transformed structure in Figure 2b requires one less register than the original structure in Figure 2a, as illustrated by Figures 3b and 3a, respectively



(a) (b)

Figure 3: Lifetime for variables for examples from Figure 2a and 2b

1.3.3 Register Minimization when arbitrary hardware model is assumed

Although in the rest of the paper we will invariably assume the dedicated register file model, all presented techniques and algorithms can be easily generalized with respect to resource utilization and testability and applied to an arbitrary hardware model. Actually, exploration of hot potato transformations is often easier and more effective in unrestricted hardware models where the interconnect network is not subject to a set of constraints. We illustrate our claims and provide the intuition behind them using the example shown in Figure 4a.

The figure shows an example which demonstrates the ability of hot potato techniques to reduce the number of required register in the unrestricted hardware model where each register can be connected to an arbitrary number of execution units. The timing constraints include the assumed available time of three control steps, and need that results O1, O2 and O3, are available in the first, second and third control step respectively. Figure 5 shows the life times of all non-output variables: V1, V2, and V3. The conflict graph, where edges indicate which variables are simultaneously alive, is shown in Figure 6a. Obviously, we need at least three registers, as indicated by the datapath shown in Figure 7a. The datapath has four interconnects and three registers. However, if we add a deflection transfer operation as shown in Figure 4b, both the interconnect and register requirements are reduced. As can be seen from the corresponding conflict graph (Figure 6b), and the corresponding datapath (Figure 7b) now only two registers and three interconnects are required. Note that as indicated by the new conflict graph we now have more variables, but their lifetimes are such that the conflict graph can be colored using only two colors.

1.3.4 Adding deflection operation to reduce partial scan overhead

Recently, several high level synthesis approaches have been proposed, exploring the relationship between the high level synthesis process and both Built-In-Self-Test (BIST) [Geb89, Avr91, Pap91,

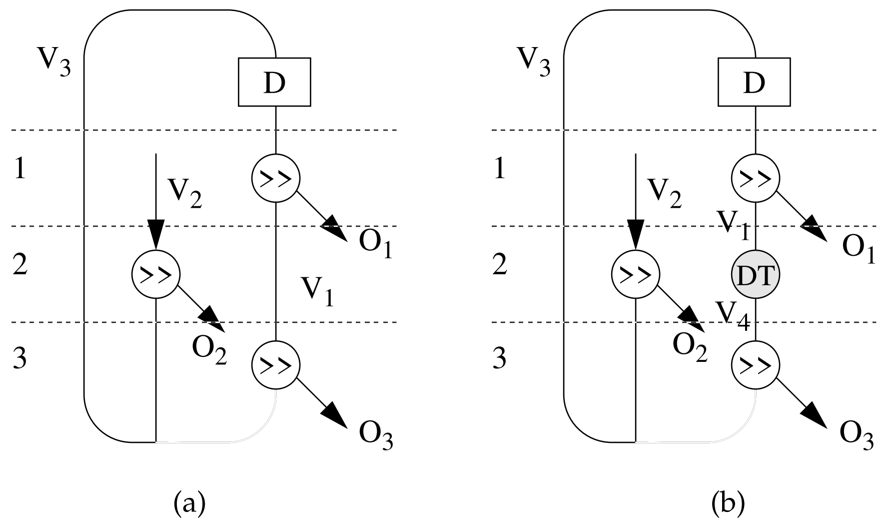


Figure 4: Saving registers and interconnects when no restrictions on interconnect network is superimposed. DT represents added deflection operation, which in this case just transfers data between two registers in the unrestricted hardware model.

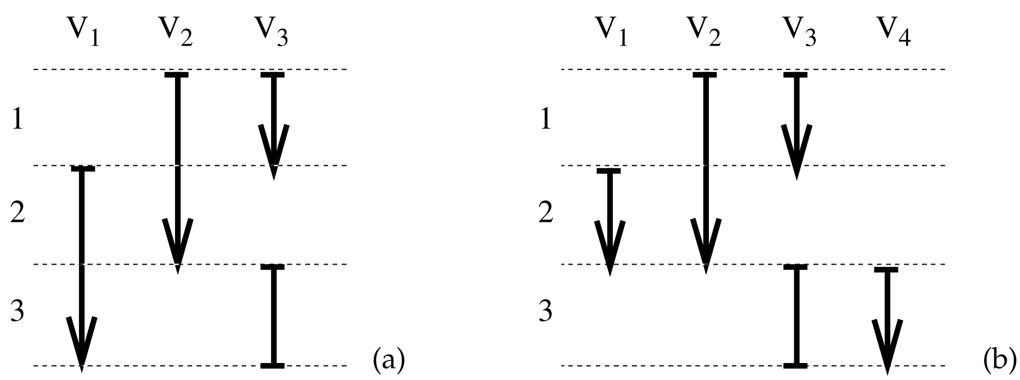


Figure 5: The lifetimes of the intermediate and input variables for the examples from Figure 4a an d Figure 4b.

Har94] and Automatic Test Pattern Generation (ATPG) methods [Maj92, Lee93, Dey93]. Also, Bhatia and Jha reported high level synthesis techniques which target hierarchical test pattern generation [Bha94].

We target gate-level sequential automatic test pattern generation (ATPG) testing methods. It is known that presence of loops in a sequential circuit makes sequential ATPG very hard. Partial scan is an effective technique to break loops by scanning a subset of flip-flops/registers [Che90, Lee90]. It has been demonstrated that it is possible to generate designs which do not compromise resource utilization while significantly reducing partial scan overhead by paying early attention to formation

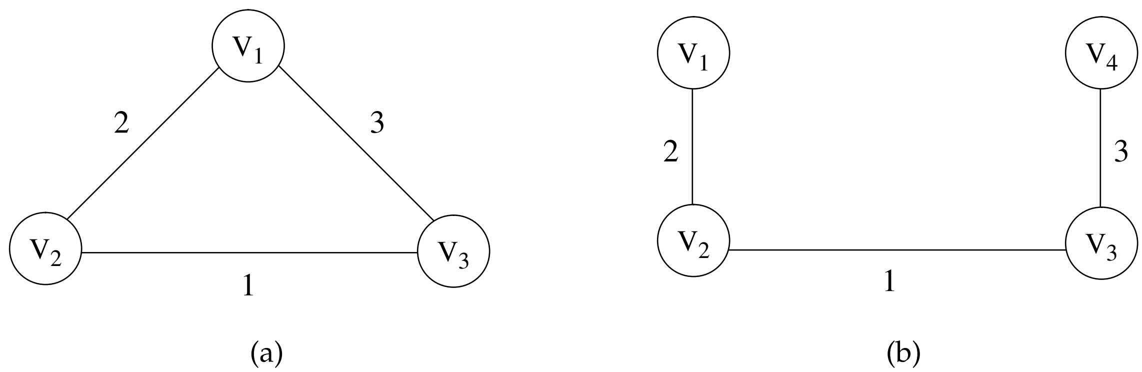


Figure 6: The register assignment conflict graph for the examples from Figures 4a and 4b. The number on the edge represents the control step in which both variables are simultaneously alive.

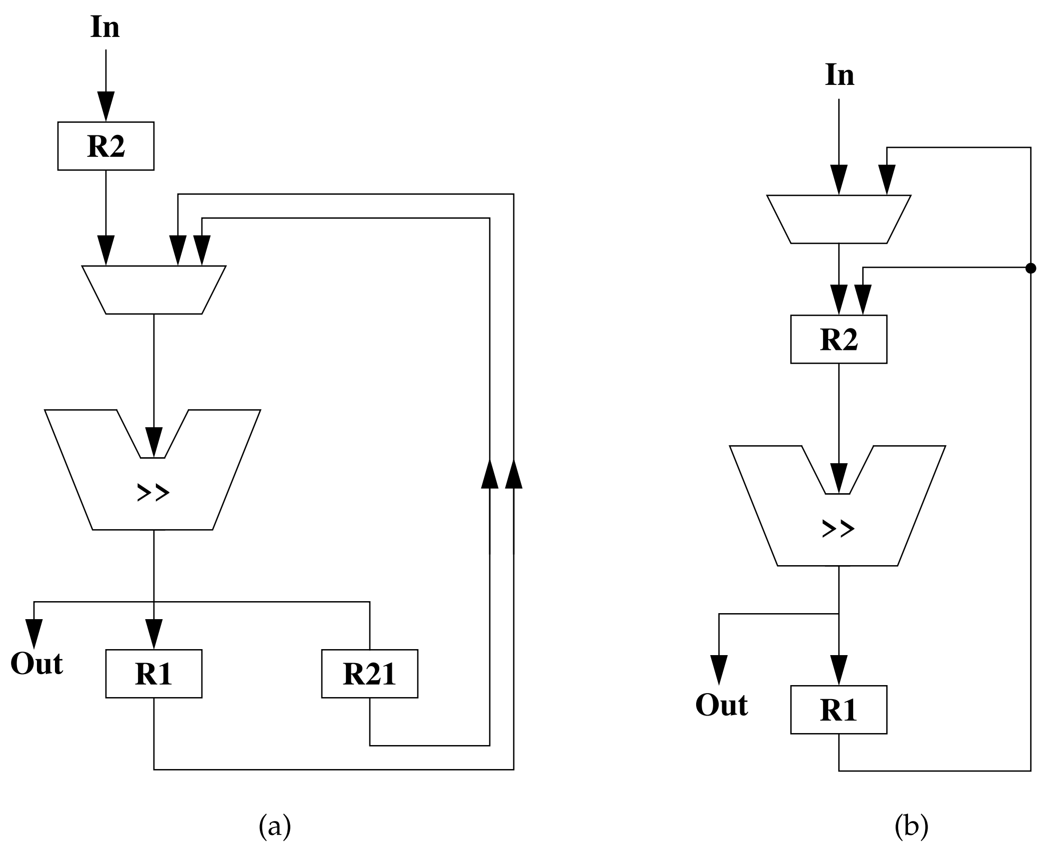


Figure 7: The datapaths for examples from Figures 4a and 4b respectively

of loops in the datapath. However, until now, the influence of only two high level synthesis tasks, scheduling and assignment, on the testability of the design has been explored [Maj92, Lee93, Dey93].

Figures 8 and 9 show two ways how the introduction of a deflection operation can reduce partial

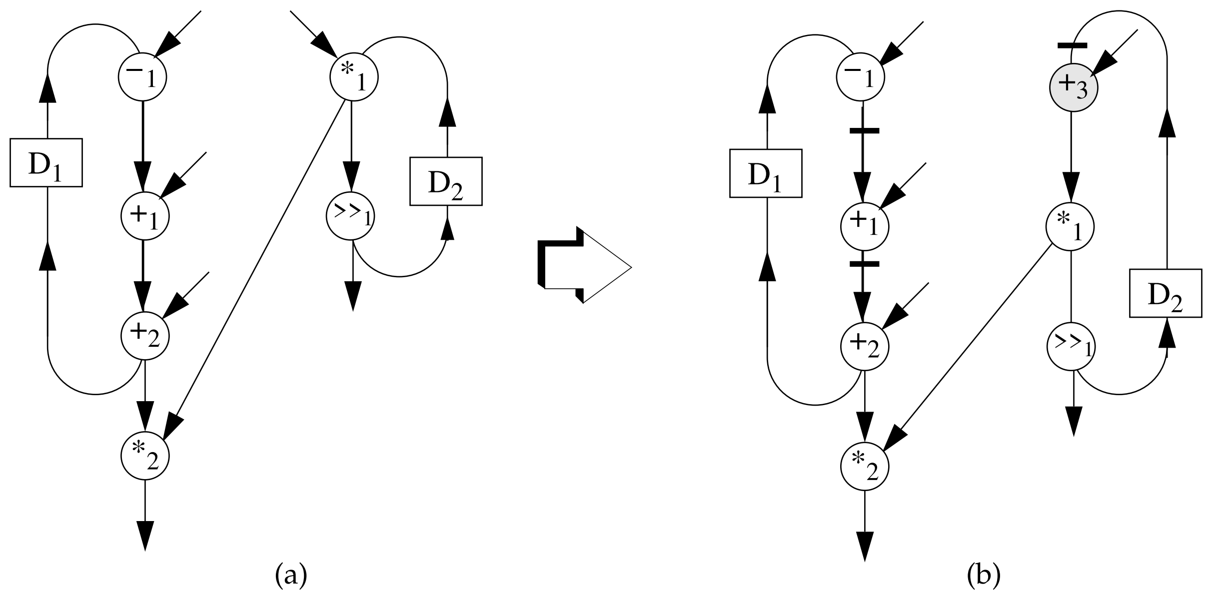


Figure 8: Using Deflection Operation For Testability

scan overhead, by preventing the formation of loops in the datapath. In the first approach, a loop in the datapath is broken using the register in the register file of newly introduced deflection operation. The deflection operation is selected so that a scan register in its register file can be reused to break as many loops as possible in datapath. The second approach is characterized by the use of deflection operation to reduce the lifetime of a variable which is stored in scan registers, so that other variables can be scanned, and the loops to which they belong are broken.

The example from Figure 8 has two CDFG loops [Dey93] which are bound to produce two loops in the datapath. Since the first loop has only addition and subtraction operations, while the second loop has only a shift and a multiplication, in the dedicated register file model it is not possible to use the same scan register for breaking both the loops. However, if we add as a deflection operation an addition with 0, as shown in Figure 8b, and select the scan variables as indicated by the crossed edge lines, all scan variables can share the same register. The resulting datapath does not have loops. An enumeration of the required interconnects for the two implementation shows that the number of interconnects needed does not change. Consequently, resource utilization is maintained, while partial scan cost is reduced by adding deflection operation.

Consider the example shown in Figure 9 with an available time of 4 control steps. Therefore all operations are on the critical path. For each operation, we indicate the control step in which the operation is bound to be scheduled. The example has two CDFG loops: (+1) -> (-1) -> (*1) -> (D1) -> (+1) and (+3) -> (*2) -> (D2) -> (+2). It is easy to see that the design requires at least two scan registers

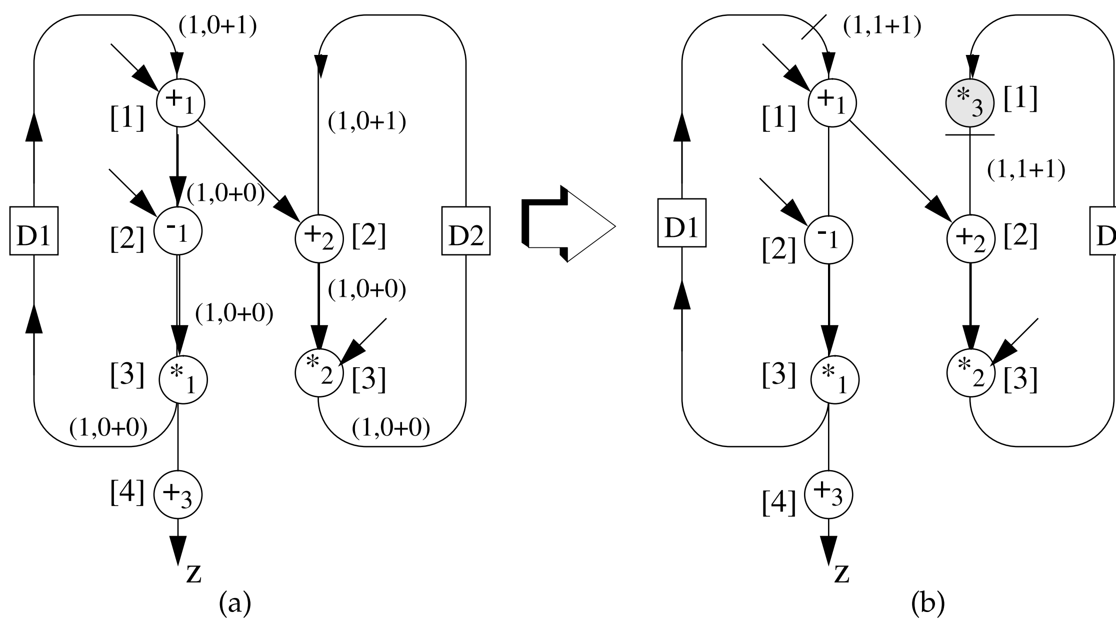


Figure 9: Using Deflection Operations for Testability by Reducing Lifetimes of scanned variables

to ensure that all CDFG loops are broken, regardless of the scheduling and assignment. This is so because it is impossible to share the same scan register to break both loops. If a defection operation *3 is added, the lifetime of variable D2 in the register file of the adder is reduced. Now the variables corresponding to the crossed edges, D1 -> +1 and *3 -> +2, can be stored in the same scan register while breaking both the CDFG loops. Therefore, the partial scan overhead is reduced.

2.0 Related Research

The idea of operations or data replication has been proposed and successfully used in several computer engineering domains, including computer networks, distributed and parallel computing, computer architecture and several CAD areas. It appears that the idea of replicating data (messages) was first proposed in computer networks research by Baran in 1964 [Bar64]. After a long hiatus, the idea was revived and widely studied recently, and practically used in a number of models and implementations of computer networks protocols [Szy90, Haj91].

More recently, in the related distributed computer architecture research on wormhole routing, it was suggested and demonstrated that replicating messages can be effectively used for avoiding or reducing probability of deadlocks [Dal87, Lin91]. The intuitive and detailed exposure of wormhole routing is presented in [Dal90]. Another interesting and powerful idea in the computer architecture domain on enhancing performance by adding dummy hardware units was proposed by Patel [Pat78]. He suggested that the minimum achievable latency in nonlinear pipelines can be significantly reduced by introducing additional delays in feedback paths. Finally, recent architecture, operating systems and compiler literature shows a flurry of research in techniques for enhancing performance of general purpose computers by speculative execution of both branches of conditional before the result of the condition is known [Ber91, Rog92, Smi92]. Often the effectiveness of this technique is augmented with conditional branches prediction mechanisms [McF86, Fis92]. Another architectural and compiler technique which dynamically explores speculative, not necessarily needed, data transfers is pre-fetching, using either software [Cal91, Kla91, Mow91] or hardware support [Che92]. Replication of data is also popular technique for performance improvement in distributed data base systems [Yu83, Gla89].

In logic synthesis, the idea of replicating logic gates so that critical path is minimized during partitioning of large designs was first proposed by Lawler at al. [Law69]. Later it was successfully modified and applied in several logic synthesis projects [Rus71, Cri91]. Recently, Hwang and El Gamal presented an optimal algorithm for replication when the goal is min-cut partitioning of a circuit of logic gates [Hwa92]. The relationship between redundant logic and the delay in designs at the logic synthesis level is discussed by Keutzer, Malik, and Saldanha [Keu91], and Cheng and Enterna [Che93].

The replication of arithmetic operations using algebraic laws so that critical path can be reduced was first proposed by Lobo and Pangrle [Lob91]. While their application was restricted to only one algebraic identity and one example (5th order wave digital filter), Potkonjak and Rabaey using theory of quantitative enabling transformations gave a firm theoretical background to operation replication using algebraic laws and demonstrated how an arbitrary linear computation can be calculated in both maximally fast and arbitrarily fast fashion [Pot92]. More recently, Srivastava and Potkonjak [Sri94] enhanced this technique and showed how the maximally fast technique based on extensive common subexpression replication can be used as a building block for the integrated set of transformation which optimize simultaneously throughput and latency, while in many cases also reducing area and power requirements.

Differences in domain, goal, and probabilistic nature of phenomena in computer network and distributed computing, and static (compile time) nature of research presented here, is very significant. The only similarity, besides the same conceptual goal and means, is the adapted name (hot potato). While nonlinear pipeline optimization techniques are used to enable high throughput by introducing additional hardware, the hot potato approaches in this paper introduce operations at the behavioral level, so that while throughput is maintained, the hardware requirements are reduced.

Finally, it is for first time that addition of new operations is used not just for the optimization of speed and area metrics, but also testability. While high level synthesis research was restricted until now on studying the influence of scheduling and assignment on testability, this paper demonstrates that transformations can be effective to ensure high resource utilization and easily testable datapaths.

3.0 Optimizing Resource Utilization using Deflection Operations

3.1 Optimization Algorithm

As was indicated by the motivational examples, two distinct components of the datapath, namely interconnect and registers, can be optimized using hot potato transformations techniques. There are at least two distinct ways how deflection operation can be used in high level synthesis. The first option is to integrate the addition of deflection operations within scheduling (or within an integrated approach which combines allocation, scheduling and assignment) algorithms in one compact tool. The advantage of this approach is that as we are proceeding with scheduling and assignment, the structure of CDFG can be dynamically altered so that resource allocation and time constraints are not violated.

The second alternative is to treat the introduction of deflection operations as a standard transformation, and apply it before scheduling is performed. This implies a static approach to the optimization problem of reducing interconnect and registers using deflection operations. The advantages include algorithmic and software modularity, and the elimination of the need for complicated bookkeeping associated with scheduling and assignment while deflection operations are dynamically added.

We believe that it is important to preserve the advantages of both approaches, while reducing their negative side effects. Therefore, hot potato transformations is applied as a fully modular step independent of scheduling and assignment. However, in order to utilize feedback information from scheduling and assignment we apply hot potato transformations within an iterative high level synthesis framework. The reasons for difficulty of assignment and scheduling to satisfy constraints are used to guide addition or removal of deflection operations in the next iteration.

The overall flow of the hot potato transformations algorithm for optimization of resource utilization is shown in Figure 10. The initial allocation of execution units is provided by standard high level synthesis. We have been using the Hyper high level synthesis system [Rab91], mainly because of its full software modularity, and already existing mechanism for the feedback information flow between different high level synthesis tasks. After each scheduling and assignment step, the process can be terminated either by the user or due to the completion of the prespecified number of attempts.

Recent studies of a large number of examples strongly indicate that most often the area of an ASIC implementation is dominated by the area of interconnect [McF91]. The primary goal during the introduction of deflection operations is to transform the computation so that the resource utilization of the interconnect network is improved. The goal is to eliminate as many interconnects as possible,

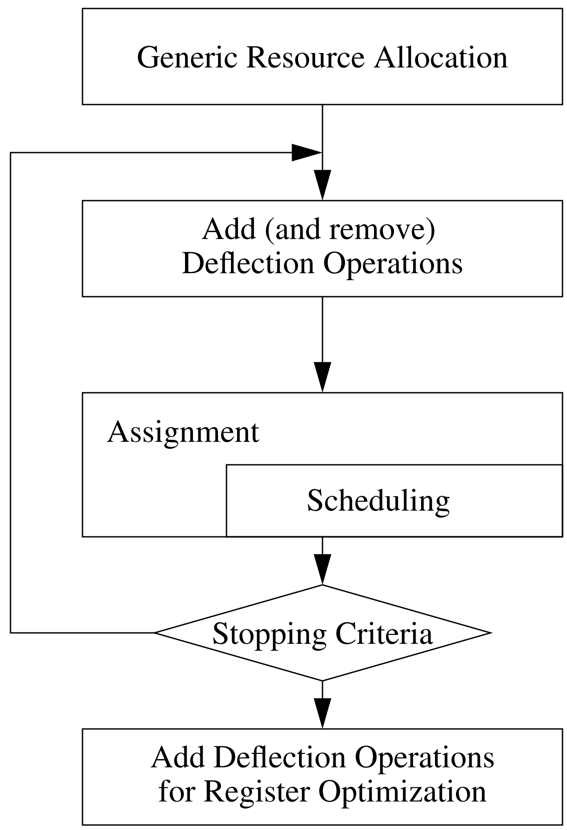


Figure 10: Hot Potato Techniques for Resource Utilization Optimization: The organization of Algorithmic Steps

while not reducing the likelihood for scheduling within the given resource (execution units) and timing (available number of control steps) constraints. Every time a decision is made to introduce a new deflection operation, we have to decide the place (edge) on which the operation will be introduced, and the type of the introduced deflection operation.

We add or delete one deflection operation at a time. In order to avoid producing only locally optimal solution, at each step we have to consider not only immediate effects of the just introduced deflection operation, but also consequences on future potential introduction of other deflection operations. The situation is further complicated due to the incomplete knowledge about effects of the later on future scheduling and assignment steps.

In order to properly address both intermediate and long term consequences, the place where a deflection operation will be introduced is decided according the following criteria which are listed in a decreasing order of importance.

1. Edges which are not on the critical paths, and which do not significantly reduce scheduling mobility [Wal91] of operations.

2. Edges which corresponds to rarely used interconnects.
3. Edges which can be replaced by combination of two other transfers (due to a deflection operation) so that new edges have high likelihoods of using already existing operations.
4. Transfers which are not promising candidates for being used as a component of deflection operations for reducing interconnect requirements in other parts of computations.
5. Deflection operation which will result in division of long lifetimes of variables.

The change in scheduling mobility is calculated as the reduction of sum of weighted slacks of all operations. The frequency of how often an interconnect is used is calculated as the ratio of the number of instances of transfers at behavioral level divided by the product of control steps and the targeted number of interconnects at the RT level, as indicated by the scheduler. The final criteria targets the reduction of register requirements and is based on the empirical intuitive observation that variables with very long lifetimes can rarely share registers with other variables.

Once the place of introduction of a deflection operation is decided, the type of the introduced deflection operation is decided using the following three criteria:

1. Operations which reduces the mobility least. Note that operations with longer execution times reduce mobility proportionally more.
2. Operations which are not competing for the critical resources (execution units) which are bound to be scheduled in the same control cycles. The level of competition is calculated as the overall of ASAP-ALAP times of operations [McF92]
3. Operations whose transfers are not good candidates for elimination as indicated by the "place" criteria.

After each assignment and scheduling attempt, the HYPER scheduler provides as feedback information an ordered list of resources which are often in high demand / low supply situation, and therefore caused scheduling and assignment difficulties. This information is used as indication which deflection operation to delete, and to update the preference that a deflection operation of a particular type is introduced.

The optimization of registers is a significantly more complex problem due to the fact that before scheduling and assignment are not completed, the information about lifetimes of variables is not available. Therefore it is difficult to estimate eventual register requirements before scheduling is done. Moreover, even when scheduling and assignment of operations has been completed, just assigning variables to registers within a register file is an NP-complete problem by itself, because it corresponds to the coloring of a circular arc problem [Sto90].

As we already indicated, one of the criteria, used during the addition of deflection operations for reducing interconnect requirements, targets reduction of registers. After the optimization of interconnect, and the synthesis of datapath using scheduling and assignment has been completed, the minimization of registers using deflection operations is attempted. During this step we impose the constraint that addition of new interconnects is not allowed.

Each register in the data path is targeted for removal, in a decreasing order of preference, according to the following three criteria:

(1) The number of variables which share the register. The registers with one variable are targeted first, followed by registers with the smallest number of variables;

(2) The sum of lifetimes of all currently stored variables in the register;

(3) The registers in register files which have the largest number of incoming interconnect, because the incoming interconnect has to be used during the introduction of deflection operations and it enables the transfer from the largest number of other register files.

The algorithm for area minimization using hot potato techniques can be summarized using the following pseudocode:

add_deflection_ops_to_reduce_the_number_of_interconnect_and_registers()

```
1.  resource_allocation();
2.  while (stopping criteria_1){  /* reduce interconnect */
3.        select edge for adding a deflection operation;
4.        select type of the  deflection operation;
5.        check  the value  of adding  the deflection operation;
6.        if (value > threshold)  augment CDFG with the deflection operation
7.        else reject the deflection operation and label the edge in the CDFG;
8. }
9.  while (stopping criteria_2){ /* reduce the number of registers */
10.       select edge for adding a deflection operation;
11.       select type of the  deflection operation;
12.       check  the value  of adding  the deflection operation;
13.       if (value > threshold)  augment CDFG with the deflection operation
14.       else reject the deflection operation and label the edge in the CDFG;
15.  }
```

The stopping criteria is user defined run time limit or the number of added deflection operations. The labeling of edges in steps 7 and 14 is required to prevent successive attempts to add the same type of operation on the same edge. When all types of operations are evaluated as candidates for addition on all edges, the labeling is reset.

3.2 Effectiveness of hot potato techniques: Low Throughput, Low Cost Designs

The intuition behind the claim about the effectiveness of hot potato techniques for area optimization is shown in Section 2, and the supporting experimental evidence is presented in Section 5. In this subsection, we will further enhance both aspects, intuition and experimental evidence, by presenting the following theorem, which analyzes the effectiveness of hot potato techniques when low throughput, low cost applications are targeted.

Before stating the theorem, we present the following definition which states that low cost, low throughput design is a design where throughput and latency constrains are so relaxed, that they do not influence scheduling and assignment decisions. This type of design is very common in practice. For example, due to advances in semiconductor manufacturing, majority of audio application have sampling rate 3 or even 4 orders of magnitude higher than clock rate and belong to this class. In the implementation of this type of design, only one instance of each type of functional unit is required.

Low Cost Theorem: An arbitrary low performance design can be transformed using hot potato techniques so that at most O(n) interconnects are needed, regardless of the number of different types of operations.

Proof: The proof is constructive. First note than for n different type of operations, a design requires O (n^2) interconnects, since each unit can send data, in a general case, to any other unit. If we add the same type of deflection operation between every two operations, then all operation send data to this type of operation and receive data from this type of operation. Since there is only one instance of execution unit of each type in the implementation, obviously we need at most O(n) interconnect in the design after the application of the hot potato techniques. Q.E.D.

Without careful consideration, it may seem that the just presented theorem is mainly of theoretical interest, since low cost, low throughput design can be implemented on general purpose ALU with O(1) interconnect requirement. However, actually, the low cost theorem can be used in several application where other goals and constrains preclude use of ALUs. One such situation is low power design, where use of ALU is precluded due to its excessive power requirements when compared to designs where dedicated functional units are used.

4.0 Using Deflection Operations to Minimize Partial Scan Overhead for Improved Testability

In this paper, we demonstrate how the hot potato transformation technique can be used to minimize the number of scan registers needed to break CDFG loops, and avoid the formation of other types of loops during assignment. The original computation structure (CDFG) is first transformed by adding suitable deflection operations, so that the number of scan registers needed to break CDFG loops is minimized. Subsequently, a second set of deflection operations are added so that the selected scan registers can be effectively used to avoid formation of loops during assignment. The synthesis for testability algorithm (SFT) presented in [Dey93] is then applied to the transformed CDFG to synthesize a testable data path with minimal number of scan registers.

4.1 A Motivational Example

Consider the CDFG of the Continued Fraction IIR filter shown in Figure 11(a). To break the CDFG loops, the SFT approach [Dey93] selects the scan variables (D1,+2), (D2,+1) and (-2,*4), requiring 3 scan registers, as the selected scan variables cannot be shared. Note that the CDFG loops cannot be broken using fewer scan registers. One more scan register will is needed to avoid formation of loops during assignment.

Assume that each operation in the CDFG takes one control cycle. Figure 11(a) shows the schedule and assignment obtained by the SFT approach [Dey93], satisfying the available time of 7 control steps, and using minimal number of execution units. For instance, the operation +1 is scheduled in control step 3, and assigned to adder A1, shown by the tuple (3,A1). The resultant data path is shown in Figure 12(a). The selected scan registers to break the CDFG loops are L3A1, L1A1 and RM1.

During the assignment phase, further loops can be introduced. For instance, in the case of the CDFG in Figure 11(a), both the operations +3 and +4 had to be assigned to the same adder, thus creating an *assignment loop* (RA1, PS1, RA1). When the operations along a CDFG path from operation u to operation v are assigned *n* separate modules, and u and v are assigned the same module, an assignment loop of length *n* is created in the data path.[Dey93]. Though there is no path in the CDFG from a - operation back to a - operation through an addition operation, a loop, (NS,RA1,NS) is formed in the data path. This is a *sequential false loop*, introduced because -1 and +1 are assigned to S1 and A1, while +3 and -2 are assigned to A1 and S1 respectively. A comprehensive analysis of the formation of loops in the data path is presented in [Dey93]. The scan register selected by SFT to avoid the assignment and false loops during assignment is RA1, for a total of 4 scan registers. Note that scanning the 4 registers L3A1, L1A1, RM1 and RA1 leaves no loops, besides self-loops, in the data path.

Figure 11(b) shows the original CDFG of Figure 11(a), with deflection operations added. The SFT approach is applied to the transformed CDFG (Figure 11(b)). All CDFG loops can be broken by selecting the scan variables (D1,-5), (+3,-4) and (D3,-3). All these scan variables can share the same scan register. Addition of deflection operations reduces the minimum number of scan registers required to break all CDFG loops from 3 in the original CDFG to 1 in the transformed CDFG. Moreover, all assignment and false loops can be avoided by assigning the variables (*3,-1) and (+3,-2) to the partial scan register selected, PS. The data path generated by the SFT technique is shown in Figure 12(b). Note that if the register PS is scanned, the data path does not have any loops. Consequently, synthesizing a testable data path from the CDFG transformed by adding deflection operations is significantly more economical than from the original CDFG, requiring only 1 scan

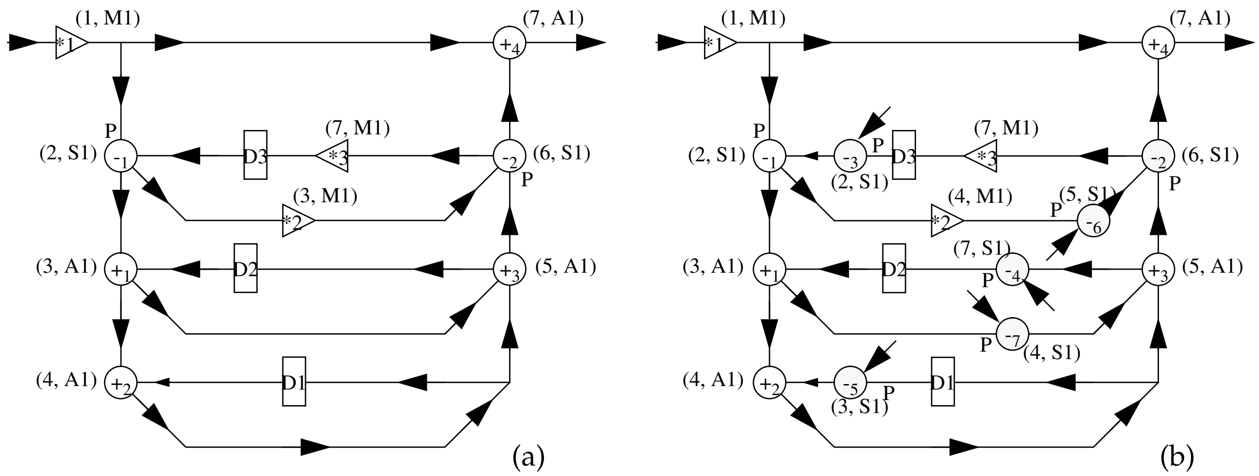


Figure 11: CDFG of the continued fraction example: (a) before and (b) after the addition of deflection operations

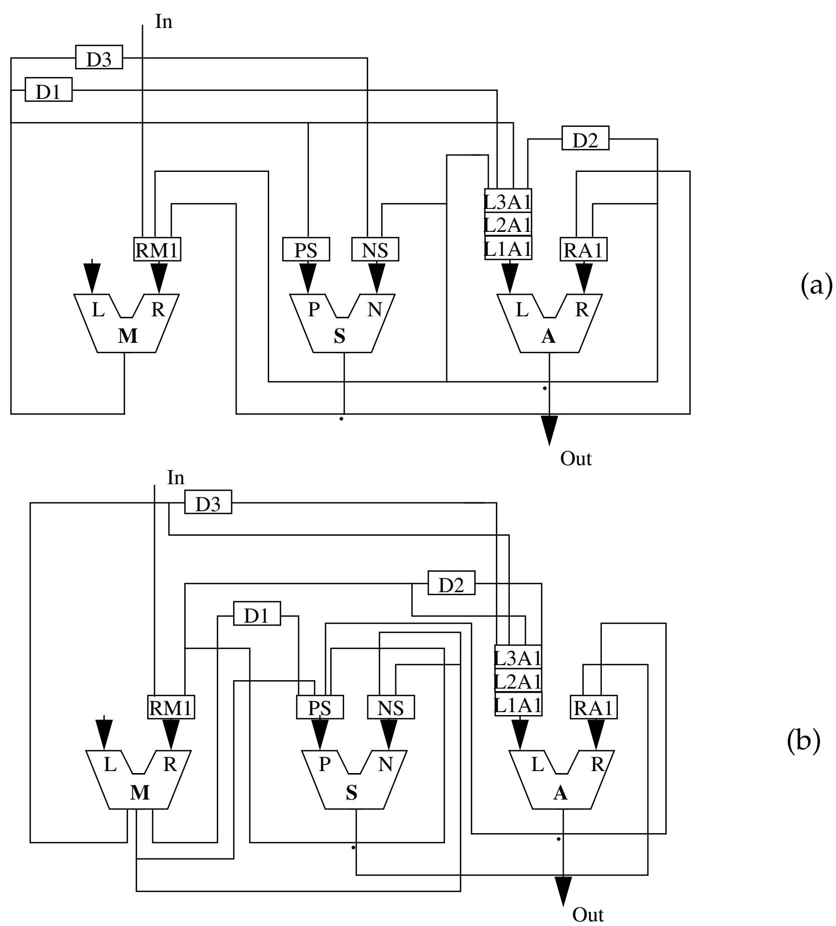


Figure 12: : The datapath for the examples of Figures 11a and 11b

register as opposed to 4 scan registers needed for the original CDFG. We describe the process of adding the suitable deflection operations to the original CDFG in the following sections.

4.2 Selecting Scan Variables to Break CDFG Loops

The problem of breaking CDFG loops using a *minimal* number of scan registers was introduced in [Dey93]. The minimum *hardware-shared cut* (HSC) problem [Dey93] is to select a set of scan variables such that the following criteria are simultaneously satisfied:

HSC1 All CDFG loops, except self-loops, are broken,

HSC2 The selected scan variables can be assigned to a minimum number of scan registers, and,

HSC3 Reusability of the scan registers, to break the other loops formed during the subsequent scheduling and assignment phase, is maximized;

Two measures can be used to capture the effectiveness of a variable in satisfying the three criteria of the minimum HSC problem. The loop cutting effectiveness (LCE) measure helps to satisfy the first criteria, HSC1, of the minimum HSC problem. The LCE measure of a variable estimates the number of loops that will be broken by assigning the variable to a scan register. Since the number of loops can be exponential, and there is no known algorithm to count them efficiently, a random walk [Hag92] based technique was proposed to calculate the LCE measure of the variables of a CDFG.

The hardware sharing effectiveness (HSE) measure satisfies criteria HSC2 and HSC3 of the minimum HSC problem. The HSE of a variable v estimates the likelihood that v can share a scan register with other variables to break the different types of loops in the data path.

In this paper, we introduce the following HSE measure. Let f_v be the type of operation to which variable v is an input. Variable v can share a scan register with all variables x such that (i) x is a corresponding input to an operation of the same type f_v, and (ii) the lifetimes of v and x do not overlap. Let set X contain those variables x which can share a scan register with variable v. Also, let Y be a subset of X such that variables in Y can cut one or more loops not broken by scanning variable v. We define HSE(v) as: $HSE(v) = w_1|Y| + w_2|X - Y|$.

The first component reflects the number of variables that can share the same scan register assigned to v, and can be used to cut CDFG loops left unbroken by v. The second component measures the likelihood that the scan register SR1, to which the variable v will be assigned, can be effectively reused later to avoid forming loops during the subsequent scheduling and assignment phase. To favor selection of scan variables good for minimizing the number of scan registers needed to break CDFG loops, as opposed to scan variables which are good to avoid formation of loops during assignment, the weight w_1 is set to be much higher than the weight w_2.

After calculating the LCE and HSE measures for each edge e which belongs to some strongly connected component (SCC), we calculate $eff_{HSC}(e) = \alpha * LCE(e) + \beta * HSE(e)$. For each SCC, we select as scan variable the edge e with highest $eff_{HSC}(e)$, and delete the selected edges from the CDFG. We repeat the process until all CDFG loops are broken. After the scan variables have been selected to cut the CDFG loops, a minimum set of scan registers is identified to which all the scan variables can be assigned. This can be done optimally by assigning all scan variables with disjoint lifetimes to the same scan register.

4.3 Using Deflection Operations to Minimize Scan Registers Needed to Break CDFG Loops

Deflection operations can be used to minimize the number of scan registers required to break CDFG loops. This is achieved by adding the deflection operations in such a way that more of the selected scan variables can share the same scan register.

Two scan variables v and w *cannot* share the same scan register if one or both of the two conditions, termed **hardware sharing bottlenecks**, is true. Note that the hardware sharing bottleneck B1 is imposed due to the hardware model adopted.

B1. Variables v and w are inputs to two operations of different types.

B2. Variables v and w have overlapping lifetimes.

4.3.1 Adding Deflection Operations to Eliminate Bottleneck B1

Consider the CDFG shown in Figure 8(a). It has two CDFG loops: loop L1 consisting of - and + operations, the other loop L2 containing operations of the type * and >>. Variables (D_1, $-_1$) and (D_2, $*_1$) have overlapping lifetimes (bottleneck B2), and hence selecting them to break the CDFG loops will require two scan registers. Any other variable v in loop L1 and variable w in loop L2, selected to break the corresponding loops L1 and L2, cannot be shared, since they have the bottleneck B1. Hence, selection of any two scan variables to break the two CDFG loops will require two scan registers.

However, if the deflection operation $+_3$ is added at the input of operation $*_1$ in loop L2, as shown in the modified CDFG in Figure 8(b), the variables ($-_1$, $+_1$) in loop L1 and (D_2, $+_3$) in loop L2, do not have either of the bottlenecks, and can share the same register. Consequently, selecting ($-_1$, $+_1$) and (D_2, $+_3$) as scan variables breaks the two CDFG loops, and requires only one scan register.

Note that for any variable v that can be used to break a CDFG loop in Figure 8(a), the set Y of variables that can be used to cut other CDFG loops not broken by v, and can share the same scan register with v, is empty. On the other hand, adding the deflection operation in the CDFG shown in Figure 8(b) increases the value of $|Y|$ to 1 for the variable ($-_1$, $+_1$), and adds a new variable (D_2, $+_3$) with $|Y| = 1$. At this point, the above two variables have the highest HSE measure. The LCE measure remaining the same for all the variables in the loops, the variables ($-_1$, $+_1$) and (D_2, $+_3$) are selected as scan variables by the scan selection process outlined in the previous section, and can be assigned to one scan register. This example demonstrates how deflection operations can be used to minimize the number of scan registers required to break CDFG loops, by eliminating bottleneck B1, and increasing the HSE measure of candidate scan variables.

4.3.2 Adding Deflection Operations to Eliminate Bottleneck B2

Deflection operations can be used to eliminate the hardware sharing bottleneck B2 also. Consider a pair of variables *v* and *w* which are inputs to the same type of operation, but cannot share the same scan register due to bottleneck B2, that is, they have overlapping lifetimes. A deflection operation can be introduced on the edge *w*, so that the lifetime of variable *w* is split, and the variables *v* and the new input *u* to the deflection operation do not have any hardware sharing bottleneck.

Consider the CDFG shown in Figure 9(a), consisting of two CDFG loops L1 and L2. Each pair of variables, (*v*,*w*), *v* in loop L1, and *w* in loop L2, needed to break the CDFG loops, have one hardware sharing bottleneck, and cannot be shared. For instance, one possible pair that are inputs to the same type of operations is: (D_1,$+_1$) and (D_2,$+_2$). However, they have overlapping lifetimes. Similarly, variables ($-_1$,$*_1$) and ($+_2$,$*_2$) are inputs to the same operation type, but have overlapping lifetimes. Any pair of scan variables selected to break the CDFG loops in Figure 9(a) would require two scan registers.

The LCE and HSE measures of the variables are shown by a tuple (LCE,HSE), with the HSE part showing both the components. For example, the variable (D_1,$+_1$) breaks one loop, and hence has an LCE value of 1. Also, $X(v) = \{*_1, +_3\}$, $Y(v) = \{\}$. Hence, the first component of HSE measure is 0, and the second component is 1, as shown by the tuple (1, 0+1) at the side of variable (D_1,$+_1$).

Consider each variable in the CDFG which can break loop L2, not broken by scanning variable (D_1,$+_1$). Consider one such variable (D_2,$+_2$). The pair of variables can break all the CDFG loops, and are inputs to the same type of operation +, but cannot be shared because they have the hardware sharing bottleneck B2. Let us add a deflection operation, $*_3$, on the edge (D_2,$+_2$), such that the lifetime of the split variable ($*_3$,$+_2$) no longer overlaps with the lifetime of variable (D_1,$+_1$). The transformed CDFG is shown in Figure 9(b). The bottleneck B2 is eliminated, and variables (D_1,$+_1$) and ($*_3$,$+_2$) can share the same scan register.

In terms of the HSC problem formulation, introduction of the deflection operation helps increase the HSE measure of variables, while keeping the LCE measure fixed. After the introduction of the deflection operation, shown in Figure 9(b), variables (D_1,$+_1$) and ($*_3$,$+_2$) can now share the same scan register, and hence, the HSE measure of variables (D_1,$+_1$) increases from (0+1) to (1+1). Similarly, the split variable ($*_3$,$+_2$) now has a HSE value (1+1). The scan selection process outlined in the previous section would select variables (D_1,$+_1$), ($*_3$,$+_2$) to break the CDFG loops, requiring only one scan register.The above examples illustrate how deflection operations can be added to the CDFG to eliminate hardware sharing bottlenecks.

4.3.3 Algorithm for Adding Deflection Operations

We briefly outline the process of adding the deflection operation, so that the number of scan registers required by the scan variables to break the CDFG loops is minimized. Let *Z(v)* be the set of variables *w*, such that *w* breaks some other CDFG loops not broken by *v*, and either both *v,w* are inputs to the same operation type, or *v,w* do not have overlapping lifetimes, but not both. That is, *Z*(*v*) is the set of all variables *w* which break other loops not broken by *v*, but cannot share the same scan register because the pair (*v,w*) have one of the two bottlenecks, B1 or B2. The aim of adding the deflection operations will be to eliminate the hardware sharing bottlenecks of those pairs of variables which are potentially good candidates for scan variables. This is achieved by attempting to increase the HSE measures of those variables *Z(v)* which can be potentially shared, after eliminating bottlenecks, with variables *v* with high LCE measure (cuts many loops). We outline below the algorithm for adding deflection operations to minimize the number of scan registers needed to break CDFG loops:

add_deflection_ops_to_break_CDFG_loops ()

```
1.  for each variable v with positive LCE measure, but low HSE measure {
2.      for each w in Z(v) {
3.          add deflection operation op_d on w;
4.          check feasibility of adding op_d, with respect to scheduling and assignment;
5.          if deflection operation op_d feasible {
6.              for all variables, calculate nHSE(u) = new HSE(u) - hardware cost of adding op_d;
7.               retain the highest nHSE values for each variable;
            }
        }
    }
8.  select scan variables (virtual operation) assuming nHSE values;
9.  for each variable v selected, add the deflection operations necessary to obtain the new HSE value,
    nHSE(v);
```

4.4 Detecting Formation of Loops in the Data Path During Assignment

The formation of different types of loops in the data path during the assignment phase can be captured by using a Data-Dependency and Compatibility Graph (DDCG). The DDCG models the data dependencies and the compatibilities of the operations of the CDFG. Each node in the graph represents an operation of the CDFG. There is a (undirected) compatibility edge between two operations if there is a non-zero probability that both the operations can be assigned to the same module. There is a (directed) data-dependency edge from operation *v* to operation *w* if operation *w* depends on data produced by operation *v*. We will denote a compatibility edge between *v* and *w* as *c*(*v,w*), and a data-dependency edge from *v* to *w* as *d*(*v,w*).

Figure 13(a) shows segments of two paths in a CDFG, and Figure 13(b) shows the corresponding data-dependency and compatibility graph. Assuming that each operation in the CDFG takes one control cycle, the longest path is 3 control steps long. For a schedule which requires 3 control steps, the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) control steps in which each operation can be scheduled [Wal91] is shown by the tuple [ASAP,ALAP] in Figure 13(a). In Figure 8(b), each compatibility edge $c(v,w)$, shown dotted, is weighted by an estimate of the compatibility, comp(v,w), between two nodes v and w, as given by:.

$$\mathrm{Comp}(v,w) = \left\{\begin{pmatrix} 1 & \exists d(v,w) \\ 1 - \dfrac{\mathrm{overlap}(v,w)}{(\mathrm{mobility}(v) * \mathrm{mobility}(w))} & \mathrm{otherwise} \end{pmatrix}\right.$$

The compatibility between nodes $+_1$ and $+_3$, Comp($+_1$,+3) = 1 - (1/2) = 0.5. All the compatibility edges $c(v,w)$ are weighted by Comp(v,w), as shown in the DDCG in Figure 13(b).

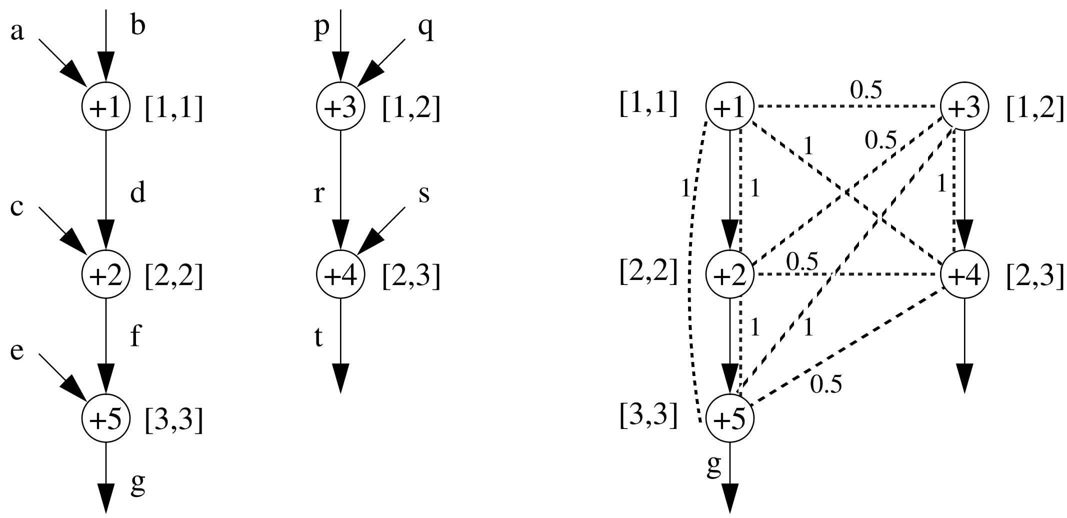


Figure 13: CDFG (a) and the corresponding DDCG (b)

The paths of the DDCG can be represented using regular expressions. d^+ represents a path consisting of a sequence of one or more data-dependency edges d. The concatenation of two paths p and q is represented by p.q, or simply, pq. For example, the regular expression cd^+ represents a path starting with a compatibility edge c, followed by one or more occurrence of data-dependency edges. In Figure 13(b), the path ($+_5$,$+_1$), ($+_1$,$+_2$), ($+_2$,$+_5$) can be represented by cd^+.

An assignment loop is formed in the data path if there exists a cycle of the form cd^+ in the DDCG, and the compatibility edge c is used during module assignment. Similarly, if there is a cycle of the form $(cd^+)(cd^+)^+$ in the DDCG, and the compatible edges c in the cycle are used during module assignment, a sequential false loop is formed in the data path.

4.5 Adding Deflection Operations to Avoid Formation of Loops During Assignment

Deflection operations can also be added to avoid the formation of loops during assignment, namely assignment loops and sequential false loops. We adopt the approach of adding all the deflection operations required in the beginning, and performing the three phases of allocation, selecting scan variables for CDFG loops, and scheduling and assignment, on the transformed CDFG. Since we want to add the deflection operations before we know which loops can be formed during assignment, we use the DDCG and the compatibility estimates to insert deflection operations. The DDCG has information about all possible loops that can be formed during assignment. We consider only those loops that have a high possibility of formation during assignment, as reflected by the high values of comp(*v,w*) on their *c* edges. We ignore those loops that have a low possibility of forming, reflected by low values of comp(v,w) on their *c* edges. To achieve this, we delete all those compatibility edges which have compatibility estimate less than a threshold.

We insert deflection operations in the CDFG so as to be able to break as many loops remaining in the DDCG. Every time a deflection operation is added, the corresponding data dependency edge in the DDCG is broken, in anticipation that the input of the deflection operation will be scanned. If the resultant DDCG contains no loops of the form cd^+ or $(cd^+)(cd^+)^+$, we are guaranteed no loops will be formed during assignment. Else, addition of the deflection operations have minimized the chances of formation of loops during the scheduling and assignment phase.

add_deflection_ops_to_avoid_loops_during_assgnment ()

1. *create DDCG from the given CDFG;*
2. *remove data dependency edges from DDCG, corresponding to scan vars selected for CDFG loops;*
3. *delete compatibility edges having compatibility estimate comp(v,w)* **<** *threshold;*
4. *for every loop L of the form* cd^+ *or* $(cd^+)(cd^+)^+$ *in DDCG {*
5. *attempt to add a deflection operation op_d on a* ***d*** *edge in CDFG, such that loop L in DDCG is broken;*
6. *check if adding op_d is feasible, in terms of scheduling, assignment and RU cost;*
7. *if deflection operation op_d is feasible {*
8. *add* op_d *in CDFG;*
9. *delete the d edge in DDCG;*

   }

}

10. *return transformed CDFG, with added deflection operations.*

5.0 Experimental Results

The hot potato transformation approach for the optimization of resource utilization was applied on the following seven high level synthesis examples: speech filter, 7th order cascade IIR filter (7IIR), 8th order parallel IIR filter (8IIR), Gray Markel filter, 3rd and 4th order Volterra filters, 100th order FIR filter, 8 point discrete cosine transform [8DCT] and 9 point Winograd fast Fourier transform

(FFT]. The initial and final number of register and interconnects, as well as relative reduction in their numbers, are given in Table 1. Both the average and the median reduction in the number of interconnects were 37% and both the average and the median reduction in the number of registers were 14%. The most important advantage of the hot potato transformations techniques for resource utilization is that they are orthogonal with other transformations (e.g. retiming and algebraic transformations), and that therefore the hot potato techniques can be effective on the already highly optimized designs.

Example	**InitReg**	**HPReg**	**InitIntc**	**HPIntc**	**RegReduc**	**IntcReduc**
speech filter	35	29	15	11	17	27
7IIR	32	23	15	10	28	33
8IIR	39	32	17	11	18	39
IIR GM	44	35	18	11	20	39
Cascade	52	44	14	11	15	21
3 Volterra	35	31	17	9	13	47
4 Volterra	48	44	37	23	8	38
100FIR	132	121	14	9	8	36
8 DCT	19	18	16	9	5	44
9 FFT	21	19	14	8	10	43

Table 1 Effect of Hot Potato Technique (HP) on Resource Utilization: InitReg and HPReg - the number of registers before and after the application of HPT; InitIntc and HPIntc - the number of interconnects before and after the application of HPT; RegR educand IntcReduc - reduction in the number of registers and interconne*ct (%)*

To evaluate the effectiveness of the hot potato transformations on the testability of the designs, we applied the technique on the following examples: the 3rd order Continued Fraction IIR filter, the Modem filter, the 5th order Elliptical Wave Digital Filter, and the 5th order IIR Cascade Filter. We first synthesize the examples using the behavioral test synthesis system BETS [Dey93] for both testability and resource utilization. Next, we transform the examples by adding deflection operations to minimize partial scan cost, using the algorithms presented in this paper, and then apply BETS on the transformed CDFG. In the following tables, **Orig** refers to the design synthesized by conventional high level synthesis (BETS without testability considerations), **BETS** refers to the design synthesized by BETS, while **HP+BETS** refers to the design synthesized by BETS after the addition of deflection operations for testability.

Table 2 shows the physical characteristics of the designs. The bit width of the synthesized designs is indicated in the column **Bits**. The number of control steps (**CS**) needed by the implementation after adding the deflection operations remains same, ensuring that the performance of the designs is not compromised while improving testability. Similarly, the number of execution units (**EXU**) needed remains same. The columns **Reg, Mux, and Int** represent the number of registers, multiplexers, and interconnects, respectively. Note that while attempting not to compromise resource utilization, BETS may increase the hardware resources used while making a design testable with reduced partial scan cost. However, deflection operations can be added such that the number of resources, like registers and interconnects, is reduced simultaneously with the number of scan registers used. Hence, in all the design cases, the hardware costs, like registers, multiplexers, and interconnects, are not compromised by the addition of deflection operations for testability.

Design	Method	Bits	CS	EXU	Reg	Mux	Int
Continued Fraction IIR	Orig	20	7	1M,1S,1A	11	11	15
	BETS	20	7	1M,1S,1A	11	11	15
	HP+BETS	20	7	1M,1S,1A	10	11	15
Modem	Orig	16	10	3M,2A,1S	18	18	25
	BETS	16	10	3M,2A,1S	18	18	25
	HP+BETS	16	10	3M,2A,1S	17	17	23
5th Order Elliptical Wave Digital Filter	Orig	20	17	3M,3A	23	29	20
	BETS	20	17	3M, 3A	24	32	27
	HP+BETS	20	17	3M, 3A	24	32	27
5th Order IIR Cascade Filter	Orig	16	7	3M,1S,2A	14	14	24
	BETS	16	7	3M,1S,2A	14	11	13
	HP+BETS	16	7	3M,1S,2A	14	11	13

Table 2: Performance and Hardware Characteristics of the Designs

Table 3 shows the number of Flip-Flops (**FFs**), and the number of scan FFs (**Scan FFs**) needed to break all the loops of the design for all the three implementations, Orig, BETS, and HP+BETS versions. In the case of the original implementation, a gate-level partial scan tool, OPUS [Chi91], is used for scan FF selection. For instance, in the case of the Modem filter, the original data path synthesized without testability considerations has 288 FFs, the data path synthesized by BETS from the original CDFG has 288 FFs, while the data path synthesized by BETS from the transformed CDFG, after addition of deflection operations, has 272 FFs. To avoid loops except self-loops in the data path, the gate-level partial scan tool OPUS needs to scan 64 FFs in the original data path, while the behavioral test synthesis system BETS needs 48 scan FFs. However, when BETS is given the transformed CDFG with deflection operations, it requires only 16 scan FFs.

Design	Method	FFs	Scan FFs
Continued Fraction IIR	Orig	220	80
	BETS	220	80
	HP+BETS	200	20
Modem	Orig	288	64
	BETS	288	48
	HP+BETS	272	16
5th Order Elliptical Wave Digital Filter	Orig	460	300
	BETS	480	60
	HP+BETS	480	40
5th Order IIR Cascade Filter	Orig	208	48
	BETS	208	32
	HP+BETS	208	16

Table 3: Effect of Adding Deflection Operations on Partial Scan Overhead

Design	Method	Total FFs	Scan FFs	Total Faults	Aborted Faults	Fault Cov (%)	Test Eff. (%)	ATPG CPU (secs)
Continued Fraction IIR	Orig	220	0	6050	133	96	98	4019
	Orig+OPUS	220	80	6050	5	98	100	55
	BETS	220	80	6058	4	98	100	89
	HP+BETS	200	20	6052	21	98	100	167
Modem	Orig	288	0	8579	8299	0	3	>6hr
	Orig+OPUS	288	64	8879	7	96	100	153
	BETS	288	48	8648	21	96	100	235
	HP+BETS	272	16	8629	21	96	100	258
5th Order Elliptical Wave Digital Filter	Orig	460	0	10364	10077	1	3	>72hr
	Orig+OPUS	460	300	10364	0	98	100	309
	BETS	480	60	10916	16	98	100	233
	HP+BETS	480	40	11130	2	98	100	200
5th Order IIR Cascade Filter	Orig	208	0	8740	8488	0	3	20446
	Orig+OPUS	208	48	8740	7	97	100	128
	BETS	208	32	7531	23	97	100	869
	HP+BETS	208	16	7575	19	97	100	124

Table 4: Sequential ATPG using Hitec [Nie91]

The testability of the synthesized designs was evaluated using the gate-level sequential ATPG tool, HITEC [Nie91], shown in Table 4. For each design, besides the rows **Orig**, **BETS**, and **HP+BETS**, the row **Orig+OPUS** refers to the data path obtained from the original data path after scanning the FFs selected by the gate-level partial scan tool OPUS [Chi91]. For each version of a design, the total number of FFs and the number of scan FFs used, are reported. The results of running HITEC on each design is also shown. The total number of faults, the number of faults aborted by HITEC, the fault coverage and test efficiency achieved, and the CPU time taken by HITEC on a SUN Sparcstation2 are reported.

As expected, though the sequential test pattern generator HITEC [Nie91] could not achieve 100% test efficiency on the original designs without scan FFs, 100% test efficiency could be obtained by HITEC on the final designs after scanning the selected FFs. For example, in the case of Modem filter, while the original design was not testable, for the Orig+OPUS, BETS and HP+BETS designs, HITEC could achieve 100% test efficiency. For the original design, gate-level scan selection needs 64 scan FFs to make the circuit 100% testable (row Orig+OPUS). For the BETS design synthesized from the original specification (row BETS), 48 scan FFs are needed to achieve 100% test efficiency. On the other hand, only 16 scan FFs are needed to achieve 100% test efficiency for the HP+BETS design, synthesized by BETS from the specification transformed by adding deflection operations (row HP+BETS).

Finally, we illustrate the effectiveness of the hot potato techniques to significantly reduce the scan overhead needed by BETS using the most popular high level synthesis benchmark: 5th order elliptical wave digital filter. We implemented the filter as specified by CCITT G712 PCM standard {Cla88]. Following most common high level synthesis benchmarking standards, the available time and critical path are 17 control cycles long, when multiplication is two cycles long and addition is unit cycle operation.

The schedule and assignment of the filter, and the corresponding datapath synthesized using BETS are shown in Figure 14 The rectangles and the circles denote multiplications and additions respectively. Note, that BETS uses commutativity in order to reduce test overhead, as indicated by "C" next several operations. The detailed datapath resource and test requirements are shown in Tables 2, 3 and 4.

Eight scan variables are required to break all CDFG loops and 3 scan registers are sufficient to produce the testable datapath. As is reported in [Dey93], 15 scan registers are required for making the datapath testable, if the Hyper system is used for producing low area implementations (row orig in Tables 3 and 4).

Figure 14 shows the CDFG and the corresponding datapath after the addition of two deflection operations (additions) which are denoted by shaded circles. The application of the hot potato transformation resulted in the computational structure more amenable for the application of BETS

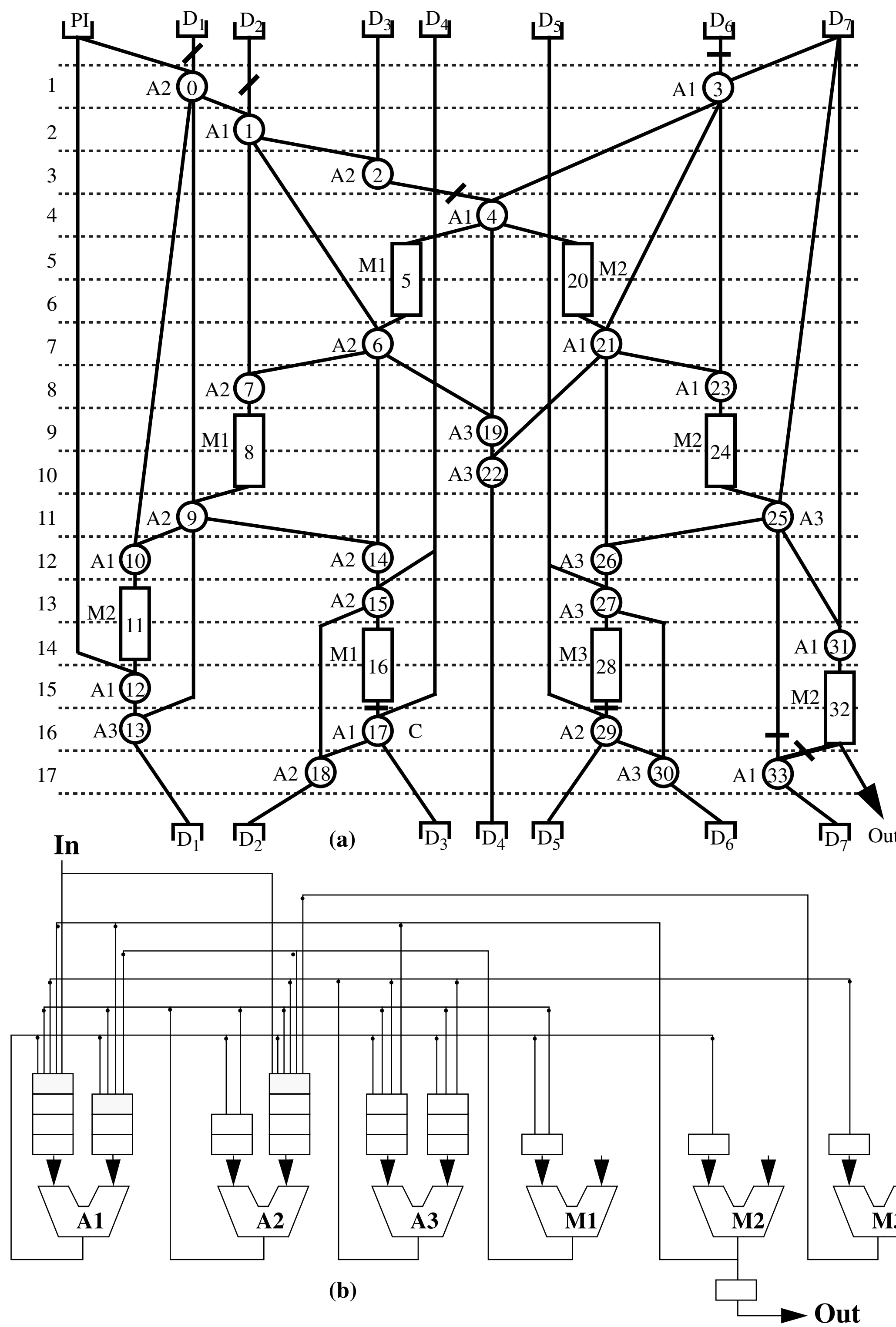


Figure 14:

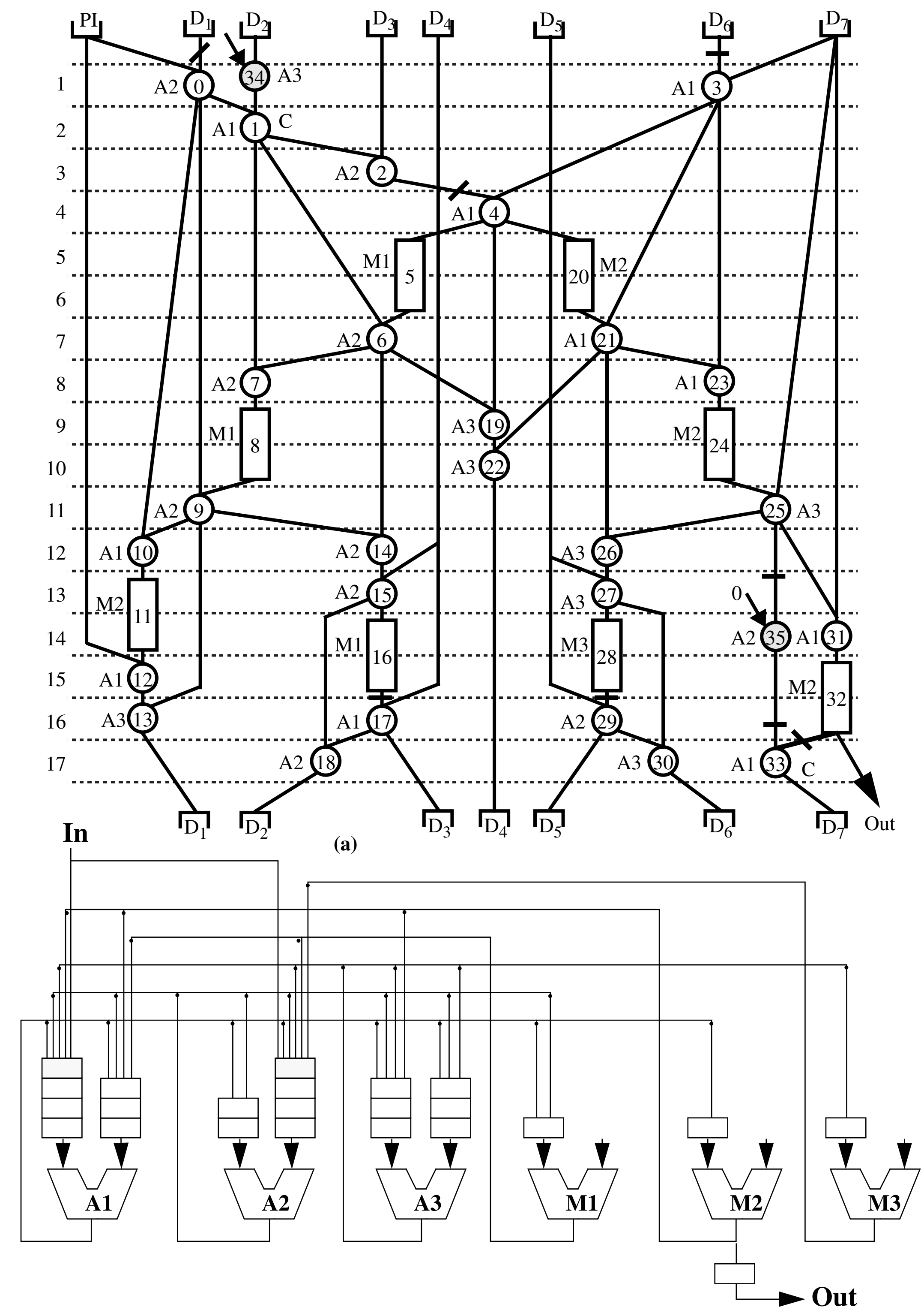

PI
D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
A2 0
34 A3
A1 3
A1 1 C
A2 2
A1 4
M1 5
M2 20
A2 6
A1 21
A2 7
A1 23
M1 8
A3 19
A3 22
M2 24
A2 9
25 A3
A1 10
A2 14
A3 26
M2 11
A2 15
A3 27
0
A2 35
A1 31
M1 16
M3 28
M2 32
A1 12
A3 13
A1 17
A2 29
A2 18
A3 30
A1 33 C
Out
D1 D2 D3 D4 D5 D6 D7 Out
(a)
In
A1 A2 A3 M1 M2 M3
Out

Figure 15:

scheduling and assignment algorithms. After the application of hot potato techniques only two scan registers are required in the testable datapath produced by BETS from the transformed CDFG in Figure 14a.

6.0 Conclusion

We have introduced a new hot potato transformation technique based on the introduction of deflection operations. We have shown the application of the technique to reduce the interconnect and register requirements, while preserving throughput and not altering the number of execution units needed. Application of the hot potato transformations to reduce the partial scan overhead for generating easily testable data paths has also been shown. Experimental results on several benchmarks demonstrate the effectiveness of the hot potato transformations to improve resource utilization and testability.

ACKNOWLEDGMENT: The authors wish to thank Vijay Gangaram for his implementation help in the project.

7.0 References

[Avr91] L. Avra: "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths", Proceedings of the International Test Conference, 1991.

[Bar64] P. Baran: "On Distributed Communication Networks", IEEE Trans. on Comm., Vol. 12, No. 1, pp. 1-9, 1964.

[Ber91] D. Bernstein, D. Cohen, H. Krawczyk: "Code Duplication: An Assist for Global Instruction Scheduling", Proceedings of MICRO-24, pp. 103-113, 1991.

[Bha94] S. Bhatia, N. K. Jha, "Genesis: A Behavioral Synthesis System for Hierarchical Testability," Proc. European Design & Test Conf., pp. 272-276, 1994.

[Bla85] R.E. Blahut: "Fast Algorithms for Digital Signal Processing", Addison-Wesley, 1985.

[Bra84] R.K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli: "Logic Minimization Algorithms for VLSI Synthesis", Kluwer, Boston, MA, 1984.

[Cal91] D. Calahan, K. Kennedy, A. Porterfield: "Software Prefetching", Fourth International Conference on Architectural Support for Programming Languages and Operating Systems", pp 40-52, 1991.

[Cha92] A.P. Chandrakasan et. al.: "Hyper-LP: A Design System for Power Minimization using Architectural Transformations", Intl Conf. Computer-Aided Design, Santa Clara, CA, 300-303, November 1992.

[Cha91] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Managan, and S. M. Watt. *Maple V: The Future of Mathematics*. Springer-Verlag, 1991.

[Che92] T-F. Chen, J-L. Baer: "Reducing Memory Latency via Non-blocking and Prefetching Caches", Fifth International Conference on Architectural Support for Programming Languages and Operating Systems", pp 51-61, 1992.

[Che90] K.T. Cheng, V.D. Agrawal: "A Partial Scan Method for Sequential Circuits with Feedback", IEEE Trans. on Computers, Vol. 39., No. 4, pp. 544-548, 1990.

[Che93] K.T. Cheng, L.A. Enterna: "Multi-Level Logic Optimization by Redundancy Addition and Removal", European Design Automation Conference, pp. 373-377, 1993.

[Chi90] V. Chickermane, J. H. Patel: "An Optimization Based Approach to the Partial Scan Design Problem", Proceedings of the International Test Conference, 377-386, September, 1990.

[Cri91] C. Cring, A.R. Newton: "A Cell-Replicating Approach to Mincut-Based Circuits Partitioning", IEEE ICCAD-91, pp. 2-5, 1991.

[Dal87] W.J. Dally, C.L. Seitz: "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", IEEE Trans. on Computers, Vol. 36, No. 5, pp. 547-563, 1987.

[Dal90] W. Dally: "Network and Processor Architecture for Message-Driven Computers", in "VLSI and Parallel Computation", ed. by R. Suaya, G. Birtwistle, pp. 140-222, Morgan Kaufmann, San Mateo, CA, 1990.

[Dar81] J. Darlington: "An experimental program transformation and synthesis system", Artif. Intell., Vol. 16, No. 1, pp. 1-46, 1981.

[DeS94] J.I. De Souza, M. Potkonjak, A. Parker: "Optimal Simultaneous Algorithm/Architecture Matching and Retiming", NEC USA Technical Report, 1994.

[Dey93] S. Dey, M. Potkonjak, R. Roy: "Exploiting Hardware-Sharing in High Level Synthesis for Partial Scan Optimization", pp. 20-25, ICCAD93, 1993.

[Fis88] C.N. Fischer, R.J. Le Blank: "Crafting a Compiler", The Benjamin/Cummings Publishing Co., Menlo Park, CA, 1988

[Fis92] J.A. Fisher, S.M. Freudenberger: "Predicting Conditional Branch Direction From Previous Runs of a Program", Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp 85-95, 1991.

[Geb89] C.H. Gebotys, M. I. Elmasry: "Integration of Algorithmic VLSI Synthesis with Testability Incorporation", IEEE Journal of Solid-State Circuits, Vol. 24, No. 2, 1989.

[Gla89] H.M. Gladney: "Data Replicas in Distributed Information Services", ACM Transactions on Database Systems, Vol. 14, No. 1, pp. 75-97, 1989.

[Gol89] G.H. Golub, C.F. Loan: "Matrix Computations", The Johns Hopkins University Press, Baltimore, MD, 1989.

[Gue93] L. Guerra, M. Potkonjak, J. Rabaey: "High Level Synthesis for Reconfigurable Datapath Structures", ICCAD93, pp. 26-29, November 1993.

[Haj91] B. Hajek: "Bounds on Evacuation Time for Deflection Routing", Distributed Computing, Vol. 5, No. 1, pp. 1-5, 1991.

[Har94] I.G. Harris, A. Orailoglu: "SYNCBIST: SYNthesis for Concurrent Built-In Self-Testability", Proc. IEEE Conference on Computer Design, 1994.

[Hwa92] J. Hwang, A. El Gamal: "Optimal Replication for Min-Cut Partitioning", IEEE ICCAD-92, pp. 432-435.

[Ioa87] Y.E. Ioannidis, E. Wong: "Query optimization by simulated annealing", Proc. of the 1987 ACM-SIGMOD Conference on the Management of Data, pp. 9-22, 1987.

[Keu91] K. Keutzer, S. Malik, A. Saldanha: "Is Redundancy Necessary to Reduce Delay?", Trans. on CAD, Vol. 10, No. 4, pp. 427-435, 1991.

[Kla91] A.C. Klaiber, H.M. Levy: "An architecture for software-controlled data prefetching", Proceedings of the 17th Annual Symposium on Computer Architecure, pp. 43-53, 1991.

[Law69] E.L. Lawler, K.N. Levitt, J. Turner: "Module Clustering to Minimize Delay in Digital Networks", IEEE Transaction on Computers, Vol. 18, No. 1, pp. 47-57, 1969.

[Lee87] E. A. Lee and D. G. Messerschmitt. "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing." *IEEE Transactions on Computers*, Vol. 36, No. 1, pp.24–35, 1987.

[Lee90] D.H. Lee, S.M. Reddy: "On Determining Scan Flip-Flops in Partial-Scan Designs", pp. 322-325, 1990.

[Lee93] T.C. Lee, N.K. Jha, W.H. Wolf: "Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and partial Scan Environment", DAC-93, pp. 292-297, 1993.

[Lin91] X. Lin, L.M. Ni: "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks" Proc. 18th Int. Symp. on Computer Architecture, pp. 116-125, IEEE CS Press, Los Alamitos, CA, 1991.

[Lob91] D. Lobo, B.M. Pangrle: "Redundant Operator Creation: A Scheduling Optimization Technique", DAC-91, pp. 775-778, 1991.

[Maj92] A. Majumdar, K. Saluja, R. Jain: "Incorporating Testability Considerations in High-Level Synthesis", Proceedings of the International Symposium on Fault-Tolerant Computing, 1992.

[McF86] S. McFarling, J. Hennessy: "Reducing the cost of branches", Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 396-403, 1986.

[McF90a] M.C. McFarland, A.C. Parker, R. Camposano: "The High Level Synthesis of Digital Systems", Proc. of the IEEE, Vol. 78, No. 2, pp. 301-317, 1990.

[McF90b] M.C. McFarland, T. Kowalski: "Incorporating bottom-up design into hardware synthesis", Trans. on CAD, Vol. 9, No. 9, pp. 938-950, 1990.

[Mow91] T. Mowry, A. Gupta: "Tolerating latency through software-controlled prefetching in shared-memory multiprocessors", Journal of Parallel and Distributed Computing, Vol. 12, No. 2, pp. 87-106, 1991.

[Mow92] T.C. Mowry, M.S. Lam, A. Gupta: "Design and Evaluation of a Compiler Algorithm for Prefetching", Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp 62-73, 1992.

[Nie91] T.M. Niermann, J. H. Patel: "HITEC: A Test Generation Package for Sequential Circuits", EDAC, pp. 214-218, 1991.

[Ott84] K. Ottenstein, L. Ottenstein: "The program dependence graph in a software development environment", ACM SIFSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, pp. 177-184.

[Pap91] C. Papachristou, et al.: "SYNTEST: a method for high-level SYNthesis with self TESTability, ICCAD, pp. 458-462, 1991.

[Par89] K. K. Parhi, D. Messerschmitt: "Pipeline interleaving and parallelism in recursive Filters, Parts 1 and 2", IEEE Trans. on ASSP, Vol. 37, pp. 1099-1117 and 1117-1134, 1989.

[Pat78] J.H. Patel: "Pipelines with Internal Buffers", Proc. 5th Symposium Computer Architecture, pp. 249-255, 1978.

[Pot92] M. Potkonjak, J. Rabaey: "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", IEEE International Conference on Computer-Aided Design, Santa Clara, CA, pp. 304-308, November 1992.

[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.

[Rog92] A. Rogers, K. Li: "Software Support for Speculative Loads", Fifth International Conference on Architectural Support for Programming Languages and Operating Systems", pp 38-50, 1992.

[Rus71] R. L. Russo, P.H. Oden, P.K. Wolff, Sr.: "A Heuristic Procedure for the Partitioning and Mapping of Computer Logic Graphs", IEEE Trans. on Computers, Vo. 20, No. 12, pp. 1455-1462, 1971.

[Smi92] M.D. Smith, M. Horowitz, M.S. Lam: "Efficient Superscalar Performance Through Boosting", Fifth International Conference on Architectural Support for Programming Languages and Operating Systems", pp. 248-259.

[Sri94] M. B. Srivastava, M. Potkonjak: "Transforming Linear Systems for Joint Latency and Throughput Optimization", EDAC-94 European Design Automation Conference, pp. 267-271, 1994.

[Sto90] L. Stok, J.A.G. Jess: "Foreground Memory Management in Data Path Synthesis", International Journal of Circuits Theory and Applications", Vol. 20, No. 3, pp. 235-256.

[Syz90] T. Szymanski: "An Analysis of Hot-Potato Routing in a Fiber Optic Packet Switched Hypercube, IEEE INFOCOM '90 Proceedings, pp. 918-925, 1990.

[Ull89] J.D. Ullman: "Database and Knowledge-Base Systems, Volume II: The New Technologies", Computer Science Press, Rockville, MD, 1989.

[Wal91] R. Walker, R. Camposano: "A Survey of High-Level Synthesis Systems", Kluwer, Boston, MA, 1991.

[Wei84] M. Weiser: "Program Slicing", IEEE Transactions on Software Engineering, Vol. 10, pp. 352-357, 1984.

[Yu83] C.T. Yu, C.C. Chang: ""On the Design of a Query Strategy in a Distributed Database Environment", ACM Proceedings of SIGMOD, pp. 30-39, 1986.