# Integrated Hardware-Software Co-Synthesis and High-Level Synthesis for Design of Embedded Systems under Power and Latency Constraints

Alex Doboli
VLSI Systems Design Laboratory
Department of Electrical and Computer Engineering
State University of New York (SUNY) at Stony Brook
Stony Brook, NY, 11794-2350
adoboli@ece.sunysb.edu

## Abstract

*This paper presents an integrated approach to hardware-software co-synthesis and HLS for design of low-power embedded systems. The main motivation for this work is that fine trade-offs between latency and power can be explored at the system level only with a detailed knowledge of used hardware resources. Integrated method was realized as a simulated annealing based* **solution-space exploration***. Exploration is guided by* **Performance Models***, that exactly capture the relationship between performances i.e. power consumption and latency and design decisions i.e. binding and scheduling. The proposed approach permits not only a more accurate latency and power estimation but also the exposure of RTL-level design decisions at the system level. As a result, more effective power-latency trade-offs are possible during co-synthesis as compared to traditional task-level methods.*

## 1. Introduction

With the impending technological revolution due to mobile computing, there is a growing demand for low-power embedded systems. Nevertheless, success of mobile systems such as cell phones, digital cameras, personal communicators etc in real-life crucially depends on the amount of energy they consume. Reducing power consumption of embedded systems lengthens the time period between two successive re-chargings of their batteries. An enhanced mobility range facilitates the employment of mobile embedded systems for a wider class of applications.

System level design tasks (including hardware-software partitioning, task scheduling, communication synthesis, etc) are critical for building high-quality low-power systems [2] [11] [17]. It is well known that by the time the RTL design of a system is finished, 80% of its power consumption is already committed. Moreover, hardware-software co-synthesis must be conducted with a detailed knowledge of the used hardware resources. This enables precise power estimations because power consumption is tightly related to hardware specifics. Hardware abstraction at a high-level is not practical for low-power co-synthesis as it offers vague power estimations [2] [5]. Finally, power minimization conflicts with latency constraints as time-critical systems dissipate more power [3]. An important conclusion that emerges from these observations is that fine trade-offs between latency and power have to be explored at the system level with a detailed knowledge of the used hardware resources. Therefore, we believe that hardware-software co-synthesis for low-power embedded systems should be performed in tight integration with high level synthesis (HLS) under the guidance of an accurate timing and power evaluation (estimation) mechanism. This is the topic of this paper.

Many of the existing hardware-software co-synthesis methodologies [1] [8] [18] contemplate an abstract modeling of hardware by considering generic properties such as its capacity to concurrently execute operations and to be shared by similar operations. As a result, co-synthesis and HLS are successive and independent activities. This is a valid assumption if objective is speeding-up process execution. The link between co-synthesis and HLS, however, has to be much stronger if fine performance trade-offs between conflicting performance constraints i.e. latency and power are examined. Also, system functionality is traditionally described as a task graph with data dependencies, only [4] [8] [10] [18]. This implies that co-synthesis does not address any conditional behavior expressed with control dependencies. Nevertheless, handling control dependencies enables more precise performance estimations, more effective design decisions i.e. scheduling and ultimately, better quality solutions [7]. Recently, Eles *et al* [7] and Strehl *et al* [16] propose scheduling techniques that consider control dependencies besides data dependencies. Targeted goal is sys-

tem latency minimization. Authors do no indicate how their methods could be extended to address power reduction.

Recent work discusses hardware-software co-synthesis for low-power embedded systems. Henkel [10] suggests a hardware-software partitioning method for low-power systems. Partitioning is at the level of operation (instruction) clusters. After cluster scheduling, clusters with a high utilization rate (thus, with less wasted energy) are moved to hardware. Still, there is no guarantee that a hardware-software partition actually meets imposed power constraints because accurate estimations are possible only at the RTL level, hence, after HLS. A tighter integration of co-synthesis and HLS is probably needed to surmount this limitation. Shin *et al* [14] describe a power efficient scheduling method that exploits slack times. Dave *et al* [4] propose a low-power co-synthesis method that includes allocation, scheduling and performance estimation. Their method is at the task level so that estimation is limited to average power. Also, authors do not consider any low-power specific design issues [12] i.e. temporary shut-down of unused resources.

This paper presents an integrated approach to hardware-software co-synthesis and HLS for design of low-power embedded systems. Goal is to find the hardware-software implementation of a system, that minimizes overall power consumption while satisfying a global latency constraint. All available hardware resources are assumed to be known, including general-purpose processing elements - PEs for executing software and functional units - FUs (adders, multipliers, etc) for hardware. Integrated co-synthesis and HLS is accomplished by describing systems as hierarchical graphs. Graphs include *operation clusters* for expressing software and *operations* for hardware. Data and control dependencies exist among the nodes. To the best of our knowledge, addressing control dependencies during co-synthesis of low-power systems is a novel research topic.

Integrated method was realized as a simulated annealing (SA) [13] based **solution-space exploration**. During exploration, operation clusters are partitioned and scheduled to PEs (as in traditional co-synthesis) and operations are bound and scheduled to FUs (like in HLS). Low-power oriented aspects such as PE shut-down are then contemplated for each partitioned (bound) and scheduled solution. Exploration is guided by **Performance Model** (PM) evaluation. PMs are a key component of the co-synthesis approach. PMs are graphs that exactly capture the relationship between latency and power consumption and design decisions i.e. binding and scheduling. PM also reflect data and control dependencies present in a system graph. We believe that PM are more suggestive and precise than traditional performance modeling through metrics such as task (operation) priorities, forces, degree of concurrency, utilization rate [6] [10].

The integrated approach to co-synthesis and HLS is an important contribution of this paper not only because it permits a more accurate latency and power estimations but also because it exposes RTL-level design decisions at the system level. As a result, more effective performance trade-offs are possible during co-synthesis.

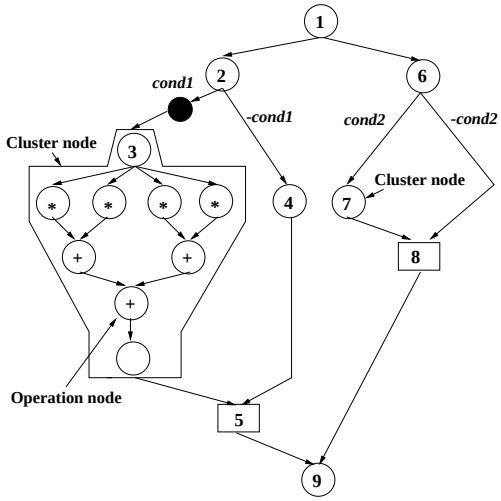


Figure 1. **Example of Hierarchical Data and Control Dependency Graph**

The paper is organized as follows. Section 2 discusses system representation for the integrated approach. Section 3 presents performance modeling for co-synthesis. Integrated methodology for co-synthesis and HLS is discussed in Section 4. Results to illustrate the efficiency of our approach are in Section 5. Finally, we putforth our conclusions.

## 2. System Representation

A *Hierarchical Data and Control Dependency Graph* (HDCG) is the input for our integrated co-synthesis and HLS methodology. HDCG offers a dual perspective on a system: a task-level description that is used for co-synthesis and an operation-level representation that is needed for HLS. Figure 1 presents an HDCG example.

HDCG is a hybrid representation with nodes of two types: operation nodes (ON) and cluster nodes (CN).

- *Operation nodes* are needed to conduct HLS activities such as operation binding and scheduling. They denote an atomic processing such as addition, multiplication, comparison, etc. Communications are described as special ONs that are not part of any CN. Communication ONs are shown as black bubbles in the figure.
- *Cluster nodes* are useful to perform hardware-software co-synthesis, including partitioning and task scheduling. Similar to [10], we assume that clusters are compiled from system specifications (i.e. VHDL specifications) and represent loops, if-then-else constructs and functions. Each CN is a polar sub-graph that is built of

ONs. For example, the detailed operation structure of cluster node 3 is depicted in Figure 1.

Both CNs and ONs are linked through data and control dependencies. Data dependencies are directed arcs that express an imposed execution order: a target can start its execution only after its source is completed. In order to express control dependencies, some arcs are annotated with condition values. In Figure 1, conditions are depicted in italics. Node 2 computes condition *cond1*, and depending on its value, *one* of its out-branches is selected. For a *true* value, the graph traversal activates the communication between nodes 2 and 3, followed by starting operations pertaining to node 3. Node 4 is executed for a *false* condition value (*false* is indicated by - *cond1*). Nodes 5 and 8 are fictitious *join nodes*, which are created for the purpose of outlining control structures. Each ON and CN is annotated with its execution times and power consumptions specific to all hardware resources that can implement it. The technique suggested by Gupta *et al* [9] can be used for ON characterization and the method by Tiwari *et al* [17] can be applied for CN characterization. Any dependency of power consumption on input data [12] [11] is addressed at this level.

The execution semantics of our model assumes that each CN is executed not more than once for each traversal of the graph. ONs can be executed multiple times with the restriction that the number of iterations is known.

Even though system functionality can be fully represented using only ONs, CNs prevent the unnecessary growth of solution-space and hence, a very difficult exploration process. If realized in software, CN performances for latency and power consumption can be estimated with a fair precision using data profiling and memory models for CPU, cache, memory and communication units [11] [17]. This permits that hardware-software partitioning and system-level scheduling is performed using clusters. A second advantage of representing software as CNs is that compiler optimizations i.e. loop unrolling and tilling can be contemplated during performance estimation. Finally, CNs offer a clear perspective on system-level issues such as inter-task communications. This is beneficial for co-synthesis tasks i.e. communication synthesis [7]. As explained in the introductory part, ONs are still required for accurate timing and power consumption estimation of the hardware component.

## 3. Performance Modeling

System descriptions as HDCGs represent functionality aspects but do not indicate any timing and power characteristics of alternative implementations. However, for selecting the best solution found during co-synthesis, a technique is needed for relating performances to HDCG characteristics and synthesis decisions. We propose Performance Models (PM) - a representation for expressing timing and power

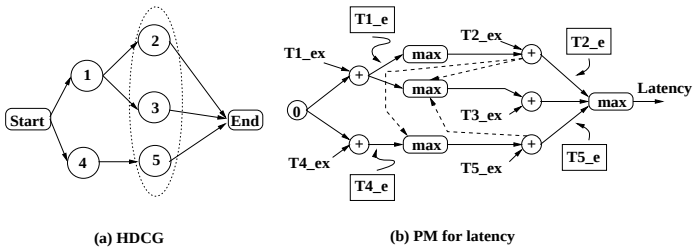


Figure 2. **Latency Performance Model for HDCG**

consumption. This section presents main characteristics and rules for automatically building PMs.

PM representation includes two parts: a *constant part* that presents HDCG characteristics and a *variable part* that reflects design decisions taken during co-synthesis. For example, a certain scheduling order of CNs mapped to the same PE is correspondingly described by the variable part of a PM. Figure 2(a) shows an HDCG. Operation nodes 2, 5 and 3 are executed in this order on a shared FU. Figure 2(b) depicts the PM of the HDCG for latency. Following are the main characteristics of the PM:

- The PM has a starting node labeled as *0*. This node indicates that all observed performances (latency in our case) are set to value 0.
- The *constant part* of the PM is represented in the figure as nodes and solid edges. It depends on invariant HDCG characteristics i.e. data and control dependencies and the semantics of performances with respect to observed metrics. For example, *max* and *addition* nodes are used in the PM in Figure 2(b) to express ON (CN) start and end times. *Max* nodes describe that an ON (CN) can not start earlier than the moment when all its predecessors were finished. Outputs of *max* nodes indicate the starting time of their corresponding ON (CN). Addition nodes describe that the end time $Ti\_e$ of ON $op_i$ is the sum between its start time and its execution time $Ti\_ex$.
- The *variable part* of the PMs shows the influence of scheduling decisions for ONs 2, 3 and 5 on the resulting latency. For instance, the execution order 2, 5, 3 is represented in the figure by dotted arcs between addition nodes that calculate end times for ON (CN) and *max* nodes that characterize starting times. Other ON schedulings are easily captured by the PM by accordingly changing the orientation of dotted arcs.
- The latency of a certain implementation solution can be precisely determined by numerically evaluating its PM.

The remaining part of this section presents our rules for automatically building PMs that express HDCG characteristics and synthesis decisions.

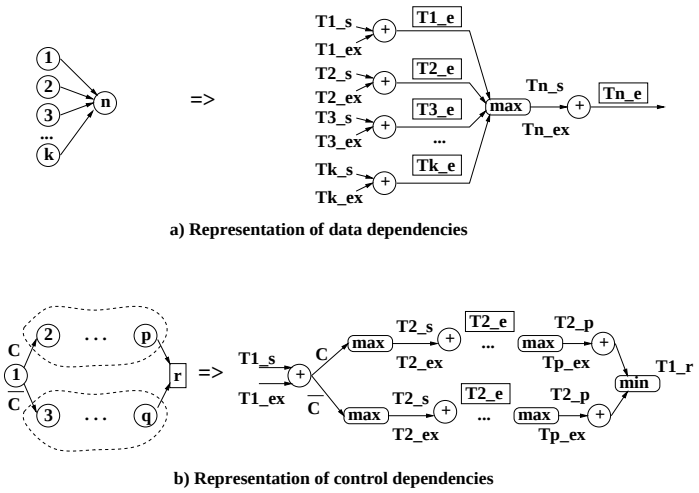


Figure 3. **Modeling of data and control dependencies**

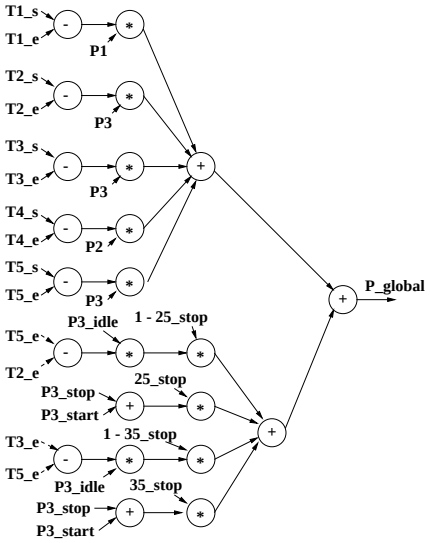


Figure 4. **Modeling of power consumption**

**Modeling of Data Dependencies**

Data dependencies introduce a required sequencing of HDCG node executions. For example, in Figure 3(a) node $n$ can be executed only after all its predecessors 1, 2, ..., $k$ are performed. The right part of Figure 3(a) depicts the PM that expresses these execution order requirements. For each operation, addition nodes indicate how node end times $Ti\_e, (i = 1, k)$ are computed depending on node starting times $Ti\_s$ and node execution times $Ti\_ex$. The *max* node models the constraint that node $n$ starts only after all its predecessors are terminated.

**Modeling of Control Dependencies**

Control dependencies describe conditional node executions in an HDCG. In Figure 3(b), for example, if the value of condition $C$ is true then nodes 2, ..., $p$ are performed. Nodes 3, ..., $q$ are executed for a false condition $C$. Join node $r$ is introduced in the HDCG to point the end of the conditional construct. The right part of Figure 3(b) introduces the PM generated for the HDCG. Similar to data dependencies, control dependencies define an execution ordering among nodes. For example, nodes 2, 3, etc can not start their executions until node 1 completed. Analogous to data dependencies, the built PM reflects this requirement through *max* nodes. Conditional execution of nodes is presented in the PMs by annotating edges with condition values. Following semantics is applied when numerically evaluating such PMs: edges annotated with a true condition will propagate the numerical values that result from evaluating the DAGs for PMs. Edges annotated with a false condition will propagate the value *infinite*. The *min* node in a PM corresponds to node $r$ in the HDCG. It is used to eliminate infinite values due to the non-selected branches. This permits calculating the correct time values for operations that succeed node *r*.

**Modeling of Power Consumption**

Global power consumption is the sum between the power required for executing ONs (CNs) on their FUs (PEs) and the power consumed during the idle periods of hardware resources [11] [12]. Power during idle periods can be diminished by shutting down PEs (FUs) [12]. However, when a PE (FU) is turned-off, the power needed for shut-down and re-start has to be added. Moreover, the latency PM has to be also updated to consider the times it takes for PE (FU) shut-down and re-start.

We discuss the total power consumption PM for the example in Figure 2(a). Similar PMs can be built to express peak power consumption as well. We consider that nodes execute in the order 2, 5 and 3 on the shared resource. Figure 4 shows the global power consumption PM for this example. The top part of the PM calculates the sum $\sum_{i=1}^{5} Pi \times (Ti\_e - Ti\_s)$ that represents the consumed power for executing the five nodes. $Pi, i = 1, 3$ is the average power consumption of a PE (FU) in a clock cycle. Techniques by Gupta [9] and Tiwari [17] can be used to find values for $Pi$. The bottom part of the PM describes power during idle periods (values $Pi\_idle$) and power for shut-down ($Pi\_stop$) and re-start ($Pi\_start$) of PEs (FUs).

Variables $25\_stop$ and $35\_stop$ model the shut-down of the shared hardware resource. Variable $25\_stop$ has value 1 if the hardware unit is shut-down between the end of node 2 and the start of node 5. Otherwise, variable $25\_stop$ has value 0. Variable $35\_stop$ has a similar definition. Section 4 explains how values for these variables are found as a part of the integrated co-synthesis process. If resources are turned-off then no idle power is consumed, otherwise idle power has to be added. Using variables $25\_stop$ and $35\_stop$, PM in Figure 4 accurately describes these requirements for eval-

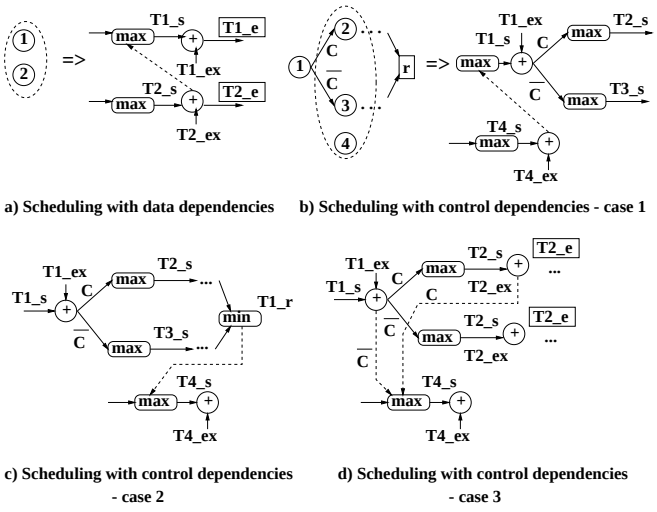


Figure 5. **Representation of scheduling decisions**

uation of consumed power.

**Modeling of Cluster Partitioning and Operation Binding**

We assume that the resource set $\mathcal{R}$ (including PEs and FUs) available for implementing an HDCG is given. Thus, the number and type of hardware resources that can realize each CN and ON in the HDCG is known. $\mathcal{R}\_op_i$ is the subset of set $\mathcal{R}$ to which node $op_i$ can be mapped. Also, we define function

$$\mathcal{R}esource : \mathcal{N}odes\ in\ HDCG \rightarrow \mathcal{R}$$

that presents the resource to which each node is bound. The goal of cluster partitioning and operation binding is finding the definition of function $\mathcal{R}esource$ (thus, its values for all nodes $op_i$). Obviously, binding has to be done such that $\mathcal{R}esource(op_i) \in \mathcal{R}\_op_i$. A consequence of cluster partitioning and operation binding is that any attribute value (in our case, execution time and power consumption) of node $op_i$ that depends on the resource type becomes well defined. For example, let's assume that node execution time $Ti\_ex$ varies with resource type. After binding, execution time becomes known and its value is updated in the PM.

**Modeling of Scheduling**

Given a HDCG and a node binding to hardware resources, scheduling decides the execution order of operations on shared resources. Depending on taken scheduling decisions, different node sequences and different timing attributes (i.e. starting time, end time) result for nodes in a HDCG. These correlations between scheduling decisions and performance metrics i.e. timing attributes have to be captured by a PM. If only data dependencies occur then the imposed ordering can be easily modeled by introducing an arc from the PM node for the end time of the first HDCG node to the PM node for the starting time of the second HDCG node. For example, in Figure 5(a) node 1 and 2 share the same resource and the scheduler decides that node 2 is executed before node 1. Accordingly, the PM is updated by introducing the dotted arc that imposes that node 1 starts only after node 2 ends. The dotted arc pertains to the variable part of the PM. Different scheduling decisions can be easily captured by PMs by simply changing the orientation of dotted arcs.

Representing scheduling decisions in the presence of control dependencies is more difficult due to the uncertain character of control dependencies. Difficulty increases if scheduling is performed across control constructs. The main challenge is to generate node schedules that maintain HDCG semantics with respect to three criteria: (1) respecting the execution order defined by data and control dependencies, (2) maintaining conditional node execution as defined by a HDCG (i.e. if a condition value is true then only nodes from the true branch must be executed) and (3) executing at most once each CN in the HDCG. The first two requirements were already captured by the modeling of data and control dependencies in a PM. To exemplify the third requirement, let's assume that for the graph in Figure 5(b) a schedule is produced such that node 4 is scheduled before node 1 if condition $C$ is true, and after node 1 if condition $C$ is false. Nodes 2, 3 and 4 share the same resource. This situation occurs if the branch for condition $C$ true is very long and the branch for condition $C$ false is very short compared to path to which node 4 pertains. However, this schedule is incorrect because if condition $C$ is false then node 4 will be executed twice, both before and after node 1.

For modeling the third requirement of scheduling correctness, we identified three different cases that are possible in a PM:

- **Case 1**: Node 4 is executed before node 1. Its scheduling does not depend on the value of condition $C$. Thus, there is no risk that node 4 is executed multiple times for a single execution of the HDCG. Figure 5(b) depicts this situation and the dotted arc is enforces so that node 1 starts only after node 4 terminates.
- **Case 2**: Node 4 executes after node $r$. Its scheduling time depends on the value of condition $C$. However, the value of condition $C$ is already known by the time node 4 starts. Thus, there is no danger of executing node 4 multiple times. This situation is reflected in Figure 5(c) by the dotted arc between the *min* node for node *r* and the *max* node for node 4.
- **Case 3**: For a given condition value (i.e. condition $C$ is true), node 4 is scheduled to execute after node 1 but before node $r$ starts. To maintain scheduling correctness, for the opposite value of condition $C$, it is also required that node 4 executes only after node 1 ends. Figure 5(d) depicts this case. Two dotted arcs are introduced so that node 4 starts after node 2 if condition $C$ is true and after node 1 if condition $C$ is false.

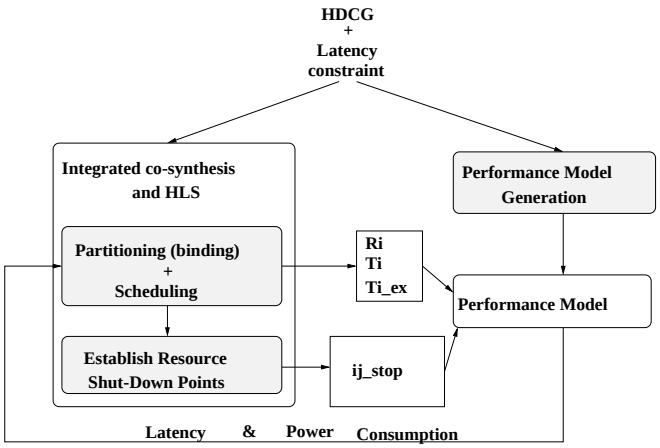


Figure 6. **Integrated approach to co-synthesis and HLS**

## 4. Integrated Approach to Hardware-Software Co-synthesis and HLS

**Co-synthesis Methodology**

The integrated approach to hardware-software co-synthesis and HLS is presented in Figure 6. Inputs to co-synthesis are an HDCG and latency constraint. The number and type of available hardware resources, including PEs and FUs, is also given. Co-synthesis problem is finding what resource executes each HDCG node and deciding the scheduling order of nodes so that power consumption is minimized while latency is below the given constraint.

The method includes two steps. First, **Performance Models (PM) are generated** for an HDCG. Rules for PM set-up were discussed in Section 3. Second step is the **integrated co-synthesis and HLS**, which was defined as an **optimization** algorithm. As explained in Section 3, for each CN (ON), attributes $R_i$ (hardware resource that executes the node) and $T_i$ (starting time of node execution) constitute unknowns for the co-synthesis process. CN partitioning to PEs and ON binding to FUs is modeled by unknowns $R_i$. CN and ON scheduling is described by unknowns $T_i$. Possible values for unknowns $R_i$ and $T_i$ are searched during exploration. Power consumption and latency are computed by numerically instantiating all node characteristics $R_i$, $T_i$ and $T_{i\_ex}$ and then evaluating their PMs.

Unknowns $ij\_stop$ model resource shut-down during the idle period defined by the end of node $i$ and the start of node $j$. Values for these variables are established by a greedy algorithm. The algorithm examines all possibilities of turning-off hardware resources and determines those that reduce power consumption (sum of shut-down and restart power is less than idle power). Then, in the order of their power savings, it retains those shut-down decisions that do not violate any timing constraints.

**Co-synthesis by Performance Model Optimization**

This section describes our approach for integrated co-synthesis and HLS as an optimization-based design-space exploration algorithm. Exploration was realized using the very popular simulated-annealing optimization algorithm (SA) [13]. The algorithm examines the quality of numerous partitioning (binding) and scheduling solutions by numerically evaluating PMs for power consumption and latency.

The SA algorithm iteratively selects a new point from the neighborhood of the current solution. We defined neighborhood as the set of points that (1) differ from the current solution by the execution order of *one* pair of nodes that share a hardware resource or (2) the resource binding of *one* node. PMs are updated for the newly selected solution and then used for numerically evaluating the resulting power consumption and latency. If the resulting solution has a better quality than the current then the new solution is unconditionally accepted. A worse solution is accepted with a higher probability at the beginning of exploration. Acceptance probability decreases as exploration progresses. Initial solution is obtained by uniformly distributing nodes to resources, and then scheduling nodes using list-scheduling. Critical path was the priority function for list-scheduling.

Partitioning (binding) and scheduling steps are executed with different probabilities. The reason is that multiple valid schedules are possible for each resource partitioning (binding) decision. A small probability $p_1$ is used to select a partitioning step, that moves a cluster from a PE to another PE or to hardware. A probability $p_2$ ($p_2 > p_1$) binds an ON to another FU. The reason for $p_2$ being greater than $p_1$ is that multiple HLS solutions are possible for each partitioning of clusters to FUs. Finally, a probability 1 - ($p_1 + p_2$) decides a scheduling action. This strategy emulates a hierarchical exploration process because for each new partition (binding) there are $\frac{1-(p_1+p_2)}{p_1+p_2}$ analyzed schedules. For example, if $p_1$ = 0.01 and $p_2$ = 0.1 then on the average, 8 schedules are examined for each partition (binding). If the execution order of a node pair is modified then the algorithm also verifies that the new ordering is feasible. This means that no cycles can occur in the updated PMs.

**Establishing Resource Shut-Down Points**

We propose a greedy algorithm for deciding what hardware resource shut-downs result in power savings and do not violate the imposed latency constraint. As discussed in Section 3, variable $ij\_stop$ models the decision to stop the hardware resource $Resource(i)$ shared by nodes $i$ and $j$ between the end of node $i$ and start of node $j$. These variables are affected only when a new scheduling decision is taken because then resource idle times are modified. More precisely, these variables correspond to the nodes that have their starting times changed by the scheduling decision. Hence, the algorithm for establishing resource shut-down points has to be

**Input** $S$
**for** $\forall$ $ij\_stop$ $\in$ $S$ **do**
Calculate power saving if resource $Resource(i)$ is shut-down between the end of node $i$ and the start of node $j$;
**if** power saving results **then**
Introduce $ij\_stop$ in set $P$;

**for** $forall$ $ij\_stop$ $\in$ $P$ in the order of their power saving **do**
**if** latency constraint is not violated by $ij\_stop$ **then**
$Resource(i)$ is shut-down between end of node $i$ and start of node $j$;

Figure 7. **Algorithm for establishing resource shut-down points**

called after each scheduling decision of the SA exploration.

Figure 7 presents the suggested algorithm. As motivated in Section 3, variables $ij\_stop$ are part of the PM for power consumption. Lets assume that the set $S$ includes all variables $ij\_stop$ that were affected by a scheduling decision. The algorithm considers each of the variables and checks if any power saving results by shutting-down the resource. Power savings are calculated by using PMs. Resource shut-downs that offer the highest power savings and do not violate the fixed latency constrained are retained in the final solution.

## 5. Experimental Results

Goal of our experimental part was to evaluate the effectiveness of the integrated approach for hardware-software co-synthesis and HLS as compared to traditional task-level co-synthesis. We analyzed the resulting power consumption of solutions found by the proposed method and a task-level method for different latency constraints and distinct hardware resources. Both loose and tight latency constraints were considered. We used two examples: the video coding algorithm H261 (video) [1] and the 4x4 determinant calculator (4x4 det) [15].

Both integrated and task-level co-synthesis methods were realized as simulated-annealing (SA) based exploration algorithms. Hence, we excluded the possibility that observed differences in solution quality were due to different exploration techniques. SA was run with the following parameters: initial temperature was 80000, temperature length was set to 200 and cooling schedule was 0.8. The SA exploration finished when 10000 solutions were analyzed or when no improvement was observed for the last 5000 moves.

Table 1 presents the results of the comparison between the integrated and the task-level co-synthesis methods. Column 2 are the latency constraints posed on the designs. The assumed hardware resource set is detailed in Column 3. At the task level, the architecture includes general purpose processing elements (PE) and an ASIC (HW) modeled abstractly. For integrated approach, Column 3 presents the number of PEs and FUs (adders, multipliers) that were available for each example. Resulting power consumption for best solutions found during exploration is shown in Column 4. Column 5 presents the latency of the solution. Relative power saving of the integrated approach with respect to the task-level method is reported in Column 6. Finally, execution times of the SA-based exploration are indicated in Column 7.

The results in Table 1 show that the integrated co-synthesis and HLS approach offers significant power savings as compared to the task-level co-synthesis method. Power savings ranged from 1.6% to 27% but were, in general higher than 10%. Obtained solutions had also a smaller latency for the integrated method. The reason for the proposed integrated method performing better may be attributed to a more efficient utilization of the available hardware resources. Better hardware utilization is due to superior FU sharing that can be revealed in the integrated method. As a result, resource idle periods are shorter so that less energy is wasted. Also, more functionality can be shifted towards slower but less power consuming resources without violating the imposed latency constraints. As a conclusion, the integrated approach offers more effective latency-power consumption trade-offs than the task-level method. An interesting observation was that relative power saving tends to decrease as imposed latency relaxes. This can be explained by the fact that for loose latency constraints functionality is mostly placed into software. In our experiments, power consumption was less in software than in hardware.

## 6. Conclusion

In this paper, we presented an integrated approach to hardware-software co-synthesis and HLS for design of low-power embedded systems. Goal is to find the hardware-software implementation of a system, that minimizes overall power consumption while satisfying a global latency constraint.

Integrated co-synthesis and HLS is accomplished by describing systems as Hierarchical Data and Control Dependency Graphs (HDCG) that include *operation clusters* for software and *operations* for hardware. Integrated approach was realized as a simulated annealing (SA) based solution-space exploration. Exploration is guided by Performance Model (PM) evaluation. PMs are graphs that express the relationship between latency and power consumption and design decisions (i.e. binding and scheduling) and HDCG characteristics (i.e. data and control dependencies). The in-

| Example (1) | Latency Constraint (msec) (2) | Resource Set (3) | Power Consumption (W) (4) | Latency (msec) (5) | Power Saving (%) (6) | Execution Time (sec) (7) |
|---|---|---|---|---|---|---|
| video | 2000 | System-level: 2 PE + 1 HW | 2.72 | 1864 | - | 275 |
| video | 4000 | System-level: 2 PE + 1 HW | 2.36 | 3486 | - | 214 |
| video | 20000 | System-level: 2 PE + 1 HW | 2.175 | 9168 | - | 225 |
| video | 2000 | 2 PE + 2 adder + 1 mult. | 2.42 | 1744 | 11 | 222 |
| video | 4000 | 2 PE + 2 adder + 1 mult. | 2.20 | 3396 | 6.7 | 241 |
| video | 20000 | 2 PE + 2 adder + 1 mult. | 2.14 | 16146 | 1.6 | 200 |
| video | 2000 | 2 PE + 2 adder + 2 mult. | 2.26 | 1744 | 18 | 218 |
| video | 4000 | 2 PE + 2 adder + 2 mult. | 1.97 | 3024 | 16 | 204 |
| video | 20000 | 2 PE + 2 adder + 2 mult. | 1.94 | 15034 | 10 | 205 |
| video | 2000 | 4 PE + 4 adder + 4 mult. | 1.98 | 1664 | 27 | 240 |
| video | 4000 | 2 PE + 4 adder + 4 mult. | 1.79 | 2944 | 24 | 205 |
| video | 20000 | 2 PE + 4 adder + 4 mult. | 1.65 | 8968 | 19.5 | 230 |
| 4x4 det | 300 | System-level: 1PE + 1 HW | 4.0 | 150 | - | 24.0 |
| 4x4 det | 1000 | System-level: 1PE + 1 HW | 3.0 | 900 | - | 20.0 |
| 4x4 det | 300 | 1 PE + 1 adder + 1 mult. | 3.69 | 110 | 7.75 | 105 |
| 4x4 det | 1000 | 1 PE + 1 adder + 1 mult. | 2.78 | 900 | 7.2 | 130 |
| 4x4 det | 300 | 1 PE + 1 adder + 2 mult. | 3.49 | 70 | 12.4 | 110 |
| 4x4 det | 1000 | 1 PE + 1 adder + 2 mult. | 2.68 | 900 | 10.3 | 112 |
| 4x4 det | 300 | 1 PE + 3 adder + 3 mult. | 3.40 | 70 | 15 | 140 |
| 4x4 det | 1000 | 1 PE + 3 adder + 3 mult. | 2.57 | 900 | 17 | 142 |

Table 1. **Experimental results for integrated co-synthesis and HLS**

tegrated approach to co-synthesis and HLS permits not only more accurate latency and power estimations but also it exposes RTL-level design decisions at the system level. Our experiments showed that more effective performance tradeoffs are possible than for task-level co-synthesis.

## References


[1] A. Bender *et al*, "MILP Based Task Mapping for Heterogeneous Multiprocessor Systems", *Proc. of EDAC*, 1996, pp.283-288.

[2] L. Benini *et al*, "System-Level Power Optimization: Techniques and Tools", *Proc. of ISPLED*, 1999, pp.283-288.

[3] M. Borah *et al*, "High-throughput and Low-power DSP using clocked CMOS Circuitry", *Proc. of ISPLED*, 1995.

[4] B. Dave *et al*, "COSYN: Hardware-Software Co-Synthesis of Embedded Systems", *Proc. of DAC*, 1997, pp.703-708.

[5] W. Fornaciari *et al*, "Power Estimation for Embedded Systems: A Hardware/Software Codesign Approach", *IEEE Trans. on VLSI*, June 1998.

[6] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, 1994.

[7] P. Eles *et al*, "Scheduling with Bus Access Optimization for Distributed Embedded Systems", *IEEE Trans. on VLSI*, November, 2000.

[8] R. Gupta *et al*, "Hardware-Software Cosynthesis for Digital Systems", *IEEE Design & Test of Computers*, September 1992, pp.29-40.

[9] S. Gupta *et al*, "Power Macro-Models for DSP Blocks with Applications to High-Level Synthesis", *Proc. of ISPLED*, 1999, pp.103-105.

[10] J. Henkel, "A Low Power Hardware/Software Partitioning Approach for Core-based Embedded Systems", *Proc. of DAC*, 1999, pp.122-127.

[11] P. Landman, "High-Level Power Estimation", *Proc. of ISPLED*, 1996.

[12] M. Pedram, "Power Minimization in IC Design: Principles and Applications", *ACM Trans. on DAES*, Vol.1, No.1, 1996, pp.3-56.

[13] C. Reeves *et al*, "Modern Heuristic Techniques for Combinatorial Problems", *J. Wiley*, 1993.

[14] Y. Shin *et al*, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems", *Proc. of DAC*, 1999, pp.134-139.

[15] V. Srinivasan *et al*, "Hardware Software Partitioning with Integrated Hardware Design Space Exploration", *Proc. of DATE*, 1998, pp.28-35.

[16] K. Strehl *et al*, "Scheduling Hardware/Software Systems Using Symbolic Techniques", *Proc. of CODES/CACHE*, 1999.

[17] V. Tiwari *et al*, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization", *IEEE Trans. on VLSI*, Vol.2, No.4, December 1994, pp.437-445.

[18] T. Y. Yen *et al*, "Hardware-Software Co-synthesis of Distributed Embedded Systems", Kluwer, 1997.