PRE-RUNTIME SCHEDULING CONSIDERING TIMING AND ENERGY CONSTRAINTS IN EMBEDDED SYSTEMS WITH MULTIPLE PROCESSORS

Eduardo Tavares, Meuse Oliveira Jr, Paulo Maciel, Bruno Souza, Silvino Neto
CIn - UFPE. Recife-PE-Brazil.
{eagt, mnoj, prmm, bs}@cin.ufpe.br, silvinovvneto@yahoo.com.br

Raimundo Barreto, Romulo Freitas, Marcelo Custodio
DCC-UFAM. Manaus-AM-Brazil
{rbarreto, devezas, mmc}@dcc.ufam.edu.br

Abstract In this paper, a pre-runtime scheduling approach for hard real-time embedded systems with multiple processors is presented considering stringent timing and energy constraints. This paper adopts a formal approach, based on time Petri nets, for synthesizing feasible schedules.

1. INTRODUCTION

Some embedded systems are classified as real-time systems, where the correct behavior depends not only on the integrity of the results, but also the time in which such results are produced. In hard real-time systems, if timing constraints are not met, the consequences can be disastrous, including great damage of resources or even loss of human lives. Due to CPU-bound tasks, some hard real-time embedded systems need to rely on multiple processors in order to meet timing constraints.

In addition to timing issues, many hard real-time systems have constraints on autonomy, since, in many cases, they need to be operated in remote areas where energy sources may be highly constrained. Therefore, such systems cannot exceed their respective energy (or power) constraints for executing their associated tasks. Mobile medical devices, for example, have both timing and energy constraints that need to be satisfied. Their tasks cannot miss their respective deadlines, and cannot exceed a specified energy constraint in order to prolong the battery charge usage.

Taking into account such needs, this paper provides a pre-runtime approach based on time Petri nets [6] for synthesizing feasible schedules considering timing and energy constraints. In order to provide a more realistic system behavior, this work models explicitly the worst-case execution time of the dispatcher, since it may affect the tasks' deadline. The approach presented in this paper is a depth-first search method that generates a partial state-space computed from a time Petri net model that represents the task's constraints, thus tackling the state-space growth inherent to such systems.

2. RELATED WORKS

Xu and Parnas [9] present a branch-and-bound algorithm that finds an optimal pre-runtime schedule on a single processor for real-time process segments with release, deadline, and arbitrary exclusion and precedence relations. Despite the importance of their work, real-world experimental results are not presented. Abdelzaher and Shin [1] extended Xu and Parnas' work in order to deal with distributed real-time systems. This algorithm takes into account delays, precedence relations imposed by interprocess communications, and considers many possibilities for improving the scheduling lateness at the cost of complexity.

In [8], Swaminathan and Chakrabarty address the problem of scheduling tasks for minimum I/O energy consumption in hard real-time systems . The work adopts a pre-runtime scheduling approach and employs pruning technique based on time and energy, as well as heuristic methods in order to reduce the problem complexity. However, the proposed approach does not support inter-task relation and does not take into account multiple processors. AlEnawy and Aydin [2] introduce static (pre-runtime) and dynamic (runtime) scheduling mechanisms for dealing with energy-constrained scheduling. The proposed approach does not guarantee that all tasks will be executed, but only selected tasks with high priority. Preemption is supported, but inter-task relations are not taken into account.

3. COMPUTATIONAL MODEL

Computational model syntax is given by a time Petri net [6], and its semantics by a timed labeled transition system. A time Petri net (TPN) is a bipartite directed graph represented by a tuple $\mathcal{P}= (P, T, F, W, m_0, I)$. P (places) and T (transitions) are non-empty disjoint sets of nodes. The edges are represented by $F \subseteq (P \times T) \cup (T \times P)$. $W : F \rightarrow \mathbb{N}$ represents the weight of the edges. A TPN marking m_i is a vector $m_i \in \mathbb{N}^{|P|}$, and m_0 is the initial marking. $I : T \rightarrow \mathbb{N} \times \mathbb{N}$ represents the timing constraints, where

$I(t) = (EFT(t), LFT(t))\ \forall t \in T$, $EFT(t) \leq LFT(t)$, $EFT(t)$ is the Earliest Firing Time, and $LFT(t)$ is the Latest Firing Time.

An extended time Petri net with energy and priorities is represented by $\mathcal{P}_{\mathcal{E}} = (\mathcal{P}, \mathcal{E}, \pi)$. $\mathcal{P}$ is the underlying time Petri net, $\mathcal{E} : T \to \mathbb{R}^+ \cup \{0\}$ is a function that assigns transitions to energy consumption values, and $\pi : T \to \mathbb{N}$ is a priority function.

A set of enabled transitions is denoted by: $ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t)\}$, $\forall p_j \in P$. The time elapsed, since the respective transition enabling, is denoted by a clock vector $c_i \in \mathbb{N}^{|ET(m_i)|}$. The dynamic firing interval ($I_D(t)$) is dynamically modified whenever the respective clock variable $c(t)$ is incremented, and t does not fire. $I_D(t)$ is computed as follows: $I_D(t) = (DLB(t), DUB(t))$, where $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$, $DLB(t)$ is the Dynamic Lower Bound, and $DLB(t)$ is the Dynamic Upper Bound.

Let $\mathcal{P}_{\mathcal{E}}$ be a time Petri net, C be the set of all clock vectors in $\mathcal{P}_{\mathcal{E}}$, and $M_{\mathcal{E}}$ be the set of reachable markings of $\mathcal{P}_{\mathcal{E}}$. The set of states S of $\mathcal{P}_{\mathcal{E}}$ is given by $S \subseteq (M \times \mathbb{N}^{|ET(M)|} \times \mathbb{R})$, that is, a single state is defined by a triple (m, c, e), where m is a marking, c is its respective clock vector for $ET(m)$, and e is the accumulated energy consumption up to this state.

$FT(s)$ is the set of fireable transitions at state s defined by: $FT(s, e_{max})$ = $\{t_i \in ET(m) \mid (e \leq e_{max}) \wedge (\pi(t_i) = \min(\pi(t_k))\wedge (DLB(t_i) \leq \min(DUB(t_k)))$ $, \forall t_k \in ET(m)\}$. The *firing domain* for t at state s, is defined by the interval: $FD_s(t) = [DLB(t), \min(DUB(t_k))], \forall t_k \in ET(m)$.

A timed labeled transition system (TLTS) is a quadruple $\mathcal{L}= (S, \Sigma, \to, s_0)$, where S is a finite set of states, Σ is an alphabet of labels representing actions, $\to \subseteq S \times \Sigma \times S$ is the transition relation, and $s_0 \in S$ is the initial state.

The semantics of a TPN $\mathcal{P}$ is defined by associating a TLTS $\mathcal{L}_{\mathcal{P}}= (S, \Sigma, \to,$ $s_0)$: (i) S is the set of states of $\mathcal{P}$; (ii) $\Sigma \subseteq (T \times \mathbb{N})$ is a set of actions labeled with (t, θ) corresponding to the firing of a firable transition (t) at time (θ) in the firing interval $FD(t)$, $\forall s \in S$; (iii) $\to \subseteq S \times \Sigma \times S$ is the transition relation; (iv) s_0 is the initial state of $\mathcal{P}$.

4. SPECIFICATION MODEL

Let $\mathcal{T}$ be the set of tasks in a system. A periodic task is defined by $\tau_i = (ph_i, r_i, c_i, d_i, p_i, proc_i)$, where ph_i is the initial phase; r_i is the release time; c_i is the worst case computation time required for execution of task τ_i; d_i is the deadline; p_i is the period; and $proc_i$ is the processor allocated to such task. A task is classified as sporadic if it can be randomly activated, but the minimum period between two activations is known. Pre-runtime scheduling can only schedule periodic tasks. However, Mok [7] has proposed a translation from sporadic to periodic tasks. A task τ_i *precedes* task τ_j, if τ_j can only start executing after τ_i has finished. This work considers that communication between tasks allocated to the same processor is treated as a precedence relation. A task τ_i *excludes* task τ_j, if no execution of τ_j can start while task τ_i is executing. If it is considered a single processor, then task τ_i could not be preempted by task τ_j.

When adopting a multiprocessing environment, all inter-processor communications have to be taken into account, since these communications affect the system predictability. A inter-processor communication is represented by a special task, namely, communication task, which is described as follows. Let $\mu_m \in \mathcal{M}$ be a communication task defined by μ_m = (τ_i, τ_j, ct_m, bus_m), where $\tau_i \in \mathcal{T}$ is the sending task, $\tau_j \in \mathcal{T}$ is the receiving task, ct_m is the worst case communication time, $bus_m \in \mathcal{B}$ is the bus, where $\mathcal{B}$ is the set of buses, and $proc_i \neq proc_j$.

5. MODELING REAL-TIME SYSTEMS

In this work, the proposed modeling adopts a formal method for describing systems with timing constraints. The proposed modeling applies composition rules on building blocks models. For lacking of space, this section aims to present just an overview. For more details the reader is referred to [3].

5.1 TASKS MODELING

The considered building blocks are: (i) Fork; (ii) Join; (iii) Periodic Task Arrival; (iv) Deadline Checking; (v) Non-preemptive Task Structure; (vi) Preemptive Task Structure; (vii) Resources; and (viii) Inter-Processor Communication. The blocks are summarized as follows: **a) Fork Block**. Let us suppose that the system has n tasks. The fork block is responsible for starting all tasks in the system. This block models the creation of n concurrent tasks. **b) Join Block**. Usually, concurrent activities need to synchronize with each other. The join block execution states that all tasks in the system have concluded their execution in the schedule period. **c) Periodic Task Arrival Block**. This block models the periodic invocation for all task instances in the schedule period (P_S). **d) Deadline Checking Block**. The proposed modeling method uses elementary net structures to capture deadline missing. The scheduling algorithm (Figure 5) must eliminate states that represent undesirable situations like this one. **e) Task Structure Block**. The task structure may implement either preemptive or non-preemptive scheduling methods. Considering a non-preemptive method, the processor is just released after the entire computation to be finished. The preemptive method implies that a task are implicitly split into all possible subtasks, where the computation time of each subtask is exactly equal to one task time unit (TTU). **g) Resource Block**. The resources modelled are processors (p_{proc_i}) and buses (p_{bus_i}). An individual resource is represented by a single place. The presence of a token in a resource place indicates the availability of this resource.

Figure 1 depicts a Petri net model considering a specification composed of two non-preemptive tasks: $\tau_0 = (0, 0, 2, 7, 8, P1)$ and $\tau_1 = (0, 2, 2, 6, 6, P1)$.

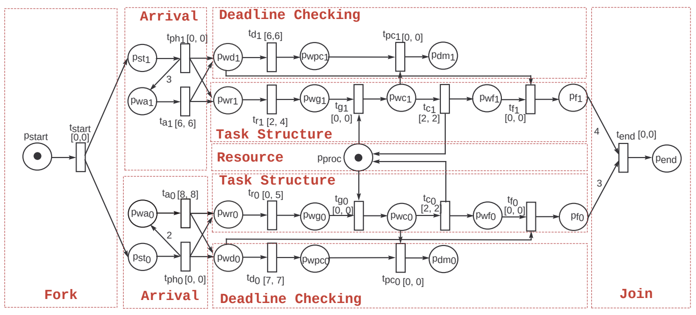

Figure 1. An example considering two tasks.

h) Inter-processor Communication Block. In this work, all inter-processor communications are treated as communication tasks, and message-passing paradigm is adopted. Additionally, the proposed approach for inter-processor communication considers that: (a) after the execution of the sending task, the message transmission is performed; (b) the receiving task can only execute after receiving the complete message; (c) both the sending and receiving processors are ready in the beginning of the communication. In other words, when the sender is transmitting the data, the receiver is prepared at the same moment for getting such data. This mechanism may be interpreted as a synchronous communication. Since interrupts may affect the system predictability, the proposed approach considers polling rather than interrupt handling to implement the receive operation; (d) point-to-point communication (or unicasting); (f) buses are reliable; (g) before communication takes place, the bus and, both sending and receiving processors have to be granted; and (h) communication time is annotated in the respective communication transition.

Figure 2 depicts the inter-processor communication building block. $t_{gb_{ij}}$ represents the granting of the sending processor, the receiving processor and the bus. $t_{send_{ij}}$ represents both the message sending and receiving. It is annotated with timing constraint specification, in this case, ct_m (worst-case communication time) of the respective communication task $\mu_m \in \mathcal{M}$. After the communication, both processors and the bus are released ($t_{comm_{ij}}$). $p_{wgb_{ij}}$ represents the waiting for bus and processors granting. $p_{ws_{ij}}$ indicates that the processors are ready to communicate. $p_{comc_{ij}}$ indicates that the communication was concluded. Lastly, $p_{rbuf_{ij}}$ represents the receiving buffer.

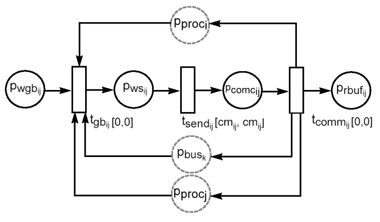

Figure 2. Building block.

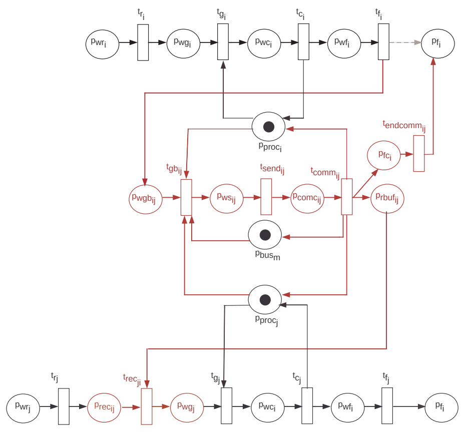

Figure 3. Modeling example.

Figure 3 applies the building block inter-processor communication for modeling the sending and receiving tasks τ_i and τ_j, respectively. It is worth observing that task τ_j has a refined place in order to consume the buffered message. More specifically, the place p_{wg_j} is substituted for the sequence ($p_{rec_{ij}}$, $t_{rec_{ij}}$, p_{wg_j}).

5.2 DISPATCHER OVERHEAD MODELING

The dispatcher overhead is captured in the grant-processor transition. When the task is non-preemptive, the timing interval of the grant-processor transition corresponds to the worst case execution time of the dispatcher. Since this is a simple solution, in the following presentation, the dispatcher overhead only considers preemptive tasks. When the task is preemptive, the model is slightly more complex. In this case, the proposed modeling adopts the TPN with priorities.

The proposed model considers two grant-processor transitions: grant-processor-with-overhead (t_{gw_i}) and grant-processor-without-overhead (t_{gwo_i}). As it can be seen in Figure 4, the timing interval ($[\alpha, \alpha]$) for transition t_{gw_i} models such timing overhead. Place $p_{proc_k T_i}$ states that task τ_i was the last executed task by the processor $proc_k$. The dispatcher overhead is considered in two situations: (1) when the next task to use the processor is different from the task that used the processor before; or (2) when a task instance ends its execution. The first situation is represented by the place $p_{proc_k T_i}$, where if such place is marked, it implies that the processor was lastly allocated to task τ_i. However, the second situation needs an explanation. Supposing that a task instance i of task τ_j ends its execution, and the following task to be executed is the task instance $i + 1$ of the same task τ_j. In this case, although the two instances are from the same task, the dispatcher calling is mandatory. As presented below, for solving this problem the model consider two final transitions, one that removes the marking in place $p_{proc_k T_i}$ and the other that does not.

In spite of this block may seem complicated, it is worth noting that this modeling is performed automatically by a tool. For more details, the interested reader is referred to [3].

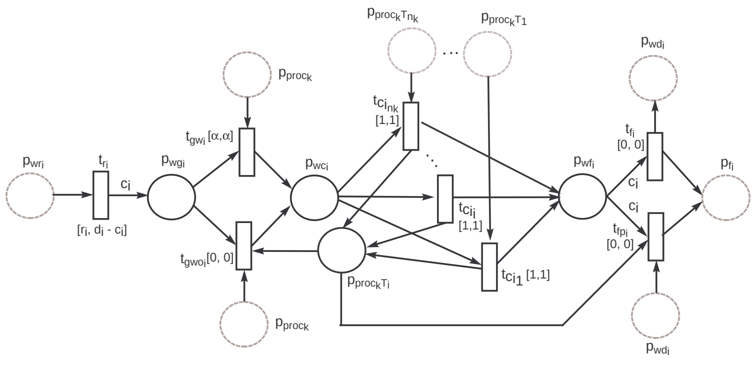

Figure 4. Building block dispatcher overhead.

6. ENERGY CONSUMPTION

Considering the Petri net task model, system energy consumption is associated with transitions representing dispatcher overhead (t_{gw}), task computation (t_c), and message transmission (t_{send}). Taking into account preemptive tasks, the energy consumption value of each computation time unit is equal to E_i/c_i, where c_i is the worst-case computation time (WCET) and E_i is the worst-case energy consumption of a task τ_i. Additionally, it is worth stating that the energy consumption value associated with t_{send} represents the sum of energy consumption values for sending and receiving the respective message.

The energy consumption for the dispatcher, the execution of tasks, and each message exchange must be known beforehand. In this work, the values were measured through a real prototype. The sum of energy dissipated in dispatcher, fired computation, and message transmission transitions results the total energy consumed during an execution of a schedule period.

The usage of the pre-runtime scheduler improves the accuracy of timing and energy consumption estimation. On the other hand, runtime approach cannot assure such an accuracy, since unpredictability of tasks arrival leads to more context-switching, increasing the energy consumption substantially.

The proposed method does not substitute other Lower-Power and Power-Aware techniques (e.g. dynamic voltage-scaling). Instead, the proposed method is a complement to such techniques, since the scheduling synthesis algorithm avoids unnecessary context-switching between tasks. Therefore, the generated schedule contains optimizations in terms of energy consumption.

7. PRE-RUNTIME SCHEDULING SYNTHESIS

This section shows a description of how to minimize the state space size, and the algorithm that implements the proposed method.

Minimizing State Space Size. The analysis based on the interleaving of actions is the fundamental point to be considered when analyzing state space explosion problem. Thus, the analysis of n concurrent actions has to verify all $n!$ interleaving possibilities, unless there are dependencies between these actions. This work proposes three ways for minimizing the state space size:

Modeling. The proposed method models dependencies between actions explicitly. **Partial-Order.** If actions can be executed in any order, such that the system always reaches the same state, these actions are *independent*. In other words, it does not matter in which order these are executed [4]. Independent actions are those that do not disable any other action, such as: arrival, release, precedence, processor releasing, and so on. This reduction method proposes to give a different *choice-priority* level for each class of independent activities. The dependent activities, like *processor granting*, have lowest choice-priority. Therefore, when changing from one state to another state, it is sufficient to analyze the class with highest choice-priority and pruning the other ones. **Removing Undesirable States.** Section 5.1 presents how to model undesirable error states, for instance, states that represent missed deadlines. The method proposed is of interest for schedules that do not reach any of these undesirable states. When generating the TLTS, transitions leading to undesirable error states have to be discarded.

Pre-Runtime Scheduling Algorithm. The algorithm adopted in this work is a depth-first search method on a TLTS. So, the TLTS is partially generated, according to the need. The *stop criterion* is obtained whenever the desirable final marking M^F is reached. For more information about this algorithm the interested reader is referred to [3].

8. CASE STUDY

In order to show the practical usability of the proposed approach in more details, a pulse-oximeter [5] is used as a case study. This equipment is responsible

```
scheduling-synthesis(S,M$^F$,TPN, $e_{max}$)
{
  if (S.M = M$^F$) return TRUE;
  tag(S);
  PT = pruning(firable($S$,$e_{max}$));
  if (|PT| = 0) return FALSE;
  for each ($\langle t,\theta \rangle$ $\in$ PT) {
    S'= fire(S, t, $\theta$);
    if (untagged(S') $\wedge$
      scheduling-synthesis (S',M$^F$,TPN,$e_{max}$)){
      add-in-trans-system (S,S',t,$\theta$);
      return TRUE;
    }
  }
  return FALSE;
}
```

Figure 5. Scheduling synthesis algorithm.

for measuring the oxygen saturation in the blood system using a non-invasive method. A pulse-oximeter may be used in many circumstances, like checking whether the oxygen saturation is lower or not than the acceptable, when a patient is sedated with anesthetics for a surgical procedure. This equipment is widely used in center care units (CCU).

For the sake of this paper, Table 1 shows the pulse oximeter task specification. In addition, the intertask relations are TE1 PRECEDES TE2, TE2 PRECEDES TE3, TE3 PRECEDES TE4, TA1 PRECEDES TA2, TA2 PRECEDES TA3, TA3 PRECEDES TA4, TA4 PRECEDES TA5, TA5 PRECEDES TA6, TA6 PRECEDES TA7, TA7 PRECEDES TA8. All tasks are preemptive. The dispatcher overhead is 200 microseconds and its respective worst-case energy consumption is 3958166,22 nJ. For this case study, the task time unit (TTU) adopted is 100 microseconds and the schedule energy constraint is 2 J .

Using the proposed approach, a feasible schedule was found in 45.9980 seconds, after visiting 42135 states. Due to lack of space, neither the Petri net model nor the found schedule is shown. The energy used by the found schedule totalizes 1,794,314,752.32 nJ ($\equiv 1.795J$). For this example, the scheduling algorithm found a schedule without any context-switching. To conclude, the scheduling synthesis algorithm was executed on a AMD Duron 1200 Mhz, 256 MB RAM, OS Linux, and compiler GCC 3.3.2.

9. CONCLUSIONS

This paper proposed a pre-runtime scheduling approach considering energy and timing constraints for embedded hard real-time systems with multiple processors. Predictability is an important concern when considering time-critical systems. The scheduling approach presented guarantees that all critical tasks meet their deadlines and the schedule satisfies the energy constraint. In spite of the analysis technique (i.e. state space exploration) is not new, to the best of our present knowledge, there is no similar work reported that uses formal

Table 1. Task specification for the pulse Oximeter

TaskID	r	c	d	p	proc/bus	from	to	Energy
TE1	0	41	1000	2500	P1	-	-	8576,79 nJ
TE2	371	41	1000	2500	P1	-	-	52,48 nJ
TE3	576	41	1000	2500	P1	-	-	8576,79 nJ
TE4	947	41	1000	2500	P1	-	-	52,48 nJ
TE5	0	45	2000	2500	P1	-	-	222,30 nJ
TA1	0	41	5000	16000	P1	-	-	55,76 nJ
TA2	141	50	5000	16000	P1	-	-	222,50 nJ
TA3	191	41	5000	16000	P1	-	-	55,76 nJ
TA4	323	50	5000	16000	P1	-	-	222,50 nJ
TA5	382	41	5000	16000	P1	-	-	55,76 nJ
TA6	523	50	5000	16000	P1	-	-	222,50 nJ
TA7	573	41	5000	16000	P1	-	-	55,76 nJ
TA8	714	50	5000	16000	P1	-	-	222,50 nJ
TC1	764	60	5000	16000	P2	-	-	430,2 nJ
TC2	0	50	10000	16000	P2	-	-	444,00 nJ
TC3	0	50	10000	16000	P2	-	-	1117,5 nJ
TC4	764	45	10000	16000	P2	-	-	935,1 nJ
TC5	0	90	10000	16000	P2	-	-	935,1 nJ
TC6	0	90	10000	160000	P2	-	-	7089,3 nJ
TC7	0	90	10000	80000	P2	-	-	935,1 nJ
M1	-	7	-	-	bus1	TA8	TC1	8797,2 nJ

methods for modeling time-critical systems with energy constraints, considers arbitrary precedence/exclusion relations, and finds pre-runtime schedules. As future work, it is proposed to generate automatically the system source code from a feasible schedule, meeting not only timing constraints but also energy constraints.

REFERENCES

[1] T. Abdelzaher and K. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel Distributed Systems*, 10(11):1179–1191, Nov 1999.

[2] T. A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. *Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS 04)*, June 2004.

[3] R. Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Software Synthesis*. PhD Thesis, Centro de Informática - UFPE, April 2005.

[4] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems*. PhD Thesis, University of Liege, Nov. 1994.

[5] M. Nogueira Oliveira Júnior. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso (in portuguese)*. MSc Thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco, August 1998.

[6] P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Trans. Comm.*, 24(9):1036–1043, Sep. 1976.

[7] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD Thesis, MIT, May 1983.

[8] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. Embedded Comput. Syst. 4(1)*, pages 141–167, 2005.

[9] J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering*, 16(3):360–369, March 1990.