

Controller and Datapath Trade-offs in Hierarchical RT-Level Synthesis *

D.Sreenivasa Rao[†] and F. J. Kurdahi[‡]

[†]EDA Labs., IBM Corp., Poughkeepsie, NY, 12601.

[‡]ECE Department, University of California, Irvine, CA 92717.

Abstract

We intend to study the impact of control logic on the RT-level design space of a class of digital systems. Such an enhancement of the design space is more accurate than several previously reported approaches since control logic has a significant impact on the total cost and performance of the circuit. We present a datapath synthesis framework that is hierarchical in nature; and thus allows the control logic overheads to be factored in hierarchically as well. We introduce hierarchy into the system dynamically by identifying regularity; as such, the proposed method is specific to the signal/image processing domain of application. The impact of control logic is studied with respect to two well known models of FSM's used in hierarchical systems – the localized model (where each hierarchical sub-unit has its own controller) and the centralized model (where the FSM's are all centrally located). We demonstrate how regularity facilitates such a study through the use of realistic area-delay estimators that lead us to better understanding of the RT design space.

1 Introduction and Motivation

An important aspect of high-level synthesis is *design space exploration*, or the study of various possible designs from a cost-performance trade-off perspective. At the high level of abstraction at which such a study has to be made, it is difficult to incorporate realistic design data into the cost and performance figures so that an informed choice can be made. Ideally, the design space should be based on accurate physical design details that include the datapath, the controller and the relevant interconnection circuitry. As can be easily seen, this data cannot be obtained in practice until after the synthesis process is complete; and the best we can do is to approximate the design space as much as possible to the realistic curve. The intent of this paper is to study one aspect of this approximation—inclusion of control logic cost and performance considerations into the design space—by considering a partitioning approach to the problem in a way that facilitates efficient hierarchical synthesis. The aspect of datapath synthesis in such a hierarchical framework was studied and presented in an earlier publication [1]; this paper examines the issue of hierarchical control synthesis, and thereby the tradeoffs involved in the design of the complete system.

Several high-level synthesis techniques base their decisions regarding the quality of a synthesis subtask (like say a schedule or allocation) by evaluating an area-delay function, the basis for which is usually a *module library* consisting of basic components like adders, multipliers, registers, multiplexers, etc. Some recent research considers the *wiring* overhead, which significantly impacts the cost and performance of the final design. However, there are not many synthesis systems that account for the *control area* overhead, although it eventually forms a significant part of the final system cost. As with wiring effects, the reason synthesis systems don't handle the control overheads is due to the difficulty of incorporating such lower level design details into the synthesis process. This points to the need for a synthesis framework that can obtain better approximations of the design space, by facilitating inclusion of control and wiring areas to a certain extent. In the current paper, we present an approach that is based on the use of proven and reliable RT-level area-delay estimators to assist in high-level decision making.

The general problem of high-level synthesis has been investigated quite extensively in the past. Researchers have also looked at *system partitioning* as a means of combating the complexity of the synthesis process. One of the first such systems is reported in BUD [2], which uses *functional proximity* as a metric for partitioning. The system allocates a generic set of functional units for each cluster; however, it does not explore the possibility of sharing the clusters themselves. The inherent advantages of an *application-specific* synthesis technique have led several researchers to investigate synthesis of specific classes of circuits, like microprocessors, DSP systems, etc. DSP system synthesis has been extensively researched (see for example [3]).

The synthesis of control logic for a given datapath is usually carried out after the RT structure of the datapath is available; this enables derivation of the required control signals. Decomposition approaches to simplify the synthesis of FSM controllers have been studied in the past [4]. In [5], different controller design styles are studied (like PLA's, microprogramming, random logic, etc.); controller implementation based on *multiple PLA's* (derived from local properties of functional units) is found to lead to minimal increase in system area compared to other design styles. In the current work, we explore the impact of different controller models on the design space of the given behavior. By this approach, a part of the controller effects are factored into the design space definition, so as to give a more accurate picture of the trade-

*This work was supported by NSF Grant # MIP-8909677 and UC-MICRO Grant # 91-080

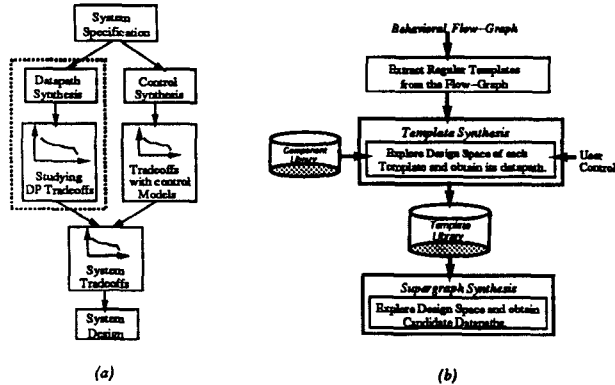


Figure 1: (a) Studying Control and Datapath Tradeoffs; (b) Overview of the Proposed Synthesis System.

offs. In the following sections, we briefly touch upon some aspects of datapath synthesis that directly influence the controller design and provide distributed FSM models to study the tradeoffs involved.

2 Overview of the Proposed Approach

We present in this section an approach to high-level synthesis of regularly specified systems (like signal and image processing circuits) that tries to factor in routing and control logic areas in order to better approximate the area-time tradeoff space. The primary premise of our approach is that a significant subset of signal and image processing circuits are characterized by an inherent *regularity* of their descriptions. By regularity, we mean that the behavioral data-flow graph of the system can be abstracted as a collection of families of *templates*, each of which represents a part of the behavior. It has been demonstrated earlier how this concept of regularity *extraction* can be used to perform tasks like system partitioning and module generation [6]. Once we are equipped with a regularity extractor, architectural synthesis can be posed as a *hierarchical* problem and all the synthesis sub-tasks (like design space exploration) can be performed hierarchically.

An overview of the work presented in this paper, in the context of high-level system synthesis, is given in Fig. 1(a). Hierarchical synthesis of the datapath and the tradeoffs involved (enclosed within the dashed box) have been explored previously [1]; this paper discusses the inclusion of control logic effects in a hierarchical fashion, studying the tradeoffs presented and utilizing this understanding to better explore the system design space. By performing the datapath synthesis hierarchically (Fig. 1(b)), we demonstrate how the control logic overhead can be accounted for in a limited way. This, of course, means that the controller implementation has to be hierarchical as well; we present an empirical study of this concept based on different models of controllers that are included in the synthesis of the “supergraph”.

The current work makes several important contributions to the field of synthesis. From a very global perspective, this is one of the few synthesis techniques that is *truly hierarchical*, in the sense that every synthesis subtask (like design space exploration, control logic synthesis, module selection) is performed hierarchically. In addition, it has the following primary features that are not present in other synthesis

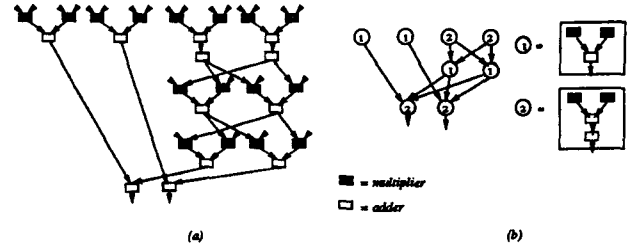


Figure 2: (a) AR Filter Flowgraph; (b) Supergraph Along With the Automatically Extracted Templates.

approaches:

- Our approach concentrates on a novel way of *modeling* the problem domain. This modeling, or abstraction, is based on the concept of *regularity*, and introduces an additional level of hierarchy into the design flow. This hierarchy extensively simplifies several design tasks. Thus our approach complements existing synthesis systems that concentrate on developing novel and complicated algorithms.
- Hierarchy also facilitates the incorporation of distributed FSM models into the design space curve; such an efficient way of accounting for control logic overhead is an important step in bringing the synthesis process closer to real world issues. As far as the authors’ knowledge goes, this issue has not been addressed previously at this level of abstraction.
- Through the introduction of hierarchy, the synthesis approach presented here provides an efficient way of accounting for physical design effects using reliable area-delay estimators. Such estimators are constructive in nature and are hence sensitive to the design size; hierarchical application of these estimators serves to reduce the overall run-time.

3 Hierarchical Synthesis Based on Regularity Extraction

The core of the proposed synthesis approach is the regularity extractor, that works on a directed graph representation of any system and identifies a set of subgraphs that, when suitably replicated, form the complete system. Thus, our objective is to identify a small set of subcircuits (*templates*), and their *instances* in the given system graph. Our proposed method for regularity extraction is a series of heuristics that first identify the candidate templates, and based on certain optimization criteria, choose a subset of these templates that have matched with the system subgraphs. The detailed description of the heuristics used for this purpose has been omitted for reasons of brevity, and can be found in [6] and [1]. We briefly present the results of regularity extraction on a well-known DSP filter example, the auto regressive (AR) filter. The initial dataflow graph and the supergraph (along with the supernodes) are shown in Figs. 2(a) and 2(b).

Once the *distinct* template types in the circuit are identified, we pose the traditional synthesis problem *hierarchically*. This concept has been employed (in a different sense) previously, for example, in [7]. Our scheme differs from the previous works in that we propose a hierarchical design space

exploration on a *small number* of subgraphs of the system, viz., the distinct templates identified above. By exploring the design space of each template type, we obtain several possible implementations for each; and can use this information to examine the scheduling alternatives for the complete graph. Assuming a library of basic modules (like adders, multipliers, etc.), we first construct the area-time characteristic for each distinct template; details are given in [1]. By retaining the final solutions obtained in a branch-and-bound search, we have a series of solutions which form the A-T characteristic of the template under question.

For a better estimate of the A-T curve of each template, it is essential that we derive an RT-level structure of each control-step assignment, so that the library based on these template implementations includes routing area and delay estimates (making it more realistic when used for synthesis of supergraph). Since the templates are typically small in size (3-5 behavioral nodes), it is easy to obtain all the possible RT datapaths. A simple greedy heuristic is employed for this purpose; details can be found in [6]. The completed datapaths can now be input to design estimators like LAST/TELE [8] so as to get an estimate of their area and delay. The total delay of each datapath, and the estimated cycle time for its operation are also obtained from the estimators.

With this information on hand, we can now move on to synthesizing the supergraph. This process is similar to the one described earlier for the templates, except that the library used now is the one derived by synthesizing the various possible datapaths for the templates. A final area-time tradeoff curve for the AR filter, obtained by synthesizing some candidate datapaths for the supergraph, is shown in Fig. 3. For comparison, a similar curve obtained on the flat flow graph (ie., without going through the regularity extraction and hence hierarchical synthesis) is also included. It can be easily seen that within experimental error, the hierarchical approach doesn't result in a significant area or performance penalty; in fact, the high performance end of the design space is better explored in the hierarchical case, demonstrating that redundant use of hardware (caused by templates) is beneficial for high-performance systems.

Having examined the issue of hierarchical datapath synthesis, we move on to the control logic implementation. In doing so, we seek to factor in this important overhead into the template library, so that the supergraph synthesis may benefit from a better approximation for its module selection.

4 Implementing Control Logic Models

4.1 Motivation

In a hierarchical implementation framework discussed in previous sections, a central control logic that supervises the whole system can be replaced by a distributed model that is simpler to synthesize. An important motivation for such an exercise is that it allows control effects to be considered in the synthesis of supergraph, just as the routing effects have been factored in into each of the templates synthesized above. In other words, the module selection and the synthesis of the supergraph is based on "super functional units" that have routing and control logic overheads incorporated into them. Evidently, this makes the design space of the system a better approximation than when synthesis is carried out on a flat data-flow repre-

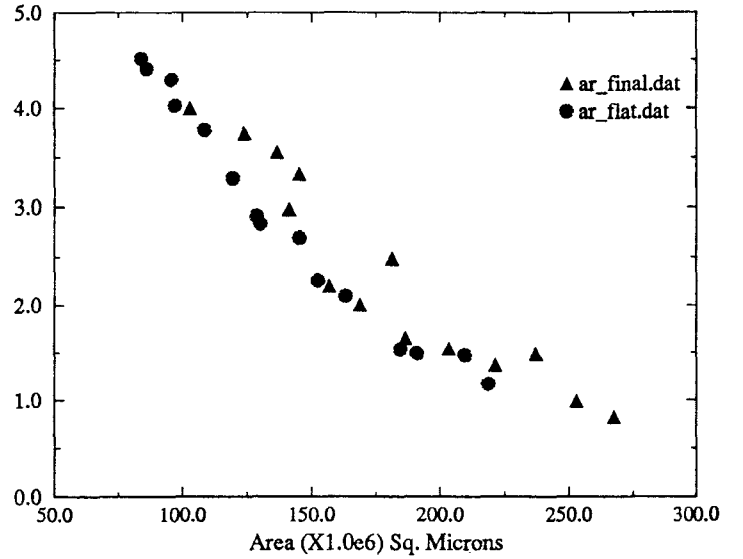


Figure 3: Final A-T curve for the AR filter, comparing flat and hierarchical cases. The Y-axis is Delay, in microsec.

sensation. This section is devoted to hierarchical controller design, by examining the various possible approaches to distributed FSM modelling.

A good formalism for *distributed control synthesis* was provided in [9]: each functional block is made up of *computation units* and *interconnection units*. The synthesis of asynchronous interface units is the main thrust of their approach. However, this system works off a structural representation of the datapath. The fact that FSM's that are smaller in size can be much easily synthesized has been exploited in [4] to derive a decomposition method that works on smaller equivalent FSM's of a larger problem.

In the hierarchical framework being discussed here, FSM's can also be modelled hierarchically, with smaller FSM's for the templates being used to form the FSM of the supergraphs. Each distinct datapath that is synthesized will have a separate FSM corresponding to it. Let us assume that the flat data-flow graph has N nodes, in which regularity extraction has identified t templates, each of average size k nodes. If we assume that only two instances of the template datapaths are used in the final implementation, then we see that only two different state machines M_1 and M_2 need to be designed. An example state machine is shown in Fig. 4(a), which has at most k states corresponding to the k nodes in the templates (assuming worst case and no conditional branches). This machine could be a simple Moore machine (in which output is a function of the state at time t ; the influence of the input on the output is only through the state). The operation of the Moore machine can be initiated by a *start* signal and whose exit would be indicated by an *end* signal. In the figure, 'start' and 'end' states are not shown for convenience. Upon receiving some specified input, state S_3 makes a transition to state S_{end} , generating an output signal 'end'. Now, the t nodes of the supergraph could give rise to t "superstates" (Fig. 4(b)). Each superstate is made up of either machine M_1 or M_2 . The supergraph FSM is fairly simple, since it has to only keep track of the *start* and *end* signals of each of the smaller machines. Once again, the final transition from SS_3 occurs

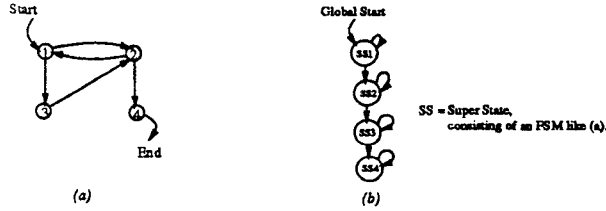


Figure 4: Concept of Hierarchical FSM's: (a) A Template FSM; (b) A Simple 'Super' State Machine built from Template FSM's.

after receiving the 'end' signal of SS2, and generates a 'global end'. Since the complexity of FSM synthesis is sensitive to the size of the specification, it is evident that the two-step procedure based on extracting regularity is much simpler compared to the one-step procedure on flat flow graphs.

4.2 Implementing FSM Models

The traditional way of implementing a control logic involves deriving all the control signals necessary for proper operation of the RT datapath and synthesizing the suitable logic by performing well known procedures like state encoding, state minimization, logic optimization, etc. In the hierarchical framework proposed here, each of the datapaths derived for the templates are considerably smaller, thus the FSM synthesis problem becomes simplified. However, it is necessary to precisely define the implementation models of such distributed FSM's. The most common model is the *centralized controller* (Fig. 5(a)), in which the FSM corresponding to the complete datapath is derived and is located centrally with control signals suitably feeding the datapath.

Two separate models of distributed FSM's used in this research are shown in Figs. 5(b) and 5(c). In the former case, the control logic, derived separately for each FSM, is physically located on the template. Thus, the routing needed for control signals of the template datapath is localized to the template itself. However, since there is no sharing of hardware modules across templates, this scheme could often need more functional units than the necessary minimum. M_1^a and M_1^b represent FSM's derived for similar templates, but they have to be located separately nevertheless, thus foregoing any possible sharing. A slightly different model is shown in Fig. 5(c), where the FSM's derived for each template are all placed at a central location. This scheme facilitates sharing of hardware constituting the FSM's [4]. However, the amount of routing needed in this case to properly distribute the control signals to the datapath could be significantly more than in the previous model. FSM's shown with similar shading in this figure can be potentially shared, thus reducing the hardware requirement. Note that in both cases, a simple global controller for the supergraph has to be derived and implemented. We will refer to the three models shown in Fig. 5 as *flat and physically centralized (FPC)*, *logically localized and physically distributed (LLPD)* and *logically localized and physically centralized (LLPC)* models respectively. In Section 6 we present an empirical evaluation of these models based on experimental study.

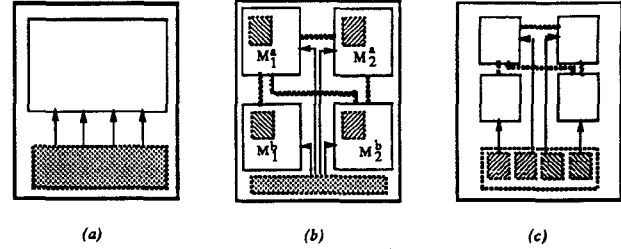


Figure 5: Three Models of Control Logic: (a) Flat datapath with Control; (b) Logically Localized; and (c) Logically Localized and Physically Centralized. In (c), FSM's With Similar Shading are Shared.

5 Architectural Synthesis of the System Graph

The objective of the system graph scheduling and allocation procedure is to allocate the operator nodes to "super" control steps and choose a suitable set of modules from the library to implement them. This has to be based on certain optimization criteria, and we define the following for our purposes:

1. minimize the maximum *finishing time*, ie., minimize $\max(T_x)$, where x is an exit point in the flow-graph; and
2. minimize the maximum *dispersion* in the completion times of the slowest and fastest modules in any control step, ie., minimize $\max(T_a^j - T_b^j)$, $\forall j$, where j is any control step.

It should be mentioned here that we haven't included cost (measured typically in terms of area) in the objectives above because the methodology emphasizes reliance on physical design details (through estimators), and there is no easy way of incorporating layout effects like wiring in a mathematical form. Thus, this formulation is intended to obtain a *set* of candidate solutions that can be analyzed for their area and delay. Several sophisticated techniques can be proposed for solving the above min-max problem. We use a straightforward greedy heuristic [1] as a first cut approach, which we found to be effective in generating good solutions. For register allocation and minimization, we use REAL [10], an algorithm based on the well known left-edge heuristic, while a variant of the algorithm reported in [11] is used to improve interconnect sharing, where an attempt is made to share patterns rather than individual operators. The FSM for the supergraph is synthesized using well-known standard procedures. An estimate of the area and delay of the resulting RT netlist is made using LAST/TELE. This procedure repeated for all the schedules obtained above gives an estimated A-T curve of the system being synthesized.

6 Experimental Results and Discussion

We have implemented the heuristics described above in C on a Sun/Sparc workstation, and have used a few typical signal processing systems (like digital filters) to test the proposed synthesis framework. The synthesis procedures were based on a basic module library that consisted of multiple instances of adders, multipliers, etc. These figures were obtained from

actual layouts of the respective functional units using 4 micron CMOS technology. We employed a bit-slice layout style for the datapath components, except for the multiplier which used a macro-style layout. The control synthesis was carried out by encoding each controller as a set of boolean equations and then optimizing the logic using MIS-II package available in the Octtools suite.

The two distributed models of control logic described in Section 4 were used to study the relative impact of these controllers. After synthesizing the individual datapaths corresponding to the templates, the boolean equations for each of the controllers were derived. For the LLPD model (Fig. 5(b)), the templates, enhanced with the controllers, are input to LAST/TELE to estimate their area and delay. Before estimating the area and delay of the completed designs, we synthesize the control logic for the supergraph as well. The final result is a collection of designs, with control logic and routing overheads partly accounted for in the synthesis decision-making process. For the LLPC model (Fig. 5(c)), a similar procedure was adopted to give a set of designs that could be compared. This experiment was done for two well known HLS examples, the AR filter and the elliptical filter. In Fig. 6(a), the curves for the AR filter that compare the area and performance in the three cases depicted in Fig. 5 are shown; Fig. 6(b) shows similar comparison for the elliptic filter example. It is clear from these curves, that within bounds of experimental error, the two distributed control models have approximately the same penalty in area and performance compared to the controller model of the flat datapath. However, for an extremely regular structure like the AR filter, the model shown in Fig. 5(b) (the LLPD model) seems to have a slight advantage in most cases over the other model in Fig. 5(c).

We further analyzed the curves of Fig. 6 to understand the impact of routing on the total area. Figs. 7(a) and (b) show the datapath and control areas for the AR filter and the elliptic filter without the routing overhead. Once again, the flat datapath and the two distributed controller models are compared. The results show convincingly that any advantage one model would have over the other arises essentially from the area consumed by routing. In other words, the control logic area itself, as evident from Fig. 7, adds approximately similar overhead in all the three models considered; however, the difference in control areas is more evident in the case of AR filter, which is inherently more regular than the elliptic filter. It is important to note that the hierarchical (datapath as well as control) models enable us to explore the high-performance design space, whereas the flat synthesis procedure handles the low-performance design space better. This can be explained by the increased redundancy in the datapath functional units (in the hierarchical case), the localizing of routing overhead in both the distributed models, etc., which contribute to lesser signal delays.

7 Conclusions

The proposed synthesis method exploits some inherent properties of the behavioral flow graphs of DSP systems to perform the synthesis tasks hierarchically. This hierarchy causes only a portion of the actual design space to be explored; however, it also allows us to incorporate physical design and controller effects into synthesis by reducing the problem size. We propose to continue this research by obtaining better formalisms

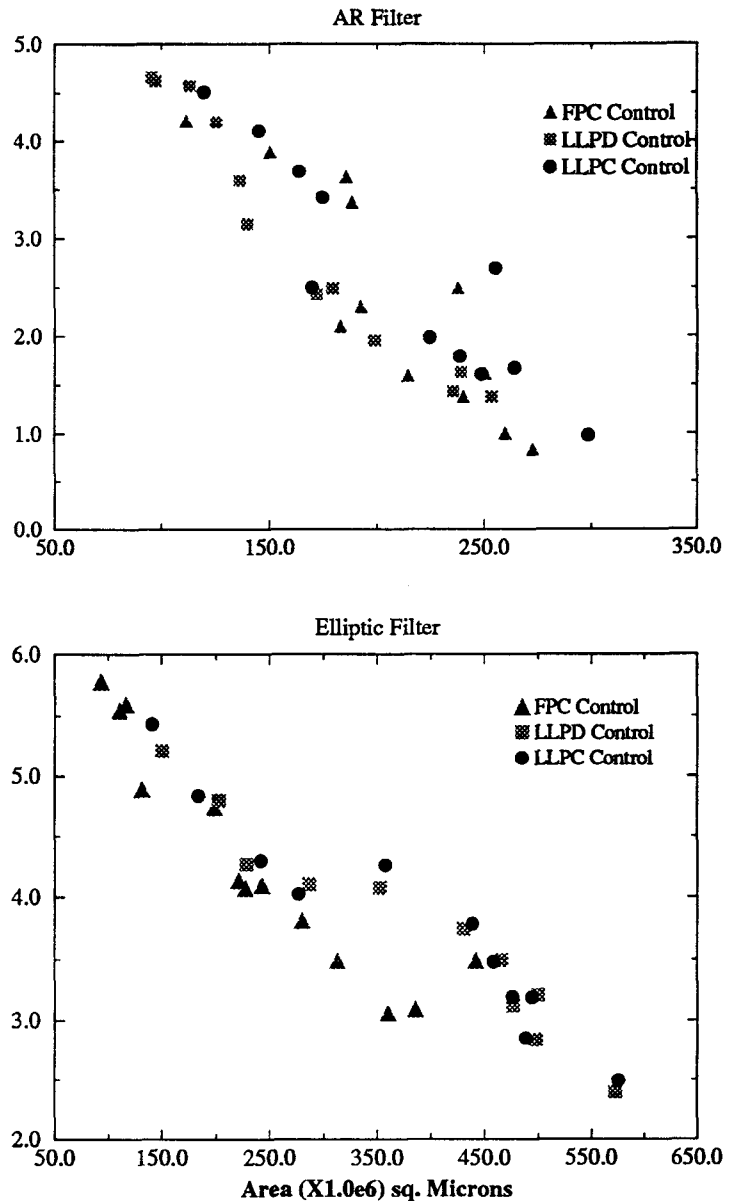


Figure 6: Comparison of Datapath and Control Areas for AR filter and Elliptic Filter, Using the Control Models Depicted in Fig. 5. The Graphs Include the Datapath Areas (Generated by Flat or Hierarchical Means) along with Respective Control and Routing Overheads. The Y-axis is Delay, in microsec.

for the hierarchical control synthesis problem; and by developing better and faster algorithms to enable handling of bigger problems.

References

- [1] D. Sreenivasa Rao and F. J. Kurdahi. Hierarchical Design Space Exploration for a Class of Digital Systems. *IEEE Trans. on VLSI Systems*, Septemeber 1993.
- [2] M. McFarland. Using Bottom-up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *Proc. 23rd DAC*, pages 474–480, June 1986.
- [3] H. DeMan, F. Cathoor, G. Goosens, J. Vanhoof, J. van Meerbergen, J. Huisken. Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms. *Proceedings of IEEE*, (2):319–335, February 1990.
- [4] S. Devadas and A. R. Newton. Decomposition and Factorization of Sequential FSM's. In *Proc. ICCAD-88*, pages 236–240, 1988.
- [5] M. Obrebska. Algorithm Transformations for Improving Control Part Implementations. In *Proc. ICCD-83*, pages 307–310, 1983.
- [6] D. Sreenivasa Rao and F. J. Kurdahi. On Clustering for Maximal Regularity Extraction. *IEEE Trans. on CAD-ICAS*, June 1993.
- [7] S. Note, F. Cathoor, G. Goosens, and H. De Man. Combined Hardware Selection and Pipelining in High-Performance Datapaths. *IEEE Trans. on CAD-ICAS*, 11(4):413–423, 1992.
- [8] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, A. Wu, V. Chaiyakul. Accurate Layout Area and Delay Modeling for System-Level Design. In *to appear in Proc. ICCAD-92*, 1992.
- [9] D. Messerschmidt Meng, T. and R. Brodersen. Automatic Synthesis of Asynchronous Circuits from High-Level Specifications. *IEEE Trans. CAD-ICAS*, 8(11):1185–1205, November 1989.
- [10] F. J. Kurdahi and A. C. Parker. REAL: A Program for REGISTER ALlocation. In *Proc. 24th DAC*, pages 210–215, June 1987.
- [11] N. Park and F. J. Kurdahi. Module Assignment and Interconnect sharing in register-transfer synthesis of pipelined data paths. In *Proc. ICCAD-89*, November 1989.

