Minimizing System Modification in an Incremental Design Approach

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 mapping and scheduling of distributed embedded systems for hard real-time applications, aiming at minimizing the system modification cost. We consider an incremental design process that starts from an already existing system running a set of applications. We are interested to implement new functionality so that the already running applications are disturbed as little as possible 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.

Keywords*: design space exploration, design reuse, distributed real-time systems, process mapping and scheduling, methodology.*

1. INTRODUCTION

Distributed embedded systems with multiple processing elements are becoming common in various application areas [4]. In [12], for example, allocation of processing elements, as well as process mapping and scheduling for distributed systems are formulated as a mixed integer linear programming (MILP) problem. A disadvantage of this approach is the complexity of solving the MILP problem. Therefore, alternative problem formulations and solutions based on efficient heuristics have been proposed [1, 2, 8, 14].

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 [13], 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 [6, 10, 9].

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 (including also upgrades to the existing one) on this system. In such a context it is very important to make as few as possible modifications to the already running applications. The main reason for this is to avoid unnecessarily large design and testing times. Performing modifications on the (potentially large) existing applications 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.

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 [7]. For the same reason we use a non-preemptive static task scheduling scheme.

In this paper, we have considered 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. already running applications are disturbed as little as possible;
b. there is a good chance that new functionality can, later, easily be mapped on the resulted system.

In [11] we have discussed an incremental design strategy which excludes any modifications on already running applications. In this paper we extend our approach in the sense that remapping and scheduling of currently implemented applications are allowed, if they are needed in order to accommodate the new functionality. In this context, we propose a heuristic which finds the set of old applications which have to be remapped together with the new one such that the disturbance on the running system (expressed as the total cost implied by the modifications) is minimized. Once this set of applications has been determined, mapping and scheduling is performed according to the requirements stated above.

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 some preliminary discussion. Section 3 introduces the detailed problem formulation and the quality metrics we have defined. Our mapping and scheduling 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 processing nodes connected by a broadcast communication channel. Communication between nodes is based on a TDMA protocol such as the TTP [7] 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. Each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i (Figure 1). In such a slot, a node can send several messages packaged in a frame. A 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.

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 [3].

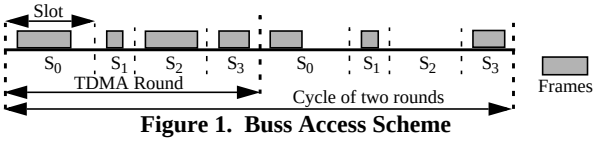


Figure 1. Buss Access Scheme

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 and Scheduling

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 $N_{Pi} \subseteq PE$ be the set of nodes to which P_i can potentially be mapped. For each $N_i \in N_{Pi}$, 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. However, finding a valid schedule is not always possible, either because there are not enough available hardware resources, or the resources are not intelligently allocated to the already running applications. Thus, in order to produce a valid solution, the resources have to be reallocated through rescheduling and remapping of some of the already running applications or, in the worst case, the architecture has to be modified by adding new resources.

In Figure 2 we consider a single processor system with three applications, A, B and C, each with a deadline D_A, D_B and D_C. Application C is depicted in more detail, showing the two processes P_1 and P_2 it is composed of. Let us suppose that the already running applications are A and B, and we have to implement C as a new application. If A and B have been mapped and schedules like in Figure 2a, we will not be able to map application C (in particular, process P_2). With a mapping of A and B like in Figure 2b and c, we are able to map both processes of C, but no schedule can be produced which meets the deadline D_C. If A and B are implemented like in Figure 2d, application C can be successfully implemented. Two aspects can be highlighted based on this example:

1. If applications A and B are implemented like in Figure 2a (or like in 3b or 3c), it is possible to correctly implement application C only with modifying the implementation of application B.
2. If during implementation of application B we would have taken into consideration that sometimes in the future an application like C will have to be implemented, we could have produced a schedule like the one in Figure 2d. In this case, application C could be implemented without any modification of an existing application.

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 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 is as presented in section 2.1. 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, considering the following requirements:

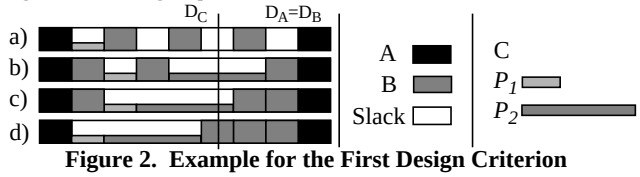


Figure 2. Example for the First Design Criterion

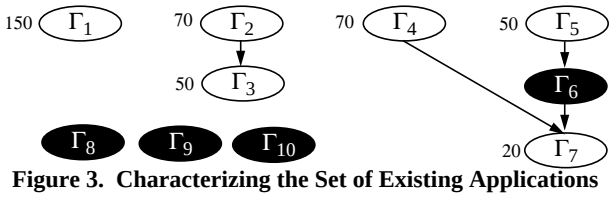


Figure 3. Characterizing the Set of Existing Applications

Requirement a: constraints on $\Gamma_{current}$ are satisfied and minimal modifications are performed to the applications in ψ.

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

If it is not possible to map and schedule $\Gamma_{current}$ without modifying the already running applications, we have to change the scheduling and mapping of some applications in ψ. 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 by, for example, adding a new processor. In this paper we will not discuss this last case, but 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 goals we need certain information to be available concerning the set of applications ψ as well as the possible future applications Γ_{future}. We consider that $\Gamma_{current}$ 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 $\Gamma_{current}$.

3.1 Characterizing Existing Applications

To perform the mapping and scheduling of $\Gamma_{current}$, the minimum information needed on the existing applications ψ consists of the local schedule tables for each node. Thus, 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. However, if the initial attempt to schedule and map $\Gamma_{current}$ does not succeed, we have to modify the schedule and, possibly, the mapping of applications belonging to ψ, in the hope to find a valid solution for $\Gamma_{current}$.

Our goal in this paper is to find that minimal modification to the existing system that leads to a correct implementation of $\Gamma_{current}$. In our context, such a minimal modification means remapping and rescheduling a subset of old applications $\Omega \subseteq \psi$ so that the total cost of reimplementing Ω is minimized. We represent a set of applications as a directed acyclic graph $G(V, E)$, where each node $\Gamma_i \in V$ represents an application. An edge $e_{ij} \in E$ from Γ_i to Γ_j indicates that any modification to Γ_i would trigger the need to also remap and schedule Γ_j. Such a relation can be imposed by certain interactions between applications[1]. In Figure 3 we present the graph corresponding to a set of ten applications. Applications Γ_6, Γ_8, Γ_9 and Γ_{10}, depicted in black, are frozen: no modifications are possible to them. The rest of the applications have the remapping cost R_i depicted on their left. Γ_7 can be remapped with a cost of 20. If Γ_4 is to be reimplemented, this also requires the modification of Γ_7, with a total cost of 90. In the case of Γ_5, although not frozen, no remapping is possible as it would trigger the need to remap Γ_6 which is frozen. Given a subset of applications $\Omega \subseteq \psi$, the total cost of modifying the applications in Ω is $R(\Omega) = \sum_{\Gamma_i \in \Omega} R_i$.

To each application $\Gamma_i \in V$ the designer has associated a cost R_i of reimplementing Γ_i. Such a cost can typically be expressed in hours needed to perform retesting of Γ_i and other tasks connected to the remapping and rescheduling of the application. Remapping of Γ_i and the associated rescheduling can only be performed if the

[1] If a set of applications have a circular dependence, such that the modification of any one implies the remapping of all the others in that set, the set will be represented as a single node in the graph.

process graphs that capture the applications and their deadlines are available. However, this is not always the case, and in such situations the application is considered frozen.

3.2 Characterizing Future Applications

What do we suppose to know about 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 could possibly be added to the current implementation. This is an assumption which is basic for the concept of incremental design. Thus, we consider that, concerning 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 used in order to provide a fair distribution of slacks.

The execution times in S_t as well as t_{need} are considered relative to the slowest node in the system. All the other nodes are characterized by a speedup factor relative to this slowest node.

3.3 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 the size and distribution of the slacks that characterizes the quality of a certain design alternative from the point of view of its potential to accommodate future applications. In this section we introduce two criteria in order to reflect the degree to which one design alternative meets the requirement b) presented at the beginning of section 3.

The first criterion reflects how well the resulted slack sizes fit to a future application. The slack sizes resulted after implementation of $\Gamma_{current}$ 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 2, where we have a single processor with the applications *A* and *B* implemented and a future application *C* which 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 2c and d, 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.

The second criterion expresses how well the slack is distributed in time. Let P_i be a process with period T_{P_i} 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 expected necessary processor time t_{need}, and bandwidth b_{need}, can be accommodated.

We have defined two metrics, C_1 and C_2, which quantify the degree to which the first and second criterion, respectively, are met. A detailed discussion about these metrics is given in [11].

3.4 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:

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

C_1^P and C_2^P are those components of the two metrics that capture the slack properties on processors, while C_1^m and C_2^m are calculated for the slacks on the bus. Our mapping and scheduling strategy will try to minimize this function.

The first two terms measure how well the resulted slack sizes fit to a future application (first criterion), while the second two terms reflect the distribution of slacks (second criterion). We call a *valid solution* that mapping and scheduling which satisfies all the design constraints (in our case the deadlines) and meets the second criterion ($C_2^P \geq t_{need}$ and $C_2^m \geq b_{need}$)[1].

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 implemented on top of ψ, we are interested to find the subset $\Omega \subseteq \psi$ of old applications to be remapped and rescheduled such that we produce a valid solution for $\Gamma_{current} \cup \Omega$ and the total cost of modification $R(\Omega)$ is minimized. Once such an Ω is found, we are interested to minimize the objective function *C* for the set $\Gamma_{current} \cup \Omega$, 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. MAPPING AND SCHEDULING STRATEGY

As shown in Figure 4, our mapping and scheduling strategy (MS) has two steps. In the first step we try to obtain a valid solution for $\Gamma_{current} \cup \Omega$ so that $R(\Omega)$ is minimized. Starting from such a solution, a second step iteratively improves on the design in order to minimize the objective function *C*.

4.1 The Initial Mapping and Scheduling

The first step of MS consists of an iteration that tries subsets $\Omega \subseteq \psi$ with the intention to find that subset $\Omega=\Omega_{min}$ which produces a valid solution for $\Gamma_{current} \cup \Omega$ such that $R(\Omega)$ is minimized. Given a subset Ω, the InitialMappingScheduling function (IMS) constructs a mapping and schedule for $\Gamma_{current} \cup \Omega$ that meets the deadlines, without worrying about the two criteria in section 3.3. For IMS we used as a starting point the Heterogeneous Critical Path (HCP) algorithm, introduced in [5]. HCP is based on a list scheduling algorithm. We have modified the HCP algorithm to consider, during mapping and scheduling, a set of previous applications that have already occupied parts of the schedule table, and to schedule the messages according to the TDMA protocol. Furthermore, for the selection of processes we have used, instead of the CP (critical path) priority function, the (modified partial critical path) MPCP priority function introduced by us in [3]. 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.

However, before using the IMS algorithm, two aspects have to

[1] This definition of a valid solution can be relaxed by imposing only the satisfaction of deadlines. In this case, the algorithm in Figure 4 will look after a solution which satisfies the deadlines and R(Ω) is minimized; the two additional criteria are only considered optionally.

be addressed. First, the process graphs $G_i \in \Gamma_{current} \cup \Omega$ are merged into a single graph $G_{current}$, by unrolling of process graphs and insertion of dummy nodes [11]. 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 $T_{\psi \setminus \Omega}$ which is the global period of the system ψ after extraction of the applications in Ω. However, the period $T_{current}$ of $G_{current}$ can be different from $T_{\psi \setminus \Omega}$. Thus, before scheduling $G_{current}$ into the existing schedule table, the schedule table is expanded to the least common multiplier of the two periods. A similar procedure is followed in the case $T_{current} > T_{\psi \setminus \Omega}$.

4.2 The Basic Strategy

If IMS succeeds in finding a mapping and schedule which meet the deadlines, this is not yet a valid solution. In order to produce a valid solution we iteratively try to satisfy the second design criterion. In terms of our metrics, that means a mapping and scheduling such that $C_2^P \geq t_{need}$ and $C_2^m \geq b_{need}$. Potential moves can 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 other nodes. Similar moves are considered for messages. SelectMoveC$_2$ evaluates these moves with regard to the second design criterion and selects the best one to be performed. Any violation of the data dependency constraints is rectified by moving processes or messages concerned in an appropriate way.

If Step 1 has succeeded, a mapping and scheduling of $\Gamma_{current} \cup \Omega$ has been produced which corresponds to a valid solution. In addition, Ω is such that the total modification cost is as small as possible. Starting from this valid solution, the second step of the MS strategy, presented in Figure 4, tries to improve on the design in order to minimize the objective function *C*. In a similar way as during Step 1, we iteratively improve the design by successive moves.

In [11] we introduced a heuristic with the goal of guiding the moves discussed above. Its intelligence lies in how the moves are selected. For each iteration a set of potential moves is selected by the PotentialMove function. SelectMove then evaluates these moves with regard to the respective metrics and selects the best one to perform.

4.3 Minimizing the Modification Cost

The first step of our mapping strategy described in Figure 4 iterates on subsets Ω searching for a valid solution which also minimizes the total modification cost $R(\Omega)$. As a first attempt, the algorithm searches for a valid implementation of $\Gamma_{current}$ without disturbing the existing applications ($\Omega=\emptyset$). If no valid solution is found successive subsets Ω produced by the function NextSubset are considered, until a terminating condition is met. The performance of the algorithm, in terms of runtime and quality of the solutions produced, is strongly influenced by the implementation of the function NextSubset and the termination condition. They determine how the design space is explored while testing different subsets Ω of applications.

4.3.1 Exhaustive Search (ES)

In order to find Ω_{min}, the simplest solution is to try successively all the possible subsets $\Omega \subseteq \psi$. These subsets are generated in the ascending order of the total modification cost, starting from $\emptyset$. The termination condition is fulfilled when the first valid solution is generated. Since the subsets are generated in ascending order, according to their cost, the subset Ω that first produces a valid solution is also the subset with the minimum modification cost.

The generation of subsets is performed according to the graph *G* that characterizes the existing applications (see section 3.1). Finding the next subset Ω, starting from the current one, is achieved by a branch and bound algorithm that in the worst case grows exponentially in time with the number of applications. For the example in Figure 3, the call to NextSubset($\emptyset$) will generate $\{\Gamma_7\}$ which has the smallest nonzero modification cost. The next generated subsets, in order, together with their corresponding total modification cost are: $R(\{\Gamma_3\})$=50, $R(\{\Gamma_3, \Gamma_7\})$=70, $R(\{\Gamma_4, \Gamma_7\})$=90 (the inclusion of Γ_4 triggers the inclusion of Γ_7), $R(\{\Gamma_2, \Gamma_3\})$=120, $R(\{\Gamma_3, \Gamma_4, \Gamma_7\})$=140, $R(\{\Gamma_1\})$=150, and so on. The total number of possible subsets according to the graph *G* is 16.

```
MappingSchedulingStrategy
  Ω=∅
  -- Step 1: try to find a valid schedule for Γcurrent that minimizes R(Ω)
  repeat
    succeeded=InitialMappingScheduling(ψ \ Ω, Γcurrent∪Ω)
    -- compute ASAP-ALAP intervals
    ASAP(Γcurrent∪Ω); ALAP(Γcurrent∪Ω)
    if succeeded then
      repeat -- try to satisfy the second design criterion
        -- find moves with highest potential to maximize C2
        move_set=PotentialMoveC2(Γcurrent∪Ω)
        -- select and perform move which improves most C2
        move = SelectMoveC2(move_set); Perform(move)
        succeeded = C2P ≥tneed and C2m ≥bneed
      until succeeded or limit reached
    end if
    if succeeded and R(Ω) smallest so far then
      Ωvalid=Ω; solutionvalid=solutioncurrent
    end if
    -- try another subset
    Ω=NextSubset(Ω)
  until termination condition
  if not succeeded then modify architecture; go to step 1; end if
  -- Step 2: try to improve the cost function C
  solutioncurrent=solutionvalid; Ωmin=Ωvalid
  repeat
    -- find moves with highest potential to minimize C
    move_set=PotentialMoveC(Γcurrent∪Ωmin)
    -- select move which improves C
    -- and does not invalidate the second design criterion
    move = SelectMoveC(move_set); Perform(move)
  until C1 has not changed or limit reached
end MappingSchedulingStrategy
```

Figure 4. MS Strategy to Support Iterative Design

This approach, while finding the optimal subset Ω, requires a large amount of computation time and can be used only with a small number of applications.

4.3.2 Ad-hoc Solution (AH)

If the number of applications is larger, a possible ad-hoc solution could be based on a greedy strategy which, starting from $\Omega=\emptyset$, progressively enlarges the subset until a valid solution is produced. The algorithm looks at all the non-frozen applications and picks that one which, together with its dependencies, has the smallest modification cost. If the new subset does not produce a valid solution, it is enlarged by including, in the same fashion, the next application with its dependencies. This greedy expansion of the subset is continued until the set is large enough to lead to a valid solution or no application is left. For the example in Figure 3 the call to NextSubset($\emptyset$) will produce $R(\{\Gamma_7\})$=20, and will be successively enlarged to $R(\{\Gamma_7, \Gamma_3\})$=70, $R(\{\Gamma_7, \Gamma_3, \Gamma_2\})$=140 ($\Gamma_4$ could have been picked as well in this step because it has the same modification cost of 70 as Γ_2 and its dependence Γ_7 is already in the subset), $R(\{\Gamma_7, \Gamma_3, \Gamma_2, \Gamma_4\})$=210, and so on.

While this approach finds very quickly a valid solution, if one exists, it is possible that the total modification cost is much higher than the optimal one.

4.3.3 Subset Selection Heuristic (SH)

An intelligent selection heuristic should be able to identify the reasons due to which a valid solution has not been found. Such a failure can have two possible causes: an initial mapping which meets the deadlines has not been produced, or the second criterion is not satisfied.

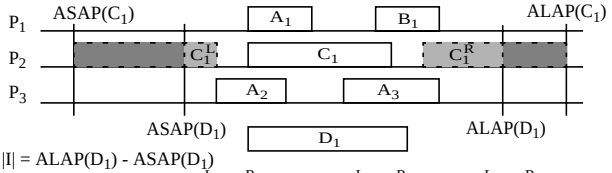
Let us investigate the first reason. If an application Γ_i is to meet its deadline D_i, all its processes $P_j \in \Gamma_i$ have to be scheduled inside their [*ASAP, ALAP*] intervals. InitialMappingScheduling (IMS) fails to schedule a process inside its [*ASAP, ALAP*] interval if there is not enough slack available on any processor, due to other processes scheduled in the same interval. In this situation we say that there is a

conflict with processes belonging to other applications. We are interested to find out which applications are responsible for conflicts encountered by our $\Gamma_{current}$, and not only that, but also which ones are *flexible* enough to move away in order to avoid these conflicts.

IMS determines a metric Δ_i that characterizes the degree of conflict and the flexibility of application Γ_i in relation to $\Gamma_{current}$. A set of applications Ω will be characterized, in relation to $\Gamma_{current}$, by $\Delta(\Omega) = \sum_{\Gamma_i \in \Omega} \Delta_i$. The metric $\Delta(\Omega)$ will be used by our subset selection heuristic if IMS has failed to produce a solution which satisfies the deadlines. An application with a larger Δ_i is more likely to lead to a valid schedule if included in Ω. In Figure 5 we illustrate how this metric is calculated. Applications *A*, *B* and *C* are scheduled on three processors P_1, P_2 and P_3, and our goal is to implement the current application *D*. At a certain moment IMS comes to the point to place process $D_1 \in D$. However, it is not able to place D_1 inside its [*ASAP, ALAP*] interval *I*, because there is not enough free slack available inside *I* on any of the processors. We are interested to determine which of the applications *A*, *B* and *C* are more likely to lend free slack for D_1 if remapped. Therefore, we calculate the slack resulted after we move away processes from the interval *I*. For example, the resulted slack available after remapping application *C* (moving process $C_1 \in C$ either to the left or to the right inside its own [*ASAP, ALAP*] interval) is of size $|I| - \min(|C_1^L|, |C_1^R|)$. Thus, we increment Δ_C with $\delta_C = |I| - \min(|C_1^L|, |C_1^R|) - |D_1|$. The increments δ_B and δ_A to be added to Δ_B and Δ_A respectively, are also presented in Figure 5. IMS continues with the other processes of application D (after assuming that process D_1 has been scheduled at the beginning of interval *I*). As result of the failed attempt to map *D*, IMS will produce the metrics Δ_A, Δ_B, and Δ_C.

If the initial mapping was successful, the first step of MS could fail during the attempt to satisfy the second criterion. In this case, the metric Δ_i is computed in a different way. It will capture the potential of an application Γ_i to improve the metric C_2 if remapped together with $\Gamma_{current}$. Thus, for the improvement of C_2 we consider a total number of moves from all the non-frozen applications (determined using Potential-Move$C_2(\psi)$). For each move that has as subject $P_j \in \Gamma_i$, we increment the metric Δ_i with the predicted improvement on C_2.

MS starts by trying an implementation of $\Gamma_{current}$ with $\Omega = \varnothing$. If this attempt fails, because of one of the two reasons mentioned above, the corresponding metrics Δ_i are computed for all $\Gamma_i \in \psi$. Our heuristic SH will then start by finding the ad-hoc solution Ω_{AH} produced by the AH algorithm (this will succeed if there exists any solution) with a corresponding cost $R_{AH} = R(\Omega_{AH})$ and a $\Delta_{AH} = \Delta(\Omega_{AH})$. SH now continues by trying to find a solution with a more favorable Ω (a smaller total cost R). Therefore, the thresholds $R_{max} = R_{AH}$ and $\Delta_{min} = \Delta_{AH}/n$ (for our experiments we considered n=2) are set. For generating new subsets Ω, the function NextSubset now follows a similar approach like ES but in a reverse direction, towards smaller subsets, and it will consider only subsets with a smaller total cost then R_{max} and a larger Δ then Δ_{min} (a small Δ means a reduced potential to eliminate the cause of the initial failure). Each time a valid solution is found, the current values of R_{max} and Δ_{min} are updated in order to further restrict the search space. The heuristic stops when no subset can be found with $\Delta > \Delta_{min}$, or a certain imposed limit has been reached (e.g. on the total number of attempts to find new sets).



$|I| = ALAP(D_1) - ASAP(D_1)$

$\delta_A = \max(|I| - |B_1| - \min(|A_1^L|, |A_1^R|), |I| - \min(|A_2^L|, |A_2^R|) - \min(|A_3^L|, |A_3^R|)) - |D_1|$

$\delta_B = |I| - |A_1| - \min(|B_1^L|, |B_1^R|) - |D_1|$

$\delta_C = |I| - \min(|C_1^L|, |C_1^R|) - |D_1|$

Figure 5. Metric for the Subset Selection Heuristic

5. EXPERIMENTAL RESULTS

For evaluation of the proposed strategies we first used process graphs of 80, 160, 240, 320 and 400 processes, representing the application $\Gamma_{current}$, generated for experimental purpose. 30 graphs were generated for each graph dimension, resulting in a total of 150 graphs. We considered an architecture consisting of 10 nodes. 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. Experiments were run on a SUN Ultra 10.

The first results concern the quality of the solution obtained with our mapping strategy MS using the search heuristic SH compared to the case when the ad-hoc approach AH and the exhaustive search ES are used. For each of the five graph dimensions for $\Gamma_{current}$ we have considered a set of existing applications ψ consisting of 320, 400, 480, 560 and 640 processes, respectively. The sets contained 6, 8, 10, 12 and 14 applications, each application with an associated modification cost assigned manually in the range 10 to 100. The available slack is of about 50% of the total schedule size. The dependencies between applications were such that the total number of possible subsets Ω resulted for each set ψ were 32, 128, 256, 1024 and 4096. We have considered that the future applications Γ_{future} consist 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 ms and b_{need}=20 ms.

MS has been used to produce a valid solution for each of the 150 process graphs representing $\Gamma_{current}$ on the target system ψ using the ES, AH and SH approaches to subset selection. Figure 6a compares the three approaches based on the total modification cost needed in order to obtain a valid solution. The exhaustive approach ES is able to obtain valid solutions with an optimal (smallest) modification cost, while the ad-hoc approach AH produces in average 3.12 times more costly modifications in order to obtain valid solutions. However, in order to find the optimal solution ES needs large computation times, as shown in Figure 6b. For example, it can take more than 2 hours in average to find the smallest cost subset to be remapped that leads to a valid solution in the case of 14 applications (640 processes). We can see that the proposed heuristic SH performs well, producing close to optimal results with a good scaling for large application sets. For the results in Figure 6 we have eliminated those situations in which no valid solution could be produced by MS.

Another important aspect to be proven by experiments is the extent to which the mapping strategy proposed in the paper really facilitates the implementation of *future* applications. For these experiments we have considered that no modifications are allowed to the applications in ψ. We have used 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% of the total schedule size. Then, 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 (for Γ_{future} we used the same experimental set as presented before). The experiments have been performed two times, using first MS* (we call MS* the version of MS in which no modification of applications in ψ is allowed), and then an *ad-hoc mapping* approach (AM), 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 IMS. The AM 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 IMS 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

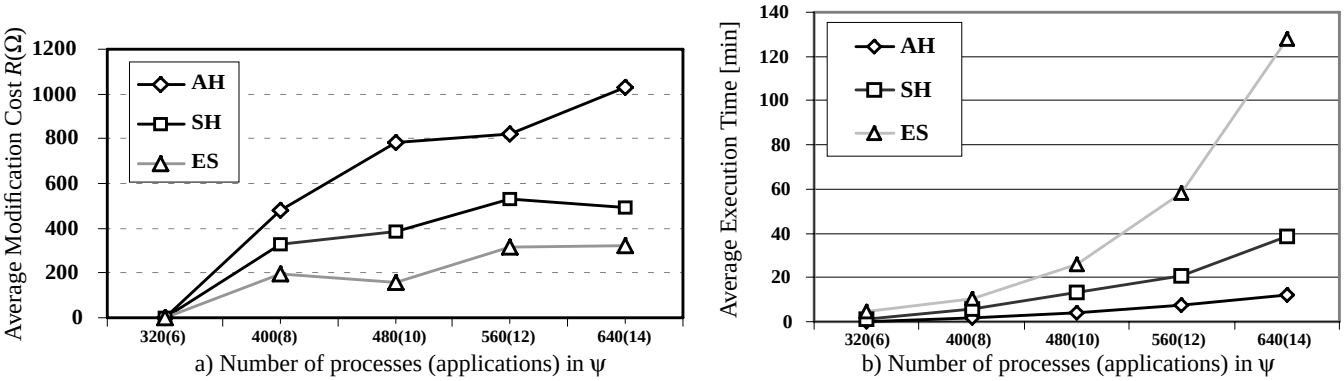


Figure 6. Average Modification Cost (a) and Execution Time (b) for MS with the AH, SH and ES Approaches to Subset Selection

for the next process, with the current larger value of *S*, as long as the resulted schedule has an $S \leq D$.

Figure 7 shows the number of successful implementations in the two cases. In the case $\Gamma_{current}$ has been mapped with MS*, this means using the design criteria and metrics proposed in the paper, we were able to find a valid solution for 65% of the total process graphs considered. However, using AM to map $\Gamma_{current}$ has led to a situation where IMS is able to find schedules which satisfy the deadlines for only 27.5% cases. When $\Gamma_{current}$ grows to 160 processes, only MS* 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.

If the mapping heuristic is allowed to modify the existing system, as discussed in this paper, then we are able to increase the total number of successfully mapped applications Γ_{future} from 65% with MS* to 77.5% with MS. For a $\Gamma_{current}$ with 160 processes the increase is from 60% to 92%. Such an increase is, of course, expected. The important aspect, however, is that it is obtained not by randomly selecting old applications to be remapped, but by performing this selection such that the total modification cost is minimized.

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 MS* we were able to later map 21 of the future applications, while using AM only 4 of the future applications could be mapped. When modifications of the current system were allowed, using MS, we are able to map 24 of the 30 future applications.

6. CONCLUSIONS

We have presented an approach to the incremental design of distributed hard real-time embedded systems. Such a design process satisfies two main requirements when adding new functionality: already running applications are disturbed as little as possible, and there is a good chance that, later, new functionality can easily be mapped on the resulted system. Our approach assumes 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 strategy to solutions supporting an incremental design process. Three algorithms have been proposed to produce a minimal subset of applications which have to be remapped and scheduled in order to implement the new functionality. ES is based on a, potentially slow, branch and bound strategy which finds an optimal solution. AH is very fast but produces solutions that could be of too high cost, while SH is able to quickly produce good quality results. The approach has been validated through several experiments.

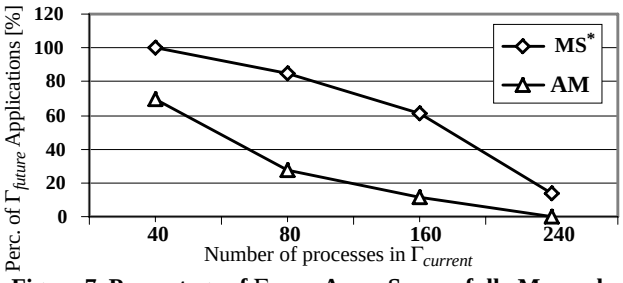


Figure 7. Percentage of Γ_{future} Apps. Successfully Mapped

REFERENCES

[1] T. Blicke, J. Teich, L. Thiele, “System-Level Synthesis Using Evolutionary Algorithms”, Des. Aut. for Emb. Syst., V4, N1, 1998, 23-58.

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

[3] P. Eles, A. Doboli, P. Pop, Z. Peng, “Scheduling with Bus Access Optimization for Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, October 2000.

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

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

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

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

[8] C. Lee, M. Potkonjak, W. Wolf, "Synthesis of Hard Real-Time Application Specific Systems", Des. Aut. for Emb. Syst., V4, N4, 1999, 215-241.

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

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

[11] P. Pop, P. Eles, T. Pop, Z. Peng, “An Approach to Incremental Design of Distributed Embedded Systems,” Submitted to the Design Automation Conference, 2001.

[12] S. Prakash, A. Parker, “SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems”, Journal of Parallel and Distrib. Comp., V16, 1992, 338-351.

[13] D. L. Rhodes, Wayne Wolf, “Co-Synthesis of Heterogeneous Multiprocessor Systems using Arbitrated Communication”, Proceeding of the 1999 International Conference on CAD, 1999, 339 - 342.

[14] T. Y. Yen, W. Wolf, "Hardware-Software Co-Synthesis of Distributed Embedded Systems", Kluwer Academic Publ., 1997.