Design Exploration for High-Performance Pipelines †

Smita Bakshi and Daniel D. Gajski
Department of Information and Computer Science
University of California, Irvine, CA, 92717-3425

Abstract

Exploration plays an important role in the design of high-performance pipelines. We propose an exploration strategy for varying three design parameters by using a performance-constrained component selection and pipelining algorithm on different "architectures". The architecture is specified manually by using a mix of behavioral and structural constructs, while the component selection and pipelining is performed automatically using our algorithms. Results on two industrial-strength DSP systems, indicate the effectiveness of our strategy in exploring a large design space within a matter of seconds.

1 Introduction

In exploring the design space of high-performance pipelines, three design features play a significant role: *architecture, pipelining,* and *component selection.* A large number of design alternatives can be first evaluated by varying the component selection and the number of pipe stages of a given architecture. The exploration can be further increased by repeating the component selection and pipelining for a variety of different architectures.

The architecture of a design refers to the type and number of its components and their interconnectivity, where the number of components in a design gives an indication of its "parallelism". A "parallel" architecture is one that exploits the inherent parallelism in a specification by computing several operations at the same time. While parallelism improves design performance, it also results in relatively expensive designs.

The parallelism of an architecture is illustrated with the help of a 4th-order (P=4) FIR filter shown in Figure 1. Consider the two designs in Figure 1(b). Design 1 has a higher degree of parallelism than Design 2, since it can compute four multiplications in parallel, while Design 2 can perform only one multiplication at a time.

The second design feature, pipelining, is another means of increasing design performance for a relatively small overhead in terms of pipelining register costs. This feature is all the more significant for DSP computations since they are regular and repetitive in nature, and yield well to pipelining techniques. Pipelining is illustrated in Figure 1(c), where Designs 3 and 4 have been obtained by pipelining Design 1 into a different number of stages.

The third design feature, component selection, adds yet another level of exploration. It involves selecting components from a realistic library with more than one implementation per operator, such that slow components are used on non-critical paths, while the faster and more expensive components are used only when necessary, on critical paths.

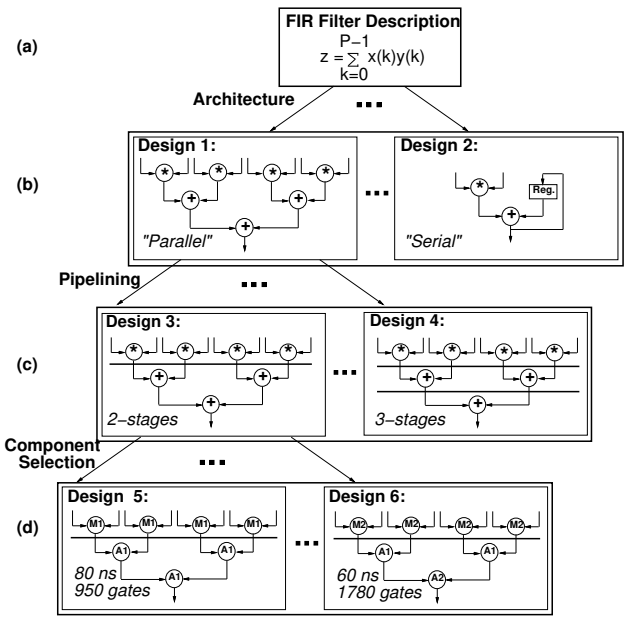

Figure 1: Design exploration for a 4th-order FIR filter.

Component selection is illustrated for the FIR filter in Figure 1(d) where Designs 5 and 6 have been obtained from Design 3 by using a different selection of adders and multipliers from a given library. Assume that the library consists of 2 multipliers, $M1$ and $M2$, (area of 200 and 400 gates and delay of 80 and 60 ns, respectively) and 2 adders, $A1$ and $A2$, (area of 50 and 80 gates and delay of 40 and 20 ns, respectively). Using this library the cheapest design for a pipe stage (PS) delay constraint of 80 ns is obtained with the component selection shown in Design 5, and for a PS delay constraint of 60 ns by the component selection

†This work was supported by the Semiconductor Research Corporation (grant #93-DJ-146).

in Design 6. Note that a design may contain different implementations of the same operator. As an example, Design 6 has two instances of $A1$ and one of $A2$, instead of three instances of $A2$, which would have resulted in a more costly design.

We have just illustrated that a large number of design alternatives can be evaluated by varying the architecture, pipelining and component selection of a given design. In this paper, we present a strategy for exploring these parameters by manually specifying an architecture and using an algorithm for component selection and pipelining.

We would like to point out that by combining pipelining and component selection, we differentiate ourselves from related research which has, (a) performed pipelining without any component selection [1], [2], [3], [4], [5], or (b) combined component selection with non-pipelined scheduling [6], [7], [8]. Furthermore, since we allow the user to specify different architectures, we can explore a larger design space than that obtained by only using one of the above algorithms.

The paper is organized as follows. We discuss the exploration strategy in Section 2 and the architecture specification in Section 3. We define and give algorithms for the component selection and pipelining problem in Section 4. Next, we present results of applying our design strategy and algorithms to explore the design space of two large DSP systems, obtained from industry.

2 Design Exploration Strategy

The FIR filter example illustrated that varying all the three design parameters leads to a very large search space. However, not all combinations of architecture, pipelining and component selection are "desirable", and instead of exhaustively searching the design space, only those designs that fit some criteria should be explored.

We now outline a design exploration procedure which generates only those designs that satisfy a given *throughput* (or sample rate) constraint on the system. We have selected the throughput as the selection criteria since the throughput of high-performance pipelines is usually decided before the exploration, by factors external to the system, and hence it is unalterable. This is unlike other parameters, such as latency and cost, which the designer may wish to minimize but not constrain.

The input consists of a specification containing architectural information, a component library, and a throughput constraint (Figure 2). The architecture specified may be the most serial or parallel one to start with. It is first pipelined into one stage (PS=1) and components are selected such that the throughput constraint is satisfied and the cost of the pipeline is minimized. An additional pipe stage is then introduced and components are re-selected for the new pipeline. The tasks of pipelining and component selection are repeated till the number of pipe stages can no longer be increased. At this point, the architecture is changed by increasing or decreasing its parallelism.

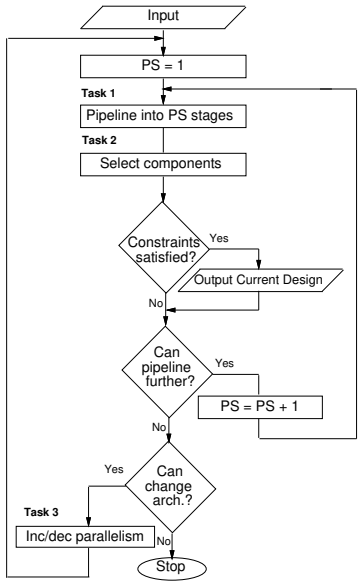

Figure 2: The exploration strategy.

Pipelining and component selection is then repeated for the new architecture. Thus, the inner loop of the search strategy changes the pipelining and the component selection, while the outer loop changes the architecture of the design. In this paper, we automate Tasks 1 and 2 by using an algorithm for performance-constrained pipelining and component selection, while we perform Task 3 manually by allowing the designer to specify the architecture in the input specification.

3 Specifying Architectures

Designers of DSP systems typically know the basic design topology or architecture they wish to use; however, they require assistance in time-consuming tasks such as pipelining and component selection. A pure structural description is inappropriate since it would require them to know the complete structure of the design. On the other hand, a pure behavioral description is also inappropriate since it would prevent them from specifying the design topology they have in mind. Thus, we propose an input format that allows a mixture of both structure and behavior in the specification. Figure 3 illustrates this feature by giving descriptions of three different 4th-order FIR filter architectures.

Designs 1 and 2 differ in their summation topology - one uses an adder "tree" and the other an adder "chain". This difference is brought out in the description by using appropriate assignment statements. Design 3 differs from Designs 1 and 2 in the number of multiply and add operators it contains, and hence in the number of iterations (or passes through the design) required to compute one output. In the description, this "behavior" (that is, the iterations) is specified by enclosing the design "structure" within a $for-loop$ statement. A designer can thus specify

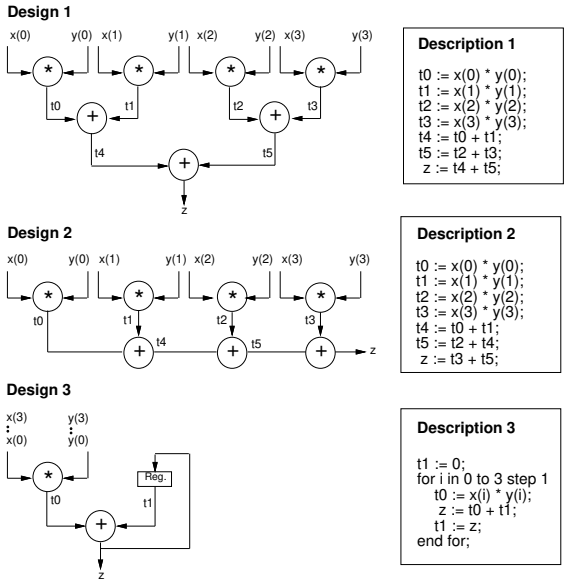

Figure 3: Input specification of three 4th-order FIR filter designs.

an architecture by mixing behavioral and structural constructs in this manner.

As stated previously, after a designer specifies an architecture manually, we utilize an algorithm for performing pipelining and component selection. In the next section, we define the problem and give a brief overview of the algorithm for component selection and pipelining. For a complete description of the algorithm please refer to [9].

4 Component Selection for Pipelines

Given a data flow graph $\mathcal{DFG}(V, E)$, where V represents a set of vertices, and $E \subseteq V \times V$ a set of directed edges, a component library $\mathcal{CL}$ consisting of a set of three tuples $\langle ComponentType,\ Area,\ Delay \rangle$, and a constraint on the *Pipe Stage (PS) delay* and *Latency*, the algorithm finds an assignment of components to vertices, and a partition of $\lfloor Latency/PS\ delay \rfloor$ stages, such that the delay of each pipe stage is less than or equal to the *PS delay* and the total area of the $\mathcal{DFG}$ is minimized. *PS delay* (or the inverse of throughput) is the delay between the arrival of two consecutive input samples, and *Latency* is the total execution time of the pipeline ($n\times$ *PS delay*, for an n-stage pipeline).

The algorithm (Figure 4) starts by mapping each vertex of the $\mathcal{DFG}$ to the fastest available component. It then slows down vertices by progressively mapping them to slower components. At each slow down the $\mathcal{DFG}$ is pipelined and if constraints are violated, the slow down is not accepted. This process is repeated until no vertex can be slowed down without a violation of constraints.

The key to the algorithm lies in judiciously selecting vertices to be slowed down in each iteration, since slowing down one vertex may prevent slowing down others due to

1. Map vertices to fastest components, pipeline $\mathcal{DFG}$, and evaluate performance.
2. **If** (*fastest design does not satisfy constraints*)
3. exit the program.
4. **Else**
5. **Loop**
6. Select the "best" vertex to slow down.
7. Pipeline the $\mathcal{DFG}$, and evaluate performance.
8. **If** (*performance constraints met*),
9. accept this slow down, **else**, reject it.
10. **Until** (*no vertex can be slowed down without violating constraints*).
11. **End if**

Figure 4: Overview of the component selection and pipelining algorithm.

graph dependencies. With every vertex, we thus associate a value, called the vertex *weight*, which is a measure of its "desirability" or priority in the slowing-down process. In each iteration of the algorithm, vertex weights are evaluated and the vertex with the highest weight is selected to be slowed down (step 6 in Figure 4). The vertex weight is defined as ADG/CF where, (1) the area delay gain (ADG) is the decrease in area per unit increase in delay when a component is replaced by a slower one, and (2) the commonality factor (CF) is a measure of the number of I/O paths containing that vertex.

Step 7 of the algorithm requires the $\mathcal{DFG}$ to be partitioned into a minimal number of stages of delay *PS delay*. This is done by traversing the graph in two directions, downward (from the input to the output nodes), and upward (from output to input nodes). During a traversal the delay from the boundary of the last pipe stage is accumulated and a new boundary is set when the performance constraint can no longer be satisfied. The traversal is repeated for both directions, and the pipeline with the fewer number of "cuts" is selected. A "cut" refers to the intersection of an edge of the $\mathcal{DFG}$ with the pipe stage partition, and it corresponds to a pipeline register. Hence, the fewer the number of cuts, the fewer the pipeline registers.

5 Experimental Results

We now apply the general exploration strategy and the component selection and pipelining algorithm, on two fairly large DSP systems, a 2-Dimensional 8×8 Inverse Discrete Cosine Transform (IDCT) [10], and a 4-element, 4-beam Beamformer [11]. For both examples, we wrote three descriptions representing different architectures. Each of the descriptions was then pipelined into a different number of stages and components selected from the library, given in Table 1 [12], such that throughput constraints were satisfied and the cost was minimized. This was done using the algorithm presented in Section 4. Whereas, experimental results in [9] demonstrate the quality of results produced by the algorithm, the results in this paper serve to demonstrate the application of the algorithm on large DSP systems.

TABLE 1
MODIFIED DTAS COMPONENT LIBRARY

Component Type	Component Name	Delay. (ns)	Cost (eqv. ND2 gates)
*	Mpy1	57.97	2368
*	Mpy2	44.21	2400
*	Mpy3	36.21	2600
*	Mpy4	32.98	2710
*	Mpy5	28.57	2978
*	Mpy6	25.00	3500
*	Mpy7	22.50	4000
*	Mpy8	20.50	4500
+/-	Add1/Sub1	25.80	62
+/-	Add2/Sub2	20.00	125
+/-	Add3/Sub3	13.50	187
+/-	Add4/Sub4	10.00	250
+/-	Add5/Sub5	5.50	375

The results of the exploration have been presented as a trade off between throughput and area for (1) different architectures, with a fixed latency, and for (2) a fixed architecture, with varying latency. Thus, for instance if a designer has a "hard" or fixed constraint on the throughput of the system that he must satisfy, he can get a good idea of the architecture he should be considering, by looking at the first graph. After narrowing down to one, or possibly two architectures, he can then study the effect of latency, and pick a design point that best optimizes his cost, which could be a function of area or latency or both.

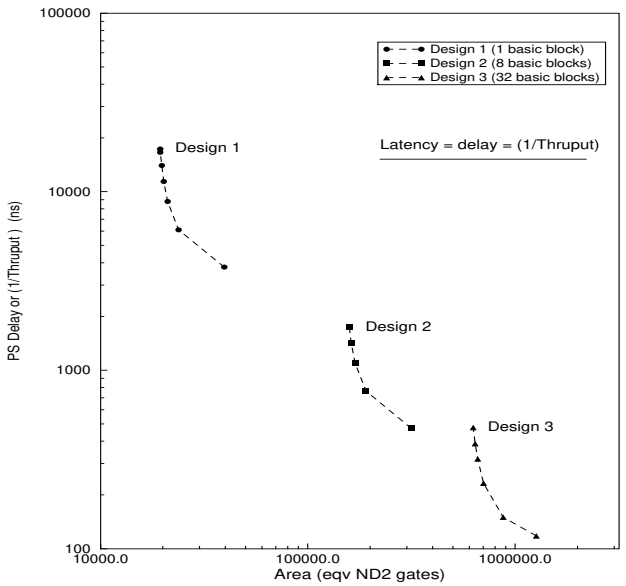

Figure 5: PS Delay vs. Area of 3 different 8×8 IDCT architectures.

In all graphs, the y-axis represents PS delay, or effectively, the inverse of throughput, while the x-axis represents area in ND2 (2-input NAND gates from LSI Logic library) gates.

IDCT

The 2-D IDCT, used to reconstruct compressed images, is represented by the following equation:

$$bd(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} BD(u,v)F(u,x)F(v,y)$$

N, the order of the IDCT, is 8 in our examples, implying that approximately 64 additions and 64×2 multiplications need to be computed for every output $bd(x,y)$, where $x, y \in 0 \ldots 7$. The algorithm we chose for evaluating the IDCT essentially consists of 3 $N \times N$ matrix multiplications. We considered 3 architectures, that compute 8×8 matrix multiplications with different "extents of parallelism". Design 1 consists of 1 basic block (BB), Design 2 of 8 BBs, and Design 3 of 32 BBs, where a BB is a block used to evaluate one term of the product matrix. Note that Design 3, consisting of about 500 nodes (15 nodes per BB), is the largest design we have considered, and our algorithm for pipelining and component selection took less than a second for this example.

Figure 5 depicts all three topologies (for a fixed latency) on the same graph (log-log scale) †, whereas Figure 6 shows the effect of varying the latency of Design 1.

As can be seen, a large design space has been explored by varying the architecture and component selection alone, where the delay of the designs ranges from about 17,000 ns to less than 100 ns (or approximately 60 KHz. to 10 MHz.), while the cost varies from about 20,000 to 1,000,000 gates (though designs are not evenly spread in this range).

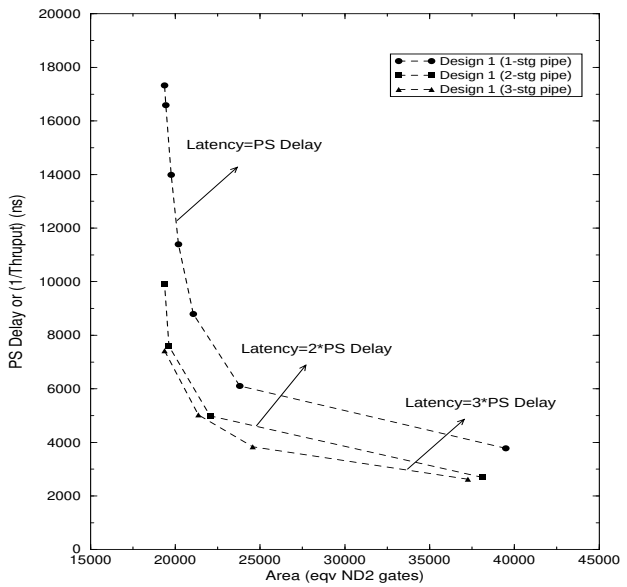

Figure 6: PS Delay vs. Area of Design 1 for 8×8 IDCT.

We would like to point out that the substantial exploration, from 100 ns to 17,000 ns, would not have been possible without the capability of using multiple implementations of operators in the design. Had our library contained just one implementation per operator, Figure 5 would have consisted of just 3 points, one for each architecture.

†Note that the design points are joined by a curve for purposes of graph readability; this does not imply a continuous design space.

Beamformer

The beamforming problem is formally described by the following equations:

$$y_e^b(i) = \sum_{k=0}^{P-1} D_e(i-k)c_e^b(k),\ \forall b \in 1 \ldots M, \forall e \in 1 \ldots N$$

$$R^b(i) = \sum_{e=1}^{N} y_e^b(i)\omega_e^b,\ \forall b \in 1 \ldots N$$

The first equation represents a FIR filter operation, while the second equation represents a phase rotation (PR), that is the multiplication of each of the FIR filter outputs with a constant (ω_e^b), and then the summation of these products. In our analysis, we have considered a 4-element (N=4), 4-beam Beamformer (M=4) with an 8th-order (P=8) FIR filter.

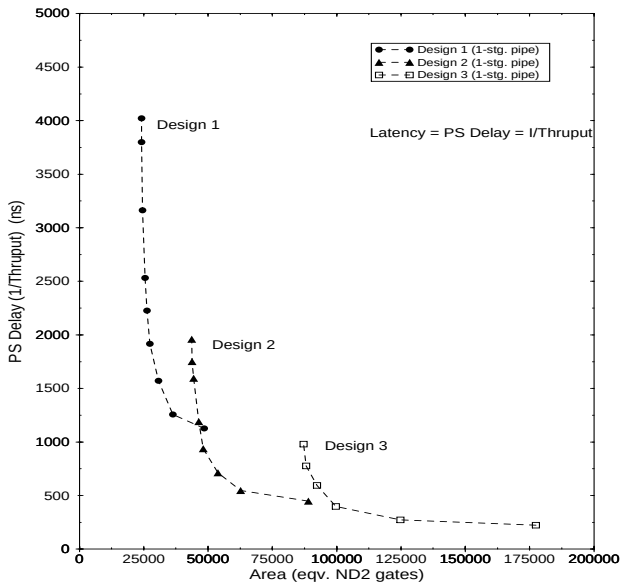

Figure 7: PS Delay vs. Area of 3 different architectures for a 4-element, 4-beam Beamformer system.

Once again we considered 3 different architectures that differed in the number of FIR filters, the number of PR blocks and the number of adders in the subsequent summation operation. Design 1 consisted of just one FIR filter and PR block (all FIR filter blocks were implemented with 8 multipliers and 7 adders), Design 2 consisted of 2 FIR and PR blocks and Design 3 consisted of 4 such blocks. Each of these designs was also pipelined into 2 and 4 stages.

The results of varying the design topology for a fixed latency are given in Figure 7. Once again, a large design space ranging from a throughput of 4000 ns to 100 ns, and a cost of 25,000 to 175,000 gates has been explored. This would not have been possible without the ability to vary all three parameters - architecture, component selection and pipelining.

6 Summary and Conclusion

To summarize, we have presented an exploration strategy for high-performance pipelines that varies three important design parameters: *architecture*, *pipelining*, and *component selection*. This is achieved by manually writing different descriptions, and then using our algorithms for pipelining and selecting components. We demonstrated the effectiveness of our exploration strategy by applying it on two industrial-strength DSP systems, the Beamformer and 2-D IDCT. For both the examples, we obtained a large spread of designs, ranging from a throughput of 100 ns to 4000 ns for the Beamformer and from 100 ns to 17,000 ns for the IDCT, within a matter of seconds.

The component selection presented in this paper has been limited to functional units alone; we are currently extending our algorithm and design strategy to include the selection of an "optimal" set of memory modules for a pipeline.

References

[1] N. Park and A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Transactions on Computer Aided Design*, vol. 7, pp. 356–370, Mar. 1988.

[2] K. S. Hwang, A. E. Casavant, C.-T. Chang, and M. A. d'Abreu, "Scheduling and hardware sharing in pipelined data paths," in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 24–27, 1989.

[3] C.-T. Hwang, Y.-C. Hsu, and Y.-L. Lin, "PLS: A scheduler for pipeline synthesis," *IEEE Transactions on Computer Aided Design*, vol. 12, pp. 1279–1286, Sept. 1993.

[4] R. Jain, A. Parker, and N. Park, "MOSP: Module selection for pipelined designs with multi-cycle operations," in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 212–215, 1990.

[5] R. Jain, A. Parker, and N. Park, "Predicting area-time tradeoffs for pipelined design," in *Proceedings of the 24th Design Automation Conference*, pp. 35–41, 1987.

[6] M. Balakrishnan and P. Marwedel, "Integrated scheduling and binding: a synthesis approach for design space exploration," in *Proceedings of the 26th Design Automation Conference*, pp. 68–74, 1989.

[7] L. Ramachandran and D. D. Gajski, "An algorithm for component selection in performance optimized scheduling," in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 92–95, 1991.

[8] A. H. Timmer, M. J. M. Heijligers, L. Stok, and J. A. G.Jess, "Module selection and scheduling using unrestricted libraries," in *Proceedings of the European Design Automation Conference*, pp. 547–551, 1993.

[9] S. Bakshi and D. D. Gajski, "A component selection algorithm for high-performance pipelines," in *Proceedings of EURO-DAC*, 1994.

[10] P. M. Embree and B. Kimble, *C Language Algorithms for Digital Signal Processing*. Englewood Cliffs, New Jersey 07632: Prentice Hall, Inc, 1991.

[11] S. Bakshi and D. D. Gajski, "Design space exploration for the beamformer system," Tech. Rep. 93-34, Dept. of Information and Computer Science, University of California, Irvine, 1993.

[12] J. Kipps, *An Approach to Component Generation and Technology Adaptation*. PhD thesis, University of California, Irvine, 1991.