Towards Optimal System-Level Design[1]

MANJOTE S. HAWORTH and WILLIAM P. BIRMINGHAM

Abstract

System-level design includes selecting optimal parts to realize a specified hardware structure. Multiple-function parts and constraints among functions make local-optimization techniques inadequate for generating globally optimal designs. Also, parts and part sets cannot be evaluated without considering the cost of implementing their support functions. Finally, for multi-attribute optimization, designers have difficulty assigning weights to the cost attributes. We present a synthesis tool that handles multiple-function parts, support functions, constraints, multi-attribute evaluation, and generates optimal designs.

1. Introduction

System-level design is the process of choosing parts to implement a specified hardware structure. In our case, this means selecting parts from a catalog. The parts can be processors, memory units, device controllers, ports, connectors, etc. Since highly sophisticated parts are available, complex systems can be designed. Example systems include computer systems, such as workstations and embedded controllers [1]. Figure 1.1 shows a sample input and output of a system-level design tool. The tool converts an abstract description of a computer board into a physical design.

It is desirable to generate designs that minimize cost. Generating these designs, however, is difficult for several reasons. First, local optimization techniques fail when there are multiple-function parts in the catalog [6]. If each function is mapped in a local perspective to the least expensive part, parts that perform only one function are chosen. When viewed globally, however, the lower per-function cost of multiple-function parts makes them more cost-effective if they have a high utilization. Similarly, constraints across subproblems make locally optimal choices globally suboptimal. Let functions X and Y, and the parts chosen for them be related through a constraint. In this case, choosing the least costly part for function X may make all low cost parts for function Y ineligible. Instead, choosing an expensive part for function X may allow a less costly part to be chosen for function Y, and therefore lead to an overall less costly design. In most design situations, parts require support functions. For example, an Intel 8086 microprocessor requires clock, reset, and ready-logic functions to support its operation; each of these functions may need its own support functions, and so on. A decision to choose an inexpensive part A over an expensive part B may not minimize cost, because A's support functions may make it more expensive than B. Further, since all functions needed in the design are not known before design begins, it is not possible to determine the utilization of a multiple-function part until the design is completed.

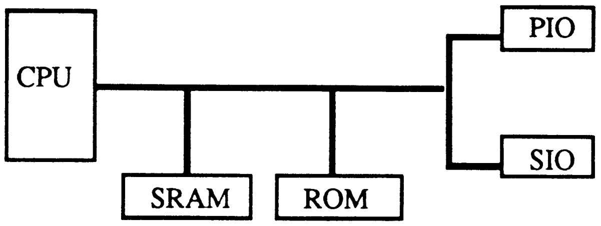
Thus, parts selected based on local or incomplete information do not always minimize the overall cost of a design. This phenomenon is called the *horizon effect*. Avoiding the horizon effect completely, however, entails searching a large design space, which is exponential in the number of parts and connections.

Typically, design tools make multi-attribute evaluation functions tractable by assigning weights to each of the attributes. Designers, however, have difficulty assigning meaningful weights [10].

We describe a synthesis tool that generates globally optimal designs. The tool uses a part-selection algorithm that handles multiple-function parts, support function requirements and dynamic constraints. It uses a multi-attribute, preference-based evaluation technique to evaluate designs.

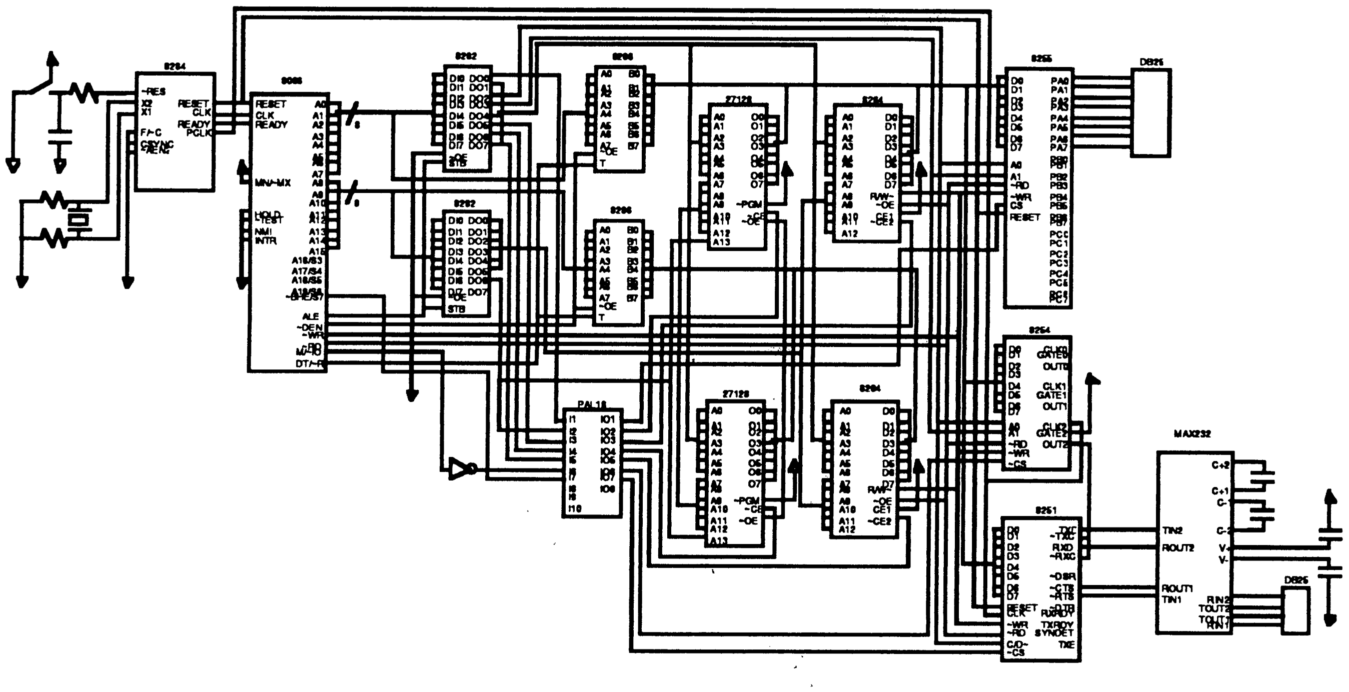
The paper is organized as follows. Section 2 defines the synthesis problem, and Section 3 presents our solution approach. Section 4 discusses related work, and Section 5 summarizes the paper.

[1]This work was funded by National Science Foundation (NSF) contract MIPS 9057981. The opinions expressed here are our own, and not necessarily those of NSF.



Desired functions: {CPU, SRAM, ROM, PIO, SIO}
Constraints: [Operating clock speed ≥ 10MHz], [Critical-path delay ≤ 100ns]
[Dollar cost ≤ \$60], [Area ≤ 10 sq.in.]
Part Library: Manufacturer's catalog

a) Input.



b) Output.

Figure 1.1 Input and output of a system-level synthesis tool.

2. Problem Definition

We define system-level synthesis as follows.

Given the following input:

- Set of desired functions, F. Each desired function $F_i \in F$ has:
    - a *function multiplicity*, $fm(F_i)$, which is the desired number of instances of that function.
    - *specifications*, which are properties of the function.
- Set of parts forming a catalog, P. Each part $P_i \in P$ has:
    - *part-function multiplicities*. A part-function multiplicity $pfm(P_iF_j)$ is the number of instances of function F_j implemented by a part P_i.
    - *characteristics*. Characteristics are properties of a part.
    - *support functions*. Support functions of a part are needed to support the part's operation. Each support function F_j has a *part-support-function-multiplicity* $psfm(P_iF_j)$ which is the number of instances of F_j required by P_i as support functions.
    - *cost-attribute-scores vector* $<c_{i1}, c_{i2}, ..., c_{in}>$ that represents costs along attributes a_1 through a_n.
- Selection constraints between function specifications and part characteristics.

- Cost constraints stating bounds on cost-attributes scores of a solution.
- A weighted utility function, U. The designer specifies the cost attributes in the function. He need not, however, specify the weights.

Find a set of designs in which each design:

- implements the desired number of instances of each function
- satisfies selection constraints
- satisfies cost constraints

The above definition lists only the conditions for a satisfying design. Our motive, however, is to generate optimal designs. A design is optimal if it is nondominated, and has better utility than other nondominated designs. A design D_i is *dominated* if there exists another design D_j such that for each cost attribute, the attribute score of D_i is worse than the attribute score of D_j, i.e., $c_{ik} > c_{jk}$ for all $k = 1, 2, ..., n$; otherwise, D_i is nondominated. The utility of a design is computed using a utility function. Thus, in addition to the conditions stated above, an optimal design PS_i also meets the following two conditions:

- PS_i is nondominated.
- PS_i has better utility than any other design.

3. Solution Approach

Our synthesis tool generates optimal designs in a two-step process. First it generates the set of nondominated designs that perform all functions and conform to cost and performance constraints. Second, it fully orders this set in accordance with the designer's preferences. The two main modules of the tool, corresponding to the two steps, are Ext_GOPS and ISMAUT. Ext_GOPS maps functions to nondominated sets of physical parts. It uses a constraint network to maintain consistency among function specifications and part characteristics. ISMAUT orders the solutions based on their cost vectors and the preference statements (i.e., not weights) input by the designer. The overall algorithm is listed in Figure 3.1.

```
SynthesisTool (F, P, cost_bounds, U)
    PS = Ext_GOPS(F, P, cost_bounds)
    OrderedPS = ISMAUT(PS, U)
    return (OrderedPS)
Ext_GOPS (F, P, cost_bounds)
    PS = {}
    subsume_flag = true
    partition_flag = true
    if ((¬NoSupportFunctions(P)) & (¬NoConstraintsAcrossFunctions(F,P)))
        subsume_flag = false
        partition_flag = false
    if (partition_flag)
        Set_of_partitions = Partitioning (P, F)
        ∀ partition_i ∈ Set_of_partitions
            partition_soln_i = OPT (partition_i.P, partition_i.F, cost_bounds, subsume_flag)
            Update cost_bounds
            PS = Combine (PS, partition_soln_i)
    else
        PS = OPT (P, F, cost_bounds, subsume_flag)
        Update cost_bounds
    if (PS = {})
        exit with failure
    if (subsume_flag)
        PS = RemoveDominatedSets(PS)
    PS = AddSupportParts(PS, P, cost_bounds)
    return (PS)
ISMAUT (PS, U)
    Prefs = QueryDesignerForPreferences(PS)
    W = DetermineRangesOfWeights(PS, U, Prefs)
    OrderedPS = OrderSolutions(PS, U, W)
    return (OrderedPS)
```

Figure 3.1. Synthesis Tool Algorithm.

3.1 Ext_GOPS

In this section, we explain the problem representation and algorithm used in Ext_GOPS to map functions to nondominated part sets.

3.1.1 Problem Representation

Information about functions and parts is organized as function and part models, respectively. Figure 3.2 shows examples of these models. The synthesis problem is formulated as a *part-function-attribute table*, as shown in Figure 3.3, where functions are listed in columns and parts in rows. A number *m* in parentheses next to a column heading F_k represents fm(F_k). A number *n* in row $P_{i,j}$ and column F_k represents pfm(P_iF_k). An *nS* in row $P_{i,j}$ and column F_k represents psfm(P_iF_k). A function is labeled *in* or *out* depending on whether it is needed in the design or not. A part is labeled *in* if it performs any *in* functions, otherwise, it is labeled *out*. In Figure 3.3, Table 1, the set of desired functions is {CPU, SIO, PIO}.

Function specifications and part characteristics are related through constraints. For example, function CPU's specification "ClockSpeed" and processor chip's characteristic "ProcChipClockSpeed" are related through constraint "ClockSpeed >= ProcChipClockSpeed". Only parts that satisfy constraints can be included in a design. Characteristics of selected parts are propagated through constraints to determine specifications of other functions. Thus, a part chosen to implement one function influences part selection for other functions through constraints.

We represent constraints in a network as shown in Figure 3.4. Elliptical nodes represent variables, and rectangular nodes represent constraints. A variable corresponds to a specification or characteristic. Specifications are assigned either by the designer, or by propagation through constraints involving other specifications and characteristics [8, 11]. Characteristics are assigned according to parts selected. In Figure 3.4, specification AddrAccessTime, Delay, NumWaitStates and characteristic ProcChipClockPeriod are related through the constraint "AddrAccessTime = (2 + NumWaitStates) * ProcChipClockPeriod - Delay". NumWaitStates is initialized to zero. Delay is computed using other characteristics (not shown). Characteristic ProcChipClockPeriod is assigned equal to the corresponding characteristic value of the selected processor. AddrAccessTime is assigned by plugging in the known values in the constraint equation.

Function Name	SIO
Function multiplicity, fm	2
Specifications	AddrAccessTime*

*The value of the specification is computed at runtime through constraints.

a) Function Model for SIO.

Part Name	8251A5
Implemented Functions, pfm	SIO (1)
Characteristics	SIOChipAccessTime = 100ns
Support Functions, psfm	Timer (1), Decoder (1), Port (1), MAX232 (1)
Cost vector	<$2.25, 1.0mW, 0.9sq.in.>

b) Part Model for 8251A5.

Figure 3.2 Example function and part models.

Constraints can be static or dynamic. Static constraints apply unconditionally. Dynamic constraints depend on the state of the design. A constraint's *inlist* specifies the conditions in which it is applicable, or *in* [4]. In Figure 3.3, the constraint with inlist "TRUE" is static, and the other constraint is dynamic, and is *in* only in designs that have processor 8086. The network propagates assigned variables through constraints to assign more variables, reports constraint violations, and *in*s and *out*s constraints according to their inlists.

		In			Out				Cost-vectors
		CPU	SIO	PIO	Timer	Decoder	Port	MAX232	<$, mW, sq.in.>
	8086	1							<10.00, 2.5, 1.4>
In	80186	1			1	8			<20.00, 3.0, 2.0>
	8251A5		1		1S	1S	1S	1S	<2.25, 1.0, 0.9>
	8255A5			1		1S	1S		<4.50, 1.0, 0.8>
	8254A5				1				<4.95, 1.0, 0.8>
Out	PAL18					1			<3.00, 0.1, 0.3>
	DB25						1		<0.65, 0.0, 1.1>
	RS232							1	<3.95, 1.0, 0.2>

a) Table 1.

Solutions to Table 1	Cost vectors	Support functions of solutions
{8086, 8251A5, 8255A5}	<$16.75, 4.5mW, 3.1sq.in.>	{Timer (1), Decoder (2), Port (2), MAX232 (1)}
{80186, 8251A5, 8255A5}	<$26.75, 5.0mW, 3.7sq.in.>	{Port (2), MAX232 (1)}*

*Support functions Timer and Decoder of 8251A5 and 8255A5 are implemented by 80186.

b) Solutions to Table 1.

		In				Out	Cost vectors
		Timer	Decode	Port	MAX232		<$, mW, sq.in.>
	8254A5	1					<4.95, 1.0, 0.8>
In	PAL18		1				<3.00, 0.1, 0.3>
	DB25			1			<0.65, 0.0, 1.1>
	RS232				1		<3.95, 1.0, 0.2>
Out							

c) Table 2

Solution to Table 2	Cost Vector	Support Functions
{8254A5, PAL18 (2), DB25 (2), RS232}	<$16.20, 2.2mW, 3.8sq.in.>	{}

d) Solution to Table 2.

		In		Out		Cost Vectors
		Port	MAX232	Timer	Decode	<$, mW, sq.in.>
In	DB25	1				<0.65, 0.0, 1.1>
	RS232		1			<3.95, 1.0, 0.2>
Out	8254A5			1		<4.95, 1.0, 0.8>
	PAL18				1	<3.00, 0.1, 0.3>

e) Table 3.

Solution to Table 3	Cost Vector	Support Functions
{DB25 (2), RS232}	<$5.25, 1.0mW, 2.4sq.in.>	{}

f) Solution to Table 3.

Complete solutions	Cost Vectors
{8086, 8251A5, 8255A5, 8254A5, PAL18 (2), DB25(2), RS232}	<$32.95, 6.7mW, 4.2sq.in.>
{808186, 8251A5, 8255A5, DB25(2), RS232}	<$32.00, 6.0mW, 6.1sq.in.>

g) Complete solutions.

Figure 3.3. Ext_GOPS refines an abstract design, {CPU, SIO, PIO}, to physical designs.

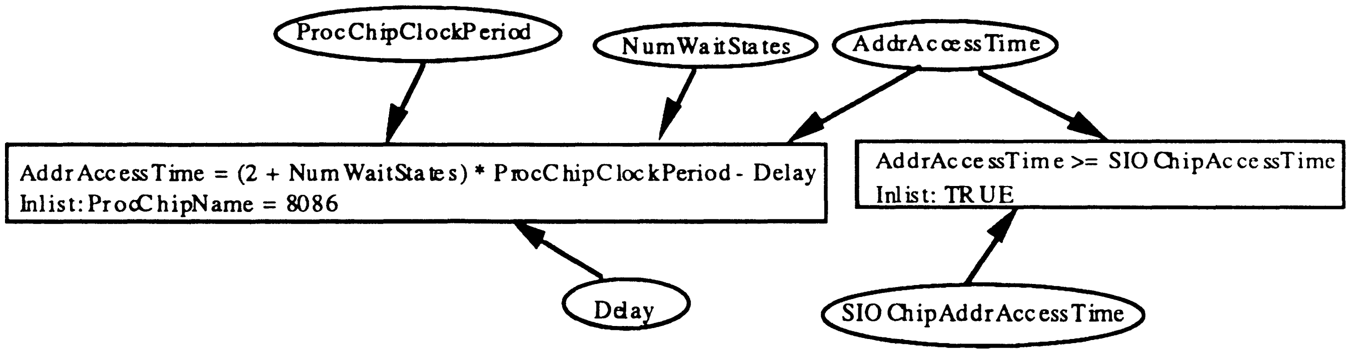

ProcChipClockPeriod
NumWaitStates
AddrAccessTime
AddrAccessTime = (2 + NumWaitStates) * ProcChipClockPeriod - Delay
Inlist: ProcChipName = 8086
AddrAccessTime >= SIOChipAccessTime
Inlist: TRUE
Delay
SIOChipAddrAccessTime

Figure 3.4: A constraint network.

3.1.2 Algorithm

Ext_GOPS uses several exact heuristics to speed-up the process of mapping functions to optimal solutions, and overcomes horizon effect due to multiple-function parts, constraints and support functions. Partitioning() divides a problem into disjunct subproblems that do not have common functions or parts. OPT() solves each subproblem, represented as a part-function table, by intelligently enumerating subsets of the set of *in* parts to cover all *in* functions [6].

Let the *in* part types be P_1 through P_n. First, OPT() enumerates all possible solutions that contain one or more instances of P_1, and zero or more instances of P_2 through P_n. Subsequently, it removes all instances of P_1 from the part-function table, and enumerates all solutions with one or more instances of P_2, and zero or more instances of P_3 through P_n. Next, all instances of P_2 are removed from the table, and enumeration continues as described. During enumeration, part sets are constructed incrementally, and a part is added to an expanding part set only if it covers a function that is not already covered by the part set.

Each time OPT() adds a part to a part set that is being expanded to a complete design, it also assigns variables in the constraint network using the new part's characteristics. The network checks that the new assignments meet all constraints, swaps *in* dynamic constraints associated with the new assignments, and propagates the assignments through constraints to determine specifications for other functions in the design. If variable assignments corresponding to a newly added part violate constraints, OPT() removes the new part from the set, and tries the next part in the library. For example, when OPT() adds processor 8086 to the set, variable ProcChipClockPeriod in the network shown in Figure 3.4 gets assigned. The constraint with inlist "ProcChipName = 8086" is swapped *in*, and propagation of assigned variables through this constraint assigns I/O specification AddrAccessTime to 110ns. Further, when OPT() adds an SIO chip S1 with access time 200ns to the design, the variable SIOChipAccessTime gets assigned to 200. The network reports violation of constraint "AddrAccessTime <= SIOChipAccessTime" to OPT(). OPT() removes chip S1 from the design, and tries chip S2 with access time 100ns. This time there is no constraint violation, and OPT() continues to build the partial design to a complete design with chip S2 in it. Alternately, if the designer wishes to use chip S1 in the design, AddrAccessTime is fixed to 200ns, and the network computes NumWaitStates, the number of wait states needed in the design to use the slow chip.

OPT() reduces search by removing parts that cannot be contained in an optimal solution (subsumed parts), or must be contained in an optimal solution (essential parts). A part type P_i is *subsumed* by another part type P_j if P_i covers a subset of the function instances covered by P_j, and P_i is cost-inferior to P_j. *Essential* part types cover at least one function that is not covered by any other part type. Removal of essential part types reduces both F and P. Essential part types are removed from P, and later appended to all solutions of the reduced F. Removing essential part types can lead to more subsumed part types, and vice-versa. During enumeration, essential and subsumed-part processing is performed every time a part type P_i is removed from P.

Besides doing subsumed and essential-part processing, OPT() prunes bad paths in the search process using the following techniques:

- When a part set covering all functions is found, no supersets of that part set are considered.
- When a part set exceeds a cost bound, that set is discarded and no supersets of it are considered.
- A partial solution is extended further only if for at least one cost attribute, its score is better than the worst score over all solutions generated so far. This is multi-attribute branch and bound.

After solving a partition, cost bounds are reduced by the least attribute scores over all solutions of the partition. Solutions to partitions are combined to form complete solutions. Dominated solutions and solutions that exceed bounds are then removed.

For each part set in PS, AddSupportParts() finds the unimplemented support functions of parts in the set, and calls Ext_GOPS with this set of support functions and part set P as input. The results returned by Ext_GOPS are combined with the corresponding part set in PS.

Each pruning technique in Ext_GOPS has a scope within which it is an exact heuristic, and outside which the technique is an inexact heuristic and may lead to elimination of optimal solutions. The scope is defined in terms of properties of data sets, i.e., the input functions, parts and constraints. The relevant properties are existence of multiple-function parts in the library, presence of support functions, and presence of constraints that cause interaction between part selection for different functions. Heuristics that remove essential parts, and prevent enumeration of supersets of complete or infeasible designs are applicable to all data sets. Subsumption, partitioning, and branch-and-bound within a partition are heuristics that have limited scope as explained below. (The scope of branch-and-bound heuristic within a partition is the same as that of subsumption.)

Subsumption and partitioning must be limited to subproblems where parts do not require support functions, and there are no constraints relating part selection for different functions. Partitioning cannot be done when parts have support functions because support functions of a part in one partition may be covered by a part selected in a different partition, but such synergy is not exploited if partitions are solved independently. Likewise, parts cannot be subsumed by other parts because the costs associated with support functions of these parts are not yet known. Also, multiple-function parts may cover support functions that are out at start of design, but later become in. This makes it difficult to evaluate parts for subsumption. Partitions, even with disjoint parts and functions, are not independent when constraints cause interaction among subproblems. When part selection for one function influences that for another function, parts cannot be subsumed without risking optimality.

Predicate NoSupportFunctions(P) is true if none of the parts in P that cover desired functions require support functions. Predicate NoConstraintsAcrossFunctions(F,P) is true if there are no constraints between functions specifications and part characteristics that cause interaction between part selection for two different functions.

Figure 3.3 illustrates how Ext_GOPS maps the set of desired functions, F, into sets of physical parts. Functions CPU, SIO, and PIO are F, whereas functions Timer, Decoder, Port, and MAX232 are support functions of parts that can cover F. Parts are labeled *in* or *out* as appropriate. NoSupportFunctions({8086, 80186, 8251A5, 8255A5}) is false, and hence, subsumption and partitioning are not performed. Assuming none of the parts violates any constraints, OPT() refines the abstract design {CPU, SIO, PIO} to PS = {{8086, 8251A5, 8255A5}, {80186, 8251A5, 8255A5}}. AddSupportParts() is called for PS. It constructs a support-function set for each of the two solutions in PS. A new table is derived from Table 1 for each support-function set. Table 2 is formed by *in*ing functions Timer, Decoder, Port and MAX232 and parts that cover these functions. Table 3 is formed in a similar way. The solutions generated by Ext_GOPS for Tables 2 and 3 are also shown. These solutions are combined by Combine() with the respective solutions of Table 1 to give two complete solutions shown in the figure.

3.2 ISMAUT

ISMAUT is a preference-based multi-attribute evaluation tool. In our synthesis tool, ISMAUT is used to determine an ordering among the nondominated sets output by Ext_GOPS.

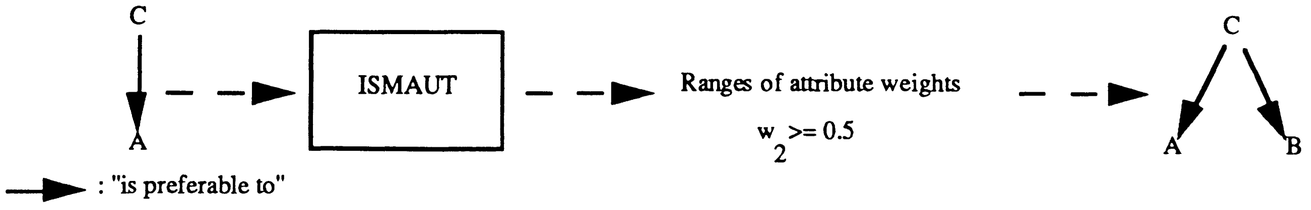
Each nondominated solution generated by Ext_GOPS is characterized by a cost vector $<c_{i1}, c_{i2}, \ldots, c_{in}>$ that represents the solution's costs along attributes a_1 through a_n. The cost vector $<c_{i1}, c_{i2}, \ldots, c_{in}>$ is transformed into utility vector $<v_{i1}, v_{i2}, \ldots, v_{in}>$ using normalizing function v_{ij} = (Highest c_j - c_{ij}) / (Highest c_j - Lowest c_j). Utility U_i of a design D_i is then computed as $v_{i1}w_{i1} + v_{i2}w_{i2} + \ldots + v_{in}w_{in}$.

The designer is shown the cost vectors of nondominated designs generated by Ext_GOPS, and asked to give as many pairwise preferences as possible among the designs and among attributes. For example, a designer may say "design D_i is preferable to design D_j, and attribute power dissipation is more important than dollar cost." From these statements, ISMAUT determines ranges of attribute weights in the utility function U. These attribute weights are used to find ordered pairs other than those already specified by the designer. A design D_i is better than design D_j if, for every possible vector of weights $<w_1, w_2, \ldots, w_n>$, for which each w_i is in its determined range and sum of all

w_i's is unity, utility of D_i is more than utility of D_j. A full ordering on alternative designs that conforms with the designer's preferences is computed. This aids the designer in selecting one design from a set of nondominated designs.

Design	Cost Vector $<c_1, c_2, c_3>$	Normalized Utility Vector $<v_1, v_2, v_3>$
A: {I8086, ...}	<\$12.00, 17.0mW, 4.0 sq.in.>	<1.0, 0.0, 1.0>
B: {I80186, ...}	<\$13.80, 14.0mW, 6.0 sq.in.>	<0.4, 0.3, 0.6>
C: {M68HC11,...}	<\$15.00, 7.0mW, 9.0sq.in.>	<0.0, 1.0, 0.0>

a)



b)

Figure 3.5. a) Cost attributes of three nondominated designs, b) use of ISMAUT to generate a preference digraph using input preferences.

Figure 3.5a) shows the cost and utility vectors of three nondominated designs, A, B, and C, output by Ext_GOPS. Suppose the designer states that he prefers design C to design A. Using this, ISMAUT infers that w_2 must be greater than 0.5. From this information, it is inferred that design C is better than design B. This gives the designer the conclusive result that C is the best of the three designs.

4. Discussion

With advances in integrated chip design, an increasing number of parts perform multiple functions. Existing synthesis tools, however, either use local-optimization techniques, which prefer single-function parts, or use inexact heuristics that prefer multiple-function parts. The example in Figure 4.1 illustrates how locally optimal part selection approach in a computer-configuration system leads to globally suboptimal solutions [1]. In a problem that requires two SIOs, the locally optimal solution has a greater number of chips, and is more costly than the globally optimal solution. One suggested heuristics is to order parts that implement a function by the number of additional functions they implement. This heuristic fails when a multiple-function part does not have high utilization, i.e., when many of the functions implemented by a multiple-function part are not needed in the design. MBESDSD selects parts based on a heuristic weighted cost function that uses per-function costs [13]. It is unclear how the cost of a multiple-function cost is distributed over the functions it covers. For example, distributing the cost of processor 80186 equally over its functions CPU, address decoder, interrupt logic, and timer underestimates the cost of implementing CPU and overestimates cost of other functions.

Given:

Parts	Functions	Attribute scores		Per-function attribute scores	
	SIO (2)	Dollars	Area (sq.in.)	Dollar	Area (sq.in.)
P_{I8251}	1	2	.75	2	.75
P_{I8274}	2	3	1.00	1.5	.50

Solutions:

Part-selection Strategy	Solution	Cost-attribute scores	
		Dollars	Area (sq.in.)
Locally optimal	$\{P_{I8251}, P_{I8251}\}$	4	1.50
Globally optimal	$\{P_{I8274}\}$	3	1.00

Figure 4.1 Locally optimal vs. globally optimal part selection.

System-level parts often have support functions. For example, in MICON the support function of an SIO chip is a Timer. In COSSACK, another computer-design system, the support functions of a printer are hardware interface, data cables, power cable, and driver software [9]. In MBESDSD, the support function for a 4-bit counter is a 4-line-to-16-

line Decode function. When evaluating competing parts, however, no existing system takes into account the cost of implementing the support functions of the parts.

In high-level synthesis, mapping of data-path modules into hardware cells is similar to the part-selection problem. High-level synthesis systems, however, do not have multiple-function parts in their libraries. DTAS maps generic RTL components into technology-specific RTL library cells [5]. Since the RTL cell library has only single-function parts, local-optimization is sufficient to generate globally optimal designs. Systems like MBESDSD that do have multiple-function parts use inexact heuristics as explained above.

Technology mapping is often treated as a DAG or tree-covering problem. Part-selection heuristics such as identification of essential parts, branch-and-bound and partitioning are also used for technology mapping. A DAG to be covered is partitioned into a forest of trees [2, 7]. This dynamic programming, however, leads to locally optimal but globally suboptimal solutions. MIS's global DAG matching algorithm uses inexact heuristics like selecting the mapping that covers the most nodes. Backtracking approaches to technology mapping attempt to perform global optimizations; the longer the time they are given, the better the results. In the technology mapping scenario, however, parts do not have support functions, and hence logic-synthesis systems do not handle horizon effect due to support functions.

Most systems optimize design along multiple, conflicting, and non-commensurate attributes. For example, MICON and MBESDSD optimize along area, power and dollar cost. These systems use a weighted evaluation function to reduce the multi-attribute optimization problem to a single-objective-function optimization problem [1, 13, 3, 12]. A major difficulty in this approach is that weights must be precisely and completely specified before any evaluation can be done. It is difficult for designers to specify meaningful weights. In our tool, ISMAUT is used to evaluate nondominated designs even if precise trade-off weights are unavailable.

In an multiobjective integer-linear-programming formulation of the synthesis problem, there is an integer variable corresponding to every part that represents the number of instances of the part that will be in the design. Constraints among these variables, part-function multiplicities, and function multiplicities ensure that the parts together implement the desired functions. Additional variables represent function specifications and part characteristics, and are related through constraints like those in our formulation. A vast majority of the former category of variables are binary (0-1), which makes the problem very difficult to solve using this approach. Further, it is unclear how to handle *in*ing and *out*ing of support functions and constraints in an integer-linear programming formulation. Our tool *ins* and *outs* parts and support functions in part-function tables depending on the design state, and *ins* and *outs* architecture-dependent performance constraints in the network to allow a designer to explore different designs.

The worst-case performance of Ext_GOPS is exponential in time and space. In data sets where all pruning techniques in the algorithm are applicable, the performance is polynomial with respect to the number of parts, i.e., $O(|P|)$. Note that the applicability of pruning techniques can be governed by predicates that are more relaxed than the ones currently used in Ext_GOPS. There are cases where partitioning can be done even if parts have support functions. For instance, part selection for functions that are not interrelated by constraints and are covered only by single-function parts can be partitioned into individual subproblems if support functions of *in* parts are also covered only by single-function parts, and the condition holds recursively for these parts. Similarly, subsumption can be performed even if parts have support functions, provided competing parts implement the same functions, have the same values for characteristics that participate in across-function constraints, and have the same support functions. These relaxed predicates are true in most data sets; *hence, in the typical case, optimal solutions can be generated with much less than exponential search.* For data sets that are outside the scope of a majority of the heuristics in Ext_GOPS, the combinatorics of finding the globally optimal solution must be handled by parallel computation.

5. Summary and Conclusion

Optimal system-level design is hard. Multiple-function parts and constraints among functions make local-optimization techniques inadequate for global optimization. Parts and part sets cannot be evaluated without considering the cost of their support functions. Also, it is difficult to evaluate multiple-function parts without knowing all the support functions needed in the design. Multi-attribute optimization is complex because designers find it hard to assign precise weights to cost attributes.

Our synthesis tool generates optimal designs in two major steps: 1) generate the set of nondominated designs that conform to cost and performance constraints (using Ext_GOPS and the constraint network), and 2) order the set in accordance with designer's preferences (ISMAUT). Our tool uses several techniques to overcome horizon effect and mitigate the problem combinatorics. These heuristics can reduce the complexity from exponential to linear in best case. The applicability of the heuristics depends on properties of data sets. Ext_GOPS and the constraint network have been implemented and used to generate optimal designs for applications such as computer boards and controllers. These experiments confirmed our performance analysis of Ext_GOPS.

References

[1] Birmingham W.P., Gupta A.P., and Siewiorek D.P., Automating the Design of Computer Systems, Jones and Bartlett Publishers, 1992.
[2] Detjens E., Gannot G., Rudell R., Sangiovanni-Vincentelli A., and Want A., "Technology Mapping in MIS," Proc. ICCAD, 1984.
[3] Dirkes E., "A Module Binder for the CMU-DA System," SRC-CMU, Dept. of ECE, CMU, Research Report No. CMUCAD-85-43, 1985.
[4] Doyle J., "A Truth Maintenance System," Artificial Intelligence 12, No. 3, 1979.
[5] Dutt N.D. and Kipps J.R., "Bridging the High-Level Synthesis to RTL Technology Libraries," Proc. 28th DAC, 1991.
[6] Haworth M.S., Birmingham W.P., and Haworth D.E., "Optimal Part Selection", Technical Report CSE-TR-127-92, Department of EECS, Univ. of Michigan, Ann Arbor, 1992.
[7] Keutzer K., "DAGON: Technology Binding and Local Optimization by DAG Matching," Proc. 24th DAC, 1987.
[8] Mackworth A.K., "Consistency in networks of relations," Artificial Intelligence, vol. 8, no. 1, 1977.
[9] Mittal S. and Frayman F., "COSSACK: A Constraints-Based Expert System for Configuration Tasks," Proc. 2nd Int'l Conf. on Applications of AI to Engineering, 1987.
[10] Stewart B.S., Scherer W.T., Sykes E.A. and White C.C., III, "Defense Communications Decision Support Using ISMAUT," Advanced Technologies for C-squared Systems Engineering (S. Andriole, Ed.), AFCEA International Press (AIP), 1990.
[11] Sussman G.J. and Steele G.L., "Constraints - A Language for Expressing Almost Hierarchical Descriptions," Artificial Intelligence, vol. 14, 1980.
[12] Thomas D.E. and Leive G.W., "A Technology Relative Logic Synthesis and Module Selection System," Proc. 18th DAC, 1981.
[13] Wu J., Hu Y.H., Ho W.P.C., and Yun, D.Y.Y., " A Model-Based Expert System for Digital System Design," IEEE Design and Test of Computers, 1990.