Bus Access Optimization for Distributed Embedded Systems Based on Schedulability Analysis

Paul Pop, Petru Eles, Zebo Peng
Dept. of Computer and Information Science,
Linköping University, Sweden

Abstract

We present an approach to bus access optimization and schedulability analysis for the synthesis of hard real-time distributed embedded systems. The communication model is based on a time-triggered protocol. We have developed an analysis for the communication delays proposing four different message scheduling policies over a time-triggered communication channel. Optimization strategies for the bus access scheme are developed, and the four approaches to message scheduling are compared using extensive experiments.

1. Introduction

In this paper we concentrate on bus access optimization and schedulability analysis for the synthesis of embedded hard real-time systems which are implemented on distributed architectures consisting of multiple programmable processors and ASICs. Process scheduling is based on a static priority preemptive approach while the bus communication is statically scheduled.

Process scheduling for performance estimation and synthesis of embedded systems has been intensively researched in the last years. Preemptive scheduling of independent processes with static priorities, running on single processor architectures has its roots in [8]. The approach has been later extended to accommodate more general computational models and has been also applied to distributed systems [17, 20]. Static non-preemptive scheduling of a set of processes on a multiprocessor architecture has been discussed in [4, 5]. Several approaches consider architectures consisting of a single programmable processor and an ASIC. Under such circumstances deriving a static schedule for the software component practically means the linearization of the dataflow graph [3].

Although different scheduling strategies have been adapted to accommodate distributed architectures, researchers have often ignored or very much simplified aspects concerning the communication infrastructure. One typical solution is to consider communication tasks as processes with a given execution time (depending on the amount of information transferred) and to schedule them as any other process [4, 20], without considering issues like communication protocol, bus arbitration, packaging of messages, and clock synchronization.

Previous works related to communication synthesis, like [1, 2, 6, 10, 11], are dealing with lower level aspects of hardware-software communication and are not addressing problems specific to distributed real-time embedded systems, based on particular communication protocols.

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints. Thus, in order to guarantee that real-time requirements are fulfilled, schedulability analysis has been done for some communication protocols [16, 18].

In this paper we consider the time-triggered protocol (TTP) [7] as the communication infrastructure for a distributed real-time system. Processes are scheduled according to a static priority preemptive policy. TTP is well suited for safety critical distributed real-time embedded systems and represents one of the emerging standards for several application areas like, for example, automotive electronics [19].

Our first contribution is to develop the schedulability analysis in the above context, considering four different approaches to message scheduling. After this, as a second contribution, we show how the bus access scheme can be optimized in order to fit the communication particularities of a certain application.

In [12] we have addressed the issue of non-preemptive static process scheduling and communication synthesis using TTP. However, considering preemptive priority based scheduling at the process level, with time triggered static scheduling at the communication level can be the right solution under several circumstances [9]. A communication protocol like TTP provides a global time base, and improves fault-tolerance and predictability. At the same time, certain particularities of the application or of the underlying real-time operating system very often impose a priority based scheduling policy at the process level.

The paper is divided into 7 sections. The next section presents the architectures considered for system implementation. The computational model assumed and formulation of the problem are presented in section 3, and section 4 presents the schedulability analysis for each of the four approaches considered for message scheduling. The optimization strategy is presented in section 5, and the four approaches are evaluated in section 6. The last section presents our conclusions.

2. System Architecture

2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a TTP controller, a CPU, a RAM, a ROM and an I/O interface to sensors and actuators. A node can also have an ASIC in order to accelerate parts of its functionality.

Communication between nodes is based on the TTP [7]. TTP was designed for distributed real-time applications that require predictability and reliability (e.g, drive-by-wire). It integrates all the 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 time-division multiple-access (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

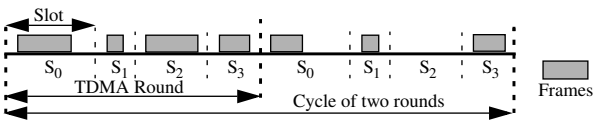

Figure 1. Bus Access Scheme

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. The sequence and length of the slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

Every node has a TTP controller that implements the protocol services, and runs independently of the node's CPU (Figure 2). Communication with the CPU is performed through a so called message base interface (MBI) which is usually implemented as a dual ported RAM.

The TDMA access scheme is imposed by a so called message descriptor list (MEDL) that is located in every TTP controller. The MEDL basically contains: the time when a frame has to be sent or received, the address of the frame in the MBI and the length of the frame. MEDL serves as a schedule table for the TTP controller which has to know when to send or receive a frame to or from the communication channel.

The TTP controller provides each CPU with a timer interrupt based on a local clock, synchronized with the local clocks of the other nodes. The clock synchronization is done by comparing the a priori known time of arrival of a frame with the observed arrival time. Thus, TTP provides a global time-base of known precision, without any overhead on the communication.

2.2 Software Architecture

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 so called tick scheduler that is activated periodically by the timer interrupts and decides on activation of processes, based on their priorities. Several activities, like polling of the I/O or diagnostics, take also place in the timer interrupt routine.

In order to run a predictable hard real-time application, the overhead of the kernel and the worst case administrative overhead (WCAO) of every system call has to be determined. Our schedulability analysis takes into account these overheads, and also the overheads due to the message passing.

The message passing mechanism is illustrated in Figure 2, where we have three processes, P_1 to P_3. P_1 and P_2 are mapped to node N_0 that transmits in slot S_0, and P_3 is mapped to node N_1 that transmits in slot S_1. Message m_1 is transmitted between P_1 and P_2 that are on the same node, while message m_2 is transmitted from P_1 to P_3 between the two nodes.

Messages between processes located on the same processor are passed through shared protected objects. The overhead for their communication is accounted for by the blocking factor, computed according to the priority ceiling protocol [14].

Message m_2 has to be sent from node N_0 to node N_1. Thus, after m_2 is produced by P_1, it will be placed into an outgoing message queue, called *Out*. The access to the queue is guarded by a priority-ceiling semaphore. A so called transfer process (denoted with T in Figure 2) moves the message from the *Out* queue into the MBI.

How the message queue is organized and how the message transfer process selects the particular messages and assembles them into a frame, depends on the particular approach chosen for message scheduling (see Section 4). The message transfer process is activated at certain a priori known moments, by the tick scheduler in order to perform the message transfer. These activation times are stored in a message handling time table (MHTT) available to the real-time kernel in each node. Both the MEDL and the MHTT are generated off-line as result of the schedulability analysis and optimization which will be discussed later. The MEDL imposes the times when the TTP controller of a certain node has to move frames from the MBI to the communication channel. The MHTT contains the times when messages have to be transferred by the message transfer process from the *Out* queue into the MBI, in order to further be broadcasted by the TTP controller. As result of this synchronization, the activation times in the MHTT are directly related to those in the MEDL and the first table results directly form the second one.

It is easy to observe that we have the most favourable situation when, at a certain activation, the message transfer process finds in the *Out* queue all the "expected" messages which then can be packed into the just following frame to be sent by the TTP controller. However, application processes are not statically scheduled and availability of messages in the *Out* queue can not be guaranteed at fixed times. Worst case situations have to be considered, as will be shown in Section 4.

Let us come back to Figure 2. There we assumed a context in which the broadcasting of the frame containing message m_2 is done in the slot S_0 of Round 2. The TTP controller of node N_1 knows from its MEDL that it has to read a frame from slot S_0 of Round 2 and to transfer it into its MBI. In order to synchronize with the TTP controller and to read the frame from the MBI, the tick scheduler on node N_1 will activate, based on its local MHTT, a so called delivery process, denoted with D in Figure 2. The delivery process takes the frame from the MBI, and extracts the messages from it. For the case when a message is split into several packets, sent over several TDMA rounds, we consider that a message has arrived at the destination node after all its corresponding packets have arrived. When m_2 has arrived, the delivery process copies it to process P_3 which will be activated. Activation times for the delivery process are fixed in the MHTT just as explained earlier for the message transfer process.

The number of activations of the message transfer and delivery processes depend on the number of frames transferred, and they are taken into account in our analysis, as well as the delay implied by the propagation on the communication bus.

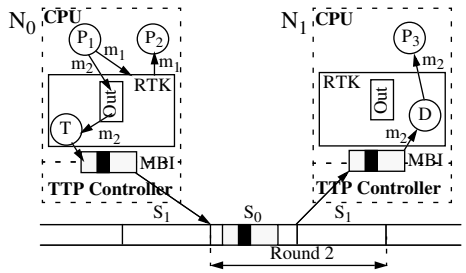

Figure 2. Message Passing Mechanism

3. Problem Formulation

We model an application as a set of processes. Each process p_i is allocated to a certain processor, has a known worst-case execution time C_i, a period T_i, a deadline D_i and a uniquely assigned priority. We consider a preemptive execution environment, which means that higher priority processes can interrupt the execution of lower priority processes. A lower priority process can block a higher priority process (e.g., it is in its critical section), and the blocking time is computed according to the priority ceiling protocol. Processes exchange messages, and for each message m_i we know its size S_{mi}. A message is sent once in every n_m invocations of the sending process, and has a unique destination process. Each process is allocated to a node of our distributed architecture, and the messages are transmitted according to the TTP.

We are interested to synthesize the MEDL of the TTP controllers (and as a direct consequence, also the MHTTs) so that the process set is schedulable on an as cheap (slow) as possible processor set.

4. Schedulability Analysis

Under the assumptions presented in the previous section Tindell et al. [17] integrate processor and communication schedulability and provide a "holistic" schedulability analysis in the context of distributed real-time systems with communication based on a simple TDMA protocol. The basic idea is that the release jitter of a destination process depends on the communication delay between sending and receiving a message. The release jitter of a process is the worst case delay between the arrival of the process and its release (when it is placed in the run-queue for the processor). The communication delay is the worst case time spent between sending a message and the message arriving at the destination process.

Thus, for a process d(m) that receives a message m from a sender process s(m), the release jitter is: $J_{d(m)} = r_{s(m)} + a_m + r_{deliver} + T_{tick}$, where $r_{s(m)}$ is the response time of the process sending the message, a_m (worst case arrival time) is the worst case time needed for message m to arrive at the communication controller of the destination node, $r_{deliver}$ is the response time of the delivery process (see section 2.2), and T_{tick} is the jitter due to the operation of the tick scheduler. The communication delay for a message m is $C_m = a_m + r_{deliver}$. a_m itself is the sum of the access delay and the propagation delay. The access delay is the time a message queued at the sending processor spends waiting for the use of the communication channel. In a_m we also account for the execution time of the message transfer process (see section 2.2). The propagation delay is the time taken for the message to reach the destination processor once physically sent by the corresponding TTP controller.

The worst case time, message m takes to arrive at the communication controller of the destination node is determined in [17] using the arbitrary deadline analysis, and is given by: $a_m = \max_{q = 0, 1, 2, \ldots} (w_m(q) + X_m(q) - qT_m)$, where the term $w_m(q) - qT_m$ is the access delay, $X_m(q)$ is the propagation delay, and T_m is the period of the message.

In [17] an analysis is given for the end-to-end delay of a message m in the case of a simple TDMA protocol. For this case, $w_m(q) = \left\lceil \frac{(q+1)P_m + I_m(w(q))}{S_p} \right\rceil T_{TDMA}$, where P_m is the number of packets of message m, S_P is the size of the slot (in number of packets) corresponding to m, and I_m is the interference caused by packets belonging to messages of a higher priority than m. Although there are many similarities with the general TDMA protocol, the analysis in the case of TTP is different in several aspects and also differs to a large degree depending on the policy chosen for message scheduling.

Before going into details for each of the message scheduling approaches, we analyze the propagation delay and the message transfer and delivery processes, as they do not depend on the particular message scheduling policy chosen. The propagation delay X_m of a message m sent as part of a slot S, with the TTP protocol, is equal to the time needed for the slot S to be transferred on the bus. This time depends on the slot size and on the features of the underlying bus.

The overhead produced by the communication activities must be accounted for not only as part of the access delay for a message, but also through its influence on the response time of processes running on the same processor. We consider this influence during the schedulability analysis of processes on each processor. We assume that the worst case computation time of the transfer process (T in Figure 2) is known, and that it is different for each of the four message scheduling approaches. Based on the respective MHTT, the transfer process is activated for each frame sent. Its worst case period is derived form the minimum time between successive frames.

The response time of the delivery process (D in Figure 2), $r_{deliver}$, is part of the communication delay. The influence due to the delivery process must be also included when analyzing the response time of the processes running on the respective processor. We consider the delivery process during the schedulability analysis in the same way as the message transfer process.

The response times of the communication and delivery processes are calculated, as for all other processes, using the arbitrary deadline analysis from [17].

The four approaches we have considered for scheduling of messages using TTP differ in the way the messages are allocated to the communication channel (either statically or dynamically) and whether they are split or not into packets for transmission. The next subsections present an analysis for these approaches as well as the degrees of liberty a designer has, in each of the cases, when synthesizing the MEDL.

4.1 Static Single Message Allocation (SM)

The first approach to scheduling of messages using TTP is to statically (off-line) schedule each of the messages into a slot of the TDMA cycle, corresponding to the node sending the message. We also consider that the slots can hold each at maximum one single message. This approach is well suited for application areas (like automotive electronics) where the messages are typically short and the ability to easily diagnose the system is critical.

As each slot carries only one fixed, predetermined message, there is no interference among messages. If a message m misses its slot it has to wait for the following slot assigned to m. The

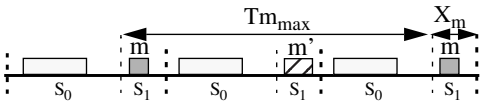

Figure 3. Worst case arrival time for SM

access delay for a message m in this approach is the maximum time between consecutive slots of the same node carrying the message m. We denote this time by Tm_{max}, illustrated in Figure 3.

In this case, the worst case arrival time a_m of a message m becomes $Tm_{max}+ X_m$. Therefore, the main aspect influencing the schedulability analysis for the messages is the way the messages are statically allocated to slots, resulting different values for Tm_{max}. Tm_{max}, as well as X_m, depend on the slot sizes which in the case of SM are determined by the size of the largest message sent from the corresponding node, plus the bits for control and CRC, as imposed by the protocol.

During the synthesis of the MEDL, the designer has to allocate the messages to slots in such a way that the process set is schedulable. Since the schedulability of the process set can be influenced by the synthesis of the MEDL only through the Tm_{max} parameters, these parameters have to be optimized.

Let us consider the simple example depicted in Figure 4, where we have three processes, p_1, p_2, and p_3 running each on different processors. When process p_1 finishes executing it sends message m_1 to process p_3 and message m_2 to process p_2. In the TDMA configuration presented in Figure 4 a), only the slot for the CPU running p_1 is important for our discussion and the other slots are represented with light gray. With this configuration, where the message m_1 is allocated to the rounds 1 and 4 and the message m_2 is allocated to rounds 2 and 3, process p_2 misses its deadline because of the release jitter due to the message m_2 in round 2. However, if we have the TDMA configuration depicted in Figure 4 b), where m_1 is allocated to the rounds 2 and 4 and m_2 is allocated to the rounds 1 and 3, then all the processes meet their deadlines.

4.2 Static Multiple Message Allocation (MM)

This second approach is an extension of the first one. In this approach we allow more than one message to be statically assigned to a slot, and all the messages transmitted in the same slot are packaged together in a frame. In this case there is also no interference, so the access delay for a message m is the same as for the first approach, namely, the maximum time between consecutive slots of the same node carrying the message m, Tm_{max}.

However, this approach offers more freedom during the synthesis of the MEDL. We have now to decide also on how many and which messages should be put in a slot. This allows more flexibility in optimizing the Tm_{max} parameter. To illustrate this, let us consider the same example depicted in Figure 4. With the MM approach, the TDMA configuration can be arranged as depicted in Figure 4 c), where the messages m_1 and m_2 are put together in the same slot in the rounds 1 and 2. Thus, the deadline is met, and the release jitter is further reduced compared to the case presented in Figure 4 b) where the deadlines were also met but the process p_3 was experiencing large release jitter.

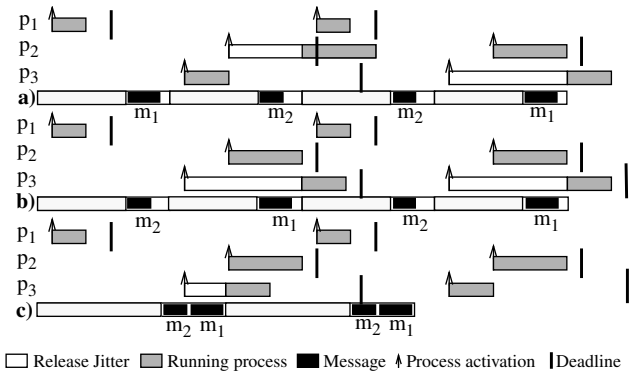

Figure 4. Optimizing the MEDL for SM and MM

4.3 Dynamic Message Allocation (DM)

The previous two approaches have statically allocated one or more messages to their corresponding slots. This third approach considers that the messages are dynamically allocated to frames, as they are produced.

Thus, when a message is produced by a sender process it is placed in the *Out* queue ordered according to the priorities of the messages. At its activation, the message transfer process takes a certain number of messages from the head of the *Out* queue and constructs the frame. The number of messages accepted is decided so that their total size does not exceed the length of the data field of the frame. This length is limited by the size of the slot corresponding to the respective processor. Since the messages are sent dynamically, we have to identify them in a certain way so that they are recognized when the frame arrives at the delivery process. We consider that each message has several identifier bits appended at the beginning of the message.

Since we dynamically package the messages into frames in the order they are sorted in the queue, the access delay to the communication channel for a message m depends on the number of messages queued ahead of it.

The analysis in [17] bounds the number of queued ahead *packets* of messages of higher priority than message m, as in their case it is considered that a message can be split into packets before it is transmitted on the communication channel. We use the same analysis, but we have to apply it for the number of *messages* instead that of packets. We have to consider that messages can be of different sizes as opposed to packets which always are of the same size.

Therefore, the total *size* of higher priority messages queued ahead of a message m in a window *w* is:

$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil S_j$ where S_j is the size of the message m_j, $r_{s(j)}$ is the response time of the process sending message m_j, and T_j is the period of the message m_j.

Further, we calculate the worst case time that a message m spends in the *Out* queue. The number of TDMA rounds needed, in the worst case, for a message m placed in the queue to be removed from the queue for transmission is $\left\lceil \frac{S_m + I_m}{S_s} \right\rceil$ where S_m is the size of the message m and S_s is the size of the slot transmitting m (we assume, in the case of DM, that for any message x, $S_x \le S_s$). This means that the worst case time a message m spends in the *Out* queue is given by $\left\lceil \frac{S_m + I_m}{S_s} \right\rceil T_{TDMA}$, where T_{TDMA} is the time taken for a TDMA round.

To determine the term $w_m(q) - qT_m$ that gives the access delay (see Section 4), $w_m(q)$ is determined, using the arbitrary deadline analysis, as being:

$w_m(q) = \left\lceil \frac{(q+1)S_m + I_m(w(q))}{S_s} \right\rceil T_{TDMA}$. Since the size of the messages is given with the application, the parameter that will be optimized during the synthesis of the MEDL is the slot size. To illustrate how the slot size influences the schedulability, let us consider the example in Figure 5 a), where we have the same setting as for the example in Figure 4 a). The difference is that we consider message m_1 having a higher priority than message m_2, and we schedule dynamically the messages as they are produced. With the TDMA configuration in Figure 5 a) message m_1 will be dynamically scheduled first in the slot of the first round, while message m_2 will wait in the *Out* queue until the next round comes, thus causing the process p_2 to miss its deadline. However, if we enlarge the slot so that it can accommodate both messages, message m_2 does not have to wait in the queue and it is transmitted in the same slot as m_1. Therefore p_2 will meet its deadline, as presented in Figure 5 b). However, in general, increasing the length of slots does not necessarily improve the schedulability, as it delays the communication of messages generated by other nodes.

4.4 Dynamic Packets Allocation (DP)

This approach is an extension of the previous one, as we allow the messages to be split into packets before they are transmitted on the communication channel. We consider that each slot has a size that accommodates a frame with the data field being a multiple of the packet size. This approach is well suited for the application areas that typically have large message sizes, and by splitting them into packets we can obtain a higher utilization of the bus and reduce the release jitter. However, since each packet has to be identified as belonging to a message, and messages have to be split at the sender and reconstructed at the destination, the overhead becomes higher than in the previous approaches.

For the analysis we use the formula from [17] which is based on similar assumptions as those for this approach:

$w_m(q) = \left\lceil \frac{(q+1)P_m + I_m(w(q))}{S_p} \right\rceil T_{TDMA}$, where P_m is the number of packets of message m, S_P is the size of the slot (in number of packets) corresponding to m, and

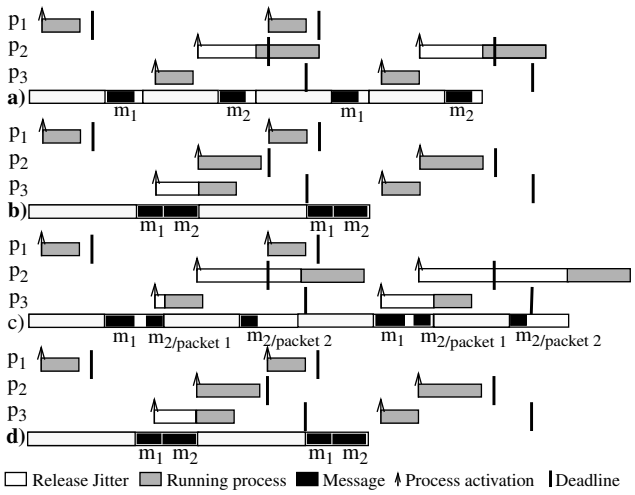

Figure 5. Optimizing the MEDL for DM and DP

$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil P_j$, where P_j is the number of packets of a message m_j.

In the previous approach (DM) the optimization parameter for the synthesis of the MEDL was the size of the slots. Within this approach we can also decide on the packet size, which becomes another optimization parameter. Consider the example in Figure 5 c) where messages m_1 and m_2 have a size of 6 bytes each. The packet size is considered to be 4 bytes and the slot corresponding to the messages has a size of 12 bytes (3 packets) in the TDMA configuration. Since message m_1 has a higher priority than m_2, it will be dynamically scheduled first in the slot of the first round, and it will need 2 packets. In the remaining packet, the first 4 bytes of m_2 are scheduled. Thus, the rest of 2 bytes from message m_2 have to wait for the next round, causing the process p_2 to miss its deadline. However, if we change the packet size to 3 bytes, and keep the same size of 12 bytes for the slot, we now have 4 packets in the slot corresponding to the CPU running p_1 (Figure 6 d). Message m_1 will be dynamically scheduled first, and will take 2 packets from the slot of the first round. This will allow us to send m_2 in the same round, therefore meeting the deadline for p_2.

In this particular example, with one single sender processor and the particular message and slot sizes as given, the problem seems to be simple. This is, however, not the case in general. For example, the packet size which fits a particular node can be unsuitable in the context of the messages and slot size corresponding to another node. At the same time, reducing the packets size increases the overheads due to the transfer and delivery processes.

5. Optimization Strategy

Our problem is to analyze the schedulability of a given process set and to synthesize the MEDL of the TTP controllers (and consequently the MHTTs) so that the process set is schedulable on an as cheap as possible architecture. The MEDL is synthesized according to the optimization parameters available for each of the four approaches to message scheduling discussed before. In order to guide the optimization process, we need a cost function that captures the "degree of schedulability" for a certain MEDL implementation. Our cost function is a modified version of that in [15]:

$$\text{cost function} = \begin{cases} f_1 = \sum_{i=1}^{n} max(0, R_i - D_i), \text{ if } f_1 > 0 \\ f_2 = \sum_{i=1}^{n} R_i - D_i, \text{ if } f_1 = 0 \end{cases}$$

where n is the number of processes in the application, R_i is the response time of a process p_i, and D_i is the deadline of a process p_i. If the process set is not schedulable, there exists at least one R_i that is greater than the deadline D_i, therefore the term f_1 of the function will be positive. In this case the cost function is equal to f_1. However, if the process set is schedulable, then all R_i are smaller than the corresponding deadlines D_i. In this case $f_1 = 0$ and we use f_2 as the cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimization parameters leading to a schedulable process set, a smaller f_2 means that we have improved the response times of the processes, so the application can be potentially implemented on a cheaper hard-

```
OptimizeSM
  for each node N_i do // set the slot sizes
    size_Si = max(size of messages m_j sent by node N_i)
  end for
  for each node N_i do // find the min. no. of rounds that can hold...
    nm_i = number of messages sent from N_i // ...all the messages
  end for
  min_rounds = max (nm_i)
  for each message m_i // create a minimal complete MEDL
    find round in [1..min_rounds] that has an empty slot for m_i
    place m_i into its slot in round
  end for
  for each rounds_no in [min_rounds...max_rounds] do
    repeat // insert messages in such a way that the cost is minimized
      for each process p_i that receives a message m_i do
        if D_i - R_i is the smallest so far then m = m_pi end if
      end for
      for each round in [1..rounds_no] do
        place m into its corresponding slot in round
        calculate the cost_function
        if the cost_function is smallest so far then
          best_round = round
        end if
        remove m from its slot in round
      end for
      place m into its slot in best_round if one was identified
    until the cost_function is not improved
  end for
end OptimizeSM
```

Figure 6. Greedy Heuristic for SM

ware architecture (with slower processors and/or bus). The release time R_i is calculated according to the arbitrary deadline analysis [17] based on the release jitter of the process (see section 4), its worst-case execution time, the blocking time, and the interference time due to higher priority processes.

For a given application, we are interested to synthesize a MEDL such that the cost function is minimized. We are also interested to evaluate in different contexts the four approaches to message scheduling, thus offering the designer a decision support for choosing the approach that best fits his application.

The MEDL synthesis problem belongs to the class of combinatorial problems, therefore we are interested to develop heuristics that are able to find accurate results in a reasonable time. We have developed optimization algorithms corresponding to each of the four approaches to message scheduling. A first set of algorithms is based on simple and fast greedy heuristics. A second class of heuristics aims at finding near-optimal solutions using the simulated annealing (SA) algorithm.

The greedy heuristic differs for each of the four approaches to message scheduling. The main idea is to improve the 'degree of schedulability' of the process set by incrementally trying to reduce the release jitter of the processes.

The only way to reduce the release jitter in the SM and MM approaches is through the optimization of the Tm_{max} parameters. This is achieved by a proper placement of messages into slots (see Figure 4).

The OptimizeSM algorithm presented in Figure 6, starts by deciding on a size *($size_{Si}$)* for each of the slots. There is nothing to be gained by enlarging the slot size, since in this approach a slot can carry at most one message. Thus, the slot sizes are set to the minimum size that can accommodate the largest message sent by the corresponding node.

Then, the algorithm has to decide on the number of rounds, thus determining the size of the MEDL. Since the size of the MEDL is physically limited, there is a limit to the number of rounds (e.g., 2, 4, 8, 16 depending on the particular TTP controller implementation). However, there is a minimum number of rounds *min_rounds* that is necessary for a certain application, which depends on the number of messages transmitted. For example, if the processes mapped on node N_0 send in total 7 messages, then we have to decide on at least 7 rounds in order to accommodate all of them (in the SM approach there is at most one message per slot). Several numbers of rounds, *rounds_no*, are tried out by the algorithm starting from *min_rounds* up to *max_rounds*.

For a given number of rounds (that determine the size of the MEDL) the initially empty MEDL has to be populated with messages in such a way that the cost function is minimized. In order to apply the schedulability analysis that is the basis for the cost function, a *complete* MEDL has to be provided. A complete MEDL contains at least one instance of every message that has to be transmitted between the processes on different processors. A *minimal complete* MEDL is constructed from an empty MEDL by placing one instance of every message m_i into its corresponding empty slot of a *round*. In Figure 4 a), for example, we have a MEDL composed of four rounds. We get a minimal complete MEDL, for example, by assigning m_2 and m_1 to the slots in rounds 3 and 4, and letting the slots in rounds 1 and 2 empty. However, such a MEDL might not lead to a schedulable system. The 'degree of schedulability' can be improved by inserting instances of messages into the available places in the MEDL, thus minimizing the Tm_{max} parameters. For example, in Figure 4 a) by inserting another instance of the message m_1 in the first round and m_2 in the second round leads to p_2 missing its deadline, while in Figure 4 b) inserting m_1 into the second round and m_2 into the first round leads to a schedulable system.

Our algorithm repeatedly adds a new instance of a message to the current MEDL in the hope that the cost function will be improved. In order to decide an instance of which message should be added to the current MEDL, a simple heuristic is used. We identify the process p_i which has the most "critical" situation, meaning that the difference between its deadline and response time, D_i - R_i, is minimal compared with all other processes. The message to be added to the MEDL is the message m=m_{pi} received by the process p_i. Message m will be placed into that round *(best_round)* which corresponds to the smallest value of the cost function. The algorithm stops if the cost function can not be further improved by adding more messages to the MEDL.

The OptimizeMM algorithm is similar to OptimizeSM. The main difference is that in the MM approach several messages can be placed into a slot (which also decides its size), while in the SM approach there can be at most one message per slot. Also, in the case of MM, we have to take additional care that

```
OptimizeDM
  for each node N_i do
    min_size_Si = max(size of messages m_j sent by node N_i)
  end for
  for each slot S_i // identifies the size that minimizes the cost function
    best_size_Si = min_size_Si
    for each slot_size in [min_size_Si...max_size] do
      calculate the cost_function
      if the cost_function is best so far then
        best_size_Si = slot_size_Si
      end if
    end for
    size_Si = best_size_Si
  end for
end OptimizeDM
```

Figure 7. Greedy Heuristic for DM

the slots do not exceed the maximum allowed size for a slot.

The situation is simpler for the dynamic approaches, namely DM and DP, since we only have to decide on the slot sizes and, in the case of DP, on the packet size. For these two approaches, the placement of messages is dynamic and has no influence on the cost function. The OptimizeDM algorithm (see Figure 7) starts with the first slot $S_i = S_0$ of the TDMA round and tries to find that size $(best_size_{S_i})$ which corresponds to the smallest *cost_function*. This slot size has to be large enough ($S_i \geq min_size_{S_i}$) to hold the largest message to be transmitted in this slot, and within bounds determined by the particular TTP controller implementation (e.g., from 2 bits up to max_size = 32 bytes). Once the size of the first slot has been determined, the algorithm continues in the same manner with the next slots.

The OptimizeDP algorithm has, in addition, to determine the proper packet size. This is done by trying all the possible packet sizes given the particular TTP controller. For example, it can start from 2 bits and increment with the "smallest data unit" (typically 2 bits) up to 32 bytes. In the case of the OptimizeDP algorithm the slot size is a multiple of the packet size, and it is within certain bounds depending on the TTP controller.

We have also developed an SA based algorithm for bus access optimization corresponding to each of the four message scheduling approaches [13]. In order to tune the parameters of the algorithm we have first performed very long and expensive runs on selected large examples, and the best ever solution, for each example, has been considered as the near-optimum. Based on further experiments we have determined the parameters of the SA algorithm, for different sizes of examples, so that the optimization time is reduced as much as possible but the near-optimal result is still produced. These parameters have then been used in the large scale experiments presented in the following section.

6. Experimental Results

For evaluation of our approaches we first used sets of processes generated for experimental purpose. We considered architectures consisting of 2, 4, 6, 8 and 10 nodes. 40 processes were assigned to each node, resulting in sets of 80, 160, 240, 320 and 400 processes. 30 sets were generated for each dimension, thus a total of 150 sets of processes were used for experimental evaluation. Worst case computation times, periods, deadlines, and message lengths were assigned randomly within certain intervals. For the communication channel we considered a transmission speed of 256 kbps. The maximum length of the data field in a slot was 32 bytes, and the frequency of the TTP controller was chosen to be 20 MHz. All experiments were run on a Sun Ultra 10 workstation.

For each of the 150 generated examples and each of the four scheduling approaches we have obtained, using our optimization strategy, the near-optimal values for the cost function, produced by our SA based algorithm. These values, for a given example, might differ from one approach to another, as they depend on the optimization parameters and the schedulability analysis determined for each of the approaches. We were interested to compare the four approaches to message scheduling based on the values obtained for the cost function.

Thus, Figure 8 a) presents the average percentage deviations of the cost function obtained in each of the four approaches, from the minimal value among them. The DP approach is generally the most performant, and the reason for this is that dynamic scheduling of messages is able to reduce release jitter because no space is waisted in the slots if the packet size is properly selected. However, by using the MM approach we can obtain almost the same result if the messages are carefully allocated to slots by our optimization strategy. Moreover, in the case of bigger sets of processes (e.g., 400) MM outperforms DP, as DP suffers form large overhead due to the handling of the packets. DM performs worse than DP because it does not split the messages into packets, and this results in a mismatch between the size of the messages dynamically queued and the slot size, leading to unused slot space that increases the jitter. SM performs the worst as it does not permit much room for improvement, leading to large amounts of unused slot space. Also, DP has produced a MEDL that resulted in schedulable process sets for 1.33 times more cases than the MM and DM. MM, in its turn, produced two times more schedulable results than the SM approach.

Together with the four approaches to message scheduling, a so called ad-hoc approach is presented. The ad-hoc approach performs scheduling of messages without trying to optimize the access to the communication channel. The ad-hoc solutions are based on the MM approach and consider a design with the TDMA configuration consisting of a simple, straightforward, allocation of messages to slots. The lengths of the slots were selected to accommodate the largest message sent from the respective node. Figure 8 a) shows that the ad-hoc alternative is constantly outperformed by any of the optimized solutions. This shows that by optimizing the access to the communication channel, significant improvements can be produced.

Next, we have compared the four approaches with respect to the number of messages exchanged between different nodes

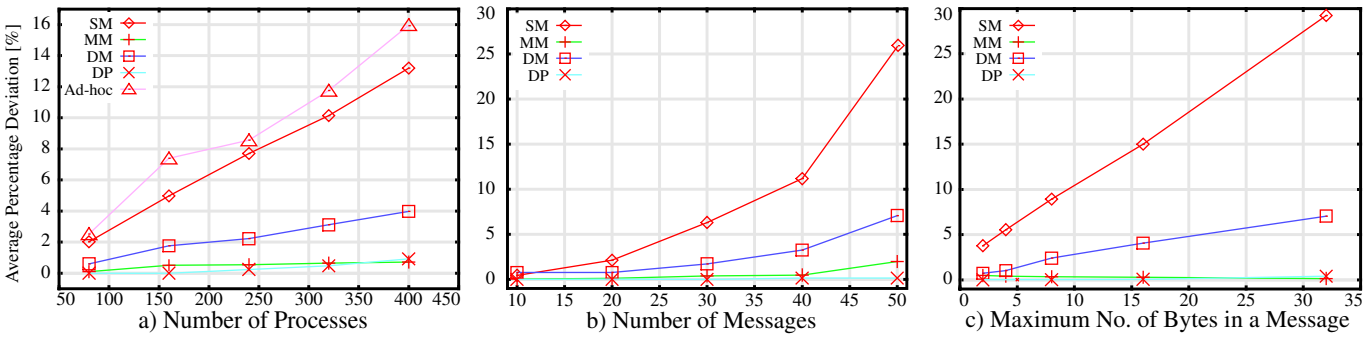

Figure 8: Comparison of the Four Approaches to Message Scheduling

and the maximum message size allowed. For the results depicted in Figure 8 b) and 8 c) we have assumed sets of 80 processes allocated to 4 nodes. Figure 8 b) shows that as the number of messages increases, the difference between the approaches grows while the ranking among them remains the same. The same holds for the case when we increase the maximum allowed message size (Figure 8 c), with a notable exception: for large message sizes MM becomes better than DP, since DP suffers from the overhead due to packet handling.

We were also interested in the quality of our greedy heuristics. Thus, we have run all the examples presented above, using the greedy heuristics and compared the results with those produced by the SA based algorithm. The table below presents the average and maximum percentage deviations of the cost function for each of the graph dimensions.

Optimize		80 procs.	160 procs.	240 procs.	320 procs.	400 procs.
SM	aver.	0.12%	0.19%	0.50%	1.06%	1.63%
	max.	0.81%	2.28%	8.31%	31.05%	18.00%
MM	aver.	0.05%	0.04%	0.08%	0.23%	0.36%
	max.	0.23%	0.55%	1.03%	8.15%	6.63%
DM	aver.	0.02%	0.03%	0.05%	0.06%	0.07%
	max.	0.05%	0.22%	0.81%	1.67%	1.01%
DP	aver.	0.01%	0.01%	0.05%	0.04%	0.03%
	max.	0.05%	0.13%	0.61%	1.42%	0.54%

All the four greedy heuristics perform very well, with less than 2% loss in quality compared to the results produced by the SA algorithms. The execution times for the greedy heuristics were more than two orders of magnitude smaller than those with SA.

Finally, we have considered a real-life example implementing an aircraft control system adapted from [17] where the ad-hoc solution and the SM approach failed to produce a schedulable solution. However, with the other two approaches schedulable solutions were produced, DP generating the smallest cost function followed in this order by MM and DM.

The above comparison between the four message scheduling alternatives is mainly based on the issue of schedulability. However, when choosing among the different policies, several other parameters can be of importance. Thus, a static allocation of messages can be beneficial from the point of view of testing and debugging and has the advantage of simplicity. Similar considerations can lead to the decision not to split messages. In any case, however, optimization of the bus access scheme is highly desirable.

7. Conclusions

We have presented an approach to static priority preemptive process scheduling for synthesis of hard real-time distributed embedded systems. The communication model is based on a time-triggered protocol. We have developed an analysis for the communication delays and optimization strategies for four different message scheduling policies.

The four approaches to message scheduling were compared using extensive experiments. We showed that by optimizing the bus access scheme, significant improvements can be produced. Our optimization heuristics are able to efficiently produce good quality results.

References

[1] P. H. Chou, R. B. Ortega, G. Boriello, "The Chinook Hardware/Software Co-Synthesis System," Proceedings of the 8th International Symp. on System Synthesis, 1995, 22-27.

[2] J. M. Daveau, T. Ben Ismail, A. A. Jerraya, "Synthesis of System-Level Communication by an Allocation-Based Approach," Proc. 8th Int. Symp. on Syst. Synth., 1995, 150-155.

[3] R. K. Gupta, G. De Micheli, "A Co-Synthesis Approach to Embedded System Design Automation," Design Automation for Embedded Systems, 1(1/2), 1996, 69-120.

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

[5] H. Kasahara, S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," IEEE Transactions on Computers, V33, N11, 1984, 1023-1029.

[6] P. V. Knudsen, J. Madsen, "Integrating Communication Protocol Selection with Hardware/Software Codesign," IEEE Transactions 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. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of the ACM, 20(1), 1973, 46-61.

[9] H. Lonn, J. Axelsson, "A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications," Proceedings of the 11th Euromicro Conf. on Real-Time Systems, 1999, 142-149.

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

[11] R. B. Ortega, G. Borriello, "Communication Synthesis for Distributed Embedded Systems," International Conference on Computer Aided Design, 1998, 437-444.

[12] 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.

[13] P. Pop, P. Eles, Z. Peng, "Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems," Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, 1999.

[14] L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Transactions on Computers, 39(9), 1990, 1175-1185.

[15] K. Tindell, A. Burns, A. J. Wellings, "Allocating Real-Time Tasks (An NP-Hard Problem made Easy)," Real-Time Systems, 4(2), 1992, 145-165.

[16] K. Tindell, A. Burns, A. J. Wellings, "Calculating Controller Area Network (CAN) Message Response Times," Control Eng. Practice, 3(8), 1163-1169, 1995.

[17] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," Microprocessing and Microprogramming, 40, 1994, 117-134.

[18] E. Tovar, F. Vasques, A. Burns, "Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach," Proceedings of the 11th Euromicro Conference on Real-Time Systems, 175-184, 1999.

[19] X-by-Wire Consortium, "X-By-Wire: Safety Related Fault Tolerant Systems in Vehicles," http://www.vmars.tuwien.ac.at/projects/xbywire/

[20] T. Y. Yen, W. Wolf, *Hardware-Software Co-Synthesis of Distributed Embedded Systems*, Kluwer Academic Publishers, 1997.