MINIMIZING GLOBAL INTERCONNECT IN DSP SYSTEMS USING BYPASSING

Jennifer L. Wong
SUNY Stony Brook University
CS Department
Stony Brook, NY 11794

Seapahn Megerian
Univ. of Wisconsin, Madison
ECE Department
Madison, WI 53706

Miodrag Potkonjak
Univ. of California, Los Angeles
CS Department
Los Angeles, CA 90095

ABSTRACT

There is a wide consensus that performance and power consumption of IC designs in deep submicron technologies is mainly dictated by interconnect requirements. Our goal is demonstrate how compilation and architectural techniques can be used to minimize and balance interconnect requirements. Specifically, we target the use of bypass units to reduce routing congestion and eliminate long interconnections while essentially preserving or even improving the throughput requirements. We formulate the bypassing problem, establish its complexity and develop an efficient integer-linear programming (ILP) formulation. In addition to satisfying user specified interconnect requirements, we simultaneously optimize the number of operations and, therefore runtime of the targeted application.The approach is prototyped and evaluated using a platform consisting of the Trimaran architecture and compilation tools and the CPLEX ILP solver.

Index Terms— Circuit synthesis, Circuit optimization, Digital Signal Processors, Multiprocessor interconnection

1. INTRODUCTION

Deep submicron technology is the key enabler of ultra large integration, GHz clock speeds, and low power designs. At the same time, technology progress drastically alters the nature of design and the importance of specific design constraints. Probably the most profound change is the rapidly increasing importance of interconnect. While in previous technologies, delay and power consumption were dominated by logic gates, in modern and pending technologies it is and will be dominated by interconnect. As a result, a number of approaches at all levels of the design process have been proposed to reduce the amount of long interconnects.

Bypassing (also known as template matching and the addition of deflection operations) is a widely used architectural technique for design optimization that mainly targets the reduction on the number of operations and improvements in clock cycle utilization. To the best of our knowledge, this is the first proposal of their use for minimization of long interconnect and the minimization of routing density. We propose an efficient ILP formulation for the identification of positions for insertion of bypassing operations in the design. The effectiveness of the approach is amplified due to the high flexibility of the technique: the user can define an arbitrary set of conditions on the structure of interconnect, including their total number and the maximal number of incoming and outgoing interconnect for each individual functional unit.

We introduce two main novelties that significantly reduce the complexity of the ILP formulations: the use of dummy transfer variables and the recursive development of ILP formulations. We compound these technique with the systematic use of schemes for the translation of Boolean constraints into an ILP format and aggressive preprocessing, in order to enable rapid derivation of optimal solutions for all design instances of interest. The effectiveness of the optimization approach and ILP formulation is demonstrated on large scale design using the TRIMARAN platform [1].

2. RELATED WORK

In this section, we briefly survey the related work on the use of bypassing operations, coordinated optimization in behavioral and physical designs, and interconnect optimization in deep submicron designs.

Bypassing was first introduced in communication literature [2]. Later, it was also used, under the name of deflection operations and bypassing in behavioral synthesis [3, 4, 5] in order to heuristically address the reduction of local interconnect and the size of registers files.

The importance of addressing long interconnect in deep submicron designs has been well established [6, 7, 8]. A number of techniques for addressing physical design problems during behavioral and architectural synthesis has been proposed [9]. We differ from the previous research not only by establishing the complexity of the problems and providing practical provably optimal solutions, but also by posing the problems in a significantly more generic and flexible way so that any specific structure of interconnect can be targeted for

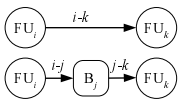


Fig. 1. Illustration of adding a bypass operation to a transfer.

optimization. Recently, bypassing attracted a lot of attention in general purpose computing community where it has been considered as one of main mechanisms for creating scalable multiprocessor architectures [10, 11].

3. CONCEPTS AND TRADE-OFFS

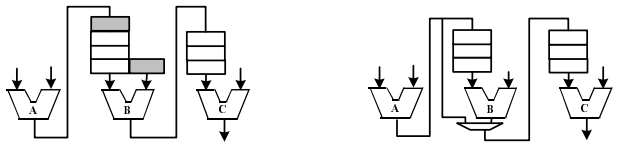
A *functional unit (FU)* is an operation such as an fixed point or floating point ALU, multiplication, branch predictor, shifter, etc. We define an *architecture* as a set of FUs and global interconnect between two distinct FUs. A *transfer* exists if the result of one FU is an input to another FU. We will define transfers between two FUs, FU_i and FU_k as $i \rightarrow k$.

Bypassing is a generic architectural transformation that can be applied on a variety of architectures. In order to ensure that our approach is directly applicable to the standard design practice, in our treatment we follow the standard industrial architectural organization where a single register file is connected to each input of an execution unit. This organization has numerous advantages including compact layout and reduced amount of interconnect. The layout is compact since all registers are aligned, and share multiplexing, demultiplexing, and control logic between registers. The amount of interconnect is reduced since all registers in one register file share the same set of interconnections.

Bypassing is an architectural transformation where we add an alternative path in conjunction with an existing functional unit. The net effect is that an interconnect from unit FU_i to FU_j is realized using two already existing interconnects: from FU_i to unit B_j and from B_j to functional unit FU_j (Figure 1). Bypassing can be accomplished in two ways: (i) using identity operations and (ii) by adding a bypassing path. The use of identity operations (e.g. 0 for addition or 1 for multiplication) results in a need for an additional register as well as an additional clock cycle for transfer of data from FU_i to FU_j through bypassing unit B_j.

Note that the bypassing path can be added either before (Figure 2(a)) or after the register file (Figure 2(b)). Each of the alternatives has certain advantages and limitations. In the scheme shown in Figure 2(a), regardless of the number of sources and destinations that use the bypassing path of unit B, there is only a need to add two sets of n registers for an n-bit wide datapath. However, in our implementation, we adopted the scheme shown in Figure 2(b) because this architecture does not introduce an additional clock cycle for completion of the data transfer through a bypassed functional unit as required by the first scheme. Finally note that the addition of a bypass path does not increase the number of multiplexers: while a set of n MUXs is added after the unit B simultaneously we eliminate a set of n MUXs in front of unit C.

The bypass of a functional unit is accomplished by adding a set of demultiplexers on the input of the unit and a set of multiplexers on its output. The alternative path consist only of local interconnect from the demultiplexers to the multiplexers. In many cases, for operations such as addition and multiplication, one can use identity elements to accomplish the same effect.



(a) (b)

Fig. 2. Realization of bypassing transformation.

4. ADDITION OF BYPASS UNITS

In this section, we present an ILP-based approach for the addition of bypass operations to eliminate all transfers between a pair of FUs and therefore long interconnect. The main idea is to use two existing local interconnects in order to eliminate a long interconnect.

We focus our attention on a program P that is compiled to an architecture with specific constraints on the number of incoming interconnect for each FU. The goal is to introduce the minimal number of bypass operations between consecutive operations (transfers) in such a way that the number of incoming interconnect for each FU is within the constraints of the architecture. Note that by minimizing the number of added bypass operations, the upper bound on the number of additional cycles which will be necessary to schedule the operations is minimized. Formally, the problem can be defined as follows.

Problem: *Addition of Bypass Operations with Maximal Incoming Interconnect per Functional Unit*
Instance: Given a set of architecture constraints A over a set of FUs M, for each $m \in M$ a maximum number of incoming interconnect $I(m) \in Z^+$, a program P with a set of consecutive operations T, for each $t \in T$ a frequency $F(t)$, positive B.
Question: Can P be scheduled with A so that the maximum number of incoming interconnect is satisfied and includes at most B bypass operations?

Note, we do not consider local interconnect, ie. transfers of type $i \rightarrow i$. Additionally, we allow multiple FUs of the same operation. In this case, the frequency as to which transfers occur for each of the duplicated units must be defined independently.

We have proved that the bypass problem is NP-complete using the degree-bounded connected subgraph (DBCS) problem as the starting point. The DBCS problem asks, Is there a subset of edges with a given cardinality in a given graph such that the subgraph is connected and no node has a degree exceeding a given number? The degree-bounded connected subgraph problem is equivalent to a special instance of the bypass problem where the goal is to minimize the number of

transfers under a uniform constraint on the number of incoming interconnect for all units, assuming that there is an equal number of transfers between all pairs of distinct FUs.

We introduce two constants for the ILP formulation for bypass addition. The first constant I_m is the number of allowable incoming interconnect for each type of FU, which is dictated by the architectural constraints. The second constant F_{ik} denotes the frequency of usage of transfer $i \rightarrow k$, which is dictated by program P. All F_{ii} are assigned to 0, since we do not consider local interconnect.

$$I_m = \text{number of incoming interconnects for FU}_m$$

$$F_{ik} = \text{original number of transfers from } i \rightarrow k$$

$$x_{ik} = \begin{cases} 1, & \text{if the transfer from } i \rightarrow k \text{ remains} \\ 0, & \text{otherwise (i.e. } i = k\text{).} \end{cases}$$

$$b_{ijk} = \begin{cases} 1, & \text{if bypass of type } j \text{ added between all} \\ & \text{transfers of type } i \rightarrow k \\ 0, & \text{otherwise.} \end{cases}$$

$$v_{ik} = \begin{cases} 1, & \text{if there is no bypass from } i \rightarrow k \\ 0, & \text{otherwise.} \end{cases}$$

$$u_{ik} = \begin{cases} 1, & \text{if transfer } i \rightarrow k \text{ is used} \\ & \text{to support a bypass} \\ 0, & \text{otherwise.} \end{cases}$$

We define a number of variables in order to formulate the problem. The first variable x_{ik} denotes the transfers from FU_i to FU_k which are selected to remain in the architecture. Since local interconnect between a FU is not considered, we assign x_{ii} to be 0. For each transfer type $i \rightarrow k$, a bypass of any type except i or k is allowable, therefore we define variable b_{ijk} to represent if bypass operations of type j are added on transfers $i \rightarrow k$. In addition to these two variables, we must define two secondary variables v_{ik} and u_{ik} which are used to determine which transfers remain in the architecture. Variable v_{ik} denotes wether a bypass operation of any type is added to transfers $i \rightarrow k$. If a transfer $i \rightarrow k$ is used to support a bypass operation used on a transfer variable u_{ik} is set to one.

There are three main types of constraints for the addition of bypass operations. The first type of constraint specifies that for each transfer type, at least one bypass must be selected. It is allowable however, for the selected bypass to be the same as one of the transfer FU (i.e. $i = j$ or $j = k$ is allowed and signifies no transfer added).

$$\sum_j b_{ijk} = 1 \quad \text{for all } i \text{ and } k \tag{1}$$

$$\sum_i x_{im} \leq I_m \quad \text{for all } FU_m \tag{2}$$

For each FU, the number of transfers into FU_m which remain in the design must be less than the allowable number. Therefore the total number of transfers into FU_m must be smaller than the allowable number of inputs (Eq. 2).

The last type of constraint enforces that a transfer x_{ik} must remain if either a bypass was added to a transfer such that interconnect $i \rightarrow k$ is required to support it (i.e. bypass of type i is added on a transfer into FU_k) or no bypass was added on the original transfers from $i \rightarrow k$. We can specify this constraint mathematically as:

$$x_{ik} = v_{ik} \vee u_{ik} \text{ where} \tag{3}$$

$$v_{ik} = b_{iik} \vee b_{ikk} \tag{4}$$

$$u_{ik} = b_{ik1} \vee \ldots \vee b_{ikk} \vee b_{1ik} \vee \ldots \vee b_{iik} \text{ for all } i \neq k \tag{5}$$

We split this expression into multiple constraints. The first set of constraints determines v_{ik} which denotes if a bypass of any type other than $i \rightarrow k$ was added between transfer $i \rightarrow k$. Since constraint (Eq. 1) specifies that one bypass must be added, the constraint determines if a pseudo-bypass was added (i.e. a bypass which is the same FU as either i or k). If one of these pseudo-bypass operations was added, then a "true" bypass was not added. Therefore, the transfer from $i \rightarrow k$ must remain. Variable v_{ik} will be set to 1 if no bypass of any type was added to transfer $i \rightarrow k$ and set to 0 if a bypass was added.

In order to implement a logical "or" operation, three constraints must be added as shown by (6).

$$v_{ik} \leq b_{iik} + b_{ikk}, v_{ik} - b_{iik} \geq 0, v_{ik} - b_{ikk} \geq 0 \tag{6}$$

The second set of constraints (Eq. 7) determines the value of u_{ik} which signifies if any bypass operation was added which is dependent on an interconnect between $i \rightarrow j$ or $j \rightarrow k$. Each logical "or" operation can be specified using the same formulation as shown above (Eq. 6).

$$u_{ik} = b_{ik1} \vee \ldots \vee b_{ikk} \vee b_{1ik} \vee \ldots \vee b_{iik} \text{for all } i \neq k \tag{7}$$

Together v_{ik} and u_{ik} in Eq. 3 will determine if the transfer from $i \rightarrow k$ must remain. If both v_{ik} and u_{ik} are 0, then a bypass was added to all original transfers between $i \rightarrow k$ and the transfer is not needed by an added bypass operation. Therefore, the interconnect from $i \rightarrow k$ is no longer needed and can be eliminated (i.e. x_{ik} equals 0). In all other cases x_{ik} will equal 1, since no bypass was added to the original transfers and/or there was an addition of bypass operations which require transfers from $i \rightarrow k$.

$$x_{ik} \leq v_{ik} + u_{ik} \quad x_{ik} - v_{ik} \geq 0 \quad x_{ik} - u_{ik} \geq 0 \tag{8}$$

$$OF : MAX(\sum_{i,k} x_{ik} F_{ik}) \tag{9}$$

The goal is to optimize the interconnect structure with minimal penalty in terms of additional operations. Therefore, the objective is to minimize the total number of bypass operations added. Hence, the OF maximizes the number of transfers which remain after the addition of bypass operations.

A variation on the original formulation is to allow the addition of multiple bypass operations per transfer. This modification allows the elimination of transfers which were used to

Instance	Most Common Transfers
Square Root (SR)	add→pred_lt (798), add_u→add (768), add→ld_f2 (732)
Convo. Encoder (CE)	add_u→add (1500), ld_i→xor (1400), add→ld_i (1300)
g721 encode (g721E)	ld_i→add (22,373), add→ld_uc2 (14914) add→pred_lt (14,914)
g721 decode (g721D)	mov→ld_i (189,758), mov→st_i (103,504) lsl→or (17,250)
g721	ld_i→add (81,583), add→ld_uc(51,759), mov→pred_ne (49,416)
Comm. Module (CM)	ld_i→add (83,183), add→ld_uc (52,759), mov→pred_ne (49,416)
Media Processor (MP)	ld_i→add (83,186), add→ld_uc (52,759), mov→pred_ne (49,416)

Table 1. Most common transfer types.

support other bypass operations. In order to model this case, we allow $m - 2$ bypass variables per transfer. Constraints (6,7, and 8) are replicated to account for each new pair for which a bypass can be introduced in between and modified to include all possible scenarios for which a bypass operation will remain included.

In addition, the dual problem can be formulated. The goal is to minimize the final number of incoming interconnect for the program P using a specified number of bypass operations. In order to formulate this dual problem in ILP form, constraint (2) is modified to specify that the sum of the bypass operations introduced must be less than the specified maximum value. Also, the OF is restated to minimize the total number of interconnect (the sum of the x_{ik}'s).

5. EXPERIMENTAL RESULTS

In this section we evaluate our bypassing approachusing a set of real life designs given in Table 1. SQRT uses Newton Raphson method to compute the square root of a number. SQRT is a standard benchmark included in Trimaran. The G721 is an implementation of the CCITT G.721 voice from the Mediabench set of benchmarks. Both encoding and decoding cases were considered. The Convolutional Encoder has rate 1/2 with 2 symbols per bit. The last three are programmable chips made of four benchmarks: g721 combines the decoder and encoder, CM adds to g721 the Convolutional decoder to form a communication module, and multimedia processor (MP) combines all four designs.

Table 1 shows the three most used transfer types for the benchmarks. Table 2 shows the results for bypassing. The second column indicates the initial number of interconnects. The third column of Table 2 shows the number of interconnect (IC) after minimization and the fourth column shows the percentage of IC reduction. Finally, the percentage reduction in runtime is indicated. The number of transfers was reduced on average by 66.1% by introducing bypassing. Note that in the case of bypassing the runtime of the benchmarks increased at average by 1.15%.

Inst.	Orig # IC	At Most 3 In. IC	% IC Reduc.	% RT Overhead
SR	56	17	69	0.6
CE	42	10	76	1.3
g721E	72	25	65	3.1
g721D	72	21	70	2.1
g721	72	26	63	1.9
CM	72	28	61	1.2
MP	72	29	59	0.9

Table 2. Experimental Results: Bypassing.

6. CONCLUSION

We have developed an ILP-based approach for effective use of bypassing during architectural space exploration in such a way that the properties of final interconnect networks are optimized. The effectiveness of the approaches and algorithms is demonstrated using the Trimaran-based platform [1]. We were able to reduce the number of interconnect by a factor of two to three times with only nominal throughput reduction with bypassing.

7. REFERENCES

[1] Trimaran, "http://www.trimaran.org/.," .

[2] P. Baran, "On distributed communication networks," *IEEE Transactions on Communications*, vol. 12, no. 1, pp. 1–9, 1964.

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

[4] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Transactions on CAD*, vol. 14, no. 1, pp. 12–31, 1995.

[5] D. Lobo and B. Pangrle, "Redundant operator creation: A scheduling optimization technique," in *IEEE/ACM Design Automation Conf.*, 1991, p. 775778.

[6] K. Banerjee et al., "3-D ICs: A novel chip design for improving deep submicron interconnect performance and systems-on-chip integration," in *Proceedings of the IEEE, Special Issue on Interconnects*, 2001, vol. 89, pp. 602–633.

[7] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Performance analysis and optimization of latency insensitive systems," in *ACM/IEEE Design Automation Conf.*, 2000, pp. 361–367.

[8] W. J. Dally and S. Lacy, "VLSI architecture: Past, present, and future," in *Adv. Research in VLSI Conference*, 1999.

[9] J. Cong et al., "Microarchitecture evaluation with physical planning," in *ACM/IEEE Design Automation Conf.*, 2003, pp. 32–35.

[10] K. Sankaralingam, V.A. Singh, S.W. Keckler, and D. Burger, "Routed inter-alu networks for ilp scalability and performance," in *Computer Design*, 2003.

[11] M.D. Taylor, W. Lee, S.P. Amarasinghe, and A. Agarwal, "Scalar operand networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2.