Optimizing Resource Utilization and Testability Using Hot Potato Techniques

Miodrag Potkonjak Sujit Dey
C&C Research Laboratories, NEC USA, Princeton, NJ 08540

ABSTRACT

This paper introduces hot potato high level synthesis transformation techniques. These techniques add deflection operations in a computation in such a way that a specific goal is optimized. We demonstrate how the requirements for two important components of the final implementation cost, registers and interconnects, are significantly reduced using new technique. It is also demonstrated how hot potato techniques can be effectively used during high level synthesis to minimize the partial scan overhead to make the synthesized design testable.

1.0 Introduction

In this paper, we introduce a new class of transformation techniques, termed **hot potato techniques**, based on adding deflection operations to the computation structure, while preserving the functionality of the computation.

Many operations used in a computation structure has an identity element associated with it. For instance, an addition operation has an identity element zero, while a multiplication operation has 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 addition to demonstrating the effectiveness of the new transformation mechanism for reducing the cost of a design, we also apply the new transformation technique to reduce the partial scan cost needed to make the resultant design testable.

We assume that the underlying hardware model is the dedicated register file model. All registers are grouped in a certain number of register files, and 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 just in several high level synthesis systems [Rab91], but also in many manual ASIC and general purpose datapaths.

In the rest of the paper, we denote an addition by +, a subtraction by -, a multiplication by *, a shift by >>, and input and output by IN and OUT. Also 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 by P and S respectively. In all motivational examples, for the sake of simplicity, it is assumed that all operations take one control step for their execution.

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. One unit of each type is sufficient to implement the computation. An inspection of the computation graph in 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.

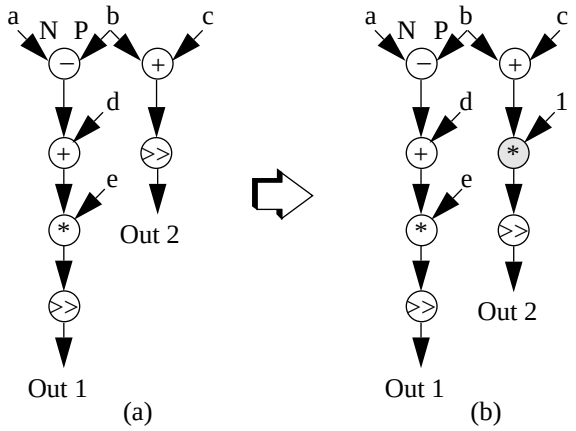

Figure 1: Reducing the number of interconnects by adding deflection operation

Figure 1b shows a computation functionally equivalent to the computation from Figure 1a, obtained by adding the deflection multiplication by 1, shown shaded. Though the new structure has 14 edges, the need for the interconnect A->SH is eliminated by reusing the existing interconnects A->LM and M->SH. Hence, 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 a better hardware sharing of interconnects is achieved. Note that neither the critical path nor the number of required execution units has been affected.

31ST ACM/IEEE Design Automation Conference ®

Figures 2a and 2b show two functionally equivalent computational structures. The structure on Figure 2b is derived from the structure on Figure 2a by adding a deflection operation on the edge V_6. The critical path in both structures is 4 control steps. The same number of

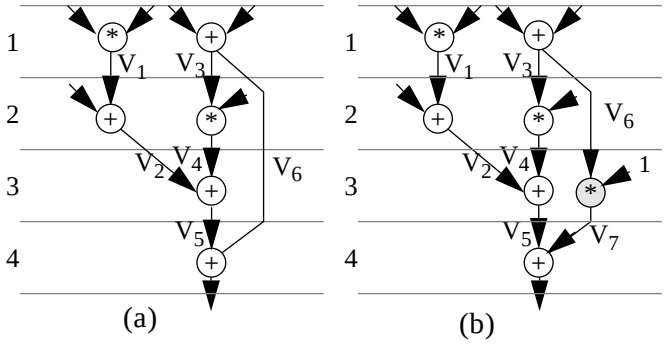

Figure 2: Reducing the number of Registers using Hot Potato Transformations

execution units, one multiplier and one adder, are sufficient for both implementations. 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.

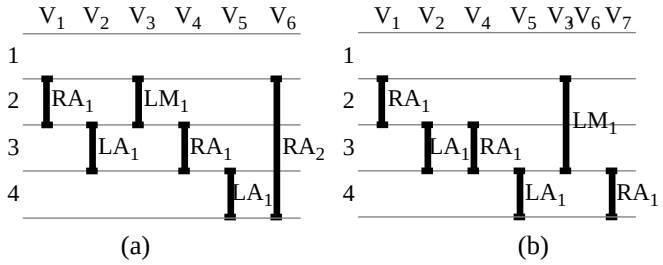

Figure 3: Lifetime for variables for examples from Figure 2a and 2b

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 LA_1, RA_1, LM_1, RM_1, represent the left and right registers of adder A and multiplier M used in the datapath. Note that in Figure 2a, variable V_6 has overlapping lifetime with variables V_1 and V_4, and hence cannot share the same register RA1 assigned to variables V_1 and V_4. In the transformed structure shown in Figure 2b, the introduction of the deflection multiplication operation reduces the lifetime of the variable V_6 at the expense of introducing a new variable V_7. However, V_6 can now be assigned to register LM_1, while the new variable V_7 can share register RA_1 with the variables V_1 and V_4. Consequently, the transformed structure requires one less register than the original structure

The idea of operation 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 the computer networks domain by Baran in 1964 [Bar64].

More recently, in the related distributed computer architecture research on wormhole routing, it was demonstrated that replicating messages can be effectively used for reducing probability of deadlocks [Dal87].Another interesting and powerful idea in the computer architecture domain on enhancing performance by adding dummy hardware units was proposed by Patel [Pat76]. In logic synthesis, the replication of logic gates is used in several projects [Cri91, Hwa92]. The replication of arithmetic operations for critical path reduction was proposed in high level synthesis [Lob91, Pot92].

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, but also testability. While high level synthesis research was restricted until now on studying the influence of scheduling and assignment on testability, this paper shows that transformations have intrinsically more power to ensure easily testable datapaths.

2.0 Optimizing Resource Utilization using Deflection Operations

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. The advantage of this approach is that as we are proceeding with scheduling and assignment, the 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.

It is important to preserve 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 difficulties of scheduling to satisfy constraints are used to guide addition or removal of deflection operation in the next iteration.

The addition of deflection operation is performed in the following way. The initial allocation of execution units is provided by standard high level synthesis. We have used Hyper [Rab91] for this task.

The primary goal during the introduction of deflection operations is to eliminate as many interconnects as possible, while not reducing the likelihood for scheduling within the given resource and timing 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. 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.

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 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 correspond 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 good 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 the use of an interconnect is calculated as 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.

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 reduce the mobility least.
2. Operations which are not competing for the critical resources which are bound to be scheduled in the same control cycles. The level of competition is calculated as the sum of ASAP-ALAP times of operations.
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 caused scheduling and assignment difficulties. This information is used as indication which deflection operation to delete or add.

The optimization of registers is a significantly more complex problem since prior to the completion of scheduling and assignment, the information about lifetimes of variables is not available. Hence it is difficult to estimate eventual register requirements before scheduling is done.

One of the criteria, used during the addition of deflection operations for reducing interconnect requirements, targets reduction of registers. After the optimization of interconnects, and the synthesis of the 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 interconnects, 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.

3.0 Minimizing Partial Scan Overhead for Improved Testability

Recently, several high level synthesis techniques have been proposed to generate easily testable data paths [Pap91,Lee93,Dey93]. Since presence of loops in a sequential circuit have been shown to be primarily responsible for making sequential automatic test pattern generation difficult, several techniques have been

suggested to synthesize data paths without loops, by using proper scheduling and assignment, and scan registers to break loops [Lee93, Dey93]. Besides the presence of CDFG loops, several other types of loops can be introduced in the data path during hardware sharing [Dey93].

Figures 4 and 5 show two ways how the introduction of a deflection operation can reduce partial 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 the 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 by reusing the same scan registers.

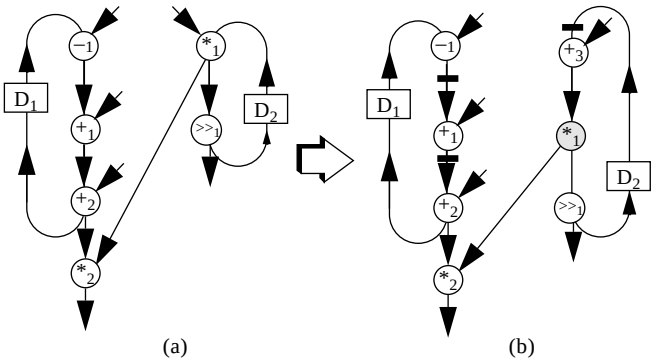

Figure 4: Using deflection operation for testability

The example from Figure 4 has two CDFG loops which are bound to produce two loops in the datapath. Since the first loop has only additions 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 shaded in Figure 8(b), 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 5 with an available time of 4 control steps. 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). At least two scan registers are needed to break the two CDFG loops, regardless of the scheduling and assignment, since it is

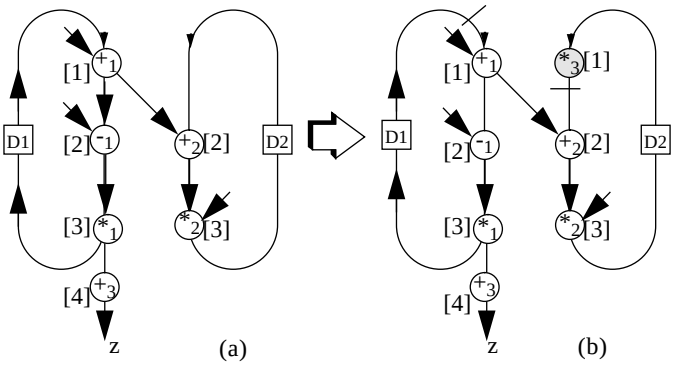

Figure 5: Minimizing partial scan cost by using deflection operations to reduce lifetimes of scanned variables

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. Consequently, the partial scan overhead is reduced from two scan registers to one scan register. We have developed an algorithm to simultaneously optimize resource utilization and testability using hot potato techniques. The pseudocode and algorithmic details of the approach are given in [Pot93].

4.0 Experimental Results

The hot potato transformation approach for the optimization of resource utilization was applied on seven high level synthesis examples. The initial and final number of registers and interconnects, as well as percentage reduction in their numbers, are shown in Table 1. The

Example	IR	HR	II	HI	RR	RI
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

Table 1 Effects of Hot Potato Technique (HPT) on Resource Utilization: IR (II)and HR (HI) - the number of registers (interconnects) before and after the application of HPT; RR and RI - reduction in the number of registers and interconnect (%)

average reduction in the number of interconnects and registers were 35% and 17% respectively. An important advantage of the hot potato transformations techniques for resource utilization is that they are orthogonal with other transformations.

To evaluate the effectiveness of the hot potato techniques on the testability of the designs, we used two examples: the Continued Fraction (CF) filter, and the Modem. We first synthesized the examples using the SFT approach [Dey93] for both testability and resource utilization (SFT). Next, we transformed the examples by

Design	Method	CS	EXU	R	M	I
C.F	SFT	7	1M,1S,1A	9	7	13
	HP+SFT	7	1M,1S,1A	8	7	12
Modem	SFT	10	3M,2A,1S	14	8	21
	HP+SFT	10	3M,2A,1S	13	8	21

Table 2: Characteristics of the Designs: R - registers, M - mixes; I - interconnect

Example	Method	FFs	Scan FFs
CF	SFT	180	80
	HP+SFT	160	20
Modem	SFT	228	48
	HP+SFT	204	16

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

adding deflection operations for minimizing partial scan cost, and then applied SFT on the transformed CDFG (HP + SFT). Table 2 shows the physical characteristics of the designs. The number of control steps (CS) and execution units (EXU) needed by the implementation after adding the deflection operations remain same. Similarly, the other hardware costs, like number of registers (R), multiplexers (M) and interconnects (I), are not compromised by the addition of the deflection operations for testability.

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. Consider the case of the Continued Fraction filter (CF). To avoid loops in the data path, SFT needed to scan 80 FFs. However, when SFT is given the transformed CDFG with deflection operations, it requires only 20 scan FFs. As expected, though the original designs without scan FFs are very hard to test, 100% test efficiency could be obtained by running the sequential test pattern generator HITEC [Nie91] on the final designs after scanning the selected FFs. The results demonstrate the effectiveness of the hot potato techniques to significantly reduce the scan overhead required to synthesize testable data paths.

5.0 Conclusion

We have proposed a hot potato transformation technique based on the introduction of deflection operations. We have shown the application of the technique to reduce interconnect and register requirements, while preserving throughput, and to minimize the partial scan overhead required to generate easily testable data paths.

6.0 References

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

[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.

[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.

[Dey93] S. Dey, M. Potkonjak, R. Roy: “Exploiting Hardware-Sharing in High Level Synthesis for Partial Scan Optimization”, ICCAD-93, pp. 20-25, 1993.

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

[Lee93] T.C. Lee, et. al.: “Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and partial Scan Environment”, DAC-93, pp. 292-297, 1993.

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

[McF92] 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.

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

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

[Pat78] J.H. Patel: “Pipelines with Internal Buffers”, Symposium Comp. Architecture, pp. 249-255, 1978.

[Pot92] M. Potkonjak, J. Rabaey: “Maximally Fast and Arbitrarily Fast Implementation of Linear Computations”, ICCAD-92, pp. 304-308, November 1992.

[Pot93] M. Potkonjak, S. Dey: “Optimizing Resource Utilization and Testability using Hot Potato Techniques”, Technical Report, NEC USA, Nov. 1993.