IEICE TRANS. INF. & SYST., VOL.E84–D, NO.3 MARCH 2001

PAPER

DESC: A Hardware-Software Codesign Methodology for Distributed Embedded Systems

Trong-Yen LEE[†], ***Regular Member***, **Pao-Ann HSIUNG**[††], ***and*** **Sao-Jie CHEN**[†], ***Nonmembers***

SUMMARY The hardware-software codesign of *distributed embedded systems* is a more challenging task, because each phase of codesign, such as copartitioning, cosynthesis, cosimulation, and coverification must consider the physical restrictions imposed by the distributed characteristics of such systems. Distributed systems often contain several similar parts for which design reuse techniques can be applied. Object-oriented (OO) codesign approach, which allows physical restriction and object design reuse, is adopted in our newly proposed *Distributed Embedded System Codesign* (DESC) methodology. DESC methodology uses three types of models: *Object Modeling Technique* (OMT) models for system description and input, *Linear Hybrid Automata* (LHA) models for internal modeling and verification, and SES/*workbench simulation* models for performance evaluation. A *two-level partitioning* algorithm is proposed specifically for distributed systems. Software is synthesized by task scheduling and hardware is synthesized by system-level and object-oriented techniques. Design alternatives for synthesized hardware-software systems are then checked for design feasibility through rapid prototyping using hardware-software emulators. Through a case study on a *Vehicle Parking Management System* (VPMS), we depict each design phase of the DESC methodology to show benefits of OO codesign and the necessity of a two-level partitioning algorithm.

key words: *distributed embedded systems, emulation, two-level partitioning, object-oriented codesign, software scheduling*

1. Introduction

Distributed systems such as distance teaching facilities, vehicle parking systems, auditorium air-conditioning, coal-mine signal systems, and others abound in our everyday life and are almost all embedded with some sort of integrated chips and/or processors for running software. These systems are difficult to design due to their physical restrictions and distributed behavior. Different from the hardware-software codesign techniques for centralized systems, symmetricities in distributed structure, hierarchical system architecture, and physical restrictions have all to be considered in our newly proposed *Distributed Embedded System Codesign* (DESC) methodology. DESC offers design reuse through its object-oriented synthesis, considers physical constraints through its hierarchical system partitioning, and generates rapid prototyping through co-emulation.

When a system consists of parts that must be located at in different physical locations, it is called a *distributed system*. The design of distributed systems has always been a challenging task. Codesign of distributed systems must solve not only hardware-software communication issues within a single embedded unit, but also the communication issues between different parts of a distributed system (which may consist of either hardware, or software, or both). Hardware-software copartitioning must take into consideration the physical restrictions of a distributed system. Besides timing and cost constraints, part modularity and physical restrictions are also important factors that must be taken into account in the codesign of distributed systems. Interface must be synthesized not only for the hardware and the software within a system part, but also for the different parts of a distributed system.

In this paper, we propose a complete codesign methodology for *distributed embedded systems* called *Distributed Embedded System Codesign* (DESC) methodology. To allow maximum exposure to all the design phases in DESC, we have omitted some technical details so that the reader can obtain a more overall picture of how distributed systems are codesigned. DESC uses three types of semantic models for different purposes, namely *Object Modeling Technique* (OMT) models for system description and input, *Linear Hybrid Automata* (LHA) models for internal modeling and verification, and SES/*workbench* simulation models for performance evaluation. The *two-level* partitioning technique used in DESC includes: (1) Design space exploration to determine the number of processors for software execution and the hardware cost in a distributed embedded system, and (2) Hardware-software copartitioning to produce a final system partition result. SES/*workbench* simulation tool [1] is used to evaluate the performance of the copartition results. Then, hardware is synthesized using a recently proposed system-level object-oriented (OO) design methodology called *Intelligent Concurrent Object-Oriented Synthesis* (ICOS) methodology [2]. And software is synthesized by task scheduling on system processor(s). The synthesized results are further validated for feasibility by rapid prototyping through hardware-software emulators. Finally, a network of LHA models for a synthesized system is used to formally verify the

Manuscript received January 19, 2000.
Manuscript revised September 18, 2000.

[†]The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.
[††]The author is with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, R.O.C.

user-given system constraints.

The contributions of our proposed DESC methodology are described as follows. From the given system specification, DESC detects real physical restrictions of distributed embedded systems to achieve a final best design, while these physical restrictions are not considered explicitly by existing methods [3]–[6]. As for system modeling, the applications of three existing models in the codesign of distributed systems are newly proposed by DESC for the following reasons. (1) The formal LHA model is suited for partitioning and analysis, and (2) the modeling capability of most existing methods based on task graphs [7]–[12] are quite limited, compared to the OMT models used in DESC. Last but not the least, a new partitioning method is proposed in DESC, which has many desirable features, such as simplicity, efficiency, and hierarchy.

The article is organized as follows. Section 2 describes some previous and related work. Section 3 describes our hardware-software codesign methodology for distributed embedded systems and design phases of cosynthesis and emulation. A case study, *Vehicle Parking Management System* (VPMS), is given in Sect. 4. Finally a conclusion is drawn in Sect. 5.

2. Related Work

A distributed system has more than one physical location, each of which may have different characteristics. Previous works related to hardware-software codesign [7], [13] have not distinguished between the different physical locations of a distributed system and their characteristics. For example, a processor for software execution, when placed at different locations, could affect the overall system performance. Further, previous work have also not considered how design parts may be reused in a distributed system.

As far as hardware design is concerned, methodologies for the system-level synthesis of general-purpose multi-processor systems have been proposed recently, for example, *Performance Synthesis Methodology* (PSM) [16], *Intelligent Concurrent Object-Oriented Synthesis* (ICOS) methodology [2], and *Parallel Object-Oriented Synthesis Enviroment* (POSE) [17] are three of the most recently proposed methodologies. A new cosynthesis methodology for parallel systems called Cosynthesis Methodology for Application-Oriented Parallel Systems (CMAPS) [18] has also been recently proposed. Some other successful methodologies for hardware design include the MICON system [19], [20] and the Megallan System [21].

As far as distributed system codesign is concerned, several related works can be found. Prakash and Parker [14] formulated heterogeneous multiprocessor system synthesis as a mixed integer linear program in SOS, a formal synthesis approach developed at University of Southern California. Further, based on Prakash and Parker's formulation, Wolf [6] developed a heuristic algorithm for architectural co-synthesis of distributed embedded computing systems. Haworth et al. [22] proposed an algorithm that chooses parts from a part library (or catalog) to implement a set of functions meeting cost bounds. D'Ambrosio and Hu [23] presented a hardware-software co-synthesis approach at the configuration level.

Recently, the design of an n-CPU/m-ASIC topology is of major concern. Yen and Wolf [13] proposed a sensitivity-driven method for the co-synthesis of real-time distributed embedded systems. The cosynthesis algorithm selects the number of processing elements (PEs), the type of each PE, as well as allocating functions to PEs and scheduling their executions. Dick and Jha [9] proposed an algorithm for hardware-software cosynthesis of distributed embedded systems, namely MOGAC which partitions and schedules embedded system specifications consisting of multiple periodic task graphs. Recently, Dave, Lakshminarayana, and Jha [10] proposed a heuristic-based constructive cosynthesis technique call COSYN, which includes allocation, scheduling, and performance estimation steps. Another constructive cosynthesis system, called COFTA, was proposed in [11], which targets fault-tolerant distributed architectures and address reliability and availability of the embedded system during cosynthesis. In [12], a heuristic-based cosynthesis technique, called COHRA, was proposed, which takes as input an embedded system specification in terms of hierarchical acyclic task graphs and generates an efficient hierarchical hardware-software architecture that meets the real-time constraints. In [24], Dick and Jha proposed a system synthesis algorithm, called MOCSYN, which synthesizes real-time heterogeneous single-chip hardware-software architectures using an adaptive multiobjective genetic algorithm. A cosynthesis system, called CORDS, which synthesizes multi-rate, real-time, periodic distributed embedded systems containing dynamically re-configurable FPGAs was proposed in [25].

In comparison to previous work on hardware-software cosynthesis, which was based on task graphs as system model, our cosynthesis method uses object oriented (OO) hierarchy, LHA, and SES as system models. The task graph model basically assumes an arbitrary system architecture, without considering any restrictions on subsystem locations. In contrast, our OO model takes *realistic physical restrictions* into consideration for the target system architecture. This is more appropriate for distributed systems because of inherent architectural restrictions. Our proposed DESC is a more complete design methodology that not only carries out hardware and software syntheses, but also includes two-level partitioning, performance analysis, and emulation of distributed embedded systems.

3. Distributed Embedded System Codesign Methodology

In this section, we explain our methodology called *Distributed Embedded System Codesign* (DESC). As shown in Fig. 1, the design flow is divided into three main phases: (1) *Specification and Mapping*, (2) *Copartitioning and Performance Evaluation*, and (3) *Cosynthesis and Emulation*. Three models are used in DESC to represent a system at different phases of design. The models are *Object Modeling Technique* (OMT) models, *Linear Hybrid Automata* (LHA) models, and SES/*workbench* models. Given OMT models as input, corresponding LHA and SES models are then generated. System is then partitioned using a two-level partitioning scheme. Software is synthesized by task scheduling and hardware is synthesized using an object-oriented system-level design methodology. Finally, the resulting system design is checked for functional feasibility by emulation. The three design phases in DESC will be described in details in the following subsections.

3.1 Specification and Mapping Phase

A system under design is represented in DESC by the following three different models. *Object Modeling Technique* (OMT) models are used for system specification input; *Linear Hybrid Automata* (LHA) are used as an internal model for partitioning; and SES simulation models are used for performance evaluation of partition results.

OMT is an object-oriented software development methodology developed by Rumbaugh et al. [26]. OMT uses three kinds of models to describe a system: *object model*, which describes components in a system and their inter-relationships; *dynamic model*, which describes temporal interactions among objects in a system; and *functional model*, which describes data transformations of a system. We use OMT to describe the specification of a system under design and as an input to DESC.

LHA was proposed for modeling and verifying reliable designs that ensure correct operation of controllers in embedded systems [27]. Since LHA is a formal model that can be used for easily modeling multi-rate hardware and software embedded systems [28], DESC uses LHA as an internal model for system evaluation during partitioning and also for formal verification.

SES/*workbench* [1] is a popular modeling and simulation tool for system performance evaluation. An SES/*workbench* model is a hierarchy of submodels. Each submodel is represented by an extended directed graph. A higher level submodel may call lower level submodels. An SES/*workbench* model can be thought of as a parallel program with multiple execution threads. The tool is an integrated collection of software tools for specifying and evaluating system designs. We use SES/*workbench* to evaluate the correctness and performance of a system design. Performance evaluation is done by simulating the model derived from the system specification. Correctness is evaluated by executing, during the simulation, assertions that we attach to each design specification component.

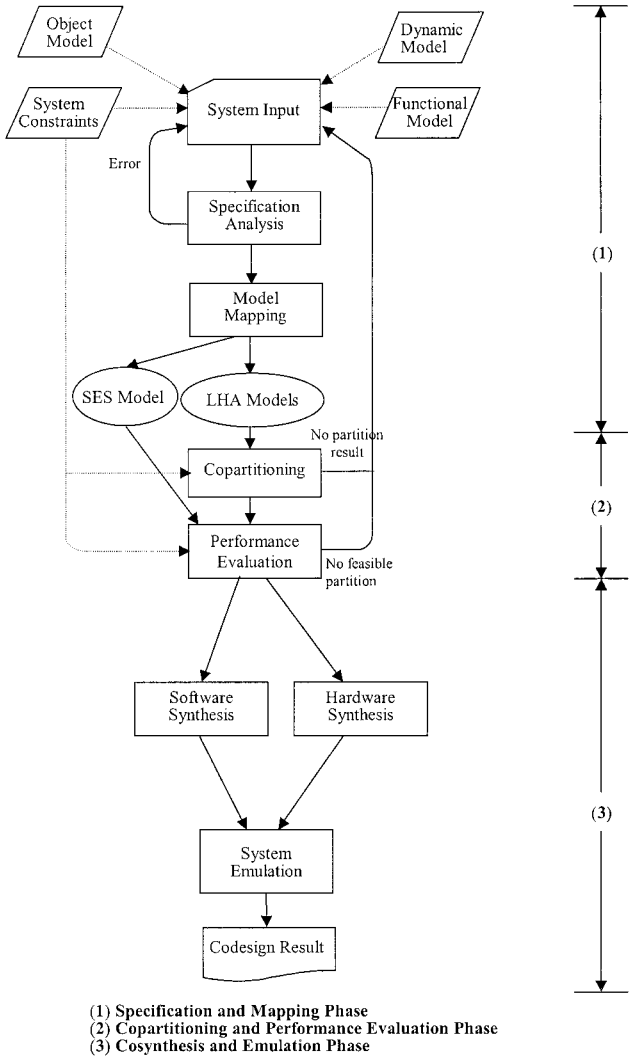


Fig. 1 Distributed embedded system codesign methodology.

In summary, a distributed system to be designed is described by a designer using OMT. DESC then transforms OMT models into LHA and SES models. Due to page-limits, we briefly mention that LHA models are based on the OMT dynamic models and SES models are based on all the three OMT models. An example will be given in Sect. 4.

3.2 Copartitioning and Performance Evaluation Phase

Once LHA and SES models are generated, DESC begins to partition a system-under-design into hardware and software parts. The copartitioning results are then

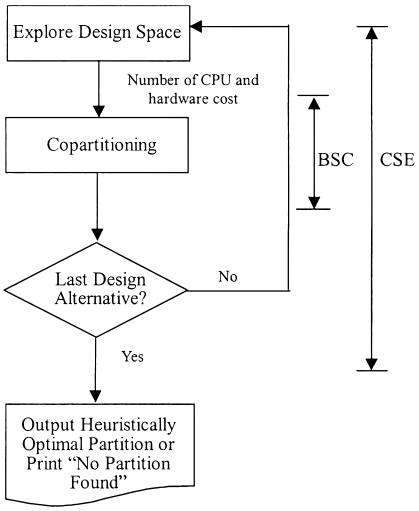


Fig. 2 Two-level partitioning.

simulated and checked whether the performance satisfies system constraints. Most of recent work studied hardware-software partitioning, that targets a one-CPU-one-ASIC topology with a predefined CPU type. Our target system consists of an n-CPU/m-ASIC topology, for which previous partitioning techniques are not adequate. Distributed embedded systems require partitioning techniques which not only considers hardware-software tradeoffs but also *system structure*.

We propose a *Two-Level Partitioning* (TLP) technique for distributed systems, which uses: (1) *codesign space exploration* to iterate over the number of CPUs and of ASICs; and (2) *binary search copartitioning* to produce hardware and software partitions that meet user-given system constraints. After partitioning, a popular simulation tool, SES/*workbench* [1], is used to simulate the partitioned results and to check whether the overall system performance of a particular partition satisfies given system constraints. This phase produces system design results in the form of hardware-software partitions with performance that meet system constraints. Due to page-limits, we have omitted the technical details of how the models are mapped to each other and the validity proofs for model equivalences.

Since our target systems are distributed with n-CPUs/m-ASICs, the inherent hierarchy in system structure necessitates a *Two Level Partitioning* (TLP) scheme. At the first level, we must explore how many CPUs and ASICs to use. The second level would be the conventional copartitioning between hardware and software in each subsystem of a distributed system. The flow diagram for TLP is shown in Fig. 2.

3.2.1 Codesign Space Exploration (CSE) Level

At the codesign space exploration (CSE) level, we must decide how many CPUs to use for software implementation and execution. In general, each location can have either zero or some positive number of processors, depending on the system cost bound. In DESC, by default the maximum number of CPUs in a system is constrained by the total number of different subsystems. System designers can easily override this default setting, but doing so lengthens the period of design space exploration because of a much larger design space size. Suppose a distributed embedded system under design has n subsystems. The CSE level iterates through $0, 1, 2, \ldots,$ and n CPUs for software implementation.

3.2.2 Binary Search Copartitioning (BSC) Level

This binary search copartitioning (BSC) level forms the core part of hardware-software copartitioning. We do not start from either of the two extreme solutions found in existing methods [4], [15]: all-hardware and all-software. Rather we start somewhere in-between and then based on two heuristic assumptions, we start moving towards the heuristically optimal feasible solution. The two heuristic assumptions are as follows. (1) First assumption is that hardware implementations always cost more than software solutions. This is true in general when costs are amortized over several components of a system. (2) Second assumption is that hardware implementations always perform better than software solutions. This is true in general because hardware ASICs can be optimized to a greater extent than software. Software optimizations are often restricted by compiler technology and microprocessor architecture that is hosting and executing it. Though the above two assumptions may sound not so realistic at first, yet they have been validated by most design experiences [2], [16], [17].

The copartitioning flow diagram and algorithm are given in Fig. 3 and Fig. 4, respectively. Two linear arrays are used to store system objects during copartitioning, namely, *Immovable Linear Array* (ILA) and *Movable Linear Array* (MLA). ILA is used to store objects that must be implemented as hardware parts, while MLA is used to store objects that could be implemented either as hardware or software parts. Copartitioning will be performed only on the objects in MLA. Each component in a system under partition is associated with a metric called *Cost-Performance Difference* (CPD) ratio defined as follows:

$$CPD(x) = \frac{[HC(x) - SC(x)] \times PB(x)}{|HP(x) - SP(x)|} \qquad (1)$$

where x is an object in MLA; $HC(x)$ is either the actual cost or the VLSI area of x; $SC(x)$ is either the cost of the main memory spent or the cost of the CPU used for executing x as a software program code; $HP(x)$ is the hardware response time; $SP(x)$ is the program execution time as implemented in a processor; and $PB(x)$

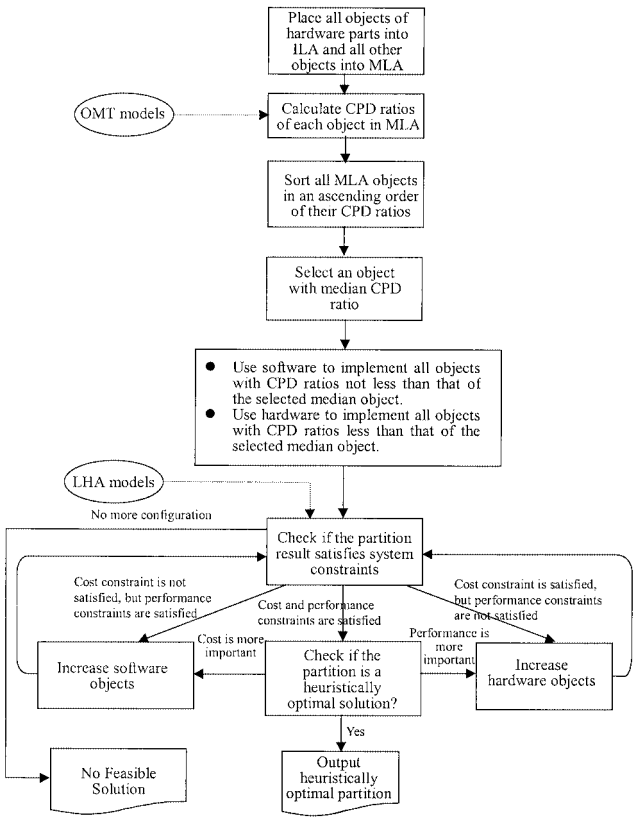


Fig. 3 The flow diagram of copartitioning methodology.

is the value of the performance bound associated with part x. Here, it is assumed that each part is associated with only one performance bound. The denominator in CPD(x) is a normalization of the performance difference, which is required for a fair comparison between different parts and performance bounds. The CPD metric is a useful hardware-software partitioning criterion. All objects are sorted in an ascending order of their CPD ratios and placed in a horizontal one-dimensional array called *Movable Linear Array* (MLA) from left to right.

The three models in OMT, namely object, dynamic, and functional, are used for calculating the CPD value of an object. The hardware cost and software cost are given as data attributes for each object. The hardware performance and software performance for an object are evaluated for an object by calling corresponding functional methods in a class description. Dynamic and functional models are, in fact, implemented as functional methods in a class description. Hence, all the three models are required for CPD calculation.

The copartitioning method begins somewhere around the middle of the sorted sequence of objects in MLA. A median object is selected as an initial *divider*. As shown in Fig. 5, the role played by a divider is that all objects to the right of the divider, including the divider, are implemented in software and the rest (at the left of the divider) are implemented in hardware. The reason that such an implementation is correct is two-folds. Firstly, the objects in the left part of the sequence have a greater gain in performance if implemented as hardware (i.e., a larger difference between hardware and software performance) and at a smaller

HW/SW_PARTITION$(N_1, N_2, N_3, \ldots, N_i, \ldots, N_r)$
/* $N_1, N_2, N_3, \ldots, N_i, \ldots, N_r$, where N_i is an object */ 1
Generate *Immovable Linear Array* (ILA) and
Movable Linear Array (MLA). 2
Calculate *Cost-Performance Difference* (CPD) ratio for each MLA object. 3
Sort all MLA objects in an ascending order of their CPD ratios such that MLA = $\langle M_1, M_2, M_3, \ldots, M_m \rangle$ 4
u:= Number of PE; Flag:=true; 5
PSS={} /* PSS:Partition Solution Set*/ 6
for $(p = 0, p \leq u, p++)$ 7
{
$HW\_Cost := Max\_Cost - Cost(PE) \times p$; 8
$k := 1, j := m$; /* where k is called the *lower bound object index*, j is called the *upper bound object index* */ 9
$i := \lceil \frac{m}{2} \rceil$ /* where i represents *divider object index* */ 10
Use software to implement objects from M_i to M_j and use hardware to implement objects from M_k to M_{i-1} 11
while Flag=true do 12
{
switch(cost and performance estimations) 13
{
Case 1: Cost constraint not satisfied, but performance constraints satisfied 14
$j := i$; 15
$i := i - \lceil \frac{i-k}{2} \rceil$; 16
break; 17
Case 2: Cost and performance constraints satisfied 18
if the partition result s' is heuristically optimal {19
Flag := false; PSS := PSS $\cup$ $\{s'\}$; break; } 20
else { if performance is more important{ 21
$k := i$; 22
$i := i + \lceil \frac{j-i}{2} \rceil$; } 23
else { $j := i$; /* cost is more important */ 24
$i := i - \lceil \frac{i-k}{2} \rceil$;} 25
} 26
break; 27
Case 3: Cost constraint satisfied, but performance constraints not satisfied 28
$k := i$; 29
$i := i + \lceil \frac{j-i}{2} \rceil$; 30
break; 31
Case 4: No satisfactory partition 32
Flag:=false; break; 33
} /* end of switch */ 34
} /* end of while */ 35
Flag:=true; 36
} 37
if PSS={} print "No partition found"; 38
else output heuristically optimal partition from PSS; 39

Fig. 4 Two-level partitioning algorithm (Algorithm 1).

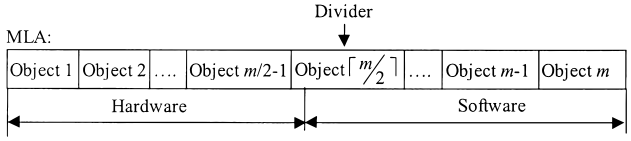


Fig. 5 Movable linear array (MLA) and divider.

expense (i.e., a smaller difference between hardware and software costs). Secondly, the objects in the right part of the sequence have a greater gain in saving costs if implemented as software (i.e., a larger difference between hardware and software costs) and at a smaller loss in performance (i.e., a smaller difference between hardware and software performance). Thus, the intuition for the CPD definition is clear from the role of the divider in copartitioning.

The initial partition obtained is then tested for feasibility under the given system constraints on cost and performance. Hardware and software LHA models corresponding to each object are used for feasibility testing and constraint satisfaction checking. Given a system partition, a set of LHA is selected corresponding to the hardware or software implementation of each object as implied by the partition. The set of LHA is then analyzed using Hytech, a popular verification and analysis tool for hybrid systems [27]. Interested readers may refer to [29] for further details.

As shown in Fig. 4 (Algorithm 1), four cases are encountered during feasibility testing. First, if the performance specifications are satisfied but cost specifications are not, then we must increase the software part by selecting a new divider towards the left of the current divider along the linear array of sorted objects. Second, if the cost specifications are satisfied but performance specifications are not, then we must increase the hardware part by selecting a new divider towards the right of the current divider along the linear array of sorted objects. Third, if both cost and performance specifications are satisfied, then depending on whether preference is given to minimizing cost or to maximizing performance we move towards the left or right, respectively. This will lead to a more cost-oriented or performance-oriented heuristically optimal solution. Finally, if either both cost and performance specifications are not satisfied or one of them cannot be satisfied, then the algorithm declares that no feasible partition can be found for the given system under the given constraints.

3.2.3 Features and Validity of TLP

There are several qualities of our algorithm that deserve further investigations. Firstly, whenever there exists a feasible solution for a system, our algorithm will definitely output the heuristically optimal solution. Secondly, if a feasible solution is found by the algorithm, then the final result will also be feasible. Thirdly, if a completely non-feasible solution (both cost and performance specifications are not satisfiable) is found, then there does not exist a feasible solution for the given system and thus our algorithm stops all further searching. The validity of these three qualities requires the two basic assumptions mentioned previously Sect. 3.2.2. They are formalized as the following three theorems.

THEOREM 1: *If a feasible partition exists, then the BSC copartitioning algorithm will find one.*

Proof: Suppose a feasible partition Z exists and BSC cannot find one, instead it stops at some object divider(Z') corresponding to some infeasible partition Z'. There are two cases here:

Case (1) divider(Z') is on the left side of divider(Z): Since Z' is not feasible, either cost(Z') or performance(Z') does not or both do not satisfy the system bounds of maximum cost or minimum performance, respectively. Three possibilities arise here.

(a) Only cost(Z') does not satisfy the bound: BSC will move further right from divider(Z'), hence divider(Z') cannot be the last object during the binary search. A contradiction follows.

(b) Only performance(Z') does not satisfy the bound: This implies performance(Z) also does not satisfy the bound because divider(Z) is on the right of divider(Z'). A contradiction follows.

(c) Both cost(Z') and performance(Z') do not satisfy the bounds: By Theorem (3), there is no feasible partition for the system. A contradiction follows.

Case (2) divider(Z') is on the right side of divider(Z): Similarly, we have three possibilities.

(a) Only cost(Z') does not satisfy the bound: This implies cost(Z) also does not satisfy the bound because divider(Z) is on the left of divider(Z'). A contradiction follows.

(b) Only performance(Z') does not satisfy the bound: BSC will move further left from divider(Z'), hence divider(Z') cannot be the last object during the binary search. A contradiction follows.

(c) Both cost(Z') and performance(Z') do not satisfy the bounds: By Theorem (3), there is no feasible partition for the system. A contradiction follows.

From the above two cases, we can see that in every possibility, a contradiction arises. Hence, our assumption is wrong, that is, if there exists a feasible partition, then BSC will definitely find one.

THEOREM 2: *Once a feasible partition is found, the final heuristically optimal partition found by the co-partitioning algorithm is always feasible.*

Proof: Suppose a feasible partition Z is found and the final partition found Z' is not feasible. Here, either one of three cases may occur:

Case (1) Only cost(Z') does not satisfy the bound: If divider(Z') is on the left of divider(Z), then BSC will return either some partition Z'' with divider(Z'') between divider(Z') and divider(Z) or Z itself. If divider(Z') is on the right of divider(Z), then this implies that cost(Z) also does not satisfy the cost bound. In either case, a contradiction arises.

Case (2) Only performance(Z') does not satisfy the bound: If divider(Z') is on the left of divider(Z), then this implies that performance (Z) also does not satisfy the performance bound. If divider(Z') is on the right of divider(Z), then BSC will return either some partition Z'' with divider(Z'') between divider(Z') and divider(Z) or Z itself. In either case, a contradiction arises.

Case (3) Both cost(Z') and performance(Z') do not satisfy the bounds: By Theorem (3), there is no feasible partition for the system. A contradiction follows.

From the above three cases, we have a contradiction arising in each case. Hence, our assumption is wrong, that is, if a feasible partition is ever found, the final partition found by BSC would be feasible.

THEOREM 3: *If a completely infeasible partition (both cost and performance constraints are not satisfied) is ever found during BSC, then there exists no feasible partition for the system. Hence, the partitioning algorithm can stop searching.*

Proof: Suppose a completely infeasible partition Z is found during the binary search and suppose there exists a feasible partition Z' for the system. The following two cases occur:

Case (1) Divider(Z') is on the left of divider(Z): This implies that performance(Z') does not satisfy the minimum performance bound of the system. A contradiction follows.

Case (2) Divider(Z') is on the right of divider(Z): This implies that cost(Z') does not satisfy the maximum cost bound of the system. A contradiction follows.

When all objects in a system satisfy the two assumptions on hardware-software cost and performance, TLP will find a heuristically optimal solution. But, if there are one or more objects whose hardware-software cost and performance do not satisfy the two assumptions, then TLP will not be able to find a feasible solution as TLP is not an exhaustive approach.

The hierarchical approach adopted in TLP is different from the traditional task-graph based hardware-software partitioning [7]–[12]. The conventional approach uses task-graphs to represent a distributed embedded system such that each task graph represents a sequential execution of processes with possible branching, either deterministic or non-deterministic. Usually cycles are not allowed in task-graphs. Two or more task-graphs execute concurrently and may communicate with each other, exchanging data. Hierarchy among task-graphs is generally represented as flattened-out graphs. This often incurs redundancy in representation and unnecessary repeated procedures in the partitioning process.

TLP is based on object-oriented design modeling and is a hierarchical process, which is more appropriate for distributed architectures because they are inherently hierarchical. Distributed architectures normally have physical constraints related to the environment, such as some subsystems must be located within some pre-specified distance or some subsystems may share or may not share an ASIC or a PE. Such physical restrictions are either difficult to model in task-graphs and not considered in previous work on partitioning. TLP considers all such physical constraints and thus TLP is more realistic when distributed architectures are targets.

In summary, the two-level partitioning algorithm presented in this section is more appropriate for distributed embedded systems than conventional partitioning algorithms due to its consideration of the distributed characteristics in the systems.

3.3 Cosynthesis and Emulation Phase

In this section, we will introduce how to cosynthesize and emulate the resulting system obtained from partitioning. Cosynthesis includes hardware design and software design. Checking for design feasibility through rapid prototyping using hardware-software emulator will be introduced in the emulation phase of codesign.

Given the partition results from the previous phase, in this phase, hardware is synthesized by an object-oriented system-level design methodology called ICOS [2] and software by scheduling tasks on processors. The synthesized system designs are then checked for feasibility by rapid-prototyping, which includes hardware and software emulation and testing. This design phase is presented as follows, which produces design results in the form of feasible synthesized distributed hardware-software systems.

3.3.1 Hardware Synthesis

The *Intelligent Concurrent Object-oriented Synthesis* (ICOS) [2] methodology was used for hardware synthesis in DESC for the following three reasons. (1)

ICOS is an object-oriented design methodology and DESC has object-oriented input models, (2) ICOS was proposed for multiprocessor synthesis and the target system of DESC are distributed with multiple processors, and (3) ICOS proposes active *self-synthesis* which is a distributed technique and thus suitable for the distributed system synthesis in DESC. In spite of various advantages, it was still required to modify ICOS because the target system in ICOS was for general-purpose multiprocessor architectures. The main modifications performed were in the specification language (from general-purpose characteristics to application-specific characteristics) and in the class hierarchy (from general-purpose components to application-specific components). Learning and fuzzy methods in ICOS were not utilized for DESC because the target systems are application-specific.

As shown in Fig. 6, the ICOS methodology [2] is divided into three design phases called *Specification Analysis*, *Concurrent Design*, and *System Integration*. In the specification analysis phase, a user's system specifications are first analyzed to check whether there are any architecture related errors, or obvious errors such as constraints that are not feasible under current technology. In the concurrent design phase, the main system synthesis is performed. Here, components (including complete subsystems) are designed concurrently. An active synthesis approach is adopted, where each component actively seeks to synthesize itself. A *design hierarchy* (DH) and a *design queue* (DQ) are utilized in the concurrent design phase of ICOS for recording the current design status and for maintaining object synthesis turns. The final phase of system integration mainly evaluates the performance of a completed design and outputs the system design that best meets the design constraints. Details can be found in [2].

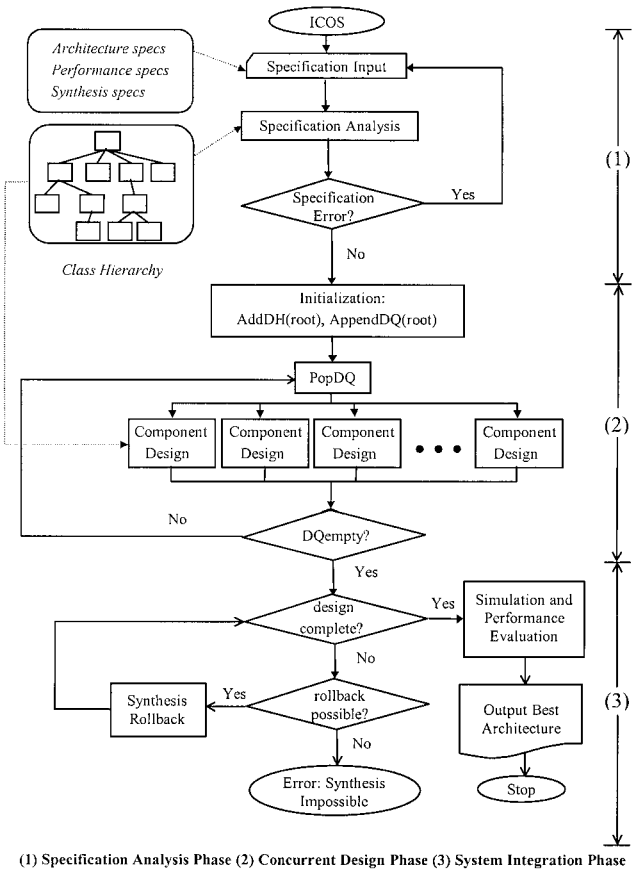


Fig. 6 Hardware synthesis flow (ICOS [2]).

3.3.2 Software Synthesis

For software synthesis, there are three issues to handle. First issue is how to produce software state diagrams from LHA software models. Second issue is how to produce feasible schedules from software state diagrams. Third issue is how to produce software pseudo code from feasible schedules. In the following, we propose our software synthesis method.

(1) *Software State Diagrams*: From the partitioning result, we know which objects are to be implemented in software. Then, we convert these software LHA models into software state diagrams. This conversion is a direct mapping from locations in an LHA to process states in software state diagrams, where each state is then implemented as a sequential process.

(2) *Software Schedules*: Software state diagrams have to be made feasible by the scheduling of processes on processors. The scheduling rules are as follows.

(a) Each state in the software state diagrams is viewed as a *process*.
(b) The initial states (initialization processes) are scheduled first.
(c) Within a group of concurrently enabled processes, higher priority processes are scheduled first. For arbitration between two or more processes with equal priority, processes that have shorter execution times are scheduled first. The complete schedule is generated using *As-Soon-As-Possible* (ASAP) algorithm [30].
(d) When two or more processes have equal priorities and equal execution times, the processes that are located in execution runs with shorter lengths are scheduled first.

(3) *Software Pseudo Code*: The software pseudo code is produced from the feasible software schedules (obtained from the previous step) by the following rules.

(a) Each process in the software schedule is implemented as a program procedure or subroutine.
(b) If a process has more than one successor processes that are triggered under different conditions, then we use a switch-case to represent

the process. Each case consists of one or more procedures or subroutines.

Here, we have only shown how software scheduling is performed on a single processor. For more than one processor, task scheduling is NP-complete [31], thus various heuristics have been proposed. For example, in [32], [33] many algorithms were proposed for periodic tasks in distributed systems, which form a big task with length of the least common multiple (LCM) of all the periods. Leinbaugh and Yamani [34] derived bounds for response times of a set of periodic tasks running on a distributed system without using the LCM enumeration. Lehoczky and Sha [35] pointed out the bus scheduling. Yen and Wolf [13] proposed the method of allocation and scheduling of processes and communication. The techniques such as fixed-point iterations, phase adjustment, and separation analysis were proposed to efficiently estimate tight bounds on the delay required for a set of multi-rate processes preemptively scheduled on a real-time reactive distributed system. In DESC, we use *Largest Scheduled Parallelism First* (LSPF) [36], *Largest Width with Largest Processing Time first* (LWLPT) [37], and a heuristic algorithm [38] for multiprocessor task scheduling.

3.3.3 Emulation

The purpose of emulation is to validate cosynthesis results such that they satisfy system constraints. The emulation flow diagram in DESC is as shown in Fig. 7. First, the cosynthesis results are refined such that hardware and software design details are all down-scaled by a suitable factor for possible prototyping. Hardware prototype fabrication includes layout, placement, and wiring. Meanwhile, the scheduled software programs are coded into the microprocessor. Finally, the system is tested using test cases to check system correctness. Based on the system performance results obtained through emulation, an optimal design solution is generated.

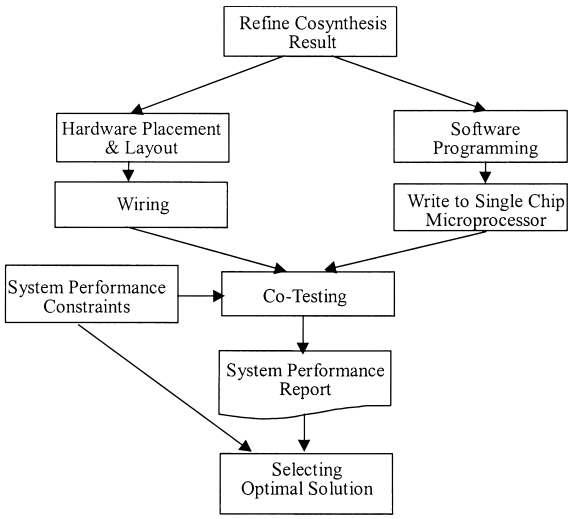


Fig. 7 Emulation flow diagram.

4. A Case Study – Vehicle Parking Management System (VPMS)

We describe a distributed embedded system for illustrating the proposed DESC methodology. This example is on the design of a vehicle parking system, called *Vehicle Parking Management System* (VPMS) [39]. VPMS consists of three subsystems: ENTRY management, EXIT management, and DISPLAY. As both of the ENTRY and EXIT subsystems allow vehicles to pass through them one by one, they are similar in most respects. DISPLAY subsystem shows the current number of vacant parking space available in a parking lot or garage.

An ENTRY (or an EXIT) subsystem consists of three parts: a ticket facility, a gate controlled by a gate-motor, and a pair of sensors. The ticket facility at the entry stamps the current date and time and gives a new ticket to an in-coming vehicle. The ticket facility at the exit checks whether the ticket (parking) fees have been paid and the current time is within 15 minutes of the ticket fee payment. After a positive response is received from the ticket facility, a gate controller opens the ENTRY (EXIT) gate to allow a vehicle to drive in (out). A pair of sensors are located after the gate (in the direction of the vehicle, that is, further in for the entry and further out for the exit). The sensors then send a signal to the gate controller to close the gate after a vehicle has passed by. At the same time, the sensors also send a signal to the display for updating the displayed number of parking vacancies.

Constraints for the VPMS system include: a maximum cost of $1,300, a maximum display response time of 14,000 μs, and a maximum ENTRY (EXIT) gate response time of 250 μs. The maximum display response time ensures that the display should not be updated too slowly. The maximum gate response time implies that the gate should not be closing or opening too slowly. A slow gate open would make the VPMS user unhappy, while a slow gate close would allow more than one vehicle to pass through the gate for each entry/exit transaction and would make the VPMS boss unhappy. Hence, gate response time is specified for correct execution of the system. The above description clearly shows that VPMS is a distributed system having some similar parts.

4.1 Specification and Mapping of VPMS

VPMS is described using OMT models consisting of object, dynamic, and functional models. The object model of VPMS is shown in Fig. 8. VPMS includes

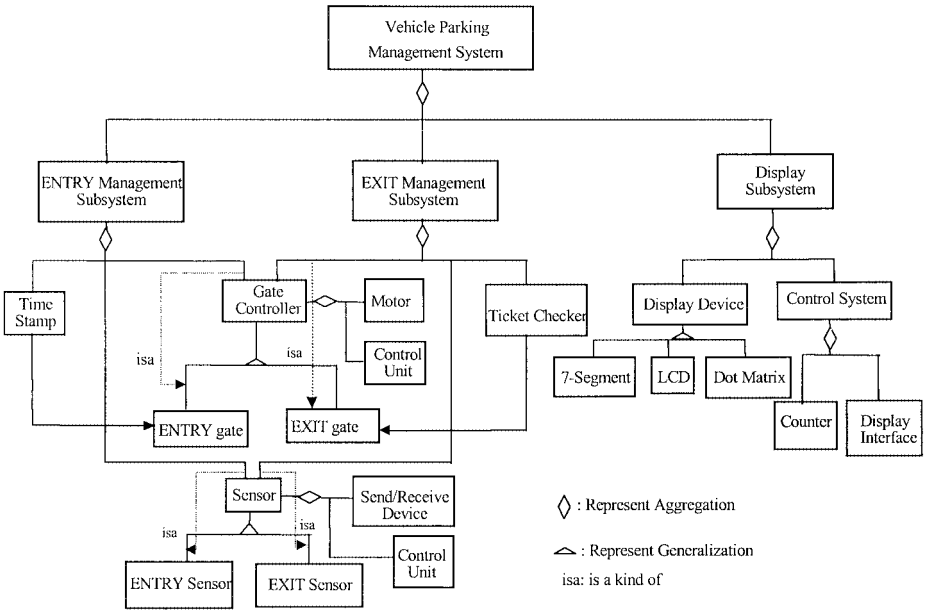


Fig. 8 Object model of VPMS.

three subsystems: ENTRY management, EXIT management, and DISPLAY. The ENTRY management subsystem includes time-stamp, gate controller, and a pair of sensors (send and receive signal devices). The EXIT management subsystem is similar to the ENTRY management subsystem, which includes ticket checker, gate controller, and a pair of sensors. The gate controller and sensor object models in ENTRY and EXIT management subsystems have many similar parts, which allow reuse of OMT models. The DISPLAY system consists of a control system (counter and display interface) and a display device such as 7-segment display, LCD, or dot matrix LED display. The counter value (*count*) indicates the number of available parking vacancies. The dynamic model of VPMS consists of three models for the three subsystems. An example for the DISPLAY is shown in Fig. 9. An example functional model of VPMS for DISPLAY subsystem is shown in Fig. 10. Due to page-limits, the dynamic and functional models for the other two subsystems are omitted. In DESC, each object uses two LHA models, called hardware LHA model and software LHA model, to represent the temporal behavior of hardware and software implementations, respectively.

A SES model for car-simulator is shown in Fig. 11. The model is used to simulate the behavior of a car entering or exiting a parking lot. It produces events that trigger ENTRY management, EXIT management, and DISPLAY subsystems. Therefore, SES models of the three subsystems are executed concurrently according to events from the car-simulator. The performance of VPMS, for example display response time and gate response time, can be collected using simulation tools supported in each SES model.

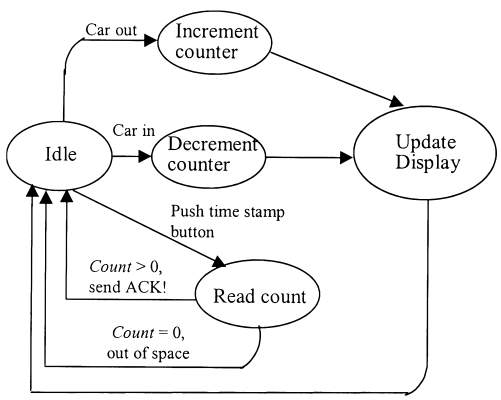


Fig. 9 Dynamic model of a DISPLAY subsystem.

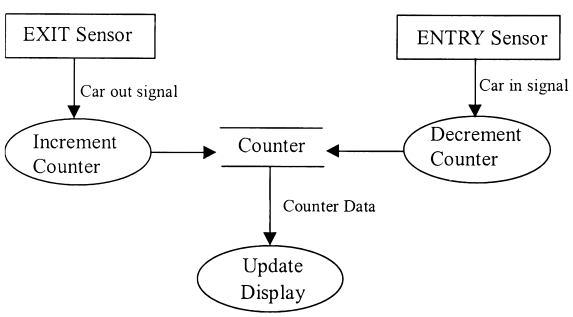


Fig. 10 Functional model of a DISPLAY subsystem.

4.2 Copartitioning and Performance Evaluation of VPMS

VPMS has five parts which can be implemented as either hardware or software, namely, counter, exit sensor

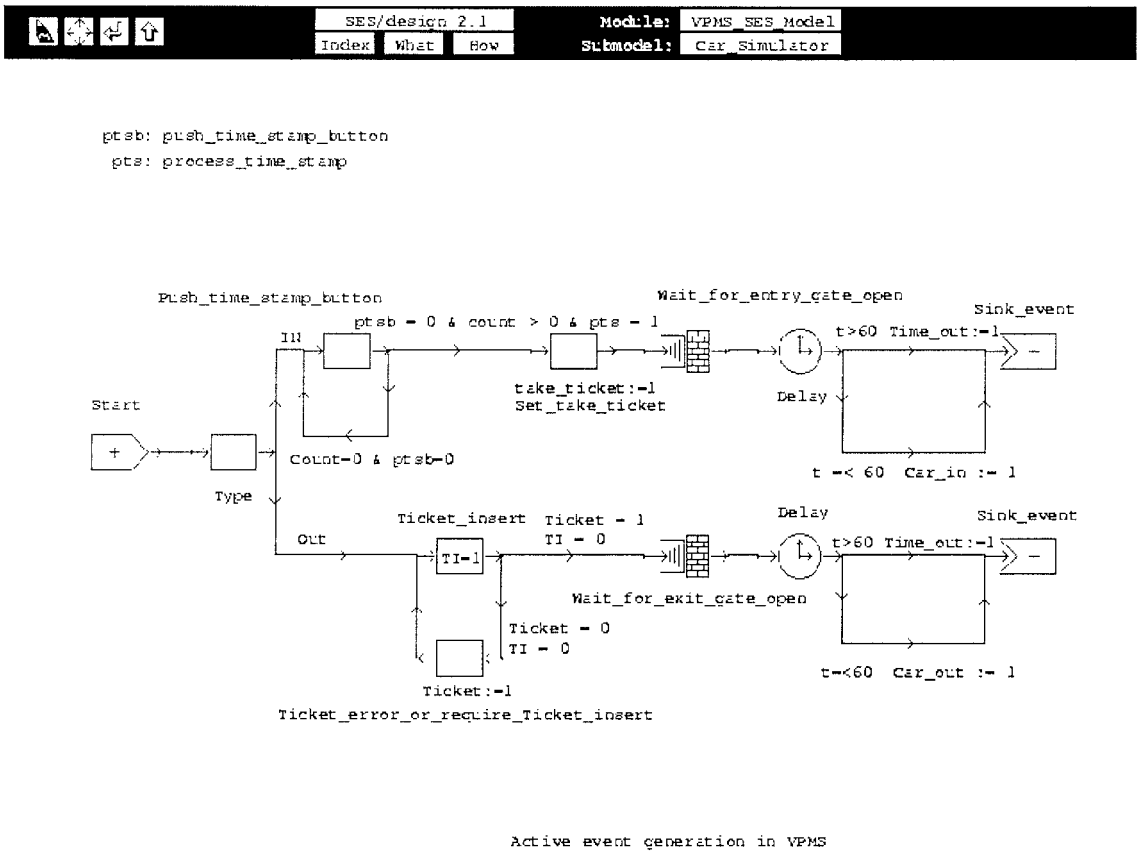


Fig. 11 SES model of a car-simulator.

Table 1 Calculation of CPD for VPMS parts hardware cost.

	Hardware Cost	Software Cost	Hardware Performance	Software Performance	CPD
Sensor Driver	115	90	210	1,030	7.622
Counter	120	90	290	13,200	32.533
Motor Driver	260	90	820	1,030	202.381

driver, entry sensor driver, exit motor driver, and entry motor driver. In this example, we assume that the ENTRY Management Subsystem and the EXIT Management Subsystem are located near each other such that the two sensor drivers can be implemented as one component (ASIC or CPU) and the two motor drivers can also be implemented as another component both to be shared by the two subsystems. Henceforth, we will consider only three parts (counter, sensor driver, and motor driver) for copartitioning.

Applying TLP to VPMS, by default TLP assumes that at most three CPUs are used for software implementation. In Table 1, the CPD values for each of the three parts were calculated according to Eq. (1) and the objects arranged in an ascending order in the MLA array. The resulting order is ⟨sensor driver, counter, motor driver⟩. In the CSE level, TLP iterates through 0, 1, 2, ..., and 3 CPUs. For each selected number of CPUs, the BSC level determines feasible partitions. Table 2 shows each iteration of applying TLP to the VPMS example. An analysis of this application shows that an exhaustive search for the heuristically optimal partitioning requires evaluating 20 different partitions. This number would be much greater when larger examples are considered.

From Table 2, we observe that when all three parts are implemented as hardware (i.e. # CPU = 0), there is no feasible partition. Here, feasibility implies satisfaction of all cost and performance constraints. For each iteration of the CSE level, the divider is selected to be the counter part. We see that out of 20 possible partitions, TLP evaluates only 6 partitions, two of which are feasible. The heuristically optimal partition is then selected to be best one of the feasible two. Here, "best" is defined in terms of minimum cost or minimum response time as preferred by the designer. From Table 3, it is observed that at least for VPMS, TLP finds the heuristically optimal partitioning by evaluating only 30% of

Table 2 Applying TLP to the VPMS example.

Codesign Space Exploration (CSE) (Number of CPU)	Binary Search Copartitioning (BSC)				Feasibility
	Partitions (SSP)	Cost ($)	Response time (μs) (sensor to display)	Response time (μs) (sensor to gate)	
0	$\mathbf{A}(H_C, H_S, H_M)$	1,450	190	0.2	No
1	$\mathbf{B}$ (H_C, H_S, S_M)	1,280	190	215	Yes
2	$\mathbf{C}$ (H_C, H_S, S_M^2)	1,370	13,200	820	No
	$\mathbf{D}($ $S_C, H_S, S_M)$	1,250	13,100	215	Yes
3	$\mathbf{E}$ (S_C^2, H_S, S_M)	1,340	13,100	210	No
	$\mathbf{F}$ (S_C, S_S, S_M)	1,225	13,200	1,030	No

H:hardware, S:software, subscripts: C=Counter, S=Sensor Driver, M=Motor Driver, superscripts: 1 ⇒ One CPU, 2 ⇒ Two CPUs, 3 ⇒ Three CPUs.

Table 3 Efficiency of TLP (VPMS example).

	Exhaustive Search	TLP Search Space	# Feasible Solutions
Number of Partitions	20	6	2
Percentage of Design Space Explored	100	30	10

all possible partitions, out of which 10% are feasible ones. This shows that TLP is an efficient partitioning algorithm for distributed embedded systems.

4.3 Cosynthesis and Emulation of VPMS

For VPMS, the hardware specification includes the following: (1) the response time from sensor to display should be at least 14,000 μs, and (2) the response time from sensor to gate motor interface should be at least 250 μs. On applying ICOS to prototype $\mathbf{B}(H_C, H_S, S_M)$, the synthesized *hardware cost* is \$1,190, where H and S represent hardware and software implementations, respectively and the subscripts C, S, and M denote the parts: Counter, Sensor Driver, and Motor Driver, respectively. For prototype $\mathbf{D}(S_C, H_S, S_M)$, the synthesized *hardware cost* is \$1,070.

In the software schedule of VPMS shown in Fig. 12, the four processes: Open EXIT gate, Open ENTRY gate, Close ENTRY gate, and Close EXIT gate, all have equal priorities and equal execution times, but the first two are located in runs with length of only two, while the other two are in runs with length of five. For example, process Close ENTRY gate is located in the run: Scan Display → Poll Signal → Close ENTRY gate → Decrement Counter → Update Display → Scan Display. Hence, the first two processes are executed before the other two.

VPMS *Emulation*: The two feasible partitioning results $\mathbf{B}(H_C, H_S, S_M)$ and $\mathbf{D}(S_C, H_S, S_M)$ for VPMS (refer to Table 2) were both emulated and their performances obtained. We can see from Table 4 that partition prototype $\mathbf{D}(S_C, H_S, S_M)$ gives a better design result in terms of a lower cost and a lower power consumption. The block diagram for partition prototype

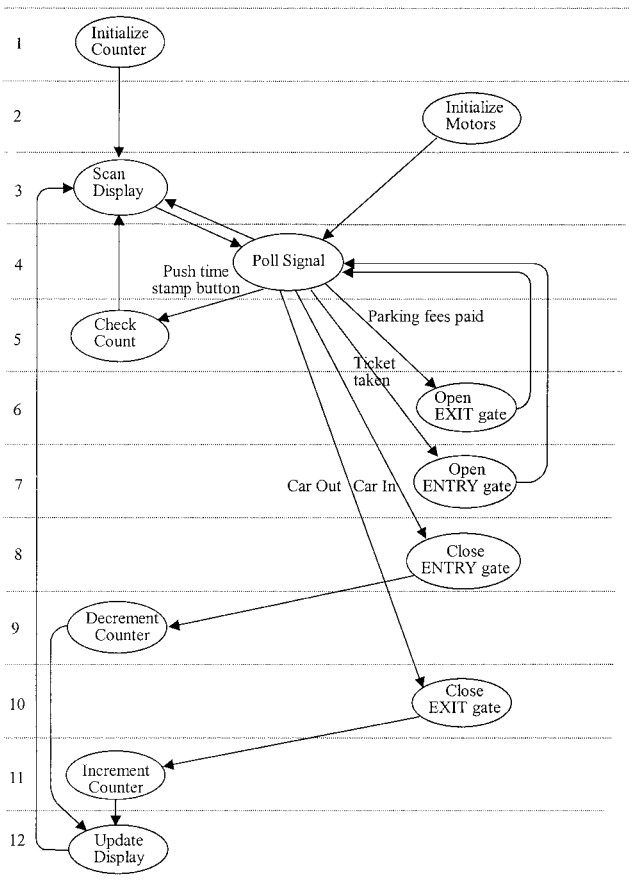


Fig. 12 Scheduling of software diagram on VPMS.

$\mathbf{D}(S_C, H_S, S_M)$ is as shown in Fig. 13. We chose a single chip integrated circuit 8051 [40] for software design which has 4 Kbytes of on-chip programmable memory for software programming, four input/output ports for data or control interface, two 16-bit timer/counters for time calculation or counting, and one series port for data transmission.

5. Conclusion

We have proposed a complete codesign methodology called *Distributed Embedded System* Codesign (DESC), which extends conventional centralized system code-

Table 4 VPMS emulation results partitions.

Partitions	$\mathbf{B}(H_C, H_S, S_M)$	$\mathbf{D}(S_C, H_S, S_M)$
Cost ($)	1278	1240
Power Consumption (W)	4.76	4.20
Response time (μs) (sensor to display)	180	13,000
Response time (μs) (sensor to gate)	210	210

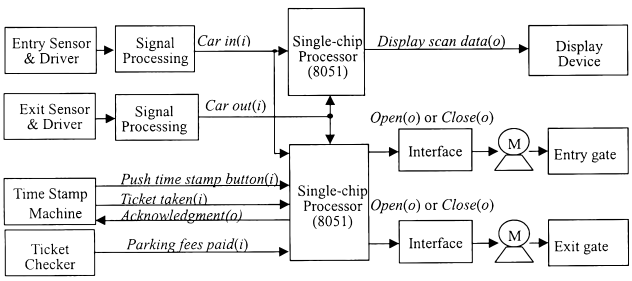


Fig. 13 Block diagram for prototype $\mathbf{D}(S_C, H_S, S_M)$.

sign methods and enhances existing distributed system codesign methods through object-oriented design reuse, hierarchical system partition, and consideration of physical constraints. In DESC, three models, namely *Object Modeling Technique*, *Linear Hybrid Automata*, and SES simulators are used for system specification, internal modeling, and performance evaluation, respectively. A two-level partitioning algorithm was proposed for distributed systems that also considered system structure besides hardware-software copartitioning. Hardware is synthesized using a recently proposed object-oriented system-level methodology called ICOS and software is synthesized by scheduling tasks on the microprocessors. Emulation is used for functional validation of our codesign results. A case study on VPMS using DESC shows its advantage in accelerating design time and decreasing design efforts.

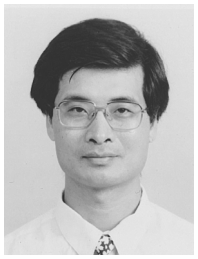
Acknowledgment

This work was supported by the National Science Council, R.O.C. under grant NSC89-2213-E002-097.

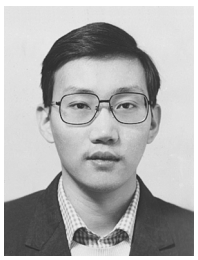
References

[1] Scientific and Engineering Software, Inc., SES/*workbench* user's manual releases 2.0, Jan. 1991.
[2] P.-A. Hsiung, C.-H. Chen, T.-Y. Lee, and S.-J. Chen, "ICOS: An intelligent concurrent object-oriented synthesis methodology for multiprocessor systems," ACM Trans. Design Automation of Electronic Systems, vol.3, no.2, pp.109–135, April 1998.
[3] A. Kalavade and E.A. Lee, "A hardware-software codesign methodology for DSP applications," IEEE Design and Test of Computers, vol.10, no.3, pp.16–28, Sept. 1993.
[4] R.K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," IEEE Design and Test of Computers, vol.10, no.3, pp.29–41, Sept. 1993.
[5] E. Barros, W. Rosentiel, and X. Xiong, "A method for partitioning UNITY language in hardware and software," Proc. European Design Automation Conference, IEEE Computer Society Press, pp.220–225, 1994.
[6] W. Wolf, "Hardware-software co-design of embedded systems," Proc. IEEE, vol.82, no.7, pp.967–989, July 1994.
[7] T.-Y. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," Proc. International Conference on Computer-Aided Design, pp.288–294, IEEE Computer Society Press, 1995.
[8] W. Wolf, "An architecture co-synthesis for distributed, embedded computing systems," IEEE Trans. VLSI Systems, vol.5, no.2, pp.218–229, June 1997.
[9] R.P. Dick and N.K. Jha, "MOGAC: A multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.17, no.10, pp.920–935, Oct. 1998.
[10] B.P. Dave, G. Lakshminarayana, and N.K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.7, no.1, pp.92–104, March 1999.
[11] B.P. Dave and N.K. Jha, "COFTA: Hardware-software cosynthesis of heterogeneous distributed embedded systems for low overhead fault tolerance," IEEE Trans. Comput., vol.48, no.4, pp.417–441, April 1999.
[12] B.P. Dave and N.K. Jha, "COHRA: Hardware-software cosynthesis of hierarchical heterogeneous distributed embedded systems," IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst., vol.17, no.10, pp.900–919, Oct. 1998.
[13] T.-Y. Yen and W. Wolf, Hardware-software co-synthesis of distributed embedded systems, Kluwer Academic Publishers, Netherlands, 1996.
[14] S. Prakash and A.C. Parker, "SOS: Synthesis of application-specific heterogeneous multiprocessor systems," J. Parallel and Distributed Computing, vol.16, no.4, pp.338–351, Dec. 1992.
[15] R. Ernst, J. Henkel, and T. Berner, "Hardware-software co-synthesis for microcontrollers," IEEE Design and Test of Computers, vol.10, no.4, pp.64–75, Dec. 1993.
[16] P.-A. Hsiung, S.-J. Chen, T.-C. Hu, and S.-C. Wang, "PSM: An object-oriented synthesis approach to multiprocessor system design," IEEE Trans. VLSI Systems, vol.4, no.1, pp.83–97, March 1996.
[17] P.-A. Hsiung, "POSE: A parallel object-oriented synthesis environment," ACM Trans. Design Automation of Electronic Systems, vol.6, no.1, Jan. 2001.
[18] P.-A. Hsiung, "CMAPS: A cosynthesis methodology for application-oriented parallel systems," ACM Trans. Design Automation of Electronic Systems, vol.5, no.1, pp.51–81, Jan. 2000.
[19] W.P. Birmingham, A.P. Gupta, and D.P. Siewiorek, "The MICON system for computer design," Proc. 26th ACM/IEEE Design Automation Conference, pp.135–140, 1989.
[20] A.P. Gupta, W.P. Birmingham, and D.P. Siewiorek, "Automating the design of computer systems," IEEE Trans. CAD of IC, vol.12, no.4, pp.473–487, April 1993.
[21] A.J. Gadient and D.E. Thomas, "A dynamic approach to controlling high-level synthesis CAD tools," IEEE Trans. VLSI Systems, vol.1, no.3, pp.328–341, Sept 1993.
[22] M.S. Haworth, W.P. Birmingham, and D.E. Haworth, "Optimal part selection," IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst., vol.CAD-12, no.10, pp.1611–1617, Oct. 1993.

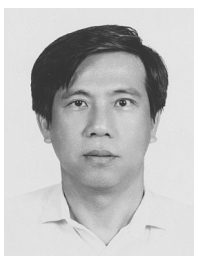
[23] J.G. D'Ambrosio and X. Hu, "Configuration-level hardware/software partitioning for real-time embedded systems," Proc. International Workshop on Hardware-Software Co-Design, pp.34–41, 1994.
[24] R.P. Dick and N.K. Jha, "MOCSYN: Multiobjective core-based single-chip system synthesis," Design Automation and Test, Europe (DATE'99), pp.263–270, ACM Press, March 1999.
[25] R.P. Dick and N.K. Jha, "CORDS: Hardware-software co-synthesis of reconfigurable real-time distributed embedded systems," IEEE/ACM International Conference on Computer-Aided Design (ICCAD'98), pp.62–68, ACM Press, Nov. 1998.
[26] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, Object-oriented modeling and design, Prentice-Hall, Englewood Cliffs, NJ, USA, 1991.
[27] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "A user guide to HyTech," Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol.1019, pp.41–71, Springer Verlag, 1995.
[28] P.-A. Hsiung, "Timing coverification of concurrent embedded real-time systems," Proc. 7th International Workshop on Hardware/Software Codesign, pp.110–114, ACM Press, May 1999.
[29] J.-M. Fu and S.-J. Chen, "Hardware-software coverification of distributed embedded systems," Proc. 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), vol.6, pp.2995–3001, June 1999.
[30] D. Landskov, S. Davidson, B. Shriver, and P. Mallet, "Local microcode compaction techniques," ACM Computing Surveys, vol.12, no.3, pp.261–294, Sept. 1980.
[31] J.Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation, vol.2, pp.237–250, 1982.
[32] K. Ramamritham, "Allocation and scheduling of complex periodic tasks," Proc. International Conference on Distributed Computing Systems, pp.108–115, 1990.
[33] D.-T. Peng and K.G. Shin, "Static allocation of periodic tasks with precedence constraints," Proc. International Conference on Distributed Computing Systems, pp.190–198, 1989.
[34] D.W. Leinbaugh and M.-R. Yamani, "Guaranteed response times in a distributed hard-real-time environment," Proc. Real-Time Systems Symposium, pp.157–169, 1982.
[35] J.P. Lehoczky and Lui Sha, "Performance of real-time bus scheduling algorithms," ACM Performance Evaluation review, May 1986.
[36] J.-F. Lin, W.-B. See, and S.-J. Chen, "Performance bounds on scheduling parallel tasks with communication cost," IEICE Trans. Inf. & Syst., vol.E78-D, no.3, pp.263–268, March 1995.
[37] J.-F. Lin and S.-J. Chen, "An analysis of multiprocessor tasks scheduling," Computer Systems Science and Engineering, vol.11, no.2, pp.117–120, March 1996.
[38] J.-F. Lin and S.-J. Chen, "Scheduling algorithm for non-preemptive multiprocessor tasks," Computers and Mathematics with Applications, vol.28, no.4, pp.85–92, 1994.
[39] T.-Y. Lee, P.-A. Hsiung, and S.-J. Chen, "A case study in hardware-software codesign of distributed systems — Vehicle parking management system," Proc. 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), vol.6, pp.2982–2987, June 1999.
[40] Intel, Embedded microcontrollers and processors, vol.1, Intel Corporation, 1993.



Trong-Yen Lee received the B.S. and M.S. degrees in industrial education (major in electronic engineering), National Taiwan Normal University, Taiwan, ROC, in 1981 and 1988, respectively, and the Ph.D. degree in electrical engineering from the National Taiwan University, Taiwan, ROC, in Jan. 2001. He has been teaching electronic engineering in Nankang Vocational High School, Taipei, Taiwan since 1980. His current research interests include hardware-software codesign, parallel architectures, simulation, and design automation systems.



Pao-Ann Hsiung received the B.S. degree in mathematics and the Ph.D. degree in electrical engineering from the National Taiwan University (NTU), Taipei, Taiwan, ROC, in 1991 and 1996, respectively. From 1993 to 1996, he was a Teaching Assistant and System Administrator in the Department of Mathematics, NTU. From 1996 to 2000, he was a post-doctoral researcher at the Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. Currently, he is an assistant professor at the Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan. His main research interests include: hardware-software codesign, real-time systems, formal verification, system-level design automation of multiprocessor systems, and object-oriented technology transfer.



Sao-Jie Chen received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1977 and 1982 respectively, and the Ph.D. degree in electrical engineering from the Southern Methodist University, Dallas, USA, in 1988. Since 1982, he has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University, where he is currently a full professor. From 1985 to 1988, he was on leave from National Taiwan University and working toward his Ph.D. at Southern Methodist University. During the fall of 1999, he was a visiting scholar in the Department of Computer Science and Engineering, University of California, San Diego. His current research interests include: VLSI ciruits design, VLSI physical design automation, object-oriented software engineering, and multiprocessor architecture design and simulation. Dr. Chen is a member of the Chinese Institute of Engineers, the Association for Computing Machinery, the IEEE, and the IEEE Computer Society.