An Approach to Incremental Design of Distributed Embedded Systems

Paul Pop, Petru Eles, Traian Pop, Zebo Peng
Dept. of Computer and Information Science, Linköping University
{paupo, petel, trapo, zebpe}@ida.liu.se

ABSTRACT

In this paper we present an approach to incremental design of distributed embedded systems for hard real-time applications. We start from an already existing system running a set of applications and the design problem is to implement new functionality so that the already running applications are not disturbed and there is a good chance that, later, new functionality can easily be added to the resulted system. The mapping and scheduling problem are considered in the context of a realistic communication model based on a TDMA protocol.

1. INTRODUCTION

In this paper we concentrate on aspects related to the synthesis of distributed embedded systems for hard real-time applications. There are several complex design steps to be considered during the development of such a system: the underlying architecture has to be *allocated* (which implies the allocation of components like processors, memories, and busses together with the decision on a certain interconnection topology), tasks and communication channels have to be *mapped* on the architecture, and all the activities in the system have to be *scheduled*. The design process usually implies an iterative execution of these steps until a solution is found such that the resulted system satisfies certain design constraints [7].

Several notable results have been reported, aimed at supporting the designer with methodologies and tools during the hardware/ software cosynthesis of embedded systems. Initially, researchers have considered architectures consisting of a single programmable processor and an ASIC. Their goal was to partition the application between the hardware and software domain, such that performance constraints are satisfied while the total hardware cost is kept at a minimum [8, 6, 10, 4]. Currently, similar architectures are becoming increasingly interesting, with the ASIC replaced by a dynamically reconfigurable hardware coprocessor [14].

Distributed embedded systems with multiple processing elements are becoming common in various application areas ranging from multimedia to robotics, industrial control, and automotive electronics. In [19] allocation, mapping, and scheduling are formulated as a mixed integer linear programming (MILP) problem. A disadvantage of this approach is the complexity of solving the MILP model. Therefore, alternative problem formulations and solutions based on efficient heuristics have been proposed [21, 13, 22, 3, 1, 2].

Although much of the above work is dedicated to specific aspects of distributed systems, researchers have often ignored or very much simplified issues concerning the communication infrastructure. One notable exception is [20], in which system synthesis is discussed in the context of a distributed architecture based on arbitrated busses. Many efforts dedicated to communication synthesis have concentrated on the synthesis support for the communication infrastructure but without considering hard real-time constraints and system level scheduling aspects [11, 16, 17].

Another characteristic of research efforts concerning the codesign of embedded systems is that authors concentrate on the design, from scratch, of a new system optimized for a particular application. For many application areas, however, such a situation is extremely uncommon and only rarely appears in design practice. It is much more likely that one has to start from an already existing system running a certain application and the design problem is to implement new functionality on this system. In such a context it is very important to operate no (or as few as possible) modifications to the already running application. The main reason for this is to avoid unnecessarily large design and testing times. Performing modifications on the (potentially large) existing application increases design time and, even more, testing time (instead of only testing the newly implemented functionality, the old application, or at least a part of it, has also to be retested). However, this is not the only aspect to be considered. Such an incremental design process, in which a design is periodically upgraded with new features, is going through several iterations. Therefore, after new functionality has been implemented, the resulting system has to be structured such that additional functionality, later to be mapped, can easily be accommodated.

The contribution of this paper is twofold. First, we consider mapping and scheduling for hard real-time embedded systems in the context of a realistic communication model. Because our focus is on hard real-time safety critical systems, communication is based on a time division multiple access (TDMA) protocol as recommended for applications in areas like, for example, automotive electronics [12]. For the same reason we use a non-preemptive static task scheduling scheme. We accurately take into consideration overheads due to communication and consider, during the mapping and scheduling process, the particular requirements of the communication protocol.

As our main contribution, we have considered, for the first time to our knowledge, the design of distributed embedded systems in the context of an incremental design process as outlined above. This implies that we perform mapping and scheduling of new functionality so that certain design constraints are satisfied and:

a. the already running functionality is not disturbed;

b. there is a good chance that, later, new functionality can easily be mapped on the resulted system.

Supporting such a design process is of critical importance for current and future industrial practice, as the time interval between successive generations of a product is continuously decreasing, while the complexity due to increased sophistication of new functionality is growing rapidly.

The paper is divided into 6 sections. The next section presents the hardware architecture, the process model and a brief introduction of the mapping problem. Section 3 presents the detailed problem formulation and the quality metrics we have introduced. Our mapping strategies are outlined in Section 4, and the experimental results are presented in Section 5. The last section presents our conclusions.

2. PRELIMINARIES

2.1 System Architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a CPU, a communication controller, a local memory and an I/O interface to sensors and actuators.

Communication between nodes is based on a TDMA protocol like, for example, the TTP [12] which integrates a set of services necessary for fault-tolerant real-time systems.

The communication channel is a broadcast channel, so a message sent by a node is received by all the other nodes. The bus access scheme is TDMA (Figure 1): each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot, a node can send several messages packaged in a frame. A

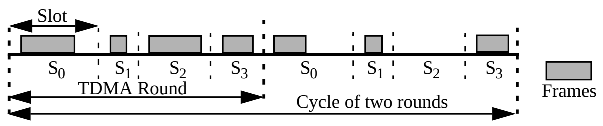


Figure 1. Buss Access Scheme

sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically.

Every node has a communication controller that implements the protocol services, and runs independently of the node's CPU. The communication controller provides each CPU with a timer interrupt based on a local clock, synchronized with the local clocks of the other nodes. Thus, a global time-base of known precision is created throughout the system.

We have designed a software architecture which runs on the CPU in each node, and which has a real-time kernel as its main component. Each kernel has a schedule table that contains all the information needed to take decisions on activation of processes and transmission of messages, based on the current value of time. For more details about the software architecture and the message passing mechanism the reader is referred to [18].

2.2 The Process Graph

As an abstract model for system representation we use a directed, acyclic, polar graph $G(V, E)$. Each node $P_i \in V$ represents one process. An edge $e_{ij} \in E$ from P_i to P_j indicates that the output of P_i is the input of P_j. A process can be activated after all its inputs have arrived and it issues its outputs when it terminates. Once activated, a process executes until it completes. Each process graph G is characterized by its period T_G and its deadline $D_G \leq T_G$. The functionality of an application is described as a set of process graphs.

2.3 Application Mapping

Considering a system architecture like the one presented in section 2.1, the mapping of a process graph $G(V, E)$ is given by a function M: $V \rightarrow PE$, where $PE = \{N_1, N_2, .., N_{npe}\}$ is the set of nodes (processing elements). For a process $P_i \in V$, $M(P_i)$ is the node to which P_i is assigned for execution. Each process P_i can potentially be mapped on several nodes. Let $\mathcal{N}_{Pi} \subseteq PE$ be the set of nodes to which P_i can potentially be mapped. For each $N_i \in \mathcal{N}_{Pi}$, we know the worst case execution time $t_{P_i}^{N_i}$ of process P_i, when executed on N_i.

In order to implement an application, represented as a set of process graphs, the designer has to map the processes to the system nodes and to derive a schedule such that all deadlines are satisfied. We first illustrate some of the problems related to mapping and scheduling, in the context of a system based on a TDMA communication protocol, before going on to explore further aspects specific to an incremental design approach.

Let us consider the example in Figure 2 where we want to map an application consisting of four processes P_1 to P_4, with a period and deadline of 50 ms. The architecture is composed of three nodes that communicate according to a TDMA protocol, such that N_i transmits in slot S_i. According to the specification, processes P_1 and P_3 are constrained to node N_1, while P_2 and P_4 can be mapped on nodes N_2 or N_3, but not N_1. The worst case execution times of processes on each potential node, the size $m_{i,j}$ of the messages passed between P_i and P_j, and the sequence and size of TDMA slots, are presented in Figure 2.

In [5] we have shown that by considering the communication protocol during scheduling, significant improvements can be made to the schedule quality. The same holds true in the case of mapping. Thus, if we are to map P_2 and P_4 on the faster processor N_3, the resulting schedule length (Figure 2a) will be 52 ms which does not meet the deadline. However, if we map P_2 and P_4 on the slower processor N_2, the schedule length (Figure 2b) is 48 ms, which is the best possible solution and meets the deadline. Note, that the total traffic on the bus is the same for both mappings and the initial processor load is 0 on both N_2 and N_3. This result has its explanation in the impact of the communication protocol. P_3 cannot start before receiving messages $m_{2,3}$ and $m_{4,3}$. However, slot S_2 corresponding to node N_2 precedes in the TDMA round slot S_3 on which node N_3 communicates. Thus, the messages which P_3 needs are available sooner in the case P_2 and P_4 are, counter-intuitively, mapped on the slower node.

But finding a valid schedule is not enough if we are to support an incremental design process as discussed in the introduction. In this case, starting from a valid design, we have to improve the mapping and scheduling so that not only the design constraints are satisfied, but also there is a good chance that, later, new functionality can easily be mapped on the resulted system.

To illustrate the role of mapping and scheduling in the context of an incremental design process, let us consider the example in Figure 3. With black we represent the already running set of applications ψ while the current application $\Gamma_{current}$ to be mapped and scheduled is represented in grey. We consider a single processor, and we present three possible scheduling alternatives for the current application. Now, let us suppose that in future a third application, Γ_{future}, has to be mapped on the system. In Figure 3, Γ_{future} is depicted in more detail, showing the two processes P_1 and P_2 it is composed of. We can observe that the new application can be scheduled only in the first two cases, presented in Figure 3a and b. If $\Gamma_{current}$ has been implemented as in Figure 3c, we are not able to schedule process P_2 of Γ_{future}. The way our current application is mapped and scheduled will influence the likelihood of successfully mapping additional functionality on the system without being forced to redesign and test already running applications.

3. PROBLEM FORMULATION

We model an application $\Gamma_{current}$ as a set of process graphs $G_i \in \Gamma_{current}$, each with a period T_{Gi} and a deadline $D_{Gi} \leq T_{Gi}$. For each process P_i in a process graph we know the set $\mathcal{N}_{Pi}$ of potential nodes on which it could be mapped and its worst case execution time on each of these nodes. The underlying architecture, as presented in section 2.1, is based on a TDMA protocol. We consider a non-preemptive static cyclic scheduling policy for both processes and message passing.

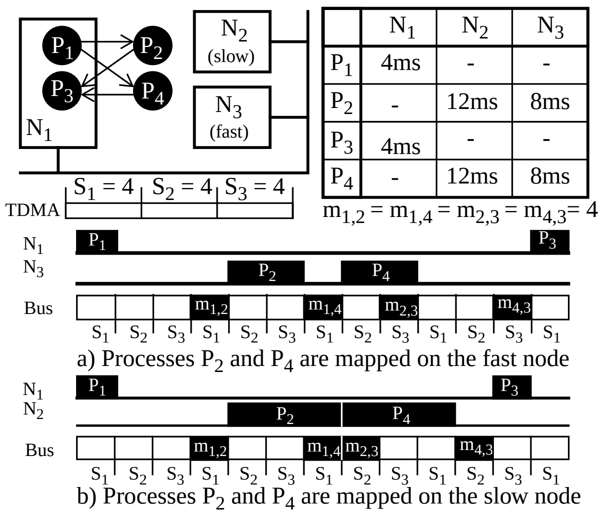
Our goal is to map and schedule an application $\Gamma_{current}$ on a system that already implements a set ψ of applications so that:

a.the constraints on $\Gamma_{current}$ are satisfied without any modification on the implementation of the set of applications ψ;

b. new applications Γ_{future} can be mapped on the resulting system.

If no solution is possible that satisfies a) (the algorithm IM discussed in section 4 fails) we have to change the scheduling and possibly the mapping of applications in ψ in order to meet the constraints on $\Gamma_{current}$. However, even with serious modifications performed on ψ, it is still possible that certain constraints are not satisfied. In this case the hardware architecture has to be changed. In this paper we will not discuss the modification of the running applications or of the hardware architecture. We will concentrate on the situation where a possible mapping and scheduling which satisfies requirement a) can be found and this solution has to be further improved by considering requirement b).

In order to achieve our goal we need certain information to be available concerning the set of applications ψ as well as the possi-



P1
P2
P3
P4
N1
N2 (slow)
N3 (fast)
S1 = 4 S2 = 4 S3 = 4
TDMA
N1 N2 N3
P1 4ms - -
P2 - 12ms 8ms
P3 4ms - -
P4 - 12ms 8ms
m1,2 = m1,4 = m2,3 = m4,3= 4
N1
N3
Bus
m1,2
m1,4
m2,3
m4,3
S1 S2 S3 S1 S2 S3 S1 S2 S3 S1 S2 S3 S1
a) Processes P2 and P4 are mapped on the fast node
N1
N2
Bus
S1 S2 S3 S1 S2 S3 S1 S2 S3 S1 S2 S3 S1
b) Processes P2 and P4 are mapped on the slow node

Figure 2. Mapping and Scheduling Example

ble future applications Γ_{future}. We assume that the only information available on the existing applications ψ consists of the local schedule tables for each node. This means that we know the activation time for each process on the respective node and its worst case execution time. As for messages, their length as well as their place in the particular TDMA frame are known.

The $\Gamma_{current}$ application can interact with the previously mapped applications ψ by reading messages generated on the bus by processes in ψ. In this case, the reading process has to be synchronized with the arrival of the message on the bus, which is easy to solve during scheduling of $\Gamma_{current}$.

What do we suppose to know relative to the family Γ_{future} of applications which do not exist yet? Given a certain limited application area (e.g. automotive electronics), it is not unreasonable to assume that, based on the designers' previous experience, the nature of expected future functions to be implemented, profiling of previous applications, available uncomplete designs for future versions of the product, etc., it is possible to characterize the family of applications which possibly could be added to the current implementation. This is an assumption which is basic for the concept of incremental design. Thus, we consider that, relative to the future applications, we know the set $S_t=\{t_{min},...t_i,...t_{max}\}$ of possible worst case execution times for processes, and the set $S_b=\{b_{min},...b_i,...b_{max}\}$ of possible message sizes. We also assume that over these sets we know the distributions of probability $f_{St}(t)$ for $t \in S_t$ and $f_{Sb}(b)$ for $b \in S_b$. For example, we might have worst case execution times S_t={50, 100, 200, 300, 500 ms}. If there is a higher probability of having processes of 100 ms, and a very low probability of having processes of 300 ms and 500 ms, then our distribution function $f_{St}(t)$ could look like this: $f_{St}(50)=0.20$, $f_{St}(100)=0.50$, $f_{St}(200)=0.20$, $f_{St}(300)=0.05$, and $f_{St}(500)=0.05$.

Another information is related to the period of process graphs which could be part of future applications. In particular, the smallest expected period T_{min} is assumed to be given, together with the expected necessary processor time t_{need}, and bus bandwidth b_{need}, inside such a period T_{min}. As will be shown later, this information is treated in a flexible way during the design process and is used in order to provide a fair distribution of slacks.

The execution times in S_t as well as t_{need} are considered relative the slowest node in the system. All the other nodes are characterized by a speedup factor relative to this slowest node. A normalization with these factors is performed when computing the metrics C_1^P and C_2^P discussed in the following section.

For the sake of simplifying the discussion, we will not address here the memory constraints during process mapping and the implications of memory space in the incremental design process.

3.1 Quality Metrics

A designer will be able to map and schedule an application Γ_{future} on top of a system implementing ψ and $\Gamma_{current}$ only if there are sufficient resources available. In our case, the resources are processor time and the bandwidth on the bus. In the context of a non-preemptive static scheduling policy, having free resources translates into having free time slots on the processors and having space left for messages in the bus slots. We call these free slots of available time on the processor or on the bus, *slack*. It is to be noted that the total quantity of computation and communication power available on our system after we have mapped and scheduled $\Gamma_{current}$ on top of ψ is the same regardless of the mapping and scheduling policies used. What depends on the mapping and scheduling strategy is the distribution of slacks along the time line and the size of the individual slacks. It is exactly this size and distribution of the slacks that characterizes the quality of a certain design alternative. In this section we introduce two criteria in order to reflect the degree to which one design alternative meets the requirement b) presented

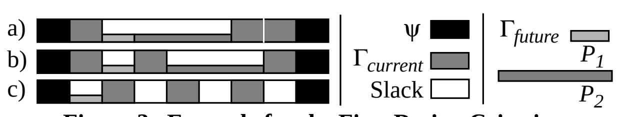


Figure 3. Example for the First Design Criterion

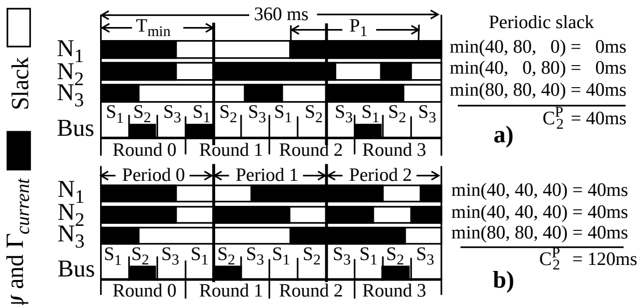


Figure 4. Example for the Second Design Criterion

above. For each criterion we provide metrics which quantify the degree to which the criterion is met. The first criterion reflects how well the resulted slack sizes fit to a future application, and the second criterion expresses how well the slack is distributed in time.

3.1.1 *Slack Sizes (the first criterion)*

The slack sizes resulted after implementation of Γ_{current} on top of ψ should be such that they best accommodate a given family of applications Γ_{future}, characterized by the sets S_t, S_b and the probability distributions f_{St} and f_{Sb}, as outlined before.

Let us consider the example in Figure 3, where we have a single processor and the applications ψ and $\Gamma_{current}$ are already mapped. Suppose that application Γ_{future} consists of the two processes P_1 and P_2. It can be observed that the best configuration, taking in consideration only slack sizes, is to have a contiguous slack. Such a slack, as depicted in Figure 3a, will best accommodate any future application. However, in reality it is almost impossible to map and schedule the current application such that a contiguous slack is obtained. Not only is it impossible, but it is also undesirable from the point of view of the second design criterion, discussed below. As we can see from Figure 3c, if we schedule $\Gamma_{current}$ so that it fragments too much the slack, it is impossible to fit Γ_{future} because there is no slack that can accommodate process P_2. A situation as the one depicted in Figure 3b is desirable, where the resulted slack sizes can accommodate the characteristics of the Γ_{future} application.

In order to measure the degree to which the slack sizes in a given design alternative fit the future applications, we provide two metrics, C_1^P and C_1^m. C_1^P captures how much of the largest future application which theoretically could be mapped on the system if the slacks would be contiguous, can be mapped on the current design alternative. C_1^m is similar relative to the slacks in the bus slots. The largest application is determined knowing the total size of the available slack, and the characteristics of the application: S_t, f_t, S_b, f_b. For example, if our total slack size on the processors is of 2800 ms then, considering the numerical example given in section 3, the largest application will result as having a total of 20 processes: 4 processes of 50 ms, 10 processes (half, $f_t(100)=0.50$) of 100 ms, 4 of 200 ms, and one of 300 and 500 ms. After we have determined the largest Γ_{future} we apply a *bin-packing algorithm* [15] using the *best-fit policy* in which we consider processes as the objects to be packed, and the slacks as containers. The total execution time of unpacked processes relative to the total execution time of the process set gives the C_1^P metric. The same applies for the C_1^m metric. Thus, C_1^P = 0% in Figure 3a and 3b (both are perfect from the point of view of slack size), and 75% -- the worst case execution time of P_2 relative the total slack size -- in Figure 3c.

3.1.2 *Distribution of Slacks (the second criterion)*

In the previous section we provided a metric of how well the sizes of the slacks fit a possible future application. A similar metric is needed to characterize the distribution of slacks over time.

Let P_i be a process with period T_{Pi} that belongs to a future application, and $M(P_i)$ the node on which P_i will be mapped. The worst case execution time of P_i is $t_{P_i}^{M(Pi)}$. In order to schedule P_i we need a slack of size $t_{P_i}^{M(Pi)}$ that is available periodically, within a period T_{Pi}, on processor $M(P_i)$. If we consider a group of processes with period T, which are part of Γ_{future}, in order to implement them a certain amount of slack is needed which is available periodically, with a period T, on the nodes implementing the respective processes.

During implementation of $\Gamma_{current}$ we aim for a slack distribution such that the future application with the smallest expected period T_{min} and with the minimum necessary processor time t_{need}, and bandwidth b_{need}, can be accommodated.

Thus, for each node, we compute the minimum periodic slack, inside a T_{min} period. By summing these minimums, we obtain the slack which is available periodically to Γ_{future}. This is the C_2^P metric. The C_2^m metric characterizes the minimum periodically available bandwidth on the bus and it is computed in a similar way.

In Figure 4 we consider a situation with T_{min}=120 ms, t_{need}=80 ms, and b_{need}=40 ms. The length of the schedule table of our system implementing ψ and $\Gamma_{current}$ is 360 ms. The system consists of three nodes. Let us consider the situation in Figure 4a. In the first period, *Period 0*, there are 40 ms of slack available on node N_1, in the second period 80 ms, and in the third period no slack is available on N_1. Thus, the total slack a future application of period T_{min} can use on node N_1 is min(40, 80, 0)=0 ms. Neither node N_2 can provide slack for this application as in *Period 1* there is no slack available. However, on N_3 there are at least 40 ms of slack available in each period. Thus, with the configuration in Figure 4a we have C_2^P =40 ms, which is not enough to accommodate t_{need}=80 ms. However, in the situation presented in Figure 4b, C_2^P =120 ms > t_{need}, and C_2^m =60 ms > b_{need}.

3.2 Cost Function and Exact Problem Formulation

In order to capture how well a certain design alternative meets the requirement b) stated in section 3, the metrics discussed before are combined in an objective function, as follows:

$$C = w_1^P(C_1^P)^2 + w_1^m(C_1^m)^2 + w_2^P max(0, t_{need} - C_2^P) + w_2^m max(0, b_{need} - C_2^m)$$

where the metric values are weighted by the constants w_i. Our mapping and scheduling strategy will try to minimize this function.

The first two terms measure how well a future application fits to the resulted slack sizes. In order to obtain a balanced solution, that favors a good fitting both on the processors and on the bus, we have used the squares of the metrics.

A design alternative that does not meet the second design criterion is not considered a valid solution. Thus, using the last two terms, we strongly penalize the objective function if either t_{need} or b_{need} is not satisfied, by using high values for the w_2 weights.

At this point, we can give an exact formulation to our problem. Given an existing set of applications ψ which are already mapped and scheduled, and an application $\Gamma_{current}$ to be mapped on top of ψ, we are interested to find a mapping and scheduling of $\Gamma_{current}$ which satisfies all deadlines and minimizes the objective function C, considering a family of future applications characterized by the sets S_t and S_b, the functions f_{St} and f_{Sb} as well as the parameters T_{min}, t_{need}, and b_{need}.

4. THE MAPPING STRATEGY

Our mapping and scheduling strategy has two steps. In the first step we try to obtain a mapping with a valid schedule (which is a schedule that meets the deadlines). Starting from such a solution, a second step iteratively improves on the design in order to minimize the objective function C. The minimization of the objective function will hopefully lead to the situation where it is possible to map new applications on the resulting system.

For the algorithm Initial Mapping (IM) that constructs an initial mapping with a valid schedule, we used as a starting point the Heterogeneous Critical Path (HCP) algorithm presented in [9]. HCP is based on the classical list scheduling algorithm, and uses a *ready list L* of processes ready to execute, i.e. all their predecessors have been scheduled. In each iteration, a process P_i is selected from L according to a *priority function* based on its critical path length (CP), and assigned to the "best processor" $M(P_i)$. Then, process P_i is scheduled on $M(P_i)$. For details on the process and processor selections the reader is referred to [9]. We have modified the HCP algorithm to consider during scheduling a set of previous

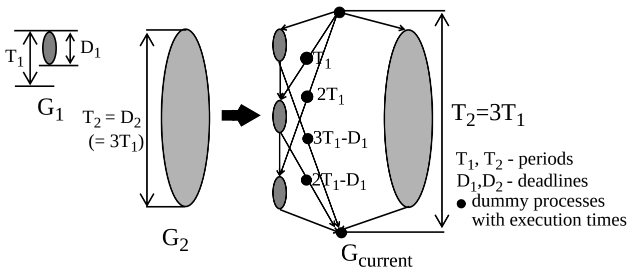


Figure 5. Graph Merging

applications ψ that have already occupied parts of the schedule table, and to schedule the messages according to the TDMA protocol. Furthermore, for the selection of the process from the ready list we have used instead of the CP priority function the MPCP priority function introduced by us in [5]. MPCP takes into consideration the particularities of the communication protocol for calculation of communication delays. These delays are not estimated based only on the message length, but also on the time when slots assigned to the particular node which generates the message, will be available. For the example in Figure 2, our initial mapping algorithm will be able to produce the optimal solution with a schedule length of 48 ms.

However, before using the IM algorithm, two aspects have to be addressed. First, the process graphs $G_i \in \Gamma_{current}$ are merged into a single graph $G_{current}$, by unrolling of process graphs and insertion of dummy nodes as shown in Figure 5. In addition, we have to consider during scheduling the mismatch between the periods of the already existing system and those of the current application. The schedule table into which we would like to schedule $G_{current}$ has a length of T_ψ which is the global period of the existing system ψ. However, the period $T_{current}$ of $G_{current}$ can be different from T_ψ. Thus, before scheduling $G_{current}$ into the existing schedule table, the schedule table is expanded to the least common multiplier of the two periods. In this context, scheduling $G_{current}$ means scheduling it into the expanded schedule table inside each period $T_{current}$. A similar procedure is followed in the case $T_{current} > T_\psi$.

Starting from the valid design produced by IM, our next goal is to improve on the design in order to minimize the objective function C. We iteratively improve the design using a transformational approach. A new design is obtained from the current one by performing a transformation called *move*. We consider the following moves: moving a process to a different slack found on the same node or on a different node, and moving a message to a different slack on the bus. In order to eliminate those moves that will lead to an infeasible design (that violates deadlines), we do as follows. For each process P_i, we calculate the $ASAP_i$ and $ALAP_i$ times considering the resources of the given hardware architecture. $ASAP_i$ is the earliest time P_i can start its execution, while $ALAP_i$ is the latest time P_i can start its execution without causing the application to miss its deadline. When moving P_i we will consider slacks on the target processor only inside the [$ASAP_i$, $ALAP_i$] interval. The same reasoning holds for messages, with the addition that a message can only be moved to slacks belonging to the same slot number, corresponding to the sender node. Any violation of the data dependency constraints is rectified by moving processes or messages concerned in an appropriate way.

For the goal of improving a design as stated above, we first propose a Simulated Annealing strategy (SA) [4] that aims at finding the near-optimal mapping and schedule that minimizes the objective function C. One of the drawbacks of the SA strategy is that in order to find the near-optimal solution it needs very large computation times. Such a strategy, although useful for the final stages of the system synthesis, cannot be used inside a design space exploration cycle.

Thus, we introduce a Mapping Heuristic (MH), outlined in Figure 6, that aims at finding a good quality solution in a reasonable time. MH starts from an initial design produced by IM and iteratively preforms moves in order to improve the design. Unlike

```
MappingHeuristic
    ASAP($\Gamma_{current}$); ALAP($\Gamma_{current}$) -- computes ASAP-ALAP intervals
    InitialMapping($\psi$, $\Gamma_{current}$)
    repeat -- try to satisfy the second design criterion
        repeat
            -- find moves with highest potential to maximize $C_2^P$ or $C_2^m$
            move_set=PotentialMove$C_2^P$ $\cup$ PotentialMove$C_2^m$
            -- select and perform move which improves most $C_2^P$ or $C_2^m$
            move = SelectMove$C_2$(move_set); Perform(move)
        until ($C_2^P \geq t_{need}$ and $C_2^m \geq b_{need}$) or limit reached
        if $C_2^P < t_{need}$ or $C_2^m < b_{need}$ then
            suggest larger $T_{min}$
        end if
    until $C_2^P \geq t_{need}$ and $C_2^m \geq b_{need}$
    repeat -- try to improve the metric of the first design criterion
        -- find moves with highest potential to minimize $C_1^P$ or $C_1^m$
        move_set=PotentialMove$C_1^P$ $\cup$ PotentialMove$C_1^m$
        -- select move which improves $w_1^P(C_1^P)^2 + w_1^m(C_1^m)^2$
        -- and does not invalidate the second design criterion
        move = SelectMove$C_1$(move_set); Perform(move)
    until $w_1^P(C_1^P)^2 + w_1^m(C_1^m)^2$ has not changed or limit reached
end MappingHeuristic
```

Figure 6. Mapping Heuristic to Support Iterative Design

SA that considers all the neighbors of a solution as potential moves, MH tries to find those neighbors that have the highest potential to improve the design, without evaluating for each of them the objective function. MH has two main iterative improvement loops. In the first loop it tries to find a solution that satisfies the second design criterion (section 3.1.2). If such a solution cannot be found, then a larger T_{min} is proposed to the designer. Proposing a larger T_{min} means that the most demanding future application with the requirements t_{need} and b_{need} that we can accommodate, without modifying the existing applications or changing the architecture, cannot have a period smaller than the suggested value. The second loop tries to improve on the metric of the first design criterion (section 3.1.1), without invalidating the second criterion achieved in the first loop. The loop ends when there is no improvement achieved on the first two terms of the objective function, or a limit imposed on the number of iterations has been reached. The intelligence of the Mapping Heuristic lies in how the potential moves are selected. For each iteration a set of potential moves is selected by the PotentialMoveX functions. SelectMoveX then evaluates these moves with regard to the respective metrics and selects the best one to be performed. We now briefly discuss the four PotentialMoveX functions with the corresponding moves:

PotentialMoveC_2^P and PotentialMoveC_2^m

Consider Figure 4a. In *Period 2* on node N_1 there is no available slack. However, if we move process P_1 with 40 ms to the left into *Period 1*, as depicted in Figure 4b, we create a slack in *Period 2* and the periodic slack on node N_1 will be min(40, 40, 40)=40, instead of 0. Potential moves will be the shifting of processes inside their [*ASAP*, *ALAP*] interval in order to improve the periodic slack. The move can be performed on the same node or to the less loaded nodes. The same is true for moving messages. For the improvement of the periodic bandwidth on the bus, we also consider movement of processes, trying to place the sender and receiver of a message on the same processor and, thus, reducing the bus load.

PotentialMoveC_1^P and PotentialMoveC_1^m

In order to avoid excessive fragmentation of the slack we will consider moving a process to a position that snaps to another existing process. A process is selected for potential move if it has the smallest "snapping distance", i.e. in order to attach it to other processes it has to travel the smallest distance inside the schedule table. For a given process such a move is considered both on its node, and to other nodes. We also consider moves that try to increase the individual slacks sizes. Therefore, we first eliminate slack that is unusable: it is too small to hold the smallest process of the future application, or the smallest message. Then, the slacks are sorted in ascending order and the smallest one is considered for improvement. Such improvement of a slack is performed through moving a nearby process or message, but avoiding to create as a result an even smaller individual slack.

5. EXPERIMENTAL RESULTS

For evaluation of our mapping strategies we first used process graphs of 40, 160, 240, 320 and 400 processes generated for experimental purpose. 30 graphs were generated for each graph dimension, thus a total of 150 graphs were used for experimental evaluation. We considered an architecture consisting of 10 nodes of different speeds. For the communication channel we considered a transmission speed of 256 kbps and a length below 20 meters. The maximum length of the data field in a bus slot was 8 bytes. All experiments were run on a SUN Ultra 10. Also, throughout the experiments we have considered an existing set of applications ψ consisting of 400 processes, with a schedule table of 6s on each processor, and a slack of about 50% the total schedule size.

The first result concerns the quality of the designs produced by our initial mapping algorithm IM (using the MPCP priority function which considers particularities of the TDMA protocol) compared to the HCP algorithm. We have calculated the average percentage deviations of the schedule length produced with HCP and IM from the length of the best schedule among the two. Results are depicted in Figure 7a. In average, the deviation with IM is 3.28 times smaller than with HCP. The average execution times for both algorithms are under half a second for graphs with 320 processes.

For the next experiments we were interested to investigate the quality of the mapping heuristic MH compared to a so called *ad-hoc approach* (AH) and to the simulated annealing based algorithm SA. The AH approach is a simple, straight-forward solution to produce designs which, to a certain degree, support an incremental process. Starting from the initial valid schedule of length *S* obtained by IM for the graph *G* with *N* processes, AH uses a simple scheme to redistribute the processes inside the [0, *D*] interval, where *D* is the deadline of the process graph *G*. AH starts by considering the first process in topological order, let it be P_1. It introduces after P_1 a slack of size min(smallest process size of Γ_{future}, *(D-S)/N)*, thus shifting all P_1's descendants to the right. The insertion of slacks is repeated for the next process, with the current larger value of *S*, as long as the resulted schedule has an $S \leq D$.

MH, SA and AH have been used to map each of the 150 process graphs on the target system. For each of the resulted designs, the objective function *C* has been computed. Very long and expensive runs have been performed with the SA algorithm for each graph and the best ever solution produced has been considered as the near-optimum for that graph. We have compared the objective function obtained for the 150 process graphs considering each of the three mapping algorithms. Figure 7b presents the average percentage deviation of the objective function obtained with the MH and AH from the value of the objective function obtained with the

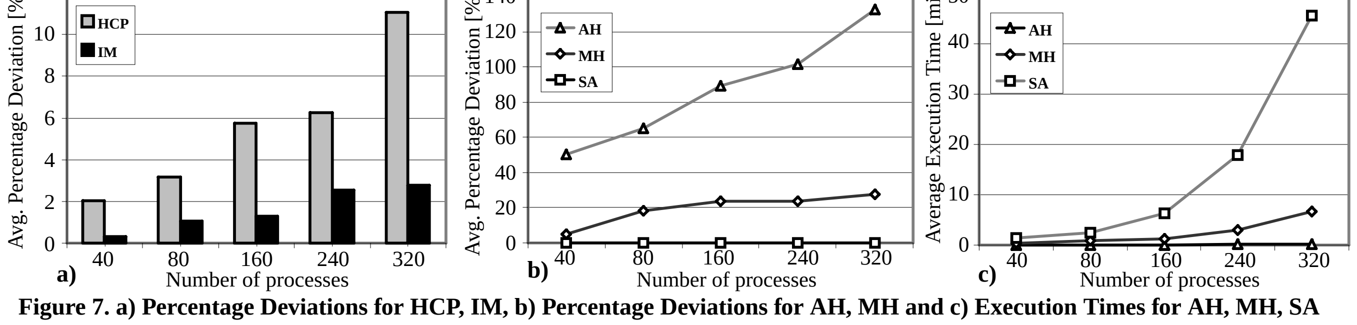


Figure 7. a) Percentage Deviations for HCP, IM, b) Percentage Deviations for AH, MH and c) Execution Times for AH, MH, SA

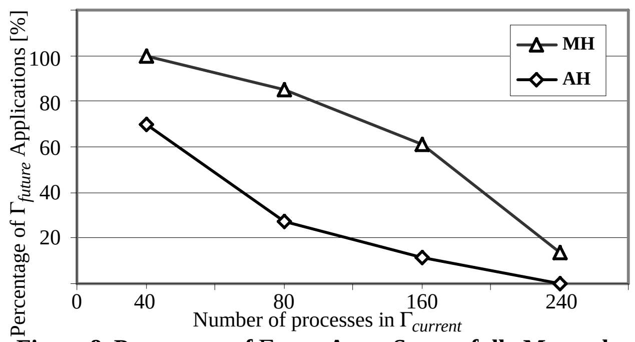


Figure 8. Percentage of Γ_{future} Apps. Successfully Mapped

near-optimal scheme. We have excluded from the results in Figure 7b, 37 solutions obtained with AH for which the second design criterion has not been met, and thus the objective function has been strongly penalized. The average run-times of the algorithms, in seconds, are presented in Figure 7c. The SA approach performs best in terms of quality at the expense of a large execution time. The execution time can be up to 45 minutes for large graphs of 320 processes. MH performs very well, and is able to obtain good quality solutions in a very short time. AH is very fast, but since it does not address explicitly the two design criteria presented in Section 3 it has the worst quality of solutions, according to the objective function.

The most important aspect of the experiments is determining to which extent the mapping strategies proposed in the paper really facilitate the implementation of future applications. To find this out, we have mapped graphs of 40, 80, 160 and 240 nodes representing the $\Gamma_{current}$ application on top of ψ. After mapping and scheduling each of these graphs we have tried to add a new application Γ_{future} to the resulted system. Γ_{future} consists of a process graph of 80 processes, randomly generated according to the following specifications: S_t={20, 50, 100, 150, 200 ms}, $f_t(S_t)$={10, 25, 45, 15, 5%}, S_b={2, 4, 6, 8 bytes}, $f_b(S_b)$={20, 50, 20, 10%}, T_{min}=250 ms, t_{need}=100 and b_{need}=20 ms. The experiments have been performed two times, using first MH and then AH for mapping $\Gamma_{current}$. In both cases we were interested if it is possible to find a valid implementation for Γ_{future} on top of $\Gamma_{current}$ using the initial mapping algorithm IM. Figure 8 shows the number of successful implementations in the two cases. In the case $\Gamma_{current}$ has been mapped with MH, this means using the design criteria and metrics proposed in the paper, we were able to find a valid schedule for 65% of the total process graphs considered. However, using AH to map $\Gamma_{current}$ has led to a situation where IM is able to find valid schedules in only 21% cases. Another observation from Figure 8 is that when the slack size available is large, as in the case $\Gamma_{current}$ has only 40 processes, it is easy for both MH and AH to find a mapping that allows adding future applications. However, as $\Gamma_{current}$ grows to 160, only MH is able to find a mapping of $\Gamma_{current}$ that supports an incremental design process, accommodating more that 60% of the future applications. If the remaining slack is very small, after we map a $\Gamma_{current}$ of 240, it becomes practically impossible to map new applications without modifying the current system.

Finally, we considered an example implementing a vehicle cruise controller (CC) modeled using one process graph. The graph has 32 processes and it was to be mapped on an architecture consisting of 4 nodes, namely: Anti Blocking System, Transmission Control Module, Engine Control Module and Electronic Throttle Module. The period was 300 ms, equal to the deadline. In order to validate our approach, we have considered the following setting. The system ψ consists of 80 processes generated randomly, with a schedule table of 300 ms and about 40% slack. The CC is the $\Gamma_{current}$ application to be mapped. We have also generated 30 future applications of 40 processes each with the characteristics of the CC, which are typical for automotive applications. By mapping the CC using MH we were able to later map 21 of the future applications, while using AH only 4 of the future applications could be mapped.

6. CONCLUSIONS

We have presented an approach to the incremental design of distributed embedded systems for hard real-time applications. Such a design process satisfies two main requirements when adding new functionality: the already running functionality is not disturbed, and there is a good chance that, later, new functionality can easily be mapped on the resulted system. Our approach was considered in the context of a non-preemptive static cyclic scheduling policy and a realistic communication model based on a TDMA scheme.

We have introduced two design criteria with their corresponding metrics, that drive our mapping strategies to solutions supporting an incremental design process. For constructing an initial valid solution, we have shown that it is needed to take into account the features of the communication protocol. Starting from an initial solution, we have proposed two mapping algorithms, SA based on a simulated annealing strategy and MH an iterative improvement heuristic. SA finds a near-optimal solution at the expense of a large execution time, while MH is able to quickly produce good quality results.

The approach has been validated through several experiments.

REFERENCES

[1] J.E. Beck, D.P. Siewiorek, “Automatic Configuration of Embedded Multicomputer Systems”, IEEE Trans. on CAD, v17, n.2, 1998, pp. 84-95.

[2] T. Blicke, J. Teich, L. Thiele, “System-Level Synthesis Using Evolutionary Algorithms”, Design Automation for Embedded Systems, V4, N1, 1998, pp. 23-58.

[3] B.P. Dave, G. Lakshminarayana, N.K. Jha, “COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, March 1999, pp. 92 -104.

[4] P. Eles, Z. Peng, K. Kuchcinski, A. Doboli, “System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search”, Des. Automation for Embedded Syst., V2, N1, 1997, pp. 5-32.

[5] P. Eles, A. Doboli, P. Pop, Z. Peng, “Scheduling with Bus Access Optimization for Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, 2000 (to appear).

[6] R. Ernst, J. Henkel, T. Benner, “Hardware-Software Cosynthesis for Microcontrollers”, IEEE Design and Test of Computers, vol. 10, Sept. 1993, pp. 64-75.

[7] R. Ernst, "Codesign of Embedded Systems: Status and Trends", IEEE Design&Test of Comp., April-June, 1998, pp. 45-54.

[8] R. K. Gupta, G. De Micheli, "Hardware-software cosynthesis for digital systems", IEEE Design and Test of Computers, vol. 10, Sept. 1993, pp. 29-41.

[9] P. B. Jorgensen, J. Madsen, “Critical Path Driven Cosynthesis for Heterogeneous Target Architectures,” Proc. Int. Workshop on Hardware-Software Co-design, 1997, pp. 15-19.

[10] A. Kalawade, E.A. Lee, “The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-Bin Selection”, Design Automation for Embedded Systems, V2, 1997, pp. 125-163.

[11] P.V. Knudsen, J. Madsen, "Integrating Communication Protocol Selection with Hardware/Software Codesign", IEEE Trans. on CAD, V18, N8, 1999, pp. 1077-1095.

[12] H. Kopetz, G. Grünsteidl, “TTP-A Protocol for Fault-Tolerant Real-Time Systems,” IEEE Computer, 27(1), 1994, pp. 14-23.

[13] C. Lee, M. Potkonjak, W. Wolf, "Synthesis of Hard Real-Time Application Specific Systems", Design Automation for Embedded Systems, V4, N4, 1999, pp. 215-241.

[14] Yanbing Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, J. Stockwood, “Hardware-Software Co-Design of Embedded Reconfigurable Architectures”; ACM/IEEE DAC, 2000, 507-512.

[15] S. Martello, P.Toth, “Kanpsack Problems: Algorithms and Computer Implementations”, Wiley, 1990.

[16] S. Narayan, D.D. Gajski, "Synthesis of System-Level Bus Interfaces", Proc. Europ. Des. & Test Conf, 1994, 395-399.

[17] R.B. Ortega, G. Borriello, "Communication Synthesis for Distributed Embedded Systems", Proc. Int. Conf. on CAD, 1998, pp. 437-444.

[18] P. Pop, P. Eles, Z. Peng, “Scheduling with Optimized Communication for Time-Triggered Embedded Systems,” Proc.

Int. Workshop on Hardware-Software Co-design, 1999,78-82.
[19] S. Prakash, A. Parker, “SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems”, Journal of Parallel and Distrib. Comp., V16, 1992, pp. 338-351.
[20] D. L. Rhodes, Wayne Wolf, “Co-Synthesis of Heterogeneous Multiprocessor Systems using Arbitrated Communication”, Proceeding of the 1999 International Conference on CAD, 1999, pp. 339 - 342.
[21] W. Wolf, “An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing systems”,IEEE Transactions on VLSI Systems, June 1997, pp. 218 -229.
[22] T. Y. Yen, W. Wolf, "Hardware-Software Co-Synthesis of Distributed Embedded Systems", Kluwer Academic Publ., 1997.