Optimizing Designs Using the Addition of Deflection Operations

Jennifer L. Wong, *Student Member, IEEE*, Miodrag Potkonjak, *Member, IEEE*, and Sujit Dey, *Member, IEEE*

***Abstract*—This paper introduces hot potato behavioral 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, we demonstrate how hot potato techniques can be effectively used during behavioral synthesis to minimize the partial scan overhead to make the synthesized design testable.**

***Index Terms*—Digital system testing, high-level synthesis, optimization, sequential logic circuit.**

I. Introduction

TRANSFORMATIONS are alternations in the structure of a control-data flow graph (CDFG) in such a way that the functionality of the initial specification is maintained [1], [2]. Transformations are standard optimization tools in many research and commercial domains, including compilers [1], symbolic mathematics [3], logic synthesis [4], and behavioral synthesis [2]. In this paper, we introduce a new redundancy replication transformation. Interestingly, it does not improve the 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 especially well suited for behavioral 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 have an identity element associated with it. For instance, addition and subtraction operations have an identity element of zero, while multiplication and division operations have an identity element of 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 previously applied in the computer networks domain [5]. 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.

The assumed computational model is synchronous data flow [6]. The synchronous data-flow model of computation assumes an infinite stream of input data, where each input has to be processed and output within the user specified amount of time. We follow standard signal processing and communication notation, where the data that is produced in one period and used in the next is denoted by the rectangle with inscribed letter "D." This element is called a delay. In this model, the speed of the design is dictated by two different metrics: throughput and latency. Throughput is equal to the longest path between two delays and latency is equal to the longest path between a primary input or delay to a primary output. We refer to the critical path as the longer of these two paths. 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 into a 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 behavioral synthesis systems [7], but also in many application specified integrated circuit (ASIC) and general purpose datapaths.

Fig. 1(a) shows an example which illustrates how a deflection operation can be used to reduce interconnect requirements, while preserving the resource utilization of other components on the datapath and maintaining the throughput requirements. Let us denote an addition by $+$, a subtraction by $-$, a multiplication by *, a shift by $>>$, and inputs and outputs by IN and OUT. Also, let A, S, M, and SH denote, respectively, an 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. Finally, the positive and negative input of a subtraction and subtracter are labeled P and N, respectively.

Inspection of the computation graph on Fig. 1(a) indicates a total of 12 edges: IN $\rightarrow$ P$-$, IN $\rightarrow$ N$-$, IN $\rightarrow$ L$+$, IN $\rightarrow$ R*,

Manuscript received November 18, 2002; revised May 6, 2003 and July 7, 2003. This paper was recommended by Associate Editor N. K. Jha.

J. L. Wong and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095–1596 USA (e-mail:jwong@cs.ucla.edu).

S. Dey is with the Electrical and Computer Engineering Department, University of California at San Diego, La Jolla, CA 92093 USA.

Digital Object Identifier 10.1109/TCAD.2003.819894

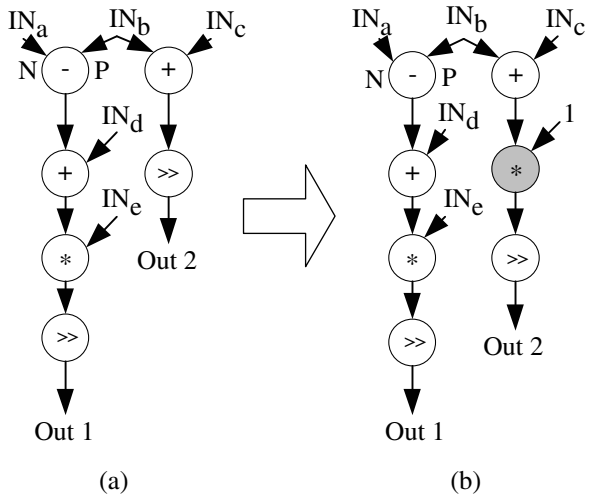

Fig. 1. Reducing the number of interconnects by adding deflection operations.

$- \rightarrow R+$, $+ \rightarrow L^*$, $* \rightarrow >>$, $+ \rightarrow >>$, and two instances of both IN $\rightarrow$ R+ and $>> \rightarrow$OUT. At the RT level there are ten corresponding interconnects, since two instances of IN $\rightarrow$ R+ can share the interconnect IN$\rightarrow$ RA and the instances of $>> \rightarrow$ OUT can share the interconnect SH $\rightarrow$ OUT.

Fig. 1(b) shows a computation functionally isomorphic to the computation of Fig. 1(a), 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 $\rightarrow$ P, IN $\rightarrow$ N, IN $\rightarrow$ RA, IN $\rightarrow$ LA, IN $\rightarrow$ RM, S $\rightarrow$ LA, A $\rightarrow$ LM, M $\rightarrow$ SH, and SH $\rightarrow$ 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.

II. 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 computer-aided design areas. It appears that the idea of replicating data (messages) was first proposed in computer networks research by Baran in 1964 [5].

In logic synthesis, the idea of replicating logic gates so that the critical path is minimized during the partitioning of large designs was first proposed by Lawler *et al.* [8]. Later, it was successfully modified and applied in several logic synthesis projects [9].

The replication of arithmetic operations using algebraic laws so that the critical path can be reduced was first proposed by Lobo and Pangrle [10]. Potkonjak and Rabaey, using the 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 a maximally fast and arbitrarily fast fashion [11]. Srivastava and Potkonjak [12] enhanced this technique and showed how the maximally fast technique, can be used as a building block for an integrated set of transformation which simultaneously optimizes throughput and latency, while in many cases also reducing area and power requirements.

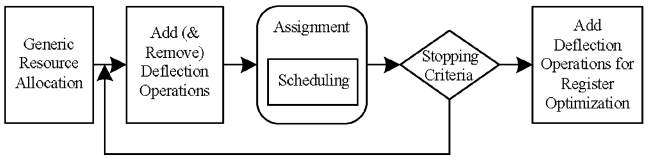

Fig. 2. Hot potato techniques for resource utilization optimization: the organization of algorithmic steps.

Recently, the interaction between synthesis, at higher-levels of abstraction and testability attracted a great deal of attention. Notable efforts in behavioral synthesis-level testing include [13], [14], in system-level testing [15], in RT-level testing [16], [17], in testing of control and interface logic [18], and in high-level testing surveys [19]. More recent research in behaviorial synthesis includes [20].

It is interesting to compare our work directly with the work presented in [21]. Herrman and Ernst also proposed a systematic algorithm for the addition of deflection operations for better sharing of interconnect. While we implicitly consider all edges for the addition of an operation, they explicitly enumerate all cases. Both algorithms are global. Herrman and Ernst are more general in the sense that they consider three different hardware models. Our approach is more general in the sense that we consider both register and interconnect optimization and interaction with other transformations. They report results with a similar level of improvement for interconnect. We targeted area optimization by conducting simultaneous reduction of both register and interconnect and achieved the same level of reduction for both entities. Two other efforts that create redundant algebraic operations in behaviorial synthesis are [10] and [22]. They focus on reduction of the critical path and interconnect. It is also interesting to note that Parulkar *et al.* [13] exploited the introduction of redundant computations in order to reduce BIST resources. A more detailed version of this work can be found at [23].

III. Optimizing Resource Utilization Using Deflection Operations

The overall flow of the hot potato transformations algorithm for optimization of resource utilization is shown in Fig. 2. The initial allocation of execution units is provided by standard behavioral synthesis. We have been using the HYPER behavioral synthesis system [7], mainly because of its full software modularity, and already existing mechanism for the feedback information flow between different behavioral 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.

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. 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, while not reducing the likelihood for scheduling within the given resource (execution units) and timing (available number of control steps) constraints.

We add or delete one deflection operation at a time. In order to avoid producing only locally optimal solutions, 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 incomplete knowledge about effects of the later deflection operations 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 to the following criteria which are listed in a order of decreasing importance ("place" criteria).

1) Edges which are not on the critical paths, and which do not significantly reduce scheduling mobility [2] of operations. This measure is mainly concerned with the potential negative impact of the deflection operations. Specifically, in principle all nodes in the transitive fan in and transitive fan out of a deflection operation will have reduced their slack.
2) Edges which correspond to rarely used interconnects. This measure is mainly concerned with the potential benefit, essentially are we able to eliminate a particular type of interconnect at the structural level. Obviously, if an interconnect is rarely used, it is more likely that we will be able to eliminate it.
3) Edges which can be replaced by a combination of two other transfers (due to a deflection operation) so that new edges have high likelihoods of using already existing operations. This measure aims to ensure that two transfers needed after the addition of deflection operations are overall easier than the initial transfer function before the operation is added.
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. While the first three measures were aiming at direct benefits, this measure is trying to ensure that some potential candidates for adding deflection operations are not eliminated when there is no immediate benefit.

Note that the above five criteria are global in the sense that at any point in time decisions are made with respect to all components in the CDFG. The change in scheduling mobility is calculated as the reduction of the 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 the 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 introduced deflection operation is decided using the following three criteria.

1) Operations which reduce 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 as soon as possible and as late as possible (ASAP-ALAP) times of operations [24].
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 situations and, therefore, caused scheduling and assignment difficulties. This information is used as an indication of 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 completed, the information about the lifetimes of variables is not available. Therefore, it is difficult to estimate the 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.

As we already indicated, one of the criteria, used during the addition of deflection operations for reducing interconnect requirements, targets the 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 the addition of new interconnects is not allowed.

Each register 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. Registers with one variable are targeted first, followed by registers with the smallest number of variables.
2) Sum of the lifetimes of stored variables. The registers that are used in a smaller number of control steps to store variables are our primary targets for early elimination.
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 pseudocode shown in Fig. 3. The stopping criteria is a user-defined runtime 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.

IV. Using Deflection Operations to Minimize Partial Scan Overhead for Improved Testability

In this section, we demonstrate how the hot potato transformation technique can be used to minimize the number of scan

```
add_deflection_ops_to_reduce_the_number_of_interconnect_and_registers(){
1.  resource_allocation();
2.  while (stopping criteria1){ /* 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 criteria2){ /* 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. }}
```

Fig. 3. Pseudocode for introducing deflection operations for interconnect and register reduction.

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 the formation of loops during assignment. The synthesis for testability algorithm (SFT) presented in [25] is then applied to the transformed CDFG to synthesize a testable data path with the minimal number of scan registers.

A. Motivational Example

Consider the CDFG of the continued fraction IIR filter shown in Fig. 4(a). To break the CDFG loops, the SFT approach [25] selects the scan variables (D1, $+2$), (D2, $+1$), and (-2, $+4$), requiring 3 scan registers, as the selected scan variables cannot be shared. The notation (a, b) indicates a variable which is produced by operation a and consumed by operation b. Note that the CDFG loops cannot be broken using fewer scan registers. An additional scan register is needed to avoid the formation of loops during assignment.

Assume that each operation in the CDFG takes one control cycle. Fig. 4(a) shows the schedule and assignment obtained by the SFT approach [25], satisfying the available time of seven control steps, and using the minimal number of execution units. For instance, the operation $+1$ is scheduled in control step three, and assigned to adder A1, shown by the tuple (3, A1). The resultant data path is shown in Fig. 5(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 Fig. 4(a), both operations $+3$ and $+4$ have 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 [25]. Though there is no path in the CDFG from a subtraction operation back to a subtraction 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 [25]. The scan register selected by SFT to avoid the assignment and false loops during assignment is RA1, for a total of four scan registers. Note that scanning the four registers L3A1, L1A1, RM1, and RA1 leaves no loops, besides self-loops, in the data path.

Fig. 4(b) shows the original CDFG of Fig. 4(a), with deflection operations added. The SFT approach is applied to the transformed CDFG [Fig. 4(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. The addition of deflection operations reduces the minimum number of scan registers required to break all CDFG loops from three in the original CDFG to one 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 Fig. 5(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 one scan register as opposed to four scan registers needed for the original CDFG. Note that three scan registers are equivalent to $3n$ scan flip-flops where n is the word length used in the design. In this example, n is equal to 16. We describe the process of adding the suitable deflection operations to the original CDFG in the following sections.

B. Selecting Scan Variables to Break CDFG Loops

The problem of breaking CDFG loops using a *minimal* number of scan registers was introduced in [25]. The minimum *hardware-shared cut* (HSC) problem [25] 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.
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 to satisfy 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-based technique [26] 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.

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: 1) x is a corresponding input to an operation of the same type f_v and 2) the

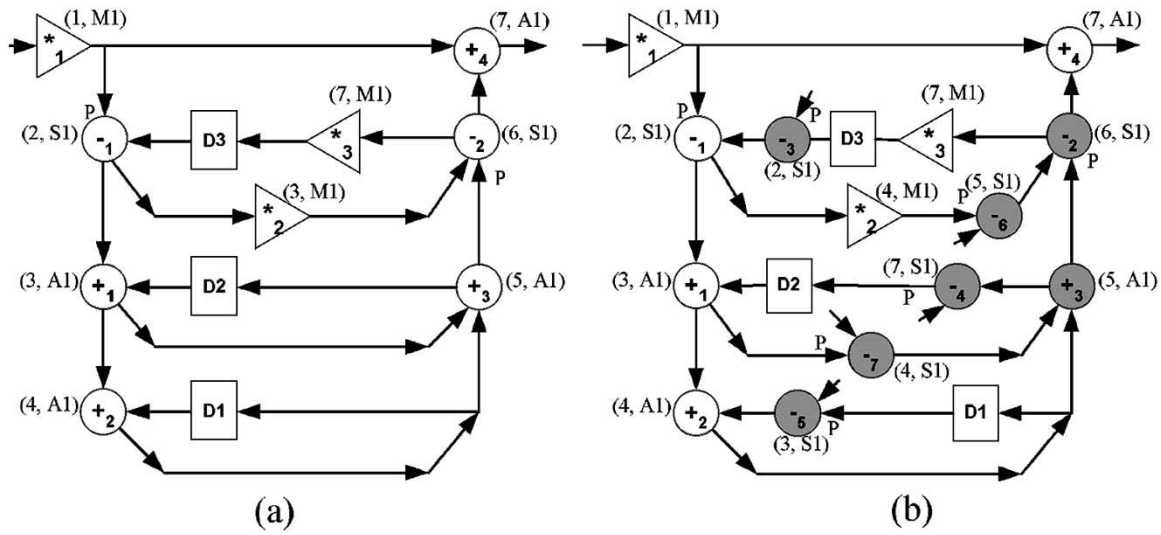

Fig. 4. CDFG of the continued fraction example (a) before and (b) after the addition of deflection operations.

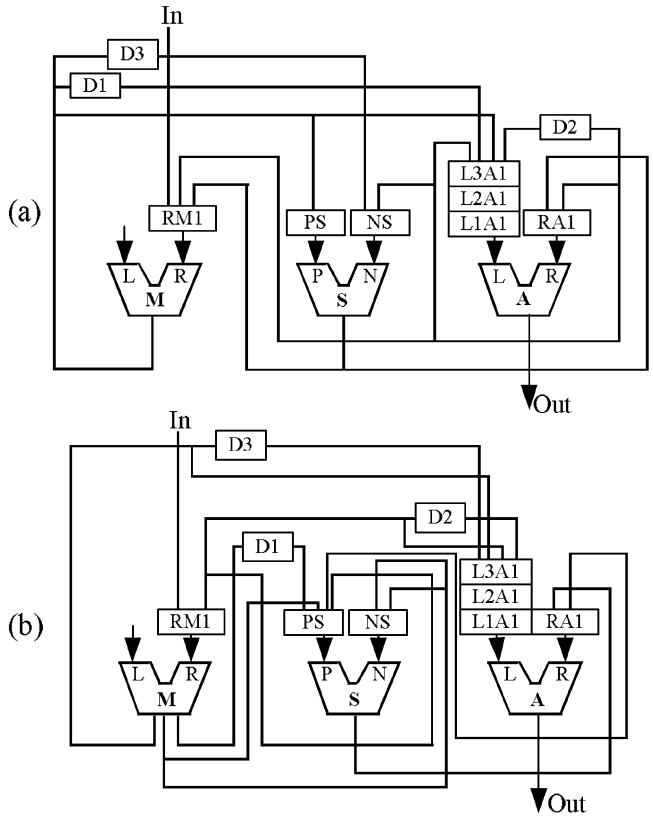

Fig. 5. Datapath for the examples of Fig. 4(a) and (b).

lifetimes of v and x do not overlap. Let set X contain those variables x which can share a scan register with variable v. 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: $\mathrm{HSE}(v) = w_1|Y| + w_2|X - Y|$.

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 $\mathrm{eff}_{\mathrm{HSE}}(e) = \alpha \times \mathrm{HSE}(e) + \beta \times \mathrm{HSE}(e)$. For each SCC, we select as scan variable the edge e with highest $\mathrm{eff}_{\mathrm{HSC}}(e)$, 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.

C. 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**, are true:

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

B2: Variables v and w have overlapping lifetimes.

1) Adding Deflection Operations to Eliminate Bottleneck B1: Consider the CDFG shown in Fig. 6(a). It has two loops: loop L1 consisting of $-$ and $+$ operations and loop L2 containing operations of the type * and >>. Variables (D1, -1) and (D2, *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,

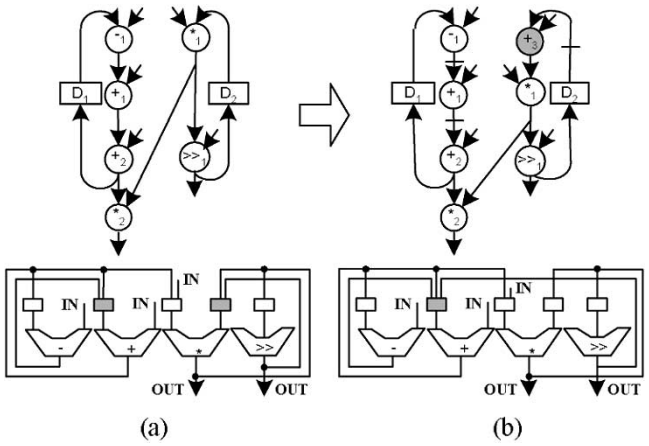

Fig. 6. Using deflection operations for testability. CDFG and RT-level specification before and after the addition of a deflection operation.

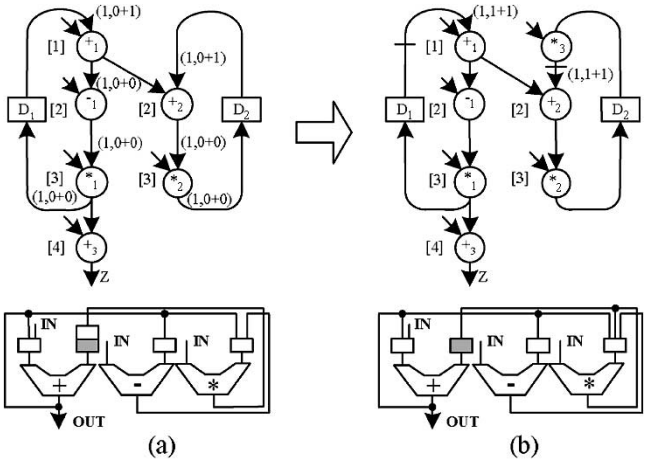

Fig. 7. Using deflection operations for testability by reducing lifetimes of scanned variables. CDFG and RT-level specification before and after the addition of a deflection operation.

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 Fig. 6(b), the variables $(-1, +1)$ in loop L1 and (D2, $+3$) in loop L2, do not have either of the bottlenecks, and can share the same register. Consequently, selecting $(-1, +1)$ and (D2, $+3$) as scan variables breaks the two CDFG loops and only one scan register is required.

Note that for any variable v that can be used to break a CDFG loop in Fig. 6(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 Fig. 6(b) increases the value of $|Y|$ to 1 for the variable $(-1, +1)$ and adds a new variable (D2, $+3$) 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 (D2, $+3$) 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.

2) Adding Deflection Operations to Eliminate Bottleneck B2: Deflection operations can also be used to eliminate the hardware-sharing bottleneck B2. 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 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 Fig. 7(a), consisting of two 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 of variables that are inputs to the same type of operations is: (D1, $+1$) and (D2, $+2$). 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 Fig. 7(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 of the components. For example, the variable (D1, $+1$) breaks one loop and, hence, has an LCE value of 1. Also, $X(v) = \{*_1, +_3\}$, $Y(v) = \{\}$. Hence, the first component of the HSE measure is 0, and the second component is 1, as shown by the tuple $(1, 0 + 1)$ at the side of variable (D1, $+1$).

Consider each variable in the CDFG which can break loop L2, not broken by scanning variable (D1, $+1$). Consider one such variable (D2, $+2$). This 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 (D2, $+2$), such that the lifetime of the split variable ($*3$, $+2$) no longer overlaps with the lifetime of variable (D1, $+1$). The transformed CDFG is shown in Fig. 7(b). The bottleneck B2 is eliminated, and variables (D1, $+1$) 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 Fig. 7(b), variables (D1, $+1$) and ($*3$, $+2$) can share the same scan register and, hence, the HSE measure of the variables (D1, $+1$) increases from $(0 + 1)$ to $(1 + 1)$. Similarly, the split variable ($*3$, $+2$) now has an HSE value $(1 + 1)$. The scan selection process would select variables (D1, $+1$), ($*3$, $+2$) to break the CDFG loops, requiring only one scan register.

3) Algorithm for Adding Deflection Operations: We briefly outline the process of adding the deflection operation, in order to minimize the number of scan registers required by the scan variables to break the CDFG loops is minimized. Let $\mathrm{Z}(v)$ 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

```
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);
```

Fig. 8. Pseudocode for introducing deflection operations for breaking CDFG loops.

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 the algorithm for adding deflection operations to minimize the number of scan registers needed to break CDFG loops in Fig. 8.

D. 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). Each node in the graph represents an operation of the CDFG. There is an (undirected) compatibility edge between two operations, if there is a nonzero 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 $d(v, w)$.

Fig. 9(a) shows segments of two paths in a CDFG, and Fig. 9(b) shows the corresponding DDCG. 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 [2] is shown by the tuple [ASAP, ALAP] in Fig. 9(a). In Fig. 9(b), each compatibility edge $c(v, w)$, shown dotted, is weighted by an estimate of the compatibility, $\mathrm{comp}(v, w)$, between two nodes v and w, as given by

$$\mathrm{Comp}(v,w) = \begin{cases} 1, & \exists\ \mathrm{d(v,w)} \\ 1 - \frac{\mathrm{overlap(v,w)}}{\mathrm{mobility(v)}^{*}\mathrm{mobility(w)}}, & \text{otherwise.} \end{cases}$$

The compatibility between nodes $+1$ and $+3$, $\mathrm{Comp}(+1, +3) = 1 - (1/2) = 0.5$. All the compatibility edges $c(v, w)$ are weighted by $\mathrm{Comp}(v, w)$, as shown in the DDCG in Fig. 9(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 $p\dot{q}$, 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 Fig. 9(b), the path $(+5, +1)$, $(+1, +2)$, $(+2, +5)$ can be represented by cd^+.

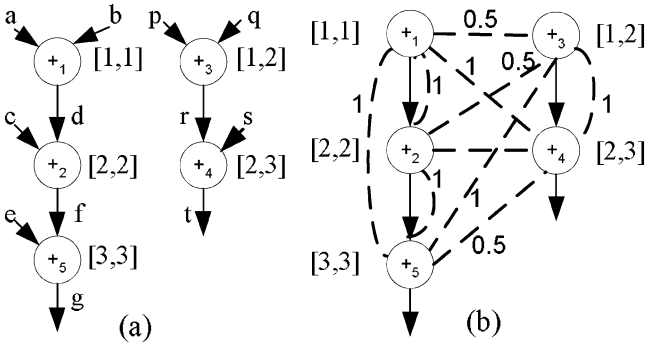

Fig. 9. CDFG (a) and the corresponding DDCG (b).

```
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 opd in CDFG;
9.     delete the d edge in DDCG;
     }}
10. return transformed CDFG, with added deflection operations.
```

Fig. 10. Pseudocode for introducing deflection operations for avoiding CDFG loops.

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, in addition to two assignment loops, a sequential false loop is formed in the data path.

E. 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 of 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 Fig. 10. 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 $\mathrm{comp}(v, w)$ on their c edges. We ignore those loops that have a low possibility of forming, reflected by low values of $\mathrm{comp}(v, w)$ on their c edges. To achieve this, we delete all those compatibility edges which have a compatibility estimate less than a threshold.

TABLE I
EFFECT OF HOT POTATO TECHNIQUE (HP) ON RESOURCE UTILIZATION

Example	InitReg	HPReg	MinReg	InitIntc	HPIntc	% RegRed	% IntcRed	% AreaRed	% PwrRed
speech filter	35	29	22	15	11	17	27	26	2
7IIR	32	23	18	15	10	28	33	31	5
8IIR	39	32	23	17	11	18	39	35	8
IIR GM	44	35	24	18	11	20	39	35	7
Cascade	52	44	30	14	11	15	21	20	2
3 Volterra	35	31	23	17	9	13	47	40	14
4 Volterra	48	44	30	37	23	8	38	32	10
100 FIR	132	121	88	14	9	8	36	23	3
8 DCT	19	18	15	16	9	5	44	33	8
9 FFT	21	19	15	14	8	10	43	33	7
Ghost	102	80	68	42	32	22	24	24	5
DAC	120	80	70	40	30	33	25	24	9
Echo	160	115	98	44	28	28	36	34	8
Honda CTL	202	170	152	45	29	16	36	33	12
Aircraft CTL	240	202	187	60	36	16	40	35	9

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. Otherwise, the addition of the deflection operations will have minimized the chances of the formation of loops during the scheduling and assignment phase.

V. EVALUATION

The hot potato transformation approach for the optimization of resource utilization was applied on the following 15 behavioral synthesis examples: speech filter, seventh order cascade IIR filter (7IIR), eighth order parallel IIR filter (8IIR), Gray Markel filter, third and fourth order Volterra filters, hundredth order FIR filter, 8 point discrete cosine transform (8DCT), 9 point Winograd fast Fourier transform (FFT), ghost signal cancellation filter (Ghost), digital to analog convertor (DAC), NEC echo canceller (Echo), Honda controller (Honda CTL), and aircraft controller (Aircraft CTL). The examples were scheduled using the HYPER behavioral-synthesis tools [7].

The initial and final number of register and interconnects, as well as relative reduction in their numbers, are given in Table I. We present the number of registers before and after the application of HP (**InitReg** and **HPReg**), the minimum number of registers required assuming data can be transferred between any two registers directly (**MinReg**), and the number of interconnects before and after the application of HP (**InitIntc** and **HPIntc**). We denote the percentage of reduction in the number of registers and interconnect in columns **RegRed** and **IntcRed**, the percentage of total area reduction in **AreaRed**, and the power reduction using HYPER-LP estimation tool in the last column. Both the average and the median reduction in the number of interconnects were approximately 36% and the reduction in the number of registers were 17% (average) and 16% (median). The overall impact of hot potato transformation on the area of the design is shown in the last column of Table I. The average and median area reduction were respectively 30.5% and 33%. Area estimated using HYPER-LP prediction is within 20% of the real value in more than 98% of situations. The average and median power reduction was 7.3% and 8%.

TABLE II
IMPACT OF HOT POTATO TECHNIQUE (HP) ON RESOURCE UTILIZATION OF VLIW AND SUPERSCALAR CONFIGURABLE ARCHITECTURES

Design	Total # of S Ops	Total # of D ops	Total # of Branch	Before / After	% Reduc
mpegenc	22129	1121265486	180263322	38/19	50
mpegdec	15626	175505114	16850074	38/21	44
pegwit	14009	34037680	3776971	35/16	54
jpegenc	21193	13905129	2298620	36/17	52
jpegdec	24513	3766034	270201	39/18	54

TABLE III
PERFORMANCE AND HARDWARE CHARACTERISTICS OF THE DESIGNS

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 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 IV
EFFECT OF DEFLECTION OPERATIONS ON PARTIAL SCAN OVERHEAD

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 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

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

TABLE V
SEQUENTIAL ATPG USING HITEC [27]

Design	Method	Total FFs	Scan FFs	Total Faults	Aborted Faults	Fault Cov(%)	Test Eff.(%)	ATPG CPU (sec)
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
5^{th} Order Wave Digital Filter	Orig	460	0	10364	10077	1	3	>72hr
	Orig+OPUS	460	30	10364	0	98	100	309
	BETS	480	60	10916	16	98	100	233
	HP+BETS	480	40	11130	2	98	100	200
5^{th} 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

transformations), and, therefore, the hot potato techniques can be effective on the already highly optimized designs.

An additional and natural question is how much the addition of deflection operations can help software compilers and compilation for reconfigurable architectures. In order to evaluate the impact of Hot Potato techniques on configurable architectures, we used the Trimaran-based evaluation platform. This platform enabled us to analyze the impact of the technique on both VLIW and superscalar architectures. For our experimentation, we used the following multimedia benchmarks IJG JPEG, MSG MPEG, and PEGWIT. IJG JPEG is an implementation of JPEG image compression and decompression. The MSG MPEG benchmark is a player for MPEG-1 and MPEG-2 video bitstreams. Lastly, Pegwit is a program for performing public key encryption and authentication.

Specifically, we used eight units and calculated the required number of interconnect before and after the application of the HP technique on each benchmark. We denote the number of static operations (**S Ops**), dynamic operations (**D Ops**), branch operations, the number of required interconnect **before** application of HP technique, the number of required interconnect **after** HP, and the percentage of area reduction (**Reduc**). Table II shows the average reduction in interconnect was by 50.8%.

To evaluate the effectiveness of the hot potato transformations on the testability of the designs, we applied the technique on the following examples: the third order Continued Fraction IIR filter, the Modem filter, the fifth order Elliptical Wave Digital Filter, and the fifth order IIR Cascade Filter. We first synthesize the examples using the behavioral-test synthesis system BETS [25] 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 behavioral synthesis (BETS without testability considerations), **BETS** refers to the design synthesized by BETS, while $\mathbf{HP + BETS}$ refers to the design synthesized by BETS after the addition of deflection operations for testability.

Table III 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 the 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 of the design cases, the hardware costs, like registers, multiplexers, and interconnects, are not compromised by the addition of deflection operations for testability.

Table IV shows the number of Flip-Flops (**FFs**), and the number of scan FFs (**Scan FFs**) needed to break all of the loops of the design for all three implementations, Orig, BETS, and HP + BETS versions. In the case of the original implementation, a gate-level partial scan tool OPUS [28] is used for scan FF selection. The testability of the synthesized designs was evaluated using the gate-level sequential ATPG tool HITEC [27] shown in Table V. For each design, besides the rows **Orig**, **BETS**, and $\mathbf{HP + BETS}$, the row $\mathbf{Orig + OPUS}$ refers to the data path obtained from the original data path after scanning the FFs selected by the gate-level partial scan tool OPUS [28]. 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 [27] 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.

VI. 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.

References

[1] C. Fischer and R. L. Blank, *Crafting a Compiler*. Redwood City, CA: Benjamin Cummings, 1988.

[2] R. Walker and R. Camposano, *A Survey of High-Level Synthesis Systems*. Norwell, MA: Kluwer, 1991.

[3] 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*. New York: Springer-Verlag, 1991.

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

[5] P. Baran, "On distributed communication networks," *IEEE Trans. Commun.*, vol. 12, pp. 1–9, Mar. 1964.

[6] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, pp. 24–35, Feb. 1987.

[7] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of data path intensive architecture," *IEEE Design Test*, vol. 8, pp. 40–51, June 1991.

[8] E. L. Lawler, K. N. Levitt, and J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Trans. Comput.*, vol. 18, pp. 47–57, Jan. 1969.

[9] C. Cring and A. R. Newton, "A cell-replicating approach to mincut-based circuits partitioning," in *Proc. Int. Conf. Computer-Aided Design*, 1991, pp. 2–5.

[10] D. Lobo and B. M. Pangrle, "Redundant operator creation: A scheduling optimization technique," in *Proc. Design Automation Conf.*, 1991, pp. 775–778.

[11] M. Potkonjak and J. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992, pp. 304–308.

[12] M. B. Srivastava and M. Potkonjak, "Transforming linear systems for joint latency and throughput optimization," in *Proc. Eur. Design Automation Conf.*, 1994, pp. 267–271.

[13] I. Parulkar, S. K. Gupta, and M. A. Breuer, "Allocation techniques for reducing BIST area overhead of data paths," *J. Electron. Testing-Theory Applicat.*, vol. 13, no. 2, pp. 149–166, 1998.

[14] S. Ravi, G. Lakshminarayana, and N. K. Jha, "TAO: Regular expression based high-level testability analysis and optimization," in *Proc. Int. Test Conf.*, 1998, pp. 331–340.

[15] M. Lajolo, L. Lavagno, M. Rebaudengo, M. S. Reorda, and M. Violante, "Behavioral-level test vector generation for system-on-chip designs," in *Proc. IEEE Int. High-Level Design Validation Workshop*, 2000, pp. 21–26.

[16] S. Dey and M. Potkonjak, "Nonscan design-for-testability techniques using RT-level design information," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1488–1506, Dec. 1997.

[17] I. Ghosh, A. Raghunathan, and N. K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," in *Proc. Int. Conf. Computer Design: VLSI Comput. Process.*, 1995, pp. 173–179.

[18] Y. Makris, J. Collins, and A. Orailoglu, "Fast hierarchical test path construction for circuits with DFT-free controller-datapath interface," *Electron. Test J.*, vol. 18, no. 1, pp. 29–42, 2002.

[19] K. D. Wagner and S. Dey, "High-level synthesis for testability: A survey and perspective," in *Proc. Design Automation Conf.*, 1996, pp. 131–136.

[20] M. Potkonjak and J. Rabaey, "Maximally and arbitrarily fast implementation of linear and feedback linear computations," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 30–43, Jan. 2000.

[21] D. Herrmann and R. Ernst, "Improved interconnect sharing by identity operation insertion," in *IEEE/ACM Int. Conf. Computer-Aided Design*, 1999, pp. 489–492.

[22] B. Landwehr and P. Marwedel, "A new optimization technique for improving resource exploitation and critical path minimization," in *Proc. Int. Symp. Syst. Synthesis*, 1997, pp. 65–72.

[23] J. L. Wong, M. Potkonjak, and S. Dey, "Optimizing Designs Using the Addition of Deflection Operations," Univ. California, Los Angeles, 2003.

[24] M. C. McFarland, A. C. Parker, and R. Camposano, "The high level synthesis of digital systems," *Proc. IEEE*, vol. 78, pp. 301–317, Feb. 1990.

[25] S. Dey, M. Potkonjak, and R. Roy, "Exploiting hardware-sharing in high-level synthesis for partial scan optimization," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 20–25.

[26] L. Hagen and A. B. Kahng, "A new approach to effective circuit clustering," in *Proc. Int. Conf. Computer-Aided Design*, 1992, pp. 422–427.

[27] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. Eur. Design Automation Conf.*, 1991, pp. 214–218.

[28] V. Chickermane and J. H. Patel, "An optimization-based approach to the partial scan design problem," in *Proc. Int. Test Conf.*, 1990, pp. 377–386.

Jennifer L. Wong (S'99) received the M.S. degree in computer science and engineering, in 2002, from the University of California, Los Angeles, where she is currently pursuing the Ph.D. degree.

Her research interests include intellectual property protection, embedded systems, and ad-hoc sensor networks.

Miodrag Potkonjak (S'90–M'91) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1991.

In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with the Computer Science Department, University of California, Los Angeles. His watermarking-based intellectual property protection research formed a basis for the virtual socket initiative alliance standard. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received the National Scientific Foundation CAREER Award, the OKAWA Foundation Award, the UCLA TRW SEAS Excellence in Teaching Award, and a number of best paper awards.

Sujit Dey (S'90–M'91) received the Ph.D. degree in computer science from Duke University, Durham, NC, in 1991.

He is an Associate Professor in the Electrical and Computer Engineering Department, University of California at San Diego, La Jolla (UCSD). Prior to joining UCSD in 1997, he was a Senior Research Staff Member with the Computers and Communications Research Laboratories, NEC USA, Princeton, NJ. His research group at UCSD develops innovative hardware-software architectures and methodologies to harness the full potential of nanometer technologies for future networking and wireless appliances. He has authored or coauthored one book and 90 journal and conference papers, and holds ten U.S. patents.