figure a design flow. One way to achieve this is by using high-level performance estimates to determine appropriate design flows.

A starting point in this direction is the "information-based" design approach proposed by Bentz *et al*. [Bentz95]. Currently, this approach attempts to provide the user with estimates. The next step is to develop techniques to allow the system to use these estimates for configuring design flows.

structures within a primarily dataflow model). The other is the *heterogeneous* approach that seeks to systematically combine disjoint semantics (for example, a combination of dataflow and FSM semantics [Chang95], where each component of the system is specified in a model best suited for it). 2. Once specified, how can these systems be partitioned into hardware and software? 3. What is the best approach for simulating such systems? Chang *et al*. discuss some of the issues related to cosimulation in [Chang95].

6.2.2 System-level Low Power Synthesis

Our partitioning techniques have focussed on two metrics: area and time. With the growing importance of low power systems, it is important to introduce a third dimension of power consumption in the system-level design process. Low power design techniques at the circuit and architecture level are gaining some maturity [Chandrakasan95]. Can we extend them to the system level? This would include developing techniques for mapping nodes into different hardware/software implementations with the additional goal of minimizing the power consumption. This would also involve developing new techniques for analyzing power consumption from high-level system specifications. Some research is underway in the area of high-level power estimation [Mehra94] [Tiwari94]. The related hardware/ software interface also contributes to the power and needs to be carefully designed. Some insight into the issues involved in designing power-sensible CAD tools is given by Singh *et al*. [Singh95].

6.2.3 Smart Frameworks

The design space exploration process can be further systematized to develop "smart" frameworks. For instance, given a user-specified performance constraint, the design methodology management system could automatically con-

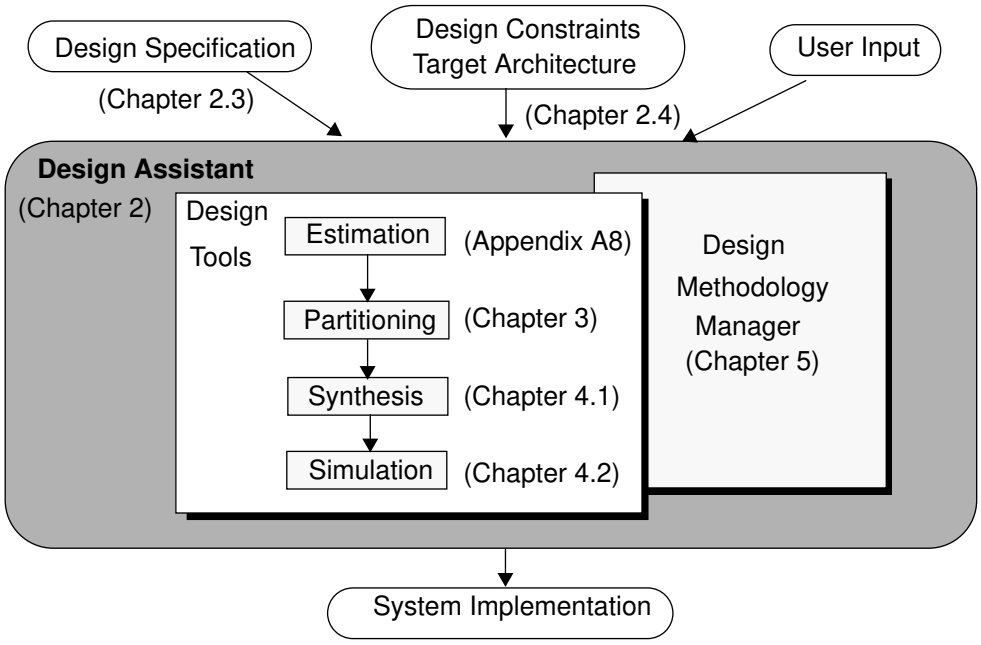

Figure 6.1. The Design Assistant

The Design Assistant operates in the DMM framework.

6.2 Future directions

6.2.1 Synthesis of Mixed Control-Dataflow Systems

The partitioning and synthesis techniques developed in this thesis apply to a class of applications that can be described using the synchronous dataflow model of computation. The SDF model is limiting in describing the dynamic behavior and control structures that are required to model typical embedded systems. The next obvious step is to extend our design techniques to a larger domain of applications. A good starting point is mixed control and dataflow systems.

Several questions need to be answered. 1. How should these systems be specified? Two opposite philosophies are emerging. One is a *unified* approach that seeks a consistent semantics for specification of the complete system (for example, boolean dataflow semantics [Buck94b], which allows specification of some control

Our approach to cosynthesis is to decompose the partitioned DAG into hardware, software, and interface graphs, where each node in the hardware and software graphs is a technology-dependent representation of the original node. Synthesis tools are used to generate the final implementation from these graphs. The emphasis in our work is on generating synthesizeable representations for the hardware and software components. We use pre-existing synthesis tools (such as Hyper and Ptolemy) to generate the final implementation.

Hardware-software *cosimulation* is the process of simulating the hardware and software components of a mixed hardware-software system within a *unified* environment. This includes simulation of the hardware modules, the processor, and the software that the processor executes.

We identify some of the key requirements of a cosimulation framework and show, with the help of an example, how the Ptolemy environment is effectively used for cosimulation.

6.1.3 Design Methodology Management

We present an infrastructure that supports efficient management of the design process, with emphasis on design flow management. This infrastructure contains powerful constructs for flow definition, dependency analysis, and automated flow execution. The infrastructure is developed as the DMM domain within Ptolemy. A tool called *DesignMaker* is responsible for automated flow management.

Design Assistant

The Design Assistant is a framework for unified system-level design. Figure 6.1 illustrates the Design Assistant. It contains specific tools for partitioning, synthesis, and simulation, which are configured by the user to create a design flow.

compute an appropriate implementation bin for the tagged node. It uses a lookahead measure, called the *bin fraction*, which estimates, for each bin of the node, the fraction of unmapped nodes that need to move to their fastest implementations so that timing constraints are met. The bin fraction is used to compute a *bin sensitivity* measure that correlates the implementation bin with the overall hardware area reduction. The bin selection procedure selects the bin with maximum bin sensitivity. The computation of bin sensitivity is simplified by assuming that the remaining free nodes are either in their slowest or fastest implementations. The tagged node becomes a fixed node once its implementation bin is determined. GCLP is then applied over the remaining free nodes and the sequence is repeated until all nodes in the graph become fixed.

The MIBS algorithm is reasonably efficient and has a complexity of $O(|N|^3 + B \cdot |N|^2)$, where $|N|$ is the number of nodes, and B is the number of implementation options per node, per mapping. In the examples tested, the MIBS solution is reasonably close to the optimal solution. We also illustrate that implementation bins can be used effectively to *reduce the overall area* over solutions generated using binary partitioning.

6.1.2 Cosynthesis and Cosimulation

Cosynthesis is the problem of synthesizing the hardware, software, and interface components of the system, starting with a partitioned DAG. Our synthesis techniques target an architecture consisting of a single programmable processor and multiple hardware modules. The architecture is assumed to be non-pipelined and nodes mapped to hardware and software communicate using a memory-mapped, self-timed, blocking mechanism. An architecture for efficiently managing hardware-software communication is described.

in the case of a node that consumes disproportionate amounts of resources in hardware and software mappings. Such a node is classified as an extremity and is quantified by an *extremity measure.* The extremity measure is used to bias the mapping selected by *GC* alone. The motivation in identifying extremities is to avoid infeasible or poor mappings that may arise if these preferences are not taken into account.

3. We also identify several intrinsic properties (such as bit manipulation operations and memory intensive operations) that make a node suited to either hardware or software. The effect of these properties is quantified by a *repeller measure*. The repeller measure is used in a novel way to effect *on-line* swaps between nodes to reduce the hardware area. This avoids post mapping swaps.

The GCLP algorithm is computationally efficient ($O|N|^2$). For the examples tested, the solution from GCLP is reasonably close to the optimal solution. We verify with examples that quantifying the local phase of nodes (i.e., extremities and repellers) helps improve the quality of the solution.

The formulation and solution of the **extended partitioning problem** is an original contribution of this thesis. We are not aware of any other work that attempts to do this at the system level. We present the MIBS heuristic to solve the extended partitioning problem.

The philosophy of the MIBS algorithm is to extend the GCLP heuristic for extended partitioning without the associated complexity buildup. The strategy is to classify nodes in the graph as *free*, *tagged*, and *fixed*. Initially all nodes in the graph are free — their mappings and implementation bins are unknown. GCLP is applied over the set of free nodes. A tagged node is then selected from this set; its mapping is assumed to be that determined by GCLP. A *bin selection procedure* is used to

design problem cannot be posed as a single well-defined optimization problem from the designer's perspective. Typically, the designer needs to explore the possible options, tools, and architectures, choosing either automated tools or manually selecting his/her choices. We believe that managing the design process plays an equally important role in system-level design, as do the tools used for different aspects of the design. To this end, we develop a framework that supports design methodology management. The key aspects of this infrastructure are summarized in Section 6.1.3.

6.1.1 Partitioning

Binary partitioning is the problem of determining, for each node in the application, a hardware or software mapping and schedule for execution. We define *extended partitioning* as the joint problem of mapping nodes in a precedence graph to hardware or software, scheduling, and selecting a particular implementation bin for each node. The end-objective in both cases is to minimize the total hardware area subject to throughput and resource constraints. Since both of these problems are NP-hard, we develop efficient heuristics.

We present the GCLP heuristic to solve the **binary partitioning problem**. This algorithm has several unique features:

1. Recognizing that binary partitioning is a constrained optimization problem, GCLP uses a *global time-criticality measure GC* to adaptively select a mapping objective at each step — if time is critical, it selects a mapping that minimizes the finish time of the node, otherwise it minimizes the resource consumption. The global time criticality measure attempts to overcome the inherent drawback of serial traversal.

2. In addition to global consideration, local preference is taken into account

software components. This allows the hardware and software designs to proceed in parallel, with feedback and interaction between the two as the design progresses. The codesign approach enables the exploration of a wide variety of implementation options. Thus, the system can be optimized in its entirety.

Four key problems are identified in the context of system-level codesign of embedded systems: partitioning, synthesis, simulation, and design methodology management. We attempt to provide solutions to all these problems. To enable meeting this ambitious objective, we have chosen to focus on a class of signal processing systems that is specified using synchronous dataflow semantics.

At the system-level, designs are typically represented as task graphs, where tasks have moderate to large granularity. Each task can be synthesized by a multiplicity of algorithms. Each algorithm selection, can, in turn, be synthesized using different transformations and resource-level options. The resulting implementations typically differ in area and execution time. The objective in system-level design is to select the "best" implementation for the system as a whole. The system-level designer is hence faced with the question of selecting the best design option for each task from a considerably large number of design options. We present a systematic approach to solving this problem by formulating the binary partitioning and extended partitioning problems. The key contribution is summarized in Section 6.1.1.

In the context of mixed hardware-software systems, system-level design offers even greater challenges. How do we take advantage of the inherently distinct properties of hardware and software and still tightly couple their design processes? We present approaches to cosynthesis and cosimulation that address this problem. The key ideas are summarized in Section 6.1.2.

The system-level design space is quite large. In general, the system-level

6

CONCLUSION

This thesis studies a systematic approach to the system-level design of embedded signal processing systems. The key ideas presented are summarized in Section 6.1. In Section 6.2, we conclude with a discussion of some of the future directions to this research.

6.1 Contribution

In this thesis we develop techniques for the design of embedded signal processing systems. The design of such systems is challenged by stringent cost, performance, and time-to-market constraints. Pure hardware or software implementations often cannot meet these constraints; besides, some parts of these applications are inherently better-suited to either hardware or software. Hence, these applications tend to have mixed hardware and software implementations. Due to their algorithmic complexity and also to avoid early commitment to a particular hardware or software implementation, these applications are best specified at a *task* level of granularity.

Our approach to designing such systems is to *codesign* the hardware and