UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

HERMES JOSÉ GONÇALVES JÚNIOR

ARQUITETURA EM HARDWARE PARA CO-PROCESSAMENTO DE TAREFAS EM SISTEMA OPERACIONAL TEMPO REAL

Porto Alegre
2004

HERMES JOSÉ GONÇALVES JÚNIOR

ARQUITETURA EM HARDWARE PARA CO-PROCESSAMENTO DE TAREFAS EM SISTEMA OPERACIONAL TEMPO REAL

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE), da Universidade Federal do Rio Grande do Sul (UFRGS), como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Automação e Instrumentação Eletro-Eletrônica

ORIENTADOR: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre

2004

HERMES JOSÉ GONÇALVES JÚNIOR

ARQUITETURA EM HARDWARE PARA CO-PROCESSAMENTO DE TAREFAS EM SISTEMA OPERACIONAL TEMPO REAL

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Carlos Eduardo Pereira, UFRGS.

Doutor pela Universität Stuttgart – Stuttgart, Alemanha

Banca Examinadora:

Prof. Dr. Walter Fetter Lages, UFRGS.

Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Prof. Dr. João César Netto, UFRGS.

Doutor pela Universite Catholique de Louvain – Bélgica

Prof. Dr. Luigi Carro, UFRGS.

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Leandro Buss Becker,UFSC.

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Coordenador do PPGEE: _____

Prof. Dr. Carlos Eduardo Pereira

Porto Alegre, setembro 2004.

DEDICATÓRIA

Dedico este trabalho aos meus pais, em especial pela dedicação e apoio em todos os momentos difíceis. A Marlise Santos de Lima pelo companheirismo nas horas mais conturbadas durante a execução deste trabalho e por estar presente em minha vida. Também gostaria de estender esta dedicatória à minha querida avó, Maria Francisca, pelos inestimáveis ensinamentos nestes longos anos de vida. Por fim, esta obra celebra minha homenagem “in memorium” a Júlia Ribeiro Marques, Osvaldo Manuel Marques e a Astrogildo Gonçalves.

AGRADECIMENTOS

Dentre tantas pessoas que me cabe destacar pelo apoio no desenvolvimento deste trabalho de dissertação estão: M.Eng. Ronaldo Husemann pela incansável e destacável dedicação para a realização deste estudo; O meu orientador Professor Dr.Ing Carlos Eduardo Pereira pelo inestimável aprendizado proporcionado durante o tempo de convivência no programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul; Dr. Leandro Buss Becker pela grande colaboração no desenvolvimento deste projeto, pela amizade e pela perseverança em nossas relações técnico científicas.

Cabe agradecer ainda, ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de realização de trabalhos em minha área de pesquisa.

Aos colegas do PPGEE pelo seu auxílio nas tarefas desenvolvidas durante o curso e apoio na revisão deste trabalho.

A CAPES pela provisão da bolsa de mestrado.

RESUMO

Os sistemas operacionais de tempo real, assim como os sistemas embarcados, estão inseridos no desenvolvimento de projetos de automação industrial segmentado em diversas áreas de pesquisa como, por exemplo, robótica, telecomunicações, e barramentos industriais. As aplicações de sistemas modernos de controle e automação necessitam de alta confiabilidade, velocidade de comunicação, além de, determinismo temporal. Sistemas operacionais de tempo real (SOTR) têm-se apresentado como uma solução confiável quando aplicadas em sistemas que se fundamentam no cumprimento de requisitos temporais. Além disso, o desempenho computacional é totalmente dependente da capacidade operacional da unidade de processamento. Em um sistema monoprocessado, parte da capacidade computacional da unidade de processamento é utilizada em atividades administrativas, como por exemplo, processos de chaveamento e salvamento de contexto. Em decorrência disto, surge a sobrecarga computacional como fator preponderante para o desempenho do sistema. Este trabalho tem por objetivo, analisar e fornecer uma arquitetura alternativa para realizar o co-processamento de tarefas em uma plataforma IBM-PC, aumentando a capacidade computacional do microprocessador principal. No presente trabalho, a plataforma de co-processamento realiza a execução do algoritmo de escalonamento do sistema operacional, desta forma distribuiu-se o gerenciamento temporal das tarefas entre a plataforma IBM-PC e a unidade de co-processamento.

Palavras-chaves: Escalonador, Sistema Operacional de Tempo Real, Co-processador.

ABSTRACT

Real time operating systems and embedded systems are increasingly using in the development of projects of industrial automation divided in several areas of research such as, for example, robotics, telecommunications, and industrial bus. These applications on modern control systems need, very high reliability, speed communication, besides temporal determinism. Real time operating systems (RTOS) has been presented as a reliable solution when applied on systems that are based in real time requirements, further on computing performance is totally dependent of the operating capacity processing unit. On monoprocessing system a portion of the computing capacity, of the processing unit, it is used in management activities such as, for example, processes of context switching, and context sefety. In consequence of this, it arise the computing overload a significant characteristic for the performance system. This work has for objective to analyze and to provide an alternative architecture, especially developed to accomplish the co-processing of tasks on platform IBM-PC, increasingly computing capacity of main processor. In the present research, the co-processing platform accomplishes the execution of algorithm scheduling on system, this way the task's management was distributed between the platform IBM-PC and co-processing unit.

Keywords: Scheduler,Real Time Operational Systems,Co-processor.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AGV: Autonomus Guided Vehicle

API: Application Programming Interface

EDS: Earliest Deadline Soonest

ASIC: Aplication Specific Intergrated Circuit

BDM: Background Debug Mode

CP: Critical Period

CPU: Central Processing Unit

DMA: Direct Memory Access

DPRAM: Dual-Port Random Access Memory

ECET:Expected Case Execution Time

EDF: Earliest Deadline First Scheduling Algorithm

EDL: Earliest Deadline Least

EEPROM: Electrically Erasable Programmable Read Only Memory

EPROM: Erasable Programmable Read Only Memory

FIFO: First-In First-Out

FPGA: Field Programmable Gate Array

LRT: Latest Release Time

LLT: Least Laxity First

MP: Main Part

OS: Operating system

PC: Personal Computer

PCI: Peripheral Component Interconnect

PDA: Personal Digital Assistant

PID: Proportional Integral Derivative

POSIX: Portable Operating system Interface

PROM: Programmable Read Only Memory

RAM: Random Access Memory

RM: Rate Monotonic

ROM: Read Only Memory

RR: Round Robin

RTLinux: Real-Time Linux

RTOS: Real Time Operating Systems

TAFT: Time Aware Fault Tolerant

TFCP: Taft Co-Processador

WCET: Worst Case Execution Time

1 INTRODUÇÃO

Segundo (STANKOVIK, 1988), um sistema de tempo real é aquele cujo suas atividades são classificadas não apenas por executarem atividades que resultem no correto processamento lógico da solução, mas que também, leve em consideração o limite de tempo que a unidade de processamento utiliza para executar cada tarefa. Não raramente ocorrem situações onde o atraso na execução de uma ação de controle pode causar graves prejuízos tanto sob o aspecto financeiro, quanto à vida humana.

Com o advento dos sistemas computacionais, muitas aplicações como, por exemplo, algoritmos de controle e sistemas de monitoração, estão sendo utilizadas nas áreas industriais, militares, e espaciais. Esses sistemas já estão inseridos no cotidiano tecnológico e os computadores têm revolucionado a produção de bens e entrega de serviço.

Um sistema operacional de tempo real deve interagir com os estímulos oriundos do seu ambiente através de sensores e atuadores. Da mesma forma, os sistemas embarcados, sistemas de supervisão e controle, utilizam as premissas dos RTOS para garantir que os requisitos temporais inerentes a estes sistemas sejam, estritamente, atendidos.

Os sistemas operacionais específicos para aplicações de tempo real são do tipo em que muitas vezes, dependendo da aplicação, o não cumprimento temporal dos requisitos é considerado inaceitável sob o aspecto dos resultados. Entretanto, há aplicações com sistemas de tempo real em que o não cumprimento dos limites temporais não é um fator preponderante. No contexto geral, tais aplicações incluem controle industrial, controle de vôo, simulações em tempo real, e também, freqüentemente, aplicações militares como, por exemplo, sistema teleguiado de mísseis (MILENKOVIC, 1992).

Sistemas operacionais de tempo real, assim como, sistemas embarcados relacionam-se com o mercado mundial através de inúmeras empresas provedoras de sistemas (OLIVE,

2000). Os negócios em torno desses sistemas, tais como, Chorus, Ecos, QNX, VxWorks, Windows CE, RTLinux e RTAI entre tantos outros existentes, movimentam grandes cifras, o que torna a concorrência muito acirrada entre as grandes corporações envolvidas no mercado dos sistemas operacionais. Os produtos desenvolvidos por essas empresas que utilizam os sistemas operacionais estão presentes numa grande variedade de produtos como, por exemplo, fax, impressoras, assistente digital pessoal (PDA), sistemas de navegação, sistemas de comunicação por satélites, além de sistemas de domótica, como por exemplo, fornos de microondas, lavadoras e secadores de roupas, ou em aplicações envolvendo o controle de plantas de manufatura com controle e supervisão.

1.1 CONTEXTUALIZAÇÃO

Os resultados científicos obtidos na área da ciência da computação, no setor da microeletrônica, tiveram uma fundamental importância para o desenvolvimento de aplicações com suporte tempo real. Através desses resultados, foi possível melhorar o desempenho com relação aos requisitos temporais nos RTOS, pois segundo (WEISS, 1999), nos últimos anos, com o rápido progresso na tecnologia da microeletrônica, têm-se reduzido os custos dos componentes utilizados na implementação dos sistemas, enquanto simultaneamente, tem-se adotado o uso de microcontroladores de 32 bits no desenvolvimento de sistemas embarcados. Além disso, características como dados e instruções que utilizam memória, interface de barramentos programáveis, maior ciclo de freqüência e desempenho de velocidade de processamento, têm influenciado significativamente na arquitetura dos sistemas embarcados, assim como, nos sistemas de tempo real, entretanto, estas características podem ser as principais causas de imprevisibilidade temporal nos RTOS. Novos conceitos e fundamentos empregados em projetos de *Hardware* permitem aos RTOS ser implementados com a orientação para o aumento do desempenho e da complexidade funcional do sistema (vide

(WEISS, 1999)). Atualmente, diversos assuntos relacionados ao campo dos RTOS como, por exemplo, particionamento de sistemas computacionais, e modelos referentes aos mecanismos de escalonamento, estão sendo explorados por parte da comunidade científica. Os resultados, provenientes dos inúmeros centros de pesquisas, são muito freqüentemente utilizados nas aplicações de tempo real para obter-se um aumento de processamento das arquiteturas computacionais. Entre os objetos-alvo de pesquisa em se tratando de RTOS estão os algoritmos de escalonamento. Concomitantemente, segundo (CHETTO, 1989), o campo relacionado ao escalonamento de tempo real é o maior foco de interesse de pesquisa. Isso se deve, segundo o autor mencionado, ao freqüente uso de computadores digitais em aplicações de tempo real, crescendo deste modo, a sofisticação do *Software* de tempo real, e o aumento no interesse de aperfeiçoar o desempenho e a confiabilidade dos sistemas. O cumprimento temporal está entre as características de maior relevância para aplicações de tempo real. Em sistemas com essas características a não execução dos requisitos temporais previamente agendados, como um *Deadline*, pode levar à instabilidade do sistema ou até mesmo causar danos imensuráveis à aplicação, e às pessoas envolvidas no processo. Daí a necessidade em se utilizar sistemas computacionais que possuam comportamento temporal determinístico. O determinismo é caracterizado pela capacidade do sistema responder, sob qualquer circunstância, dentro de uma faixa temporal pré-definida, a qualquer estímulo vinculado ao sistema. Além disso, o resultado de uma operação num sistema de tempo real não depende exclusivamente do correto valor funcional, mas também, do instante de tempo que o resultado foi produzido (STÄRNER, 1998).

As arquiteturas dos sistemas embarcados encontrados na literatura consistem em uma aplicação específica de *Hardware* que interagem com o ambiente. E ao mesmo tempo, a uma aplicação específica de *Software* implementada em um microcontrolador (vide (NITSCH, 2000)). Com o transcorrer dos anos, tem-se aumentado a complexidade da arquitetura dos

microcontroladores e dos circuitos de aplicação específica (ASIC's). As implementações possuem funções específicas na constituição da arquitetura embarcada de modo geral, sendo suas funcionalidades divididas da seguinte maneira: a parte do *Hardware* fica restrita a função de prover operações que requeiram a utilização de técnicas tais como, VHDL (VSIC Hardware Description Language), e ASIC's (vide (NITSCH, 2000)) e (WEIß, 1999). Já, por outro lado, a parte relativa ao *Software* fica responsável pelo fornecimento de suporte à política de escalonamento, além de inúmeros outros atributos funcionais, como a de conter as aplicações e interface com o usuário. As conexões de comunicação nos sistemas embarcados podem ser executadas por protocolos dedicados como, por exemplo, os protocolos PCI e USB (VOIGT, 1999). A principal justificativa para utilização de RTOS em sistemas embarcados é que se pode, estrategicamente, alterar características específicas do sistema de tempo real a fim de obter-se melhorias de desempenho temporal. Essa premissa de pesquisa vem tornando-se inspiração para muitos pesquisadores na área de sistemas de tempo real. Pesquisas recentes (PILLAI, 2001) têm explorado a linha de pesquisa referente à redução de carga computacional, através de diferentes políticas de escalonamento, bem como, utilizando técnicas de multiprocessamento para melhorar o desempenho das funcionalidades operacionais. Outro objeto de pesquisa correlacionado com RTOS diz respeito à redução na carga computacional nos sistemas embarcados, este assunto é de suma importância devido ao fato de que o consumo de energia dos sistemas embarcados está intimamente concatenado, em parte, ao tipo de estratégia de escalonamento adotado (PILLAI, 2001).

Já, em (BECKER, 2001) e (BECKER, 2003) é analisado o algoritmo TAFT (*Time-Aware Fault-Tolerant*) onde são mostradas suas características e soluções flexíveis quanto à utilização de recursos do processador principal considerando-se a estratégia de tolerância à falha.

1.2 MOTIVAÇÃO

Esta seção aborda os principais fatores que motivaram a realização deste trabalho de pesquisa. Primeiramente, cabe destacar a falta de uma análise temporal detalhada com relação à comunicação, determinismo e previsibilidade em sistemas cooperantes de processamento, bem como, a falta de estimativas da sobrecarga computacional relacionada a arquitetura proposta.

Os projetistas de sistemas embarcados, de maneira geral, têm como grande desafio em se tratando de projetos relacionados a *Hardware/Software* a redução no tempo de projeto. Isto se faz necessário nos dias atuais, devido ao fato da grande concorrência no mercado mundial de sistemas embarcados entre diversas empresas do ramo. E atrelado a esse fato, vence a concorrência a empresa que constantemente está lançando no mercado novos produtos.

Tradicionalmente, a adoção de sistemas operacionais de tempo real RTOS vem sendo utilizada como uma solução bastante viável, em todos os aspectos temporais e nas aplicações que requeiram previsibilidade e determinismo. Arquiteturas alternativas vêm sendo utilizadas para auxiliar o processador, através do algoritmo de escalonamento, a atender todos os requisitos temporais referentes às tarefas a serem escalonadas pelo sistema. Muitos autores como em (COOLING, 1997), estão propondo como solução, uma arquitetura de cooperação computacional na qual venha auxiliar a unidade principal de processamento a garantir o escalonamento das tarefas dentro dos seus limites de prazos finais. Liberando, desta maneira, a CPU para realizar outras tarefas no sistema. Nesta mesma seqüência, o autor (STANKOVIC, 1999b) também propõe uma arquitetura multiprocessada dedicada, exclusivamente, para o processamento de políticas de escalonamento de um sistema operacional de tempo real. Busca-se com essas arquiteturas melhorar o desempenho

computacional quanto à execução do algoritmo de escalonamento, e a redução no tempo utilizado nas operações inerentes do processador principal.

Destaca-se ainda, a falta de um ambiente baseado numa plataforma de *Hardware* para suportar as funcionalidades dos diversos algoritmos de escalonamento, existentes na literatura de tempo real como, por exemplo, o LRT, EDF (vide (MILENKOVIC, 1992)) e o TAFT (vide (BECKER, 2003)).

1.3 OBJETIVOS E CONTRIBUIÇÕES DESTA DISSERTAÇÃO

Com base no contexto apresentado anteriormente, esta dissertação tem como proposta desenvolver e analisar uma arquitetura multiprocessada para auxiliar o processador principal no escalonamento de tarefas em sistemas de tempo real, bem como, estimar as possíveis sobrecargas computacionais decorrentes da comunicação entre os dispositivos constituintes da arquitetura proposta. Além disso, objetiva-se especificar o comportamento temporal (*Deadlines, Jitter*) de processamento no ambiente de tempo real proposto.

De modo específico, pretende-se abordar nesta obra uma alternativa para a redução da sobrecarga computacional. O objetivo principal deste estudo é o de fornecer uma arquitetura alternativa para aumentar o determinismo do sistema computacional, através da retirada de algumas funcionalidades inerentes do algoritmo de escalonamento TAFT sob a plataforma RTlinux-3.0 (BARABANOV, 1997) e, também, permitir o uso de algoritmos de escalonamento mais sofisticados. Além disso, deseja-se estimar na arquitetura proposta, a previsibilidade temporal do sistema quanto à interface de comunicação e transferência de dados.

Portanto, propõe-se neste trabalho o desenvolvimento de uma arquitetura em *Hardware* para auxiliar no escalonamento de tarefas. Os aspectos fundamentais que se visa melhorar com esta arquitetura são: o escalonamento de tarefas, o aumento de desempenho e da

previsibilidade e a redução do “jitter” decorrente dos instantes de ativações do processo de escalonamento. O principal fator a ser salientado é a redução da sobrecarga computacional pela transferência de funcionalidades entre unidade de processamento e a unidade co-processadora.

1.4 ORGANIZAÇÃO DO VOLUME

O restante deste volume está organizado da seguinte maneira: O capítulo 2 apresenta uma revisão dos conceitos relacionados à literatura dos sistemas embarcados. O capítulo 3 apresenta uma análise do estado da arte sobre sistemas operacionais de tempo real (RTOS). O capítulo 4 apresenta uma visão geral da arquitetura proposta. Já, no capítulo 5 é apresentado o ambiente de programação e validação da arquitetura proposta. Por fim, no capítulo 6 são resumidas as conclusões obtidas, e realizadas as devidas menções com relação aos trabalhos futuros.

2 REVISÃO DE CONCEITOS

Os conceitos relacionados aos sistemas operacionais de tempo real são apresentados neste capitulo a fim de elucidar os principais fatores relativos à teoria de escalonamento.

Em sistemas de aplicações industriais, de maneira geral, a noção de tempo e concorrência são de suma importância, para um dado conjunto de tarefas, no que se refere ao cumprimento de suas atribuições num certo limite temporal. Os conceitos e técnicas relacionados ao escalonamento são fundamentais para o determinismo e a previsibilidade do comportamento dos sistemas que requeiram restrições temporais em suas atividades.

Com o advento das pesquisas realizadas nesta área, uma significativa quantidade de algoritmos com diversas estratégias de escalonamento têm surgido na literatura. Porém, alguns desses resultados mostraram-se inviáveis quanto da sua utilização em aplicações reais, pelo fato de serem restritos com relação à técnica empregada além de não suprirem requisitos desejáveis em se tratando de RTOS. Atualmente, com relação à padronização, o mercado de sistemas operacionais tem adotado o padrão POSIX (GALLMEISTER, 1995). POSIX é uma proposta da IEEE que visa padronizar o desenvolvimento de qualquer produto comercial.

Na seqüência deste trabalho, são mostrados alguns conceitos teóricos que permitem caracterizar e modelar as tarefas dedicadas para aplicações em tempo real.

2.1 MODELO PARA PROCESSOS COMPUTACIONAIS CONCORRENTES

O conceito de tarefa é uma abstração que está inserida na análise de escalonamento. Tarefas ou processos, como são conhecidas na literatura tempo real, nada mais são do que uma unidade de processamento seqüencial que disputam os recursos computacionais dos sistemas. Uma aplicação de tempo real é constituída por várias tarefas. E uma tarefa sob a condição tempo real tem que satisfazer seus prazos finais (*Deadlines*) apresentando correção temporal (*Timeliness*), além de, correção lógica (*Correctness*).

As tarefas são, geralmente modeladas por uma tupla, $\Pi = \{\tau = (T,D,C), i= 1$ até $n\}$ que tem por finalidade descrever as características temporais de uma tarefa qualquer do sistema. Analisando-se mais especificamente uma tupla de modelos para tarefas de tempo real, tem-se o período representado por T, D designa o prazo final, e C o tempo de computação que uma unidade de processamento utiliza em uma determinada tarefa para cada instante de ativação. A título de análise considera-se, na maioria dos casos, o período igual ao *Deadline*.

2.2 RESTRIÇÕES TEMPORAIS

As aplicações de tempo real são caracterizadas por restrições temporais, as quais devem ser cumpridas. Todas as tarefas possuem um prazo final a respeitar (*Deadline*), e por definição, elas devem concluir suas funções antes destes prazos. Assim, classificam-se as tarefas conforme o cumprimento de seus requisitos temporais. Deste modo, elas podem ser especificadas em termos de segurança, com relação às características temporais como tarefas com os requisitos temporais críticos (*Hard*), onde os sistemas não aceitam falhas no atendimento aos prazos finais, sob pena de submeter perdas temporais aos processos controlados, como exemplo, sistemas de controle de navegação. Tarefas brandas (*Soft*) são denominadas àquelas tarefas em que os requisitos temporais têm a real necessidade de atender os seus prazos finais, como por exemplo, sistemas de áudio e vídeo. Para as definições dos dois tipos de tarefas têm-se à classificação que segue.

Tarefas críticas (*Hard*): são denominadas àquelas tarefas que devem cumprir estritamente os seus respectivos prazos finais. Se outra situação, com relação aos instantes de ativação de cada tarefa, vier a ocorrer, uma falha de temporização deve ser computada, invalidando deste modo, o processamento do sistema. Nesta situação, uma falha pode até mesmo ser catastrófica, dependendo da vulnerabilidade do sistema. Além de, muitas vezes, submeter danos irreversíveis ao sistema.

Tarefas brandas (*Soft*): a essas tarefas é dado o direito de atendimento em média do seu respectivo prazo final, pois, apenas ocorre a redução de rendimento temporal do sistema não representando perdas significativas.

Outras características são definidas e observadas com relação às tarefas nos sistemas de tempo real, entre elas cabe realçar a regularidade dos instantes de ativações. Os modelos de tarefas distinguem-se conforme as freqüências de ativações:

Tarefas periódicas: neste caso, os instantes de ativações de processamento das tarefas ocorrem em intervalos regulares de tempo, chamados de períodos.

Tarefas aperiódicas: são caracterizadas quando os instantes de ativações de processamento de uma determinada tarefa podem responder aleatoriamente as ativações temporais.

Tarefas esporádicas: correspondem a uma variação das tarefas aperiódicas, e o ponto central diz respeito ao não conhecimento prévio dos instantes de ativações. Para tarefas periódicas tem-se que o prazo final (*Deadline*) igual ao período á título de modelagem, entretanto, para as tarefas esporádicas, esta premissa torna-se inválida.

As tarefas periódicas regulares nos instantes de ativações, portanto, com previsibilidade temporal, são associadas a "*Deadline Hard*", sendo desta maneira denominadas de periódicas. Já por outro lado, às tarefas ditas aperiódicas, pela inexistência de previsibilidade em seus instantes de ativações, quase sempre têm "*Deadline Soft*" associados as suas execuções. Sendo, portanto, classificadas como tarefas brandas, ou seja, não necessitam respeitar estritamente os seus instantes de ativações.

Outras definições importantes quanto ao comportamento temporal das tarefas de tempo real que merecem destaque são:

Tempo de computação (*Computation Time*) : é o tempo necessário para a realização de uma tarefa.

Tempo de início (*Start Time*): é o instante inicial de ativação no processamento de uma tarefa.

Tempo de término (*Completion Time*): é o instante em que se finaliza a execução de uma tarefa.

Tempo de chegada (*Arrival Time*): é o instante em que o escalonador toma conhecimento de ativação de uma tarefa.

Tempo de liberação (*Release Time*): é o instante de tempo da inclusão da tarefa na fila de execução.

2.3 ALGORITMOS DE ESCALONAMENTO TEMPO REAL

Nos sistemas de tempo real, os algoritmos de escalonamento têm a mais importante tarefa a desempenhar no funcionamento do sistema, pois cabe a eles gerenciar todos os recursos quanto aos requisitos temporais relacionados aos processos, como prazos finais e instantes de ativações. Esse gerenciamento é de suma importância para atender a demanda temporal de todas as tarefas do sistema de tempo real. Assim, os algoritmos de escalonamento são considerados como o problema a ser resolvido nos sistemas de tempo real. Uma aplicação é representada como sendo um grupo de tarefas, cujas integrações do tempo de computação dessas tarefas expressam a carga computacional do sistema.

Escalonar é o procedimento de ordenação, pelas unidades de processamento em relação ao tratamento dos requisitos temporais, referentes às tarefas de um sistema sob condições de tempo real. A simples execução de um algoritmo de escalonamento, nada mais é do que a coordenação de ocupação do processador por um grupo de tarefas. O escalonador é a entidade do sistema computacional responsável pela gestão temporal do processador, sendo nele implementadas as políticas de escalonamento, possibilitando desta forma, a execução de um conjunto de tarefas. Segundo (LIU, 1973), um algoritmo de escalonamento nada mais é do

que o procedimento de ajuste temporal que determina quais as tarefas serão executadas num momento particular.

Os critérios de escalonamento são definidos como sendo regras a serem seguidas pela unidade de processamento na ordenação das tarefas de tempo real. Deste modo, os escalonadores utilizam as políticas que garantem, ou devem garantir, o cumprimento dos requisitos temporais das tarefas. A estratégia de escalonamento é ótima se, conforme os critérios estabelecidos previamente pela política implementada, as restrições temporais especificas das tarefas do sistema forem totalmente atendidas. Os algoritmos de escalonamento podem ser preemptivos, se permitirem num dado momento de execução de um conjunto de tarefas, a ocorrência de interrupções sobre as tarefas com menor índice de prioridade, fazendo com que o processador passe a tratar imediatamente as tarefas com maior prioridade em detrimento das demais. Já, com relação aos algoritmos não preemptivos, a eventual interrupção de uma tarefa de menor prioridade por outra de maior prioridade, jamais ocorre durante a execução. Além disso, os algoritmos podem ser estáticos ou dinâmicos. Algoritmos são ditos estáticos quando os parâmetros são atribuídos às tarefas em tempo de projeto e não podem ser modificados em tempo de execução. (LIU, 1973) denomina como estáticos os algoritmos de escalonamento cujas prioridades são definidas no início do processo e não mais são alteradas. Esse tipo de algoritmo é conhecido como sendo de prioridade fixa. Já os de prioridade dinâmica são os algoritmos em que as propriedades das prioridades sofrem alterações desde o instante inicial de ativação da tarefa.

Um importante conceito, em se tratando de escalonamento de sistemas de tempo real, é o teste de escalonabilidade. Este teste define se as restrições dos requisitos temporais de um grupo de tarefas podem ser atendidas pela unidade de processamento, quando um determinado algoritmo de escalonamento é utilizado. Logo, o teste tem uma função importantíssima com relação à escalonabilidade e atendimento aos requisitos temporais de um

sistema, pois ele determina se um conjunto de tarefas é escalonável. Na literatura encontram-se umas séries de testes, os quais variam conforme o modelo adotado para a modelagem das tarefas e a política de escalonamento utilizada. Em (CHETTO, 1989) definem-se alguns teoremas e conceitos relativos aos instantes críticos de ativações e zonas de período crítico. O instante crítico para qualquer tarefa ocorre quando há simultaneidade no instante de ativação com outra tarefa de mais alta prioridade.

O escalonamento de um conjunto de tarefas é realizável, por uma unidade processadora, se a taxa de utilização for menor ou igual a 100 por cento. Segundo (LIU, 1973), o algoritmo RM “Rate Monotonic” de prioridade fixa apresenta um índice máximo de utilização da CPU em torno de 69 por cento, já o de prioridade dinâmica, como o EDF apresenta um índice de 100 por cento. Em (LIU, 1973) é provado analiticamente que com algoritmos de prioridade fixas o máximo fator de utilização de CPU converge para o logaritmo neperiano de dois, ou seja, há um desperdício no fator de utilização da unidade processadora. Já, nos algoritmos de prioridades dinâmicas, pode-se atingir até 100 por cento de fator de utilização. Em suma, neste trabalho os autores discutem algumas premissas associadas a multiprogramação dos sistemas do tipo “*hard real time*”em aplicações típicas como processos de controle e monitoração. Nos algoritmos de prioridade fixa, como por exemplo, o RM, o fator de utilização é de 70 por cento, já nos algoritmos dinâmicos pode-se ter a total utilização do processador.

O algoritmo “Rate Monotonic” ou RM é um algoritmo de escalonamento que produz escalas em tempo de execução através de mecanismo de preempção. O RM apresenta um esquema onde as tarefas com as maiores taxas de pedido de ativações terão as mais altas prioridades. Ele é considerado ótimo entre todos os algoritmos da sua classe. Pode-se inferir que não haja algoritmo de escalonamento que resolva algum problema de restrições temporais de um certo conjunto de tarefas que o RM não resolva também (BUTTAZZO, 1996).

Algumas premissas devem ser consideradas quanto ao algoritmo RM com relação à análise de escalonabilidade: as tarefas são sempre periódicas e independentes entre si, e os seus respectivos prazos finais são iguais aos períodos. Além disso, os tempos de computações são sempre constantes e os tempos de chaveamentos entre as tarefas são considerados nulos. As prioridades das tarefas são baseadas com relação aos seus respectivos períodos, conforme esta regra de ordenação, quanto menor o período de ativação de uma tarefa, menor também será a sua prioridade. Os autores (vide (LIU, 1973)) e (SANTOS, 1993) analisam em seus estudos um método para fornecer resultados prévios para o escalonamento de múltiplas tarefas com características tempo real numa única unidade processadora as quais podem ser escalonadas pelo algoritmo RM. Segundo os autores, os principais problemas com relação as multitarefas nos sistemas multiprocessados são: alocação de tarefas, e os seus respectivos escalonamentos no processador conforme a política adotada.

Considerando-se o prazo final igual ao período, o algoritmo RM tem um fator de utilização (vide (LIU, 1973)) limite de utilização teórico e conservativo.

À medida que a política de escalonamento baseada no algoritmo RM dá maior prioridade de escalonamento para as tarefas com um menor período, propiciando a preemptibilidade, por outro lado, aumenta carga computacional do sistema. O fator mais importante para as tarefas sob plataforma tempo real é que elas sempre sejam executadas antes do seu prazo final. A complexidade computacional é descrita pela expressão O(m*Tm) onde m é o número de tarefas e Tm é o máximo período das tarefas (SANTOS, 1993).

Tem-se com relação à análise de escalonabilidade quanto ao algoritmo RM que as tarefas possuem *Deadline* sempre menor ou igual ao seu respectivo período e que as tarefas sejam sempre periódicas, porém, em (AUDSLEY, 1992) os autores tratam da análise de escalonamento de tarefas esporádicas. Segundo eles, se as conseqüências de uma falha temporal num sistema de tempo real for catastrófica, então o sistema referido é “*hard real*

time" e esse tipo de sistema é empregado em uma vasta variedade de aplicações modernas como, por exemplo, sistemas embarcados e controles de plantas industrias. Segundo (vide (STANKOVIC, 1996a)) núcleos de tempo real estão sendo estendidos para operar com multiprocessadores altamente cooperativos em ambientes distribuídos.

O algoritmo "Deadline-Monotonic", analisado em (AUDSLEY, 1992), mostrou-se um ótimo esquema de escalonamento de prioridade fixa. Nesta obra, apresenta-se um teorema que diz: "se qualquer algoritmo de escalonamento com prioridade estática pode escalonar um processo onde o prazo final é menor que o período, um algoritmo usando o "Deadline-Monotonic" também escalonará tal processo" essa premissa é valida somente para tarefas periódicas. As tarefas sem periodicidade, ou seja, as não-periódicas são subdivididas em duas categorias, aperiódicas e esporádicas.

Já, por outro lado, o EDF (vide (LIU, 1973)) é um algoritmo baseado em prioridade dinâmica, ele é considerado um algoritmo de escalonamento dinâmico ótimo e as atribuições das prioridades no escalonador sob a política EDF são ordenadas segundo o prazo final das tarefas. A prioridade mais alta é dada à tarefa que tiver o seu *Deadline* mais próximo do tempo atual. A cada instante quando da chegada de uma nova tarefa, há uma nova redistribuição das prioridades. Outro aspecto importante que deve ser realçado nos algoritmos de escalonamento, é o fator de utilização do processador, no algoritmo EDF essa grandeza é considerada importantíssima para a obtenção do resultado final do teste de escalonabilidade. Sendo um conjunto de tarefas periódicas escalonável pelo algoritmo EDF, se o somatório do dividendo dos tempos de computação de cada tarefas, pelos seus respectivos períodos for menor ou no máximo igual a um. Além disso, levando-se em conta suas funcionalidades baseadas nos aspectos operacionais como preemptibilidade onde o processamento de uma tarefa pode ser interrompido pelo pedido de ativação do escalonador por outra tarefa de maior prioridade. Comparando-se o algoritmo EDF com o RM, o primeiro produz menos preempção

do que o RM, em contrapartida o último é muito mais simples quanto a sua implementação. Algoritmos escalonadores, como o EDF (vide (Liu, 1973)) em que os autores demonstram analiticamente todas as faculdades desta estratégia de escalonamento e comprovam matematicamente que se existir uma solução de escalonamento ótima para um determinado conjunto de tarefas, certamente o algoritmo EDF também irá encontrá-la.

2.4 ALGORITMO TAFT

Nesta seção apresenta-se o mecanismo de escalonamento para o TAFT – *Time Aware Fault Tolerant* – utilizado em (BECKER, 2003), cujo objetivo é manter o índice de utilização da CPU elevado, além de certificar que os prazos finais sempre serão atendidos. Além disso, o mecanismo adotado permite tratar situações de sobrecarga transiente (GERGELEIT, 2002). Em síntese, a proposta de escalonamento é um mecanismo que oferece garantias relacionadas para o cumprimento de requisitos temporais especificado na etapa de modelagem de uma aplicação tempo real.

Na política de escalonamento TAFT – *Time Aware Fault Tolerant* – , cada tarefa é designada como um par, denominado como *TaskPair* – TP –, que tem um único Deadline. Este par de tarefas tem definido uma parte principal – *MainPart*– e uma parte de exceção – *ExceptionPart*– como mostra a FIGURA 1

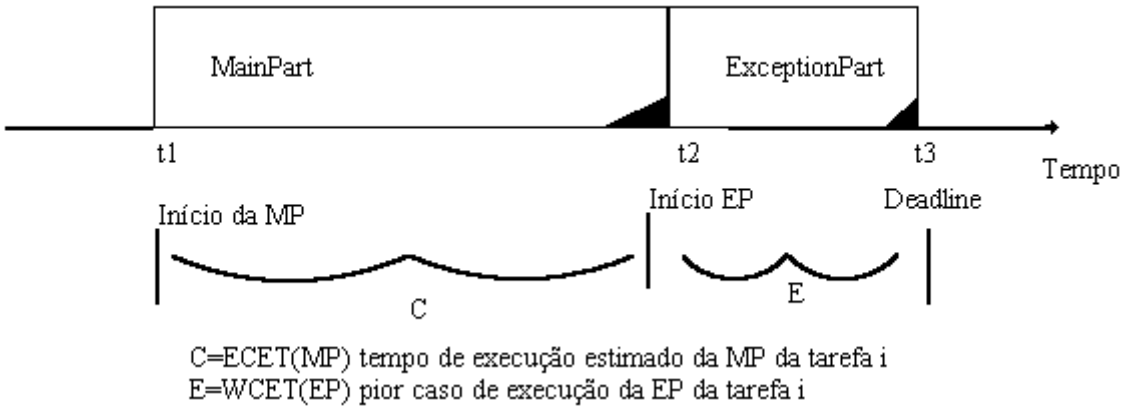

Figura 1 Estrutura de um *TaskPair*

Reside nestas duas partes o aspecto tolerante a falhas da política de escalonamento TAFT. Na MP encontram-se as funcionalidades das aplicações, a EP somente é acionada se a sua MP correspondente não for finalizada antes do seu respectivo *Deadline*. Assim, o mecanismo deve garantir que a EP será concluída antes do seu *Deadline*. Contrário senso da MP, que possui o código da aplicação, na EP tem-se o mínimo de funcionalidade implementada, assegurando-se apenas que a aplicação controlada fique num estado seguro e que o sistema de controle esteja num estado consistente.

De maneira geral, pode-se definir uma aplicação TAFT como sendo um conjunto de tarefas independentes, onde a cada tarefa associa-se um TP, ou seja, uma tupla composta por uma MP e uma EP com seus atributos como *Deadlines* e tempo de ativação T. O tempo de execução E da EP é definido como sendo o “tempo de execução de pior caso” (WCET), e na MP o tempo de execução C é o “tempo esperado de execução” (ECET). Em resumo, representa-se uma aplicação TAFT com *n TaskPairs* como : $\Pi = \{\tau = (T,D,C,E), i= 1$ até $n\}$

As (MP’s e EP’s) são tratadas como entidades de escalonamento distintas com seus próprios atributos. O escalonador do TAFT deve sempre garantir que, ou a MP ou a EP termine antes do *Deadline*, sendo este o aspecto tolerante a falhas. Percebem-se duas estratégias distintas para o escalonamento das MP’s e EP’s, isso define um escalonamento hierárquico.

3 ANÁLISE DO ESTADO DA ARTE

Este capítulo trata da análise do estado da arte sobre sistemas operacionais de tempo real. De um modo geral, são explorados os aspectos relevantes da teoria de escalonamento em sistemas operacionais de tempo real.

O estado da arte e as perspectivas para sistemas embarcados e RTOS são tratados em (OLIVE, 2000) da seguinte maneira. Os autores abordam em sua obra, diferentes aspectos considerados em tratando-se de (RTOS) tais como: tamanho, modularidade, preço, confiabilidade e qualidade de projeto. Os sistemas operacionais dedicados para sistemas embarcados diferem-se dos demais nos seguintes quesitos: custos, confiabilidade e qualidade. As principais propriedades em (RTOS), segundo os autores são: deadline, previsibilidade e confiabilidade.

Dispositivos lógicos programáveis, como, por exemplo, (FPGA's) – que permitem a integração lógica acelerando a prototipação do sistema – têm sido utilizados em projetos de sistemas embarcados. A alta densidade de integração lógica, fator que é inerente às FPGA, possibilita que certas funções de *Software* sejam implementadas em *Hardware*, assim, influenciando diretamente na redução do tempo de desenvolvimento do sistema. Além disso, como parte do projeto de *Software* é executado em *Hardware* através de descrição de *Hardware* (VHDL), obtém-se um ganho de desempenho devido ao particionamento do (RTOS), onde algumas funcionalidades do *Software* são transferidas para o *Hardware*. Como resultado, tem-se a redução da carga de processamento – overhead – bem como, a compactação no fluxo de tempo na execução do projeto.

Ainda nesta seqüência, relacionado ao desenvolvimento de projeto, o artigo de (NITSCH, 2000) aborda o particionamento do desenvolvimento em: *Hardware* responsável pelas funcionalidades implementadas em (ASICs) – circuitos de aplicação específica – , e

Software que tem a responsabilidade sobre o código executivo do microcontrolador. Os autores consideram como sendo a maior desvantagem no método convencional de implementação de projetos, o fato de não se poder iniciar o desenvolvimento de *Software* antes que o desenvolvimento de *Hardware* esteja inicializado. Esse fato eleva os custos do projeto, para minimizar essa desvantagem, é proposta pelos autores a utilização de métodos matemáticos para descrever analiticamente o sistema a ser projetado, dentre estes, destacam-se redes de petri, máquinas de estado e linguagem de programação paralela. Isso, segundo os autores, acarreta numa melhor descrição do sistema, possibilitando desta forma, maior probabilidade de êxito no desenvolvimento do projeto, bem como, no particionamento dos processos. Os pesquisadores, ainda, defendem em seu trabalho uma metodologia baseada em emulação, tanto de *Hardware*, como de *Software*, para evitar deste modo o risco de re-projeto na fase de desenvolvimento.

Já por outro lado, com a finalidade de analisar as características temporais que são inerentes aos sistemas de tempo real, tais como, previsibilidade e determinismo, os autores em (CHETTO, 1989) analisam o problema do escalonamento referente ao tratamento de tarefas periódicas em sistemas monoprocessados. Mais especificamente, o estudo dos pesquisadores é voltado para análise do período crítico, sendo investigados os problemas da estimação e a duração do tempo de ociosidade enquanto as tarefas são escalonadas pelo processador através do algoritmo (ED) "*Earliest Deadline*". Duas variações para o algoritmo (ED), uma chamada de (EDS) "*Earliest Deadline Soonest*", e a outra, de (EDL) "*Earliest Deadline Latest*", significando, tão breve quanto e tão tarde quanto são analisadas respectivamente. Os pesquisadores procuram analisar um algoritmo que mantenha um elevado fator de utilização do processador no tratamento de tarefas periódicas, buscando uma solução para resolução dos problemas que surgem no algoritmo "*Earliest Deadline*", com relação ao tratamento de tarefas aperiódicas, e também nos sistemas tolerantes a falhas que utilizam mecanismos de

escalonamento baseados em *Deadlines*, são investigados os problemas da estimação e duração do tempo de ociosidade do processador quando o mesmo estiver utilizando a política de escalonamento ED.

Cabe destacar, entre as várias obras estudadas para a realização desta análise do estado da arte (LU, 1999), onde os autores propõem um controlador PID – proporcional integral e derivativo- dedicado ao algoritmo de escalonamento, com a finalidade de reduzir a taxa de perda de *Deadlines*. Segundo os autores, o fator de perda de *Deadlines* depende da carga computacional do sistema, bem como, o fator de utilização de CPU. Assim, é apresentado um algoritmo de escalonamento baseado em um controlador PID com o EDF (FC-EDF) e todas as análises de estabilidade decorrentes da teoria de controle válidas são para o sistema proposto. Os resultados obtidos neste estudo mostraram que a taxa de perda de *Deadlines*, quando da utilização do (FC-EDF), foi reduzida sensivelmente quando comparado com outros algoritmos.

3.1 SISTEMAS OPERACIONAIS DE TEMPO REAL

(MILENKOVIC, 1992) caracteriza um sistema operacional como sendo um conjunto organizado de *Software* que executa o controle de rotinas e fornece um ambiente de execução para os demais processos do sistema. Em resumo, segundo o autor citado, os OS atuam como uma interface entre o usuário e o *Hardware* nos sistemas computacionais. A faixa e extensão dos serviços dos OS dependem de vários fatores. Internamente um OS atua como um gerente de recursos do sistema computacional, tais como, memória, dispositivos de entrada e saída. Cabe ao sistema operacional decidir sobre os acessos aos recursos de processamento. Além disso, ele define por quanto tempo e quando o dispositivo poderá utilizar-se de tais recursos.

Nos sistemas que suportam a concorrência entre os processos, o OS mantém a integridade do sistema.

Não obstante, a infra-estrutura da computação tempo real existente freqüentemente introduz formidáveis barreiras à continuação de fornecimento, atualização de equipamentos e agilidade em resposta a mudanças abruptas do mercado, referente ao aumento da competição em decorrência da globalização (STANKOVIC, 1996b).

Nos sistemas operacionais de tempo real, a entidade tempo recebe uma especial consideração, quando relacionado à execução de tarefas ele é especificado como um recurso a ser gasto durante a execução de um processo. Por outro lado, o tempo lógico é caracterizado a partir das relações de procedências entre os eventos. Através do tempo lógico é que o escalonador define as ordens de ativações de cada tarefa, que devem ser seguidas com a finalidade de alcançar o objetivo final, neste caso o escalonamento, sobre um conjunto de eventos. Em contrapartida, o tempo físico permite expressar quantitativamente à distância entre eventos, estabelecendo uma relação de ordem entre os mesmos.

Os sistemas operacionais de tempo real oferecem serviços aos quais possibilitam o funcionamento e a construção das aplicações com características tempo real. As garantias de atendimentos dos requisitos temporais dependem somente da aplicação, do sistema operacional empregado, do *Hardware*, e da rede de comunicação. No qual cabe exclusivamente ao SO possibilitar a previsibilidade.

Em muitos casos, os núcleos tempo real não possuem sistemas de arquivos ou gerência de memória para poder suprir as necessidades temporais das aplicações. Entretanto, mesmo oferecendo a mínima funcionalidade operacional esses núcleos são capazes de fornecer excelente comportamento temporal.

O que difere os sistemas operacionais convencionais de um RTOS é que estes, os convencionais, não têm a finalidade de apresentar um comportamento temporal previsível e

determinístico, mantendo o compartilhamento dos recursos computacionais do sistema de forma semelhante entre todas as tarefas, não tendo comprometimento com a previsibilidade e o determinismo temporal. Entretanto, os RTOS não distribuem uniformemente os recursos, e são dedicados a atender os requisitos como instantes de ativação e *Deadline*. Assim, os sistemas operacionais de tempo real têm suas funcionalidades voltadas para o comportamento temporal, sendo que os serviços são definidos basicamente em termos temporais.

Todo o sistema operacional tem como finalidade tornar a utilização do computador eficiente, por isso, algumas funcionalidades fornecidas por um sistema operacional de propósito geral são utilizadas em um RTOS. Todavia, deve-se observar que algumas funcionalidades que visam aumentar o desempenho médio do sistema operacional, podem gerar atrasos, como, por exemplo, mecanismo de memória virtual, e sistemas de arquivos.

Um sistema é previsível quando ele não estiver susceptível a qualquer variação, podendo ser o comportamento do sistema antecipado e de antemão conhecido. Para isso algumas premissas referentes às cargas e falhas devem ser explanadas: A carga é a entidade que determina a quantidade de processos que uma unidade de processamento, pode atender em um certo período correspondendo ao pico gerado numa certa variação de tempo originado pelo ambiente, pelos eventos periódicos e até mesmo aperiódicos. Já, as falhas descrevem a freqüência com que o sistema deve conviver em tempo de execução atendendo os requisitos temporais.

Estes dois fatores, carga e falhas, devem ser utilizados para a modelagem dos RTOS, assim, é necessário conhecer-se previamente o comportamento do sistema considerando-se a pior situação de carga, juntamente com as hipóteses de falhas.

O *Hardware* empregado nos sistemas, geralmente embarcados, é preponderante para a previsibilidade do RTOS. As instruções dos processadores, bem como, acesso direto à memória (DMA) e mecanismo de memória "*Cache*" são fontes de imprevisibilidade temporal.

A utilização desses expedientes dificulta a determinação com exatidão dos tempos de computação.

Sistemas operacionais de tempo real (vide (MILENKOVIC, 1992)) são usados em ambientes em que eventos computacionais, geralmente externos ao sistema, devem ser aceitos e processados num curto período de tempo com um certo *Deadline*.

Os computadores estão cada vez mais rápidos com relação à velocidade de processamento e, menores com relação à área ocupada e a potência dissipada pelos seus dispositivos. Estas características são preponderantes para a utilização dos sistemas computacionais em diversos processos produtivos. Os sistemas computacionais têm influenciado todas as atividades da vida moderna desde o controle de tráfico aéreo até o gerenciamento de usinas hidrelétricas.

Os sistemas operacionais de tempo real devem suportar características específicas para atender os requisitos mínimos exigidos pelos sistemas embarcados. Segundo (OLIVEIRA, 2003) as aplicações são mais facilmente construídas se puderem aproveitar os serviços de um sistema operacional convencional. Deste modo, o usuário não necessita preocupar-se com a gerência dos recursos, sendo essa função estritamente do SO. A análise da escalonabilidade é outro fator de muita relevância com relação aos RTOS. Nesta análise, é que se estima o tempo máximo de resposta para cada tarefa para comparar com os respectivos prazos finais, validando a escalonabilidade do sistema. Dentre os principais aspectos construtivos de um sistema operacional, a política de escalonamento utilizada é o fator mais preponderante com relação à carga computacional.

Outros aspectos muito importantes em se tratando de SO são o chaveamento de contexto e a latência. Dentre tantas métricas existentes na literatura dos RTOS cabe destacá-los pelo fato de que nestas métricas estão incluídos os tempos necessários para salvar os registradores decorrentes da tarefa em execução e carregar os novos valores relativos à nova

tarefa que será executada pela unidade de processamento. Cabe a ressalva que nesta métrica, chaveamento de contexto, não está incluído o tempo necessário para decidir qual a tarefa que será executada, sendo que isto, depende exclusivamente da política de escalonamento utilizada. Com relação a estas métricas, a qualidade de um RTOS se resume na menor quantidade de tempo tanto para a latência quanto para o chaveamento de contexto.

3.2 TRABALHOS RELACIONADOS

Visando auxiliar o desenvolvimento de sistemas tempo real vários estudos têm surgido nos últimos anos relacionados à arquitetura de sistemas embarcados. A comunidade científica de maneira geral tem dado especial atenção para os assuntos relacionados a RTOS. Logo, por isso, cabe ressaltar dois trabalhos, em nível de dissertação, desenvolvidos no Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, (PONTREMOLI, 1998) e (GÖTZ, 2001). Assim, como em (COOLING, 1997), (HALANG, 1992), (CRESPO, 2003), e (STANKOVIC, 1999b). Cabe realçar que na primeira obra citada o autor desenvolveu em seus estudos uma arquitetura de *Hardware* de baixo custo para desenvolvimento de aplicações com requisitos de tempo real para sistemas de controle distribuídos. Propõe-se o particionamento das atividades e funções específicas executadas pelo sistema operacional de tempo real. Com o intuito de minimizar a carga na CPU e, reduzir conseqüentemente, o escalonamento intrusivo de tarefas, bem como, garantir o melhor gerenciamento de tarefas com requisitos assíncronos, e a ativação das tarefas com requisitos temporais. As atividades funcionais que outrora eram executadas apenas pela unidade de processamento nas arquiteturas convencionais, na arquitetura proposta pelo o autor, são divididas em três grupos distintos: Bloco administrador, bloco executor e bloco comunicador. O bloco administrador tem a responsabilidade sobre as funções de escalonamento das tarefas, assim como, a gerência

sobre os eventos síncronos a assíncronos. O bloco executor é o responsável pelo processamento das tarefas da aplicação. E o bloco comunicador tem a responsabilidade sobre qualquer tipo de comunicação entre tarefas, pela comunicação entre diferentes unidades de processamento e com dispositivos de *Hardware* diversos como, por exemplo, sensores e atuadores inteligentes.

Os resultados obtidos com a nova arquitetura, segundo o autor, reduziram e mantiveram estável o “overhead” do sistema distribuído. Esse fato se dá, principalmente, pela inclusão de um bloco dedicado exclusivamente para o gerenciamento do escalonamento e tratamento de eventos síncronos e assíncronos. Ainda, sendo destacado no trabalho a redução no período “*polling*”, sem a presença dos efeitos colaterais de sobrecarga do sistema, o que ocorre em uma arquitetura convencional.

Na obra de (GÖTZ, 2001) utiliza-se o µClinux, uma variação do sistema operacional Linux para aplicações embarcadas. Na qual, tratando-se em nível de *Hardware* propõem uma arquitetura alternativa a fim de garantir o determinismo temporal, com dois microcontroladores de 32 bits, um deles dedicado às tarefas da aplicação e o outro sendo responsável pelas atividades de comunicação, gerenciamento temporal, escalonamento, e do tratamento sinais assíncronos. Logo, pela proposta do autor, o particionamento se dá em duas partes; uma sendo o bloco principal e o outro bloco secundário. As principais atividades do bloco principal são: execução das tarefas de aplicação, gerenciamento da área de memória, troca de contexto e a comunicação com o bloco secundário. Por outro lado, são de responsabilidade do bloco secundário o gerenciamento de sistemas de arquivo, execução do algoritmo de escalonamento, tratamento de sinais assíncronos e o gerenciamento da comunicação com o meio externo. Cabe ressaltar que no projeto de *Hardware,* o autor utiliza memórias DPRAM, pois esse tipo de dispositivo proporciona maior versatilidade no sistema, já que os dados podem ser lidos de forma arbitrária em relação à seqüência de escrita. Outro

fator preponderante no projeto de *Hardware* é a comunicação com o meio externo, que se dá por um canal de comunicação CAN.

O autor no seu estudo apresentou uma arquitetura para aumentar a previsibilidade e o determinismo temporal. Os testes realizados mostraram que o canal de comunicação entre os dois módulos pode limitar a capacidade do sistema. Ainda se obteve, segundo o autor, com a arquitetura, uma redução no tempo de processamento do algoritmo de escalonamento.

Os autores em (ANDREWS, 2004) estão propondo e desenvolvendo pesquisa similar ao deste trabalho, objetivando obter-se melhor desempenho de processamento através do particionamento das atividades entre as unidades de processamento. Basicamente, os autores propõem sintetizar numa FPGA as operações inerentes de um RTOS, migrando desta forma funções como tarefas de escalonamento e gerenciador de filas de eventos entre outros. Os autores acreditam desta forma aumentar o desempenho e previsibilidade de sistema pois, em tratando-se de FPGA, propriedades como resolução e precisão são melhorados. Deste modo, aumentado a eficiência do evento de escalonamento, aspecto fundamental para qualquer sistema de tempo real.

Em (CRESPO, 2003) é analisado o percalço decorrente da segurança na execução das tarefas aperiódicas com relação ao cumprimento dos seus respectivos prazos finais. A garantia dos prazos finais é avaliada através de um teste de aceitação baseado na extensão do algoritmo “Slack Stealing” para o EDF, de tal modo que ele garanta o tempo de resposta para uma determinada tarefa aperiódica. Dentre as características desta obra cabe destacar que a analise de escalonamento é efetuada em uma arquitetura multiprocessada explorando a máxima utilização das unidades processadoras.

Em (HALANG, 1992) é apresentada uma arquitetura multiprocessada especialmente dedicada para aplicações “*Hard Real Time*”. Algumas características relacionadas aos RTOS são implementadas em um processador. Basicamente as arquiteturas multiprocessadas são

projetadas para fornecerem maior taxa de utilização dos recursos computacionais do sistema. Dentre as propriedades indesejáveis, inerentes às arquiteturas computacionais, os autores destacam a imprevisibilidade. Segundo os autores o aumento de desempenho é diretamente influenciado por dois fatores, um sendo os constantes avanços tecnológicos na área da ciência dos materiais proporcionando, deste modo, a construção de novos componentes utilizados nas arquiteturas. E as novas concepções de projetos como processamento paralelo e sistemas de memórias "C*aches*". Partindo da premissa de que todas as tarefas de um sistema de tempo real devem ser completadas antes de seus prazos finais. Em um sistema multi-tarefa cabe ao algoritmo de escalonamento dinâmico gerar o escalonamento viabilizando a execução de todas as tarefas pelo processador. Para isso os autores elaboram uma arquitetura hierárquica, tendo-se um processador de tarefas e um processador de "*kernel*" separados um do outro. Os autores defendem que os processos externos e processos provenientes dos periféricos sejam controlados pelo processador de tarefas. A arquitetura do sistema proposto é subdividida hierarquicamente em três camadas. Uma chamada de *Hardware* e as outras duas denominadas de primeira e segunda camada respectivamente. Como a transferência de dados é o maior problema causado pelo barramento das arquiteturas comuns. No sistema proposto, as aplicações são executadas pelo "*Task Processor*" sem a ocorrência de sobrecarga decorrente do sistema de comunicação, assim a tarefa é inicializada pelo processador e é apenas preemptada se realmente necessária.

Na arquitetura proposta em (COOLING, 1997) são analisados os aspectos essenciais no que diz respeito ao determinismo em tratando-se de sistemas embarcados modernos. Os autores descrevem o uso de um co-processador, baseado em um microcontrolador Intel "8032", para efetuar os serviços relativos ao escalonamento de tarefas, como funções básicas de centralização das tarefas de controle do sistema, além do suporte algoritmos da literatura de tempo real como, por exemplo, o RR. Os atributos como tempo e criticabilidade

juntamente com o escalonador são tratados como tópicos relevantes neste estudo. Sendo a unidade de escalonamento tratada como uma entidade que decide quando uma determinada tarefa deve ser executada. Segundo os autores a comunicação pode tornar-se, dependendo de aplicação, no fator preponderante no cumprimento dos prazos finais das tarefas. Objetivamente os autores buscam reduzir a sobrecarga computacional com a implementação das atividades de escalonamento em *Hardware.* Na concepção da arquitetura os autores consideram o chaveamento de contexto como o principal responsável pelo aumento da sobrecarga.

Dentre os principais fatores relevados na arquitetura proposta os autores destacam o desempenho computacional, interface de comunicação e a linguagem utilizada para implementar as funções no microcontrolador. Com relação ao desempenho, sendo este considerado o principal aspecto da arquitetura multiprocessada, os autores reduziram as operações de multi-tarefa do sistema, desta forma, foram dirimidos os efeitos provenientes das interrupções. Na arquitetura proposta, os autores decidiram que no máximo 32 tarefas seriam atendidas pelo co-processador. As razões para essa especificação da arquitetura, segundo os autores, provêem de que na industria raramente utiliza-se mais do que 30 tarefas e, em segundo que para aplicações com criticabilidade "*Hard*" deve-se manter o mínimo de tarefas possível sob comando do escalonador.A interface de comunicação é outro aspecto de suma importância com relação à interconexão entre as unidades cooperantes no que tange o envio de dados entre as plataformas processadoras constituintes de sistema.

Em (STANKOVIC, 1999b) é proposto um co-processador dedicado para auxiliar a unidade de processamento principal na execução do algoritmo de escalonamento do sistema operacional *SPRING* (STANKOVIC, 1999a). Os autores utilizaram-se de tecnologia CMOS para a implementação do módulo de co-processamento da arquitetura. Nesta obra são

analisados os pontos cruciais em relação à constrição no fluxo de dados decorrente da comunicação entre as unidades de processamento.

Segundo os pesquisadores, em tratando-se da nova geração de RTOS diferentes novas características devem ser atendidas pelo algoritmo de escalonamento.Incluem-se dentre estas determinismo, e previsibilidade, além de suporte as variações temporais provenientes dos ambientes dinâmicos. Nos sistema complexo não somente previsibilidade é exigida quanto ao gerenciamento de estratégia de escalonamento, mas também adaptabilidade e flexibilidade. A adaptabilidade refere-se às mudanças que o sistema deve sofrer quando no tratamento de ambientes dinâmicos a fim de atender, completamente, todos os requisitos temporais, enquanto que flexibilidade demanda das alterações na estrutura de *Hardware* e *Software*. Os autores justificam em sua obra a utilização de um modelo cooperante de processamento devido ao fato de que a aplicabilidade dos algoritmos dinâmicos é determinante na sobrecarga computacional do sistema. Em uma arquitetura monoprocessada e, com tarefas independentes entre si os algoritmos do tipo EDF e LLF são ótimos, porém para sistemas dinâmicos multiprocessados e com tarefas que utilizam sistema de exclusão mutua não há algoritmo ótimo. Partindo deste principio e com o objetivo de minimizar este problema os autores utilizaram-se de meios heurísticos na implementação do sistema *SPRING* de co-processamento.

Os pesquisadores têm por objetivo principal, na sua pesquisa, buscar uma solução que utilize o mínimo tempo necessário para a execução do algoritmo de escalonamento assim, são considerados como aspectos a serem melhorados as respostas temporais, os requerimentos de pré-processamento, e a própria operação de escalonamento. Os três elementos principais de operação de escalonamento, na arquitetura proposta, são a execução do módulo de co-processamento *SPRING* e as atividades de pré e pós-processamento.

Os resultados da análise realizada pelos autores foram obtidos de três protótipos VLSI onde, os mesmos consideraram as diferentes densidades de área e performance de processamento entre os parâmetros de prototipação para síntese práticas co-processador em FPGA. Os autores utilizaram a título de resultado, para a validação da arquitetura proposta, a variação na taxa de freqüência de operação do circuito, além da variabilidade na quantidade de tarefas do sistema multiprocessado.

4 ARQUITETURA PROPOSTA

A troca de funcionalidades entre *Software* e *Hardware* em arquiteturas computacionais é uma técnica utilizada pelos projetistas de sistemas embarcados para agilizar e, conseqüentemente, reduzir o tempo de projeto. Através deste método algumas funcionalidades dos (RTOS) que, outrora eram implementadas em plataforma baseada em *Software*, passam a ser executadas em *Hardware*. Desta maneira podem-se eliminar possíveis sobrecargas computacionais com a redução de tarefas implementadas por *Software*, pois uma arquitetura em *Hardware* especialmente projetada comportará algumas atribuição e funções das tarefas reduzindo, desta forma, em parte a sobrecarga computacional vide ((ANDREWS, 2004)).

Os mecanismos de escalonamentos estão entres as principais partes funcionais dos sistemas embarcados, pois cabem a eles, conforme a política empregada, garantir que todas as tarefas consigam executar suas atividades antes dos seus respectivos *Deadlines*.

4.1 PROJETO CONCEITUAL

A nova concepção de arquitetura baseou-se na alteração da plataforma RTLinux, onde algumas atividades como, por exemplo, o processamento do algoritmo de escalonamento, passou a ser executado, em parte, em um novo dispositivo de *Hardware* dedicado especialmente para essa tarefa. Conceitualmente a FIGURA 2 mostra as diferenças básicas entre a arquitetura existente anteriormente e a nova arquitetura após as mudanças realizadas. Onde se tinha apenas uma unidade de processamento, agora, tem-se um *Hardware* para auxiliar o processador principal nas tarefas relacionadas ao escalonamento de processos.

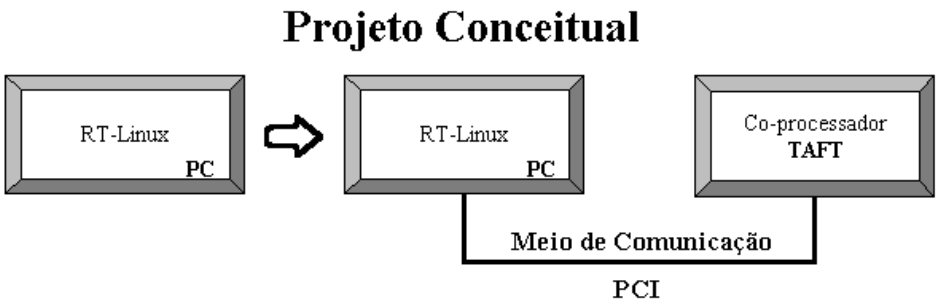

Figura 2 Organização Conceitual da Arquitetura

A unidade de escalonamento, na arquitetura proposta, é constituída por três dispositivos: Placa de interface *PLX* PCI, placa com bloco de memórias DPRAM e a placa de processamento com o processador "*ColdFire*" como mostra, de maneira mais específica, a FIGURA 3.

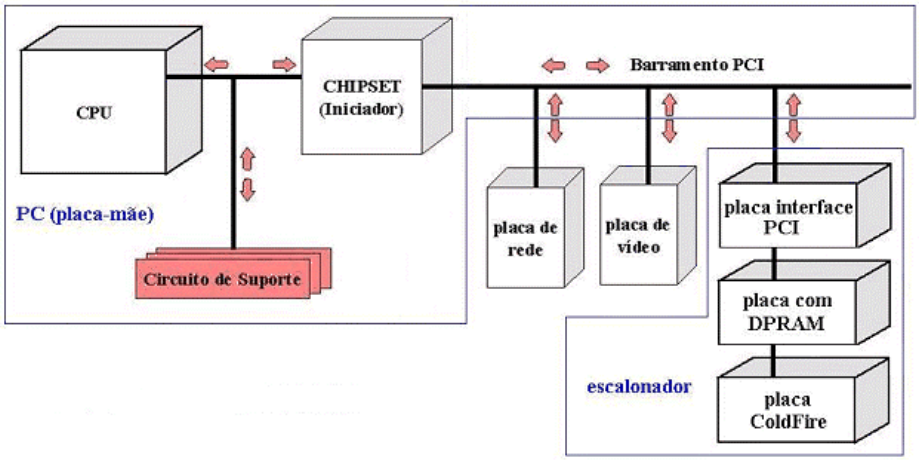

Figura 3 Co-processador de Escalonamento

Na arquitetura proposta, o algoritmo de escalonamento do RTOS é executado no *Hardware* de co-processamento. Desta forma, o processador principal se responsabiliza apenas pelo despacho das tarefas (ao invés de ter que escaloná-las). Isto significa que, o mesmo tem por função executar todas as tarefas na sequência definida pelo escalonador.

Para validação da proposta utiliza-se um computador IBM-PC compatível equipado com o *Hardware* de co-processamento proposto, onde se executa o sistema operacional tempo real RTlinux-3.0, o qual precisou ser adaptado para realização da interação entre a unidade de processamento principal e a plataforma de co-processamento, permitindo, deste modo, que o algoritmo escalonador fosse implementado e posteriormente executado em uma plataforma de *Hardware*. O algoritmo empregado no co-processador de escalonamento foi o TAFT pelas razões já mencionadas. Além disso, foram realizados experimentos práticos, os quais comprovam as possíveis vantagens da abordagem proposta.

O *Dispatcher* tem por função executar o chaveamento de contexto entre as tarefas, conforme definido pelo escalonador. A FIGURA 4 ilustra a configuração da arquitetura proposta em comparação com uma arquitetura tradicional.

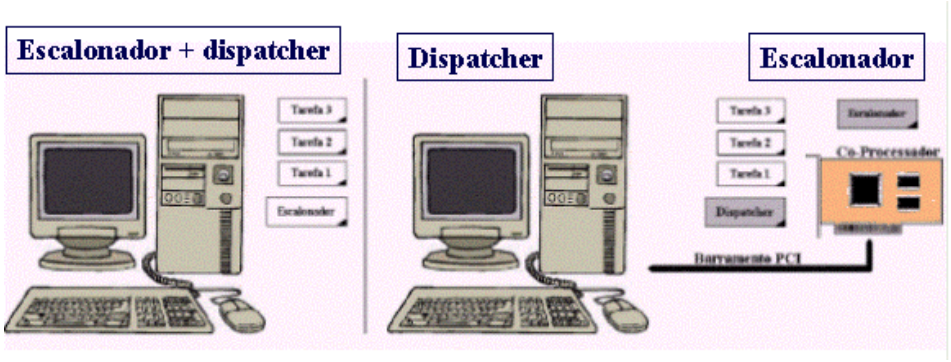

Figura 4 Comparação entre arquitetura tradicional e a arquitetura proposta

Seguem, respectivamente, as descrições em maiores detalhes das duas partes principais da arquitetura desenvolvida: A estrutura de *Hardware* da placa de co-processamento e a estrutura do *Software* de gerenciamento (RTOS modificado).

Pode-se observar, com o auxílio da FIGURA 4, que na nova concepção de arquitetura o dispatcher e o escalonador estão separados, já na concepção tradicional ambos coexistem sob mesma plataforma computacional. Deste modo, a premissa da arquitetura proposta está relacionada ao particionamento das atividades específicas relacionadas à unidade de

processamento responsável por algumas funções específicas como, por exemplo, o tratamento dos algoritmos de escalonamento.

Verifica-se que na arquitetura proposta o *Dispatcher* continua no mesmo ambiente operacional, neste caso o sistema RTLinux. Porém, transferiu-se a unidade de escalonamento para uma unidade auxiliar de processamento. Cabe destacar que o *Dispatcher* é o responsável pelo gerenciamento das atividades operacionais necessárias para o funcionamento pleno da arquitetura. As principais vantagens desta participação residem no fato de que o RTOS não compete com as tarefas.

Tabela 1 Status de Eventos operacionais do módulo Co-processador

Eventos	Valores de Status	Ação
TASK_CREATE	0x0034	Cria uma nova tarefa
TASK_START	0x0010	Ativa a execução de tarefa
TASK_STOP	0x0001	Pára a execução de tarefa
TASK_DELETE	0x0057	Retira uma tarefa da estrutura

Para a detecção dos eventos gerados no co-processador foi elaborada uma tabela com os valores utilizados para a monitoração e o reconhecimento dos mesmos. Estes valores são códigos de informação padronizados referentes a cada evento específico dos *TaskPair's* de cada tarefa existente a ser escalonada. A TABELA 1 mostra os valores referentes aos eventos no dispositivo co-processador, enquanto que a TABELA 2, logo a seguir, refere-se aos valores de status dos eventos gerados pelo *Dispatcher*.

Tabela 2 Status de Eventos operacionais da plataforma RTLinux

Eventos	Valores de Status	Ação
CREATE_ID	0x30	Retorna identificação da tarefa criada
START_ID	0x15	Indica ativação da tarefa identificada
STOP_ID	0x40	Indica desativação da tarefa identificada
DELETE_ID	0x12	Retorna identificação da tarefa criada
SCHED_MP_ID	0x52	Indica escalonamento da MP identificada
SCHED_EP_ID	0x25	Indica escalonamento da EP identificada
FIN_TASK_ID	0x93	Indica sinalização da tarefa identificada

As TABELAS 1 e 2 mostram os valores de comandos de execução gerados no módulo co-processador e na plataforma RTLinux, com seus respectivos valores. Estes valores são utilizados na monitoração e detecção dos eventos. Entre os eventos descritos por esses valores estão a criação de uma nova tarefa, escalonamento de uma tarefa, e identificação de uma tarefa, entre outros. Tanto o código do escalonador residente no processador "*ColdFire*" quanto o código no "Driver" *PLX* existente na plataforma RTLinux foram instrumentados de maneira a reconhecer e a gerenciar a ocorrência dos eventos em ambos os dispositivos. Cada evento, como por exemplo, "SCHED_MP_ID", possui o seu respectivo valor de identificação, que é o valor "0x52" neste caso, fornecido para placa de co-processamento e enviado para a plataforma RT-Linux através do banco de memória. Todo a tráfego de dados entre os dispositivos da arquitetura proposta é executado com o auxílio das memórias do tipo FIFO. Estes dispositivos de memória são partes integrantes do banco de memória.

Os quatro primeiros eventos da TABELA 1 TASK_CREATE, TASK_START, TASK_STOP e TASK_DELETE são eventos enviados pela plataforma RTLinux e recebidos pelo co-processador. Estes eventos têm como finalidade configurar o co-processador referente às características agendadas pelo sistema operacional RTLinux. Já, os outros comandos, como por exemplo, CREATE_ID, START_ID, STOP_ID e DELETE_ID são gerados e enviados pelo co-processador à plataforma RTLinux como forma de sinalização do atendimento das configurações enviadas. Além disso, os comandos SCHED_MP_ID, SCHED_EP_ID e FIN_TASK_ID servem para indicar ao *Software* "*Dispacher*" executando na plataforma RTLinux da necessidade de ativação de novas tarefas (SCHED_MP_ID para MPs ou SCHED_EP_ID para EPs) ou desativação de uma dada tarefa em execução (FIN_TASK_ID). Estes eventos têm a função de manter informado o sistema RTLinux das atividades executadas pelo escalonador a cada instante pela unidade co-processadora. Seguem as especificações das ações referentes a cada evento envolvido na arquitetura de co-

processamento. Primeiramente, são especificados os eventos que a plataforma, sob o sistema operacional RTLinux, envia para a unidade de co-processamento:

TASK_CREATE: comando criado para informar ao co-processador a requisição da criação de uma nova tarefa.

TASK_START: ativa a execução da tarefa possibilitando que a mesma possa concorrer pelo uso da unidade de processamento.

TASK_STOP: comando que solicita a desabilitação de uma determinada tarefa que estava sendo escalonada.

TASK_DELETE: remoção da tarefa da estrutura de escalonamento.

Os eventos restantes são referentes aos comandos gerados pelo co-processador os quais são enviados para a plataforma RTLinux:

CREATE_ID: gera uma identificação da tarefa recentemente criada.

START_ID: confirma a colocação de uma dada tarefa na lista de escalonamento.

STOP_ID: confirma a desabilitação de uma dada tarefa (modo de espera).

DELETE_ID: confirma que a tarefa foi apagada e retira da estrutura de escalonamento.

SCHED_MP_ID: sinaliza ao sistema operacional que a MP indicada foi escalonada e deve ser posta em execução.

SCHED_EP_ID: sinaliza ao sistema operacional que a EP indicada foi escalonada e deve ser posta em execução.

FIN_TASK_ID: sinaliza ao sistema operacional que a tarefa indicada teve seu tempo de execução cumprido e deve ser finalizada.

4.2 PLANEJAMENTO EXPERIMENTAL

Para avaliar-se a arquitetura proposta, foram definidos alguns ensaios aos quais utilizou-se para validar a arquitetura proposta. Para a realização destes testes, acoplou-se ao módulo de co-processamento em um computador IBM-PC equipado com processador ATHLON operando com freqüência de 1096,136 MHz, 256 MBytes de memória RAM e disco rígido de 40 GBytes.

Com a finalidade de analisar a nova arquitetura proposta, foram planejados os seguintes testes primeiramente, foram mensuradas as variações nas ativações periódicas entre duas tarefas executadas exclusivamente no co-processador. Para este teste, o ambiente operacional foi configurado de maneira que não houvesse nenhuma comunicação entre o co-processador e a plataforma RTlinux. Deste modo, foram configurados os atributos de maneira “off-line”, ou seja, diretamente com o auxílio do *Software* “Codewarrior” os valores desejados para os atributos das tarefas como Deadlines, ECET e WCET foram atribuídos através de uma tabela no código do algoritmo de escalonamento. Assim, não sendo ativada a troca de mensagens entre o sistema RTlinux e o co-processador, logo, a utilização do banco de memórias (DPRAM) neste procedimento de teste não se fez necessário. Levou-se em conta nesta validação temporal os seguintes aspectos, o período do processador “*ColdFire*” foi fixado em 40 MHz, e mantiveram-se inalterados os valores atribuídos para cada tarefa como *Deadline*, WCET e ECET. Ressalta-se que para este ensaio o algoritmo de escalonamento foi executado exclusivamente pelo processador “*ColdFire*”.

Além disso, para este ensaio definiu-se um conjunto com dois pares de tarefas (TP’s), as quais são caracterizados na TABELA 3. Esta tabela contém informações sobre o período, o *Deadline*, o tempo de execução esperado da MP e o tempo de execução de pior caso da EP.

Tabela 3 Parâmetros dos TaskPairs utilizados nos ensaios

TP	Período	Deadline	Tp. MP	Tp. EP
1	1 s	1 s	120 ms	16 ms
2	2 s	2 s	520 ms	4 ms

Procurou-se caracterizar, com os parâmetros da TABELA 3, o comportamento do algoritmo de escalonamento implementado no módulo de co-processamento. Para tanto, procurou-se observar a existência de anomalias de ativações nas instâncias dos TP's em decorrência dos chaveamentos de contexto.

Já, por outro lado, com relação ao número de tarefas procurou-se estimar a influência do aumento no número de tarefas na sobrecarga computacional do sistema RTLinux.

Para a obtenção destes dados, foram realizados ensaios similares para todos os algoritmos analisados na plataforma RTlinux-3.0 (BARABANOV, 1997). Os dados relacionados a cada algoritmo escalonador foram obtidos com o auxílio da ferramenta de monitoração [1]JEWEL (GERGELEIT, 2001). A título de cálculo, foram usados cerca de 300 eventos referentes à variação entre duas e nove tarefas tempo real (*Threads*) no sistema. O ambiente utilizado nos ensaios de sobrecarga, referente a cada algoritmo, foi igualitário em todos os aspectos, não havendo disparidade nos métodos empregados de forma que viessem a comprometer o desempenho dos algoritmos envolvidos na análise. Assim, todos os algoritmos foram submetidos ao mesmo ambiente computacional, não ocorrendo nenhum procedimento que invalidasse os resultados obtidos. Os algoritmos analisados neste ensaio, foram o EDF, e duas variações do algoritmo TAFT, implementado com o mecanismo de prioridade e sem o mecanismo de prioridade

Planejou-se também uma análise estatística semelhante àquela especificada nos primeiros ensaios, porém com o objetivo de determinar a diferença entre os instantes em que

os eventos são detectados pelo “Driver” PCI *PLX* no processador principal (PC) e os instantes em que são gerados pelo escalonador no co-processador.

As análises dos resultados de todos ensaios realizados nesta obra foram executadas com o auxílio da ferramenta de análise BR-Tool (HUSEMANN, 2002), através desta ferramenta tornou-se possível quantificar os tempos de comunicação entre o “Driver” PCI *PLX* e o mecanismo de despacho.

Posteriormente, na seção análise dos resultados, são mostrados os resultados obtidos para este ensaio, assim como, para os demais testes planejados, com a finalidade de caracterizar a plataforma de co-processamento.

4.3 IMPLEMENTAÇÃO

4.3.1 Módulo do Co-Processador

Para desenvolvimento do co-processador de escalonamento foi utilizada como base uma placa microprocessada desenvolvida originalmente para executar a monitoração e a captura de eventos em uma rede de barramentos industriais (HüSEMANN, 2003). Esta reutilização foi possível graças a sua arquitetura genérica. Uma representação da arquitetura de *Hardware* utilizada pode ser observada na FIGURA 4, sendo subdividida em três partes funcionais: interface PCI, memória compartilhada e a unidade de processamento.

[1] O software JEWEL é utilizado para a monitoração e validação temporal dos eventos em ambiente tempo real.

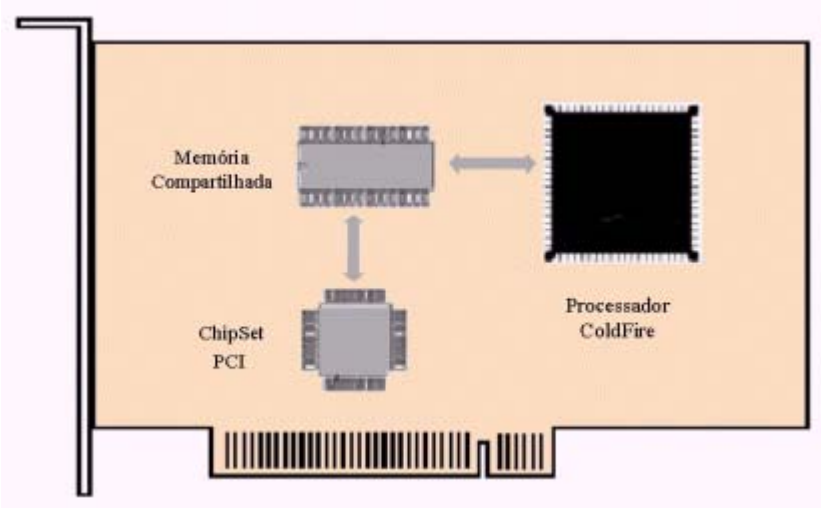

Figura 5 Representação do Módulo de Co-processamento

A placa do co-processador utiliza um microprocessador de 32 bits da família “Coldfire” da Motorola como unidade de processamento (HüSEMANN, 2003). A freqüência de operação do microprocessador é de 40MHz, como performance de até 33MIPS, o que permite atender requisitos temporais na faixa de 50 microssegundos. Esta é uma resolução considerada satisfatória, principalmente se comparada aos sistemas operacionais tradicionais que garantem resolução temporal de algumas unidades de milissegundos. Para os testes de validação da arquitetura proposta foi implementado no co-processador o mecanismo de escalonamento TAFT. Este microprocessador é o principal elemento da placa de co-processamento, pois nele são executados os algoritmos de escalonamento que interagem com o RTOS. A FIGURA 6 descreve a arquitetura funcional em diagrama de blocos.

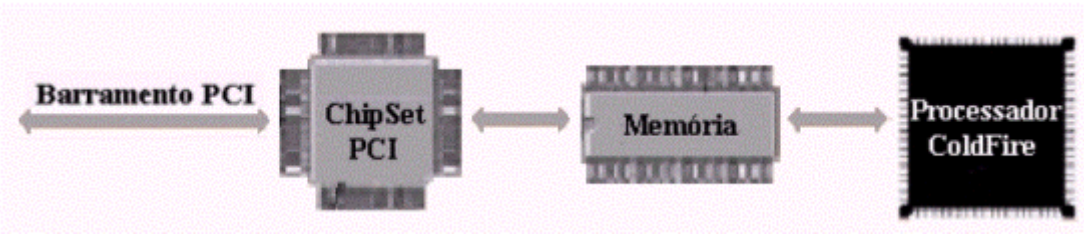

Figura 6 Arquitetura de Hardware

4.3.2 Barramento PCI

O barramento PCI é a solução para adaptadores que necessitam de velocidade na troca de dados com outros dispositivos. Através dele, as transferências ocorrem na forma denominada modo rajada (ZELENOVSKY, 2002). Dentre as características deste barramento cabe destacar a sua universalidade. Além disso, todas as leituras e escritas são realizadas em pacotes de dados, o que proporciona uma taxa de comunicação entre os dispositivos próximos a 132 MB/s (33MHz e 32 bits), 264 MB/s (33MHz e 32 bits) ou 528 MB/s (66MHz e 64 bits) e prevê detecção de erros de paridade em dados ou endereços durante as transações.

(HüSEMANN, 2003) utiliza este protocolo para o desenvolvimento de sistemas embarcados. O mesmo autor usa esta interface de comunicação na implementação de uma placa de aquisição dedicada à monitoração de sistema de validação para aplicações desenvolvidas sobre tecnologias de barramento de campo.

Ainda, relacionada com as arquiteturas desenvolvidas sob a interface de comunicação PCI é destacada a obra (MINK, 1998), onde se utilizou um *Hardware* dedicado com o barramento PCI, chamado de MultiKron, para executar a monitoração e controle em ambientes distribuídos que requeiram aplicações de tempo real, e processamento paralelo.

4.3.3 Arquitetura de Hardware PLX

A placa *PLX* 9052 é uma solução em *Hardware* dedicada à interconexão de dispositivos para a comunicação de dados via interface PCI, tendo como principais características, a alta velocidade na transmissão de dados, e o baixo custo de desenvolvimento. A placa *PLX* utiliza o protocolo PCI para realizar a comunicação entre os diversos adaptadores de I/O. A sua arquitetura, para alguns modelos, pode ser configurada para usar até 32 bits do barramento PCI com velocidade máxima de 33 MHz. A FIGURA 7 mostra o diagrama de blocos do dispositivo *PLX* 9052.

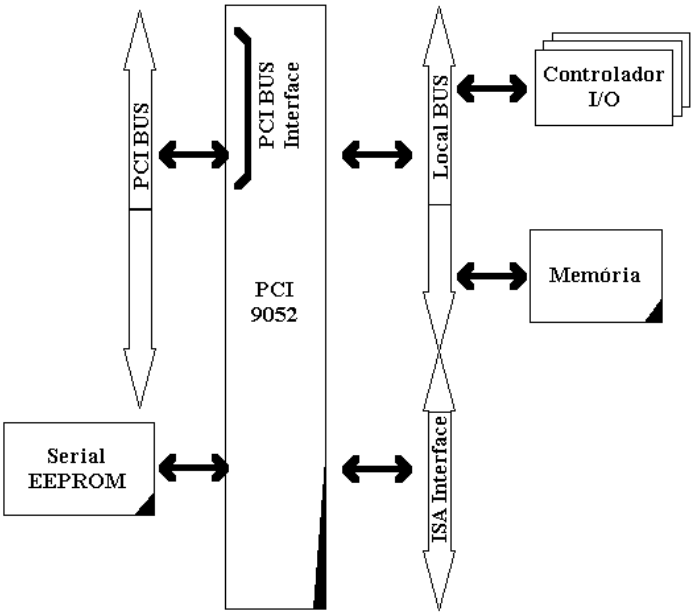

Figura 7 Diagrama de Blocos PLX 9052

A placa *PLX* 9052 foi configurada, para os ensaios deste estudo de maneira a acessar o barramento PCI com uma configuração no modo escrita/leitura em 16 bits. Utilizou-se essa configuração de 16 bits para escrita e leitura de dados na placa PLX pelo fato de ser a máxima configuração aceita por este modelo de *Hardware* chipset PCI. Como a arquitetura computacional IBM-PC é configurada em 64 bits para a execução das transações de fluxo de dados, a arquitetura projetada necessita de no mínimo quatro acessos para enviar qualquer tipo de mensagem de dados entre os dispositivos através do barramento PCI. Para reduzir-se o número de acessos se faz necessário expandir o banco de memória, para isso deve-se aumentar a quantidade de memórias DPRAM. Para efetuar-se uma expansão para 32 bits a arquitetura necessariamente terá dois acessos e, um único acesso caso fosse implementado uma arquitetura de 64 bits.

4.3.4 Banco de Memórias

Para comunicação de dados entre os dispositivos, placa *PLX* 9052 e o processador "*ColdFire*", utilizou-se de um banco de memórias para viabilizar a troca de mensagens (dados e [2]sinais de comandos), com os atributos das tarefas, entre as aplicações em RTlinux e o processador. A FIGURA 8 mostra o dispositivo de memória utilizado para as transações de dados para as trocas de mensagens entre o Co-processador e o escalonador "*Dispatcher*". O banco de memória é um módulo em *Hardware* adicional ao sistema, que concatena, através do barramento "*ColdFire*" e PCI, as unidades de processamento da arquitetura proposta.

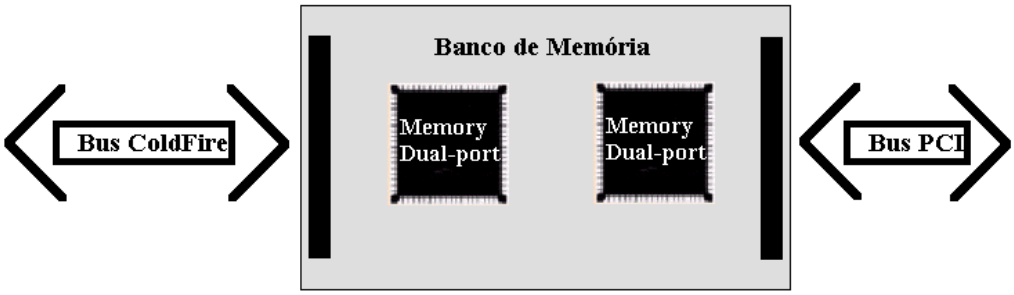

Figura 8 Banco de memória

As memórias empregadas no banco de memória são do tipo DPRAM de 8 bits, sendo que elas foram organizadas de forma a acessar os barramentos "*Bus ColdFire*" e o "Bus PCI" no modo (escrita/leitura) em 16 bits. Os barramentos "*Bus ColdFire*" e o "Bus PCI" são constituídos por cabos do tipo "Flat". Os cabos fazem a interconexão elétrica entre o banco de memória e a unidade de co-processamento através do "*Bus ColdFire*" além da ligação entre o "Bus PCI" e a plataforma IBM-PC respectivamente. Fez-se necessário o dispositivo de memória para garantir a segurança no fluxo de mensagens, coibindo, deste modo, uma

[2] Entende-se por sinais de comandos como sendo eventos de monitoração.

eventual perda de dados durante o processo de comunicação entre os dispositivos (dispatcher e o co-processador [3]TFCP).

[3] É uma sigla utilizada nesta obra para denominar o algoritmo TAFT implementado no Co-processador.

5 AMBIENTE DE PROGRAMAÇÃO E VALIDAÇÃO

A FIGURA 9 mostra, de maneira geral, a implementação da arquitetura proposta para a execução da política de escalonamento distribuída entre as unidades de processamento. Observa-se através da figura, que o banco de memórias coexiste à parte com a unidade de co-processamento. Nota-se que foram necessários dois sistemas operacionais distintos, o RTLinux para suportar as aplicações de tempo real juntamente com o *Software* de monitoração JEWEL vide ((GERGELEIT, 2001)), e o sistema operacional Windows que suporta o ambiente de desenvolvimento do processador "*ColdFire*", e o *Software* de análise [4]BR-TOOL (HUSEMANN, 2002).

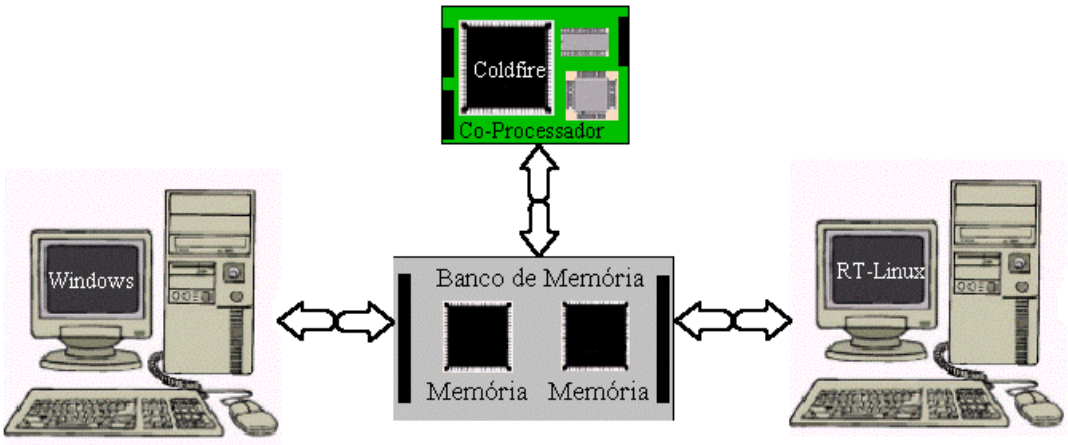

Figura 9 Sistema de Co-processamento

O *Software* de integração do processador "*ColdFire*" é executado sob plataforma Windows, assim como, o *Software* utilizado para a análise dos resultados BR-TOOL. Já, o sistema operacional RTLinux é usado para dar início ao processo de escalonamento

[4] O software BR-TOOL é usado para a análise de resultados.

cooperante, e é sob essa plataforma que o sistema de monitoração JEWEL é executado. No sistema de monitoração projetado os eventos gerados no módulo de co-processamento "*ColdFire*" são capturados pelo ambiente RTLinux. Estes eventos são transmitidos através do co-processador para o banco de memória, ficando deste modo disponível para a consulta do JEWEL. Já, o sistema monitorador é comandado a fazer uma leitura no banco de memória pelo "*Dispatcher*". O "*Dispatcher*" foi instrumentado de maneira que a cada execução, do algoritmo de escalonamento, pela plataforma RTlinux, seja executada a instrução de leitura do banco de memória. Deste modo, habilitando o sistema a adquirir qualquer valor de status de eventos decorrente da interação entre as unidades cooperativas de processamento para a monitoração das atividades executadas pelo co-processador.

O código do escalonador, neste caso o TAFT, foi implementado na plataforma Windows através do *Software* de desenvolvimento "[5]CodeWarrior". Cabe destacar que, neste ambiente de desenvolvimento foi possível utilizar-se da linguagem de programação C para o desenvolvimento do código do algoritmo escalonador, e além disso, monitorar o banco de memória interno e externo do processador. No ambiente "CodeWarrior" pela interface [6]BDM (Background Debug Mode) a conexão entre o módulo BDM e a plataforma windows é feita através da porta paralela. Já, o barramento "*ColdFire*" (vide FIGURA 8) é dividido em duas vias. Uma das vias faz a interconexão entre o Co-processador e o banco de memória, e a outra, da mesma forma, interliga ao módulo BDM. O barramento PCI proveniente da plataforma IBM PC através da placa *PLX*-9052 é estendido até o banco de memória pelo Bus PCI (vide FIGURA 8).

[5] O software CodeWarrior é a plataforma de desenvolvimento para os processadores Coldfire da Motorola.
[6] É uma interface de comunicação que pode ser utilizada para diversas famílias de processadores.

5.1 ESTRUTURA DO SOFTWARE DE GERENCIAMENTO

5.1.1 Driver PLX

Para a realização dos serviços do mecanismo de escalonamento no processador "*ColdFire*" foram criados procedimentos específicos a fim de transferir ao *Hardware* tarefas relacionadas à política de escalonamento. Esses serviços, em alguns casos, foram implementados apenas com alterações na API original. As instruções relacionadas com o "Driver" *PLX* são executadas, pelo sistema, como sendo tarefa não tempo real. Essa informação é de suma importância, uma vez que, este fato se configura em uma limitação de desempenho, pois pelos conceitos da arquitetura do RTLinux o "Driver", neste caso o *PLX*, sempre tem desvantagem na concorrência com outras tarefas na disputa pela CPU. A FIGURA 10 mostra a arquitetura PLX.

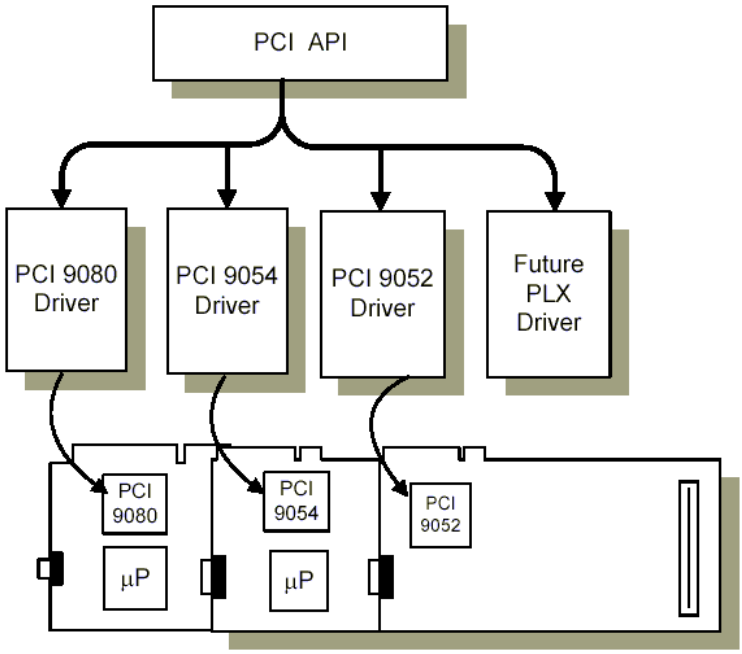

Figura 10 Uma visão hierárquica do sistemas *PLX*

Pode-se verificar, na FIGURA 10, que diversas placas PCI *PLX* (PCI 9080, 9054, 9052) podem coexistir no mesmo sistema computacional, bem como inúmeros “Drivers”.

A FIGURA 10 mostra a hierarquia do sistema genérico *PLX* de “Drivers” onde se tem a arquitetura particionada em três níveis: No primeiro nível, superior, encontra-se a API. Já, no intermediário o “*Driver*”, e por último no terceiro nível o *Hardware* da placa *PLX*. Um sistema similar a este, residente na plataforma RTLinux, foi utilizado para implementação da arquitetura de co-processamento contendo porém, um único “*Driver*” PLX 9052.

5.1.2 Algoritmo de Escalonamento do Co-processador

Devido às restrições de 8Kbytes de memória interna no processador (MCF5206e) a programação do algoritmo de escalonamento no “*ColdFire*” sofreu [7]alterações, com relação à sua estrutura original, para ocupar a mínima área de memória interna. As mudanças se limitaram na implementação das funcionalidades vitais para o funcionamento da política de escalonamento pois, ao contrário do TAFT sob plataforma Rtlinux-3.0, os serviços específicos à chamada do sistema operacional não se fizeram necessários no processador “*ColdFire*”.

O mecanismo de escalonamento TAFT permite a execução de tarefas caracterizadas por um tempo de execução estimado. Este mecanismo é capaz de tratar situações de sobrecarga, evitando a perda de *Deadlines*.

É da seleção entre MP e EP que se resume o aspecto tolerante a falhas da política de escalonamento TAFT. Na MP encontram-se as funcionalidades das aplicações, enquanto que a EP somente é acionada se a sua MP correspondente não puder ser finalizada antes do seu respectivo *Deadline*. Assim, o mecanismo de escalonamento ativa a EP de forma a garantir que pelo menos ela seja concluída antes do seu *Deadline*. Ao contrário da MP, que possui o

[7] As alterações efetuadas se restringiram apenas às funcionalidades inerentes à plataforma RTLinux.

código da aplicação, na EP tem-se o mínimo de funcionalidade implementada, assegurando-se apenas que a aplicação vá para um estado seguro.

5.1.3 Elemento de Software no PC

A arquitetura de software desenvolvida consiste de um novo módulo de escalonamento para o RT-Linux "*Dispatcher*", o qual opera em conjunto com diversos módulos adicionais. A FIGURA 11 apresenta uma visão geral destes módulos.

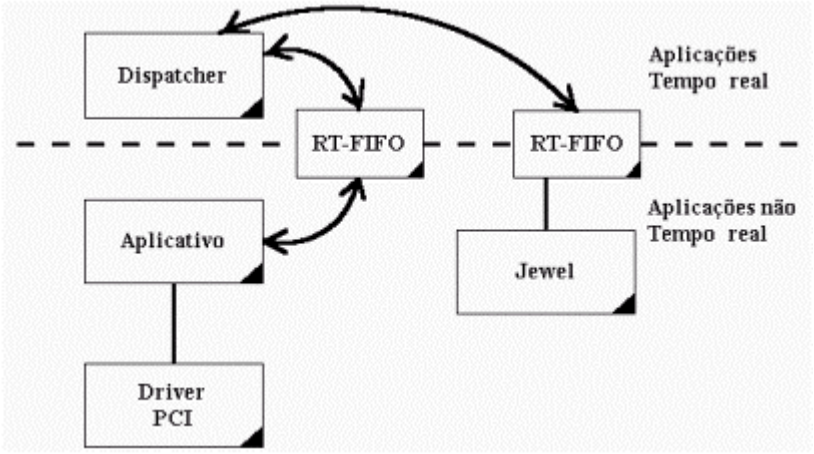

Figura 11 Arquitetura de Software desenvolvida

O módulo "*Driver*" PCI é responsável por garantir a comunicação de dados entre o PC e o *Hardware* de co-processamento. O mesmo foi desenvolvido com base em um pacote de desenvolvimento adquirido da empresa fabricante do "chipset" *PLX* 9052 utilizado no projeto. Este pacote foi adaptado para a arquitetura proposta, permitindo acesso de leitura e escrita no banco de memórias (DPRAM) da placa co-processadora. Isto é feito através de um aplicativo próprio desenvolvido para servir de interface entre o escalonador, que roda na placa externa, e o programa residente no PC. Devido ao fato deste pacote ter sido inicialmente projetado para trabalhar em plataformas não tempo real, este módulo segue a mesma restrição.

Já, o módulo *Dispatcher* atende às solicitações do escalonador (que executa no co-processador externo). As funções deste módulo são de receber dados gerados pelo escalonador quando da necessidade de ativar ou desativar alguma dada tarefa e garantir sua execução no processador principal. Em resumo, sua função é a de despachar os comandos gerados pelo escalonador e por isso foi chamado de "*Dispatcher*". Este módulo pode também enviar informações para o escalonador, como por exemplo, à indicação da finalização de uma tarefa. Como o aplicativo de leitura do "Driver" PCI é uma tarefa não tempo real e o "Dispatcher" é uma aplicação tempo real, a comunicação entre ambos ocorre através de uma RT-FIFO (BARABANOV, 1997).

Além disso, utilizou-se a ferramenta de monitoração descrita em (GERGELEIT, 2001) para validar o sistema desenvolvido. Através dos dados obtidos com a monitoração foi possível validar o comportamento temporal da arquitetura desenvolvida.

Uma segunda variação de arquitetura de *Software* foi projetada e analisada na tentativa de melhorar o desempenho do sistema gerencial inicialmente proposto. Cabe enfatizar que esta variação no módulo de escalonamento é uma alternativa que visa substituir as funcionalidades do "driver" não tempo real. Pelo fato do mesmo não suportar aplicação tempo real o desempenho do sistema pode sofrer um melhora na previsibilidade com essa proposta na arquitetura de software. A idéia principal deste novo modelo é criar-se uma *Thread* de sincronismo, onde esta teria através de RT-FIFO, a capacidade de influenciar diretamente no aplicativo e por conseqüência no "Driver" PCI como mostra a FIGURA 12.

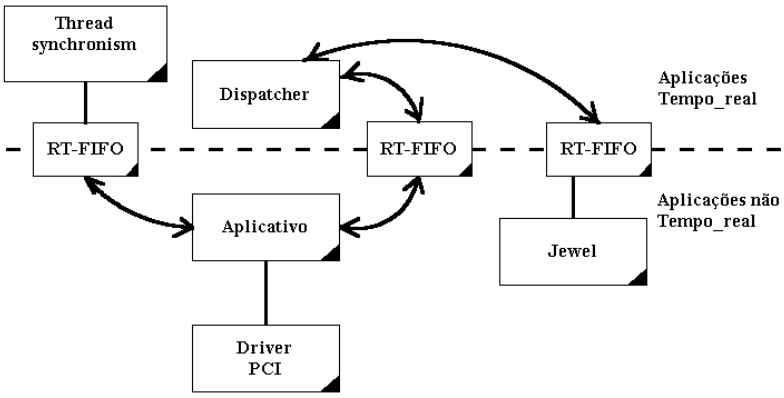

Figura 12 Arquitetura de Software desenvolvida com Sincronismo

Como pode ser observado no esquemático, a *Thread* de sincronismo pode coordenar os instantes de ativações do "driver" PCI através de uma RT-FIFO, com a qual está concatenada diretamente na aplicação não tempo real. Acredita-se que esse mecanismo pode acarretar na redução substancial das variações provenientes dos instantes de ativações no "Driver PCI" que são característico do sistema proposto, pois este novo método pode alterar e tornar a concorrência dos processos, entre o "Driver" *PLX* e as tarefas de tempo real do sistema, mais justa. Tornando-se deste modo uma alternativa em substituição ao "Driver" PLX não tempo real. No capítulo subseqüente relativo à análise dos resultados, é mostrada a análise estatística realizada com os dados referentes a esta alteração na arquitetura de *Software*.

5.2 VALIDAÇÃO DO SISTEMA

5.2.1 Apresentação do Estudo de Caso

Para título de estudo de caso utilizou-se a visualização das EP's provenientes de um conjunto de tarefas executadas pelo algoritmo de escalonamento residente no Co-processador. Foram utilizados os valores dos atributos conforme mostra a TABELA 4. Os atributos usados na constituição dos requisitos temporais pelas tarefas forçam à execução de EP's.

Tabela 4 Parâmetros dos TaskPairs com ocorrência de EP

TP	Período	Deadline	Tp. MP	Tp. EP
1	2 s	2 s	800 ms	100 ms
2	1 s	1 s	600 ms	50 ms

Com esses atributos, escolhidos após uma análise prévia, para as tarefas TP1 e TP2, o co-processador, o qual fora implementado com o mecanismo de escalonamento TAFT, não consegue atender os requisitos temporais para as duas tarefas, como pode ser observado na FIGURA 13. Onde são mostradas em destaque as ocorrências das EP's e suas respectivas MP's. O resultado para o respectivo teste, mostrado na FIGURA 13 mencionada, foi obtido com auxílio de um osciloscópio.

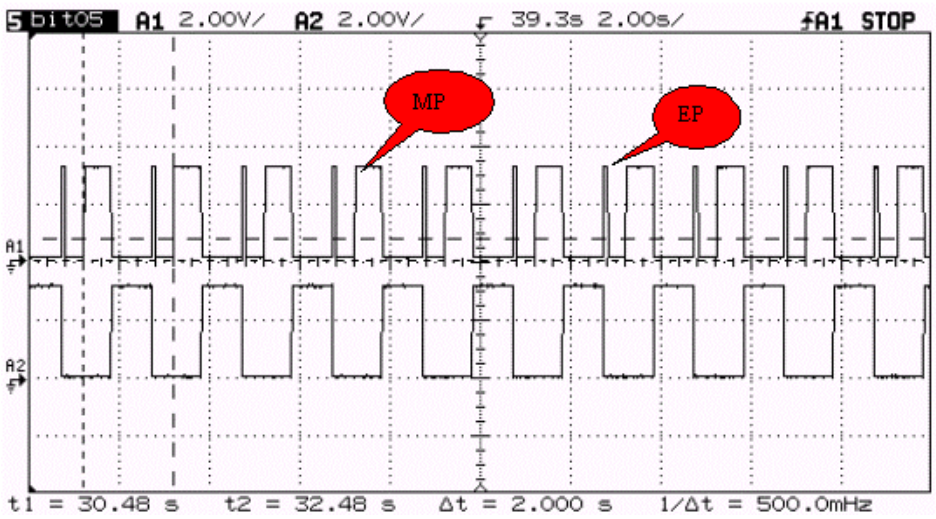

Figura 13 Forma de onda capturada do TFCP

Pode-se observar na imagem adquirida, por um osciloscópio, parte da evolução temporal das duas tarefas utilizadas para a realização dos ensaios. A legenda (A1), no lado esquerdo da figura, representa a tarefa 1 (TP1) da TABELA 4, enquanto a legenda (A2) representa a tarefa 2 (TP2) respectivamente. A FIGURA 13 mostra, em destaque, o aspecto tolerante a falha da estratégia de escalonamento TAFT. Onde as execuções das EP's relativas a tarefa (TP1) são realizadas pelo fato do co-processador não atender completamente os requisitos temporais especificados na TABELA 4. Nota-se, com o auxílio da FIGURA 13,

que o escalonador não consegue atender a MP da tarefa (TP1) especificada com 800 milisegundos e periodicidade de 2 segundos.

No segundo ensaio realizado no estudo de caso, foram analisadas estatisticamente as aquisições de 100 amostras. A TABELA 5 mostra a análise dos instantes de início das MP's no co-processador de escalonamento, determinando a média e o desvio padrão destes instantes. Analisando estes resultados, é possível concluir que o co-processador trabalha com uma boa precisão, dada a variação observada entre os instantes de ativação de cada MP. Para a efetivação do teste foram desabilitados os procedimentos de comunicação entre a unidade de co-processamento e a plataforma RTLinux.

Tabela 5 Análise estatística do início de cada MP no co-processador de escalonamento

TP	Período Médio	Desvio Padrão
1	2,001 s	602 μs
2	1,005 s	577 μs

5.3 ANÁLISE DOS RESULTADOS

Conforme, previamente explanado na seção planejamento experimental, são apresentados os resultados decorrentes dos ensaios efetuados na arquitetura proposta, com a finalidade de validar e justificar a implementação proposta. Os testes efetuados para avaliar experimentalmente a eficiência e o comportamento do ambiente proposto basearam-se no conjunto de tarefas proposto no *benchmark Hartstone* utilizado em (BECKER, 2003).

Em um segundo experimento, realizou-se um estudo com a finalidade de caracterizar os tempos de comunicação entre o escalonador (localizado na placa de co-processamento) e o mecanismo de despacho do RTOS. Mais precisamente determinou-se o "jitter" entre o instante de tempo em que o co-processador de escalonamento determina o início de uma tarefa (MP ou EP neste caso) e o instante de tempo em que o *dispatcher* atende tal solicitação.

No último experimento, comparou-se o desempenho do dispatcher em relação ao algoritmo de escalonamento. Neste ensaio, manteve-se o escalonador desenvolvido originalmente para o TAFT no ambiente RT-Linux. Além disso, testou-se também o uso de escalonadores mais simples, como o EDF.

Dentre os principais algoritmos de escalonamento reconhecidos na literatura cabe destacar as características com relação à sobrecarga computacional do EDF e do TAFT como pode ser analisado na TABELA 6, a qual exibe a sobrecarga característica na plataforma RTLinux.

Tabela 6 Análise da Sobrecarga

	EDF	TAFT c/Prio	TAFT s/Prio
Média	2307,07µs	2342,31µs	2311,18µs
Desvio Padrão	466,23ns	365,01ns	499,27ns

A título de comparação mostra-se no gráfico a relação média da carga computacional com a variação da quantidade de tarefas no sistema. Para essa avaliação foram utilizadas conforme o eixo das abscissas, duas e nove tarefas. A sobrecarga nesta figura é mostrada com base de tempo em µs no eixo das ordenadas.

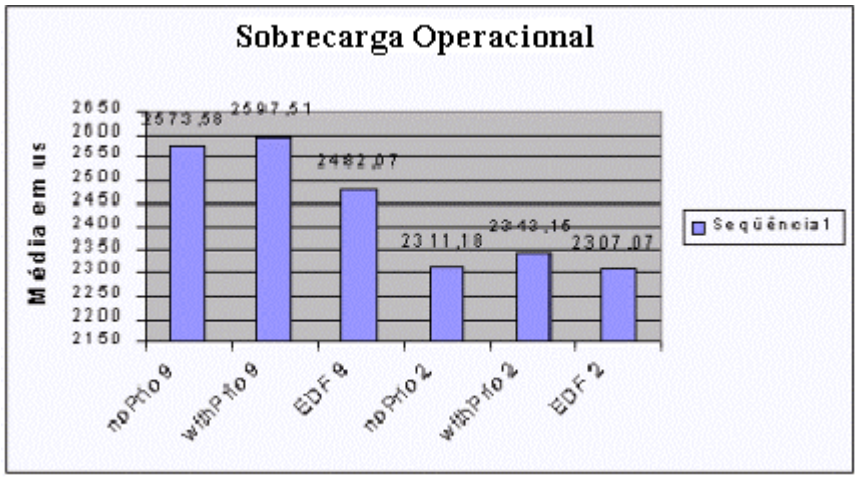

Figura 14 Carga Computacional

Pode-se averiguar com o auxílio da TABELA 6 e do gráfico (Análise de Sobrecarga no algoritmo TAFT[8]) na FIGURA 14, que a sobrecarga computacional no algoritmo TAFT com o mecanismo de prioridade implementado (c/Prio) é aproximadamente (2%) superior aos algoritmos EDF e TAFT sem o mecanismo de prioridade (s/Prio). E aumentando-se o número de tarefas, incrementa-se o "*Overhead*" dos algoritmos analisados em torno de (7%). De duas para nove tarefas, observa-se um aumento médio em torno de (3%) na diferença entre o algoritmo mais rápido que neste caso é o (EDF) e o mais lento (withPrio) - acredita-se, com base no gráfico, que tal diferença tende a aumentar com o acréscimo no número de tarefas. Cabe destacar que os mesmos procedimentos, com relação ao ambiente computacional, foram adotados para a estimação da sobrecarga..

Através dos ensaios realizados, observou-se que os tempos de comunicação variam entre poucos microssegundos até 5 milisegundos. A FIGURA 15 mostra a distribuição dos tempos medidos nesta faixa, tomando-se como base o envio do sinal que indica a ativação da MP do TP-1. A explicação para tal variação vem do fato que os eventos produzidos pelo co-processador são enviados para o processador principal e, neste último, bufferizados em uma RT-FIFO. Esta última, só é lida a cada ativação do *dispatcher*, cujo "tick" está calibrado em 5 ms. Uma forma de otimizar este tempo, é evitar a comunicação entre o "Driver" PCI *PLX* e o "*Dispatcher*" através de RT-FIFO. Entretanto, isto só seria possível caso o "Driver" fosse implementado também como um módulo tempo real, assim como o "*Dispatcher*".

[8] Obs: A legenda especificada no gráfico como withPrio diz respeito ao algoritmo TAFT.

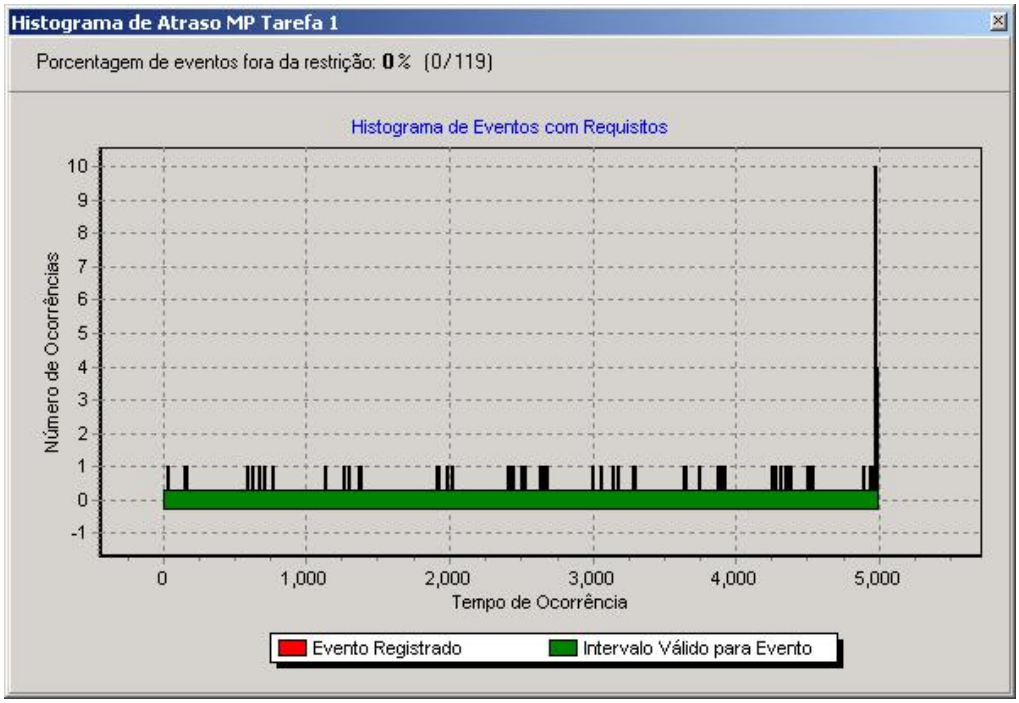

Figura 15 Atraso na marcação dos eventos entre dispatcher e aplicativo

As FIGURAS 16,17,18 e 19 exibem a distribuição dos eventos no tempo em ambas localizações, ou seja, “Driver” PCI *PLX* e *Dispatcher*. Observa-se que no *Dispatcher* os eventos são mais concentrados. Isto pode ser explicado pelo fato do *Dispatcher* executar periodicamente a cada 5 ms, sendo que os eventos são lidos e processados apenas nestes instantes. Já, o “Driver” PCI *PLX* encontra-se em execução sempre que não houver outro aplicativo tempo real concorrendo pelo uso do sistema, o que ocorre com bastante freqüência neste experimento, devido à baixa carga do sistema.

Iniciando-se uma análise mais detalhada das FIGURAS 16 e 17, pode-se observar que as distribuições dos eventos ao longo do tempo não são uniformes. Conforme comparação entre os mesmos eventos mensurados em diferentes dispositivos, neste caso, o “Driver” e o “*Dispatcher*”, como mostram as FIGURAS 16 e 17. Pode-se averiguar que o histograma da distribuição dos eventos relativos ao “Driver” não possui uma uniformidade, contrário senso à distribuição revelada na FIGURA 17.

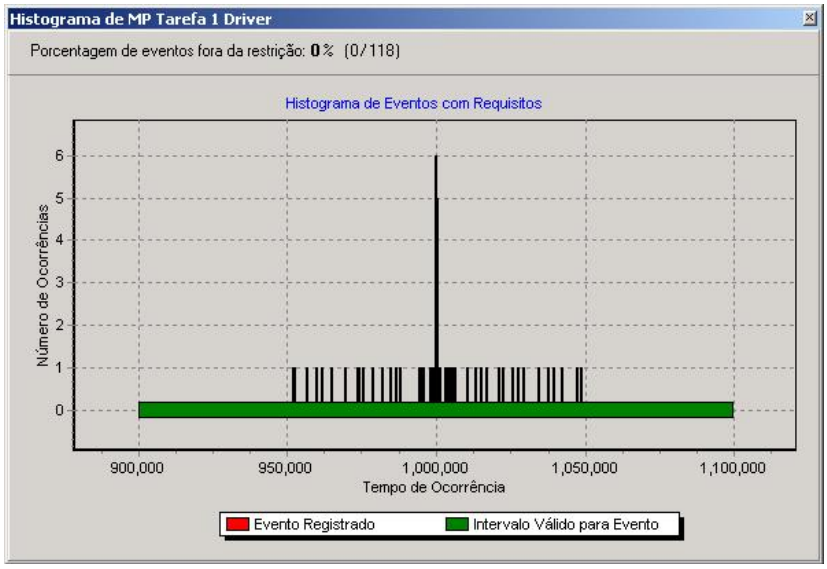

Figura 16 Evento de início da MP do TP-1 no driver PCI

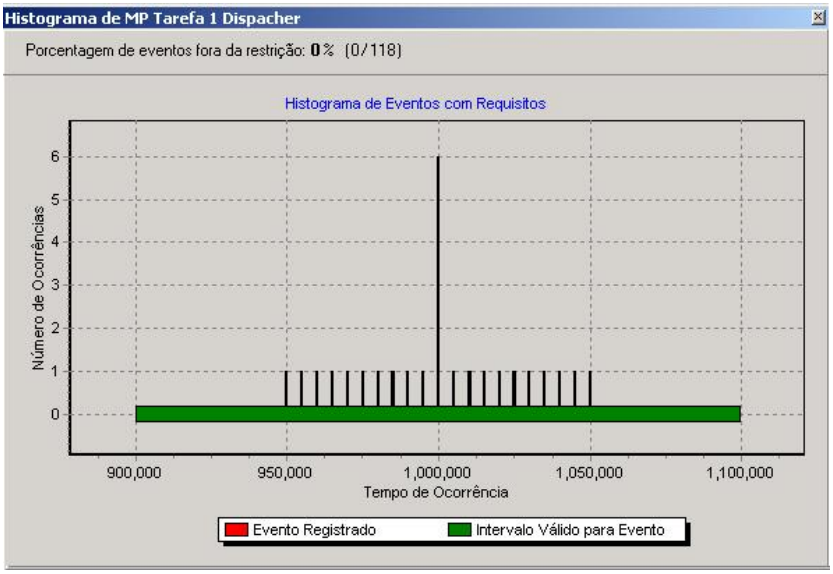

Figura 17 Evento de início da MP do TP-1 no dispatcher

Apesar dos atrasos mostrados nos períodos da TP-1, em decorrência do “jitter” de comunicação, a uniformidade da distribuição é um aspecto a ser destacado na FIGURA 18.

Explica-se esse fenômeno pelo fato do módulo “Dispatcher” estar implementado como uma “*Thread*” juntamente com o algoritmo de escalonamento da plataforma RTLinux. Assim, o “*Dispatcher*” tem prioridade na concorrência pelos serviços do processador.

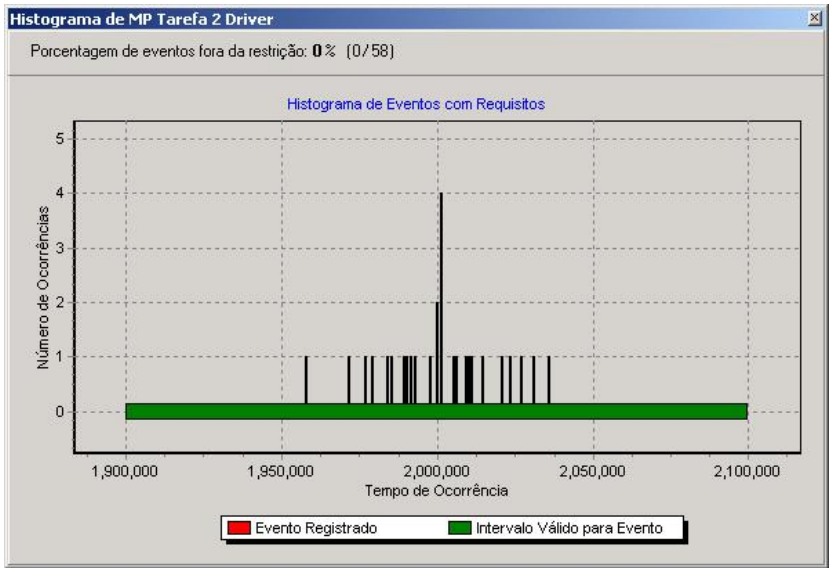

Figura 18 Evento de início da MP do TP-2 no driver PCI

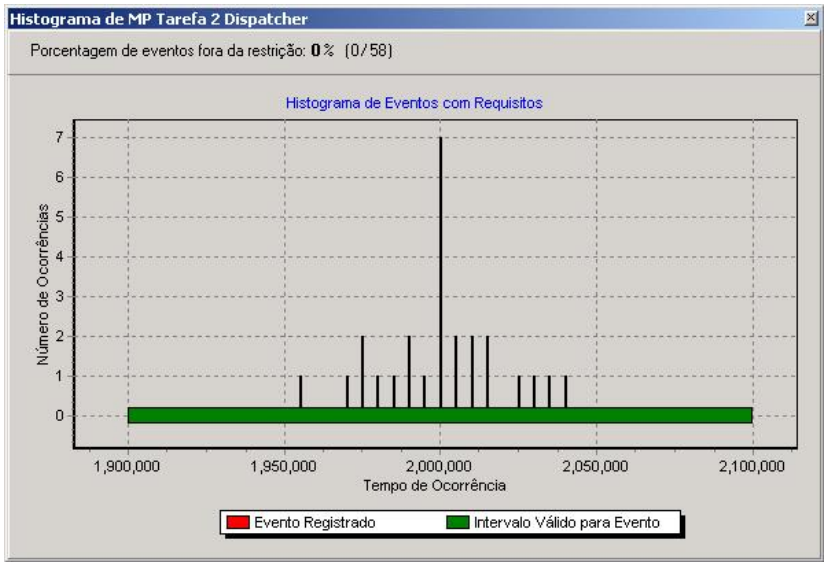

Figura 19 Evento de início da MP do TP-2 no dispatcher

Observa-se que para a TP-2, FIGURAS 18 e 19, a ocorrência do mesmo fenômeno detectado para a tarefa TP-1, através das figuras, observa-se um jitter de até 50ms em torno dos instantes de ativação das tarefas.

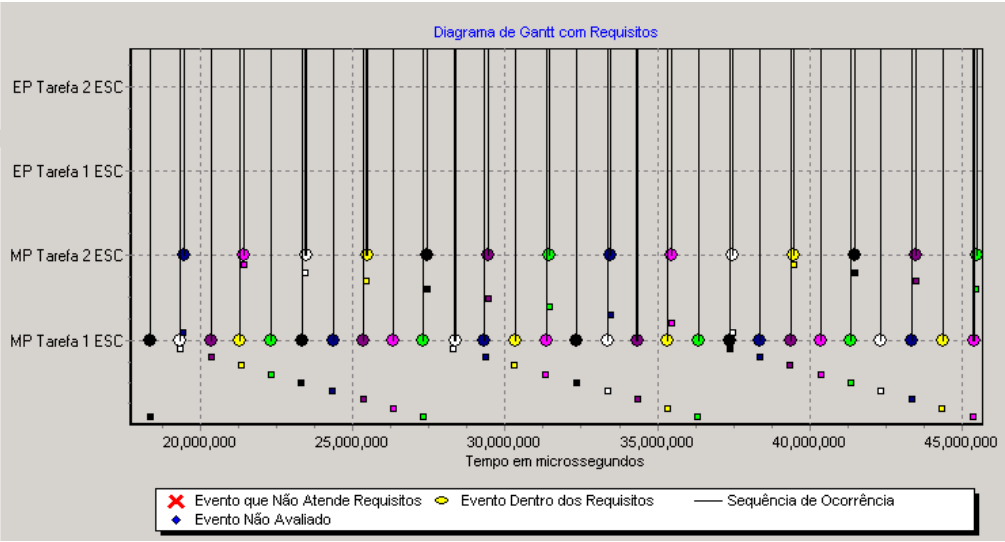

Figura 20 Análise seqüencial dos eventos

O diagrama de Gantt na FIGURA 20, mostra uma seqüência de eventos numa certa variação temporal. Observa-se que não há ocorrências de EP's , tendo-se apenas eventos relativos as MP's executadas pela unidade de co-processamento.

Uma análise mais apurada destes dados é apresentada na TABELA 7, interpretando tais resultados nota-se que esta variação ocorre devido aos processos de comunicação e aquisição no processador principal (PC), isto porque os dados apresentados na TABELA 5 indicam que as tarefas são escalonadas com uma boa precisão na placa co-processadora. Observando os eventos no "Driver" PCI (FIGURA 18), verifica-se que os mesmos já são detectados com uma defasagem de até 50 ms. Portanto, o "Driver" PCI é o causador principal da defasagem verificada. Isto se deve ao fato do "Driver" PCI não ser um aplicativo tempo real, possuindo prioridade baixa em relação às demais tarefas do sistema. Para finalizar, ressalta-se que a maior variação temporal ocorre no "*Dispatcher*" devido aos atrasos de comunicação do mesmo com o "Driver" PCI, conforme exibido na TABELA 7.

Tabela 7 Análise estatística dos eventos no processador principal (PC)

	TP-1		TP-2	
	Período Médio	**Desvio Padrão**	**Período Médio**	**Desvio Padrão**
Driver-PLX	1.000.726.353ns	17.419.560,26 µs	2.001.452.766ns	13.176.426,61 µs
Dispatcher	1.000.742.482ns	18.681.784,39 µs	2.001.501.075ns	14.812.224,74 µs

No último ensaio realizado, comparou-se o desempenho do dispatcher em relação ao algoritmo de escalonamento. No ensaio, percebeu-se que no *dispatcher* o desvio padrão é 11% menor do que o desvio padrão do escalonador TAFT na plataforma RTLinux. Em uma primeira análise, este fator parece ser baixo. Entretanto, leva-se em consideração o fato de que neste ensaio executam apenas 2 tarefas. Assim, a sobrecarga introduzida por algoritmos de escalonamento mais complexos (como o TAFT) torna-se baixa devido ao pequeno número de tarefas no sistema. Pode-se confirmar, baseado no gráfico, (vide (FIGURA 14)) que esta diferença aumenta consideravelmente com o acréscimo no número de tarefas.

Dentre as análises estatísticas realizadas sobre os resultados experimentais para validar a arquitetura, cabe mostrar a distribuição da quantidade dos eventos gerados no co-processador e na plataforma RTLinux nos ensaios como mostra a FIGURA 21.

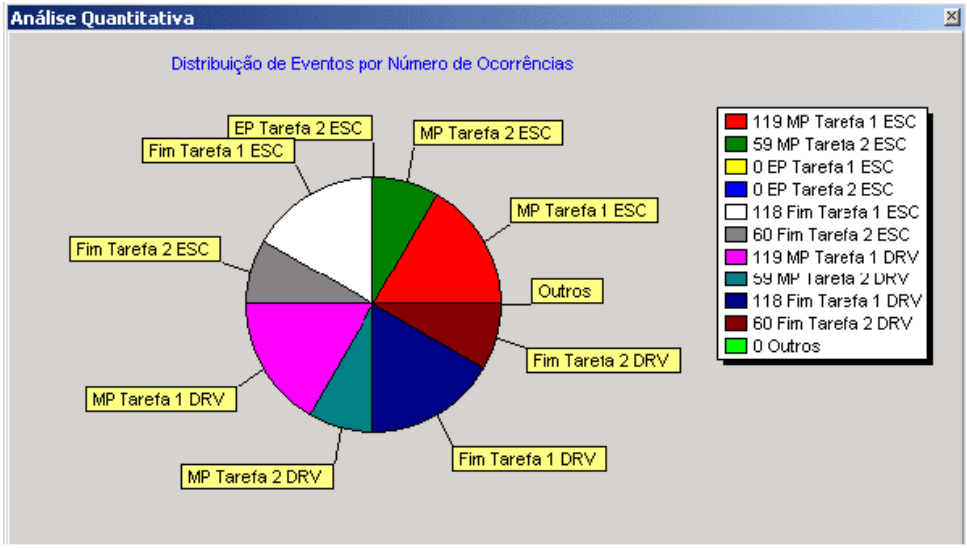

Figura 21 Análise quantitativa da Distribuição de eventos por ocorrência

Essa análise mostra de forma geral todos os eventos envolvidos no ensaio para a caracterização do determinismo e previsibilidade temporal da arquitetura proposta. Destaca-se na FIGURA 21, a totalização das ocorrências dos eventos, e, pode-se através desta, comprovar que nenhuma EP ocorreu no decorrer deste teste, indicando deste modo, que todos os Deadlines foram estritamente cumpridos.

Nesta parte é realizada uma análise estatística sobre a arquitetura com a utilização da "*Thread*" de sincronismo. Esta verificação do sistema implementado serve para verificar os resultados obtidos com essa nova alternativa.

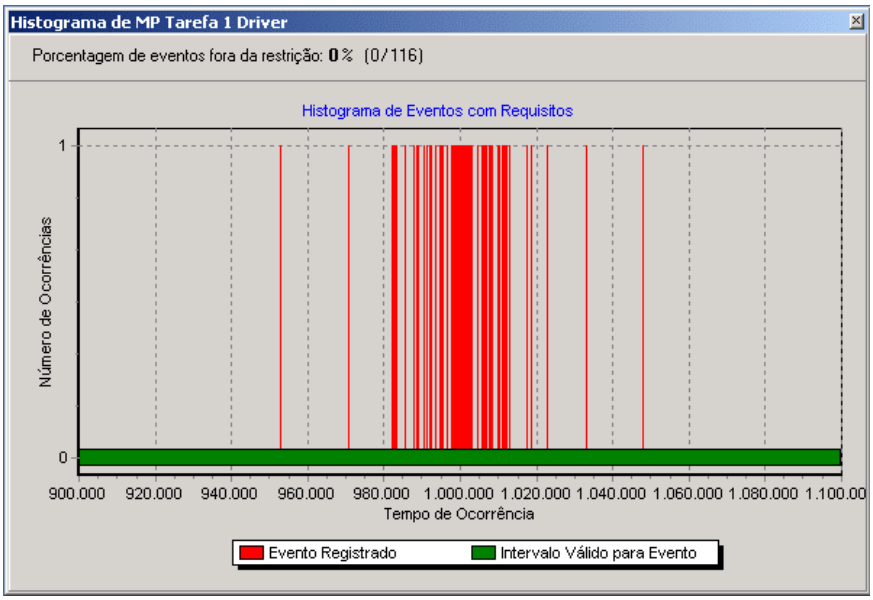

Figura 22 Evento de início da MP do TP-1 no driver PCI com SINC

Conforme os eventos mostrados na FIGURA 22, que corresponde ao histograma dos eventos referentes a MP da tarefa TP-1 executada no co-processador, tem-se para estes eventos a média e o desvio padrão com o sistema de sincronismo implementados na tabela abaixo;

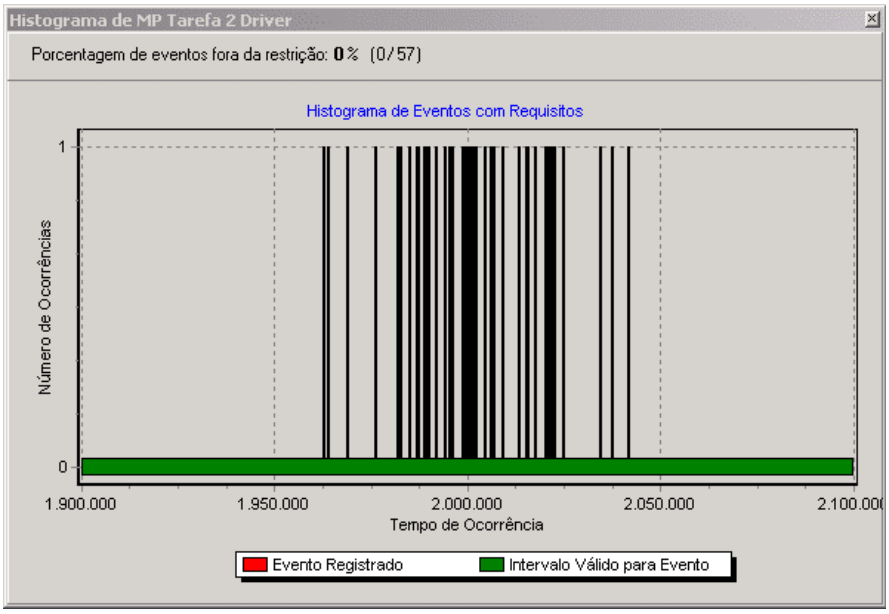

Figura 23 Evento de início da MP do TP-2 no driver PCI com SINC

Os eventos mostrados na FIGURA 23 são correspondentes a MP da tarefa 2 com o sistema de sincronismo. Os valores obtidos para a média e desvio padrão estão na TABELA 8 que reflete aos dados referentes à análise estatística para a TP-2:

Tabela 8 Análise estatística dos eventos com *Thread* de sincronismo

	TP-1		TP-2	
	Período Médio	**Desvio Padrão**	**Período Médio**	**Desvio Padrão**
RT-PLX	1.000.719.619ns	10.661.387,89 µs	2.001.415.523ns	16.360.890,54 µs
Driver-PLX	1.000.726.353ns	17.419.560,26 µs	2.001.452.766ns	13.176.426,61 µs

Visualmente pode-se verificar conforme as FIGURAS 21 e 22, que as distribuições ficaram mais compactas em torno dos valores médios dos períodos de TP-1 e TP-2, porém a TABELA 7 contradiz essa verificação mostrando que o resultado para a análise estatística realizada não detecta nenhum ganho com relação ao desvio padrão. Assim, infere-se que a arquitetura com mecanismo de sincronismo, não introduziu melhoras significativas no sistema. Apenas para a tarefa TP-1, ocorreu uma redução de 17ms para 10ms aproximadamente.

5.4 CONSIDERAÇÕES FINAIS

Cabe destacar como consideração final neste projeto de pesquisa que, os resultados obtidos foram condizentes com o esperado pelo autor, no que se refere à comunicação de dados entre os dispositivos envolvidos, tanto na plataforma RTLinux, quanto na base co-processadora. Apesar do alto "jitter" mostrado pelas análises de comunicação de dados na arquitetura. Credita-se a isso o fato de que uma aplicação, neste caso o driver *PLX*, considerada crucial para a comunicação entre as unidades cooperantes não suportar plataforma tempo real. Tornando-se, deste modo, o principal "gargalo" com relação à comunicação de dados entre as unidades de processamento do sistema proposto. Reside neste fato o aspecto mais importante que deve ser melhorado na arquitetura proposta, para que a mesma possa executar e, de forma satisfatória garantir os requisitos temporais. Assim, presume-se reduzir o "jitter" possibilitando ao sistema uma performance aceitável quanto à previsibilidade e ao determinismo temporal com a arquitetura mostra na FIGURA 24.

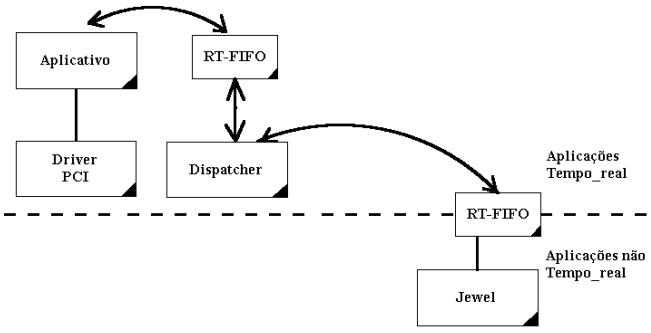

Figura 24 Concepção ideal de Arquitetura

Neste esquema proposto tem-se o "driver" e a aplicação, respectivamente, como uma tarefa em modo tempo real.

Atualmente, o sistema projetado está trabalhando de forma parcial, onde tanto a arquitetura de co-processamento, quanto a plataforma RTLinux estão escalonando as tarefas do sistema. Não havendo ainda uma forma de distribuição de “trabalho” entre os dispositivos cooperantes, de maneira a otimizar o processamento tempo real da arquitetura proposta. Logo, os dois sistemas, unidade de processamento principal e o co-processador, estão realizando processamento paralelo.

Na seção trabalhos futuros, o autor sugere algumas implementações que devem ser colocadas em prática para tornar a arquitetura funcional conforme as especificações iniciais do projeto.

6 CONCLUSÃO E TRABALHOS FUTUROS

Ao longo do texto desta dissertação foram analisados e apresentados os elementos que fazem parte da arquitetura proposta neste trabalho. Tratando o panorama geral do estado da arte relacionado a uma arquitetura de *Hardware* para co-processamento em sistemas operacional de tempo real. O tema apresentado percorre todas as abrangências com relação à área estudada, incluindo as etapas de projeto, execução e validação de uma aplicação de tempo real.

Aplicações em sistemas de tempo real exigem, pela sua complexidade e característica, a utilização de técnicas de desenvolvimento baseadas em *Software* e *Hardware.* Entre os benefícios do uso destas técnicas para a implementação da arquitetura proposta, cabe destacar, o particionamento onde algumas funções específicas do algoritmo de escalonamento são implementadas e executadas em *Hardware* dedicado que tem a faculdade de cooperar com a unidade de processamento principal.

A principal desvantagem com relação à arquitetura de proposta refere-se às exigências de comunicação entre as unidades de processamento. Todas as obras em que os autores baseavam-se em arquiteturas de multiprocessamento sempre destacavam um cuidado especial com relação a este quesito, pois a interface de comunicação pode tornar-se o maior empecilho em se tratando de ambiente de tempo real, para o cumprimento dos requisitos temporais. O determinismo temporal, neste tipo de arquitetura, não pode ser totalmente previsível em decorrência do processo de comunicação envolvido entre as partes cooperantes. Em relação aos RTOS, diversas funcionalidades temporais foram avaliadas pelo sistema proposto nesta obra, a fim de cumprir com os requisitos mínimos necessários específicos dos sistemas embarcados.

Para a elaboração da arquitetura de processamento considera-se, nesta pesquisa, a capacidade de processamento como ponto de partida para melhorar a performance da plataforma IBM-PC, mostrando-se a correlação existente no desenvolvimento de projetos tanto relacionado a *Software,* dedicado mais especificamente ao algoritmo de escalonamento, quanto ao *Hardware* através do particionamento. A implementação e a troca de funcionalidades entre as arquiteturas de *Software* e *Hardware* pode reduzir substancialmente imperfeições temporais, como por exemplo, “jitter”.

Dentre as vantagens alcançadas, pela arquitetura de processamento, cabe realçar: o possível aumento no poder de processamento da CPU, além da obtenção de previsibilidade e do determinismo temporal. Por último, e não menos importante, destaca-se a capacidade adicionada para se utilizar algoritmos de escalonamentos mais refinados sem causar sobrecarga de processamento, fator indesejável para o desempenho dos sistemas de tempo real. As funcionalidades do mecanismo de escalonamento do sistema operacional de tempo real RTLinux-3.0, no qual fora adaptado para suportar o algoritmo de escalonamento TAFT, foram avaliadas possibilitando após criteriosa análise, a implementação de algumas das principais funções em um processador de 32 bits. Assim, essa nova plataforma de co-processamento tem como principal atividade à cooperação na computação do algoritmo de escalonamento.

Neste estudo, é apresentada uma arquitetura para o co-processamento de tarefas, a qual pode ser adaptada em qualquer RTOS onde se possa alterar o comportamento do mecanismo de despacho. A principal vantagem desta arquitetura encontra-se em permitir uma diminuição na sobrecarga causada pelo algoritmo de escalonamento, na forma de “jitter”, deixando assim mais tempo de CPU disponível para o processamento das tarefas do sistema.

Com base nos resultados obtidos, com os experimentos realizados, pode-se concluir que o modelo de arquitetura proposto pode ser utilizado como um módulo de processamento

adicional para o escalonamento de tarefas de tempo real. Entretanto, algumas alterações devem ser aplicadas na arquitetura de *Software*, no que se refere ao Driver PCI. Com essa mudança pode-se inferir que as respostas temporais com relação aos instantes de ativações tornar-se-ão aceitáveis quanto à comunicação entre as unidades de processamento.

Os resultados preliminares discutidos neste trabalho são motivadores, uma vez que ,comprovam a diminuição de sobrecarga esperada. Além disso, acredita-se que o ganho percebido (em torno de 11% no desvio padrão) venha a ter maiores proporções com o acréscimo no número de tarefas do sistema.

Os resultados também demonstram que, na atual implementação, existe uma grande defasagem de tempo em relação ao processo de comunicação entre o escalonador localizado no co-processador e o processador principal (PC). Isto se deve principalmente ao fato do "Driver" *PLX* PCI utilizado no sistema não ser um aplicativo tempo real, possuindo prioridade baixa em relação às demais tarefas do sistema.

Como trabalho futuro, destaca-se a migração do atual "Driver" PCI *PLX* 9052 para um módulo de tempo real. Com isto, acredita-se obter uma melhora significativa com relação ao "jitter" observado na arquitetura. Além disso, para que a arquitetura proposta esteja totalmente operacional, com a descrição da proposta deste trabalho, algumas partes funcionais devem ser implementadas. Pode-se incluir nesta situação partes operacionais como, uma estrutura de comunicação entre a plataforma RTLinux e a unidade co-processadora, para que haja uma transmissão eficiente no envio de dados e atributos entre as unidades cooperantes de processamento.

Como continuação do trabalho desenvolvido, ainda pode-se reduzir a dependência da plataforma co-processadora do sistema operacional RTLinux, onde a unidade auxiliar de processamento seria a responsável por todas as funcionalidades como, por exemplo, teste de escalonabilidade, portanto, não havendo nenhuma relação entre as plataformas de

processamento auxiliar e principal. Assim, o co-processador teria total autonomia operacional, podendo tornar-se parte integrante de um sistema embarcado. Seguindo este raciocínio, após esta modificação concluída, adicionalmente, pode-se implementar e migrar a estrutura proposta e analisada nesta pesquisa para um dispositivo de controle distribuído, onde a arquitetura de co-processamento poderá ser uma unidade cooperante na execução de tarefas como, por exemplo, fusão de dados. Espera-se com essa prática explorar, melhorias de desempenho do algoritmo TAFT na unidade de processamento.

Alterando-se a tônica dos trabalhos futuros pode-se utilizar a base de *Hardware* de co-processamento para estimar e analisar o consumo de energia e potência com relação aos diversos algoritmos de escalonamento, primeiramente ao mecanismo TAFT e posteriormente, aos demais mecanismos de escalonamento com prioridade fixas e dinâmicas reconhecidos na literatura de sistemas de tempo real, como, por exemplo, EDF, RM, LRT e RR.

REFERÊNCIAS

ANDREWS, D.; NIEHAUS, D.; ASHENDEN, P. Programming Models for Hybrid CPU/FPGA Chips. **IEEE Computer**, USA, v. 37, n. 1, p. 118-120, jan. 2004.

AUDSLEY, N. C. et al. Hard Real-Time Scheduling: the deadline monotonic approach. In: HALANG, W. A. ; RAMAMRITHAM, K. Real-Time Programming, 1992, England, **Proceedings…** England: Pergamon Press, 1992. p. 127-132.

BARABANOV, M. **A Linux-based Real-Time Operating Systems**, 1997. New Mexico. M.Sc. (Thesis), New Mexico Institute of Technology, 1997.

BECK, M.; BOHME, H.; DZIADZKA, M. **Linux Kernel Internals.** 2. ed. New York: Addison-Wesley, 1998. 480p,. ISBN: 0-201-33143-8.

BECKER, L.; GERGELEIT. M. Execution Environment for Dynamically Scheduling Real-Time Tasks. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM (RTSS), WIP-Session, 22., 2001, London. **Proceedings…** London: IEEE, dec. 2001, p. 3-6.

BECKER, L. **Um Método para Abordar todo o Ciclo de Desenvolvimento de Aplicações tempo real**, 2003. Tese (Doutorado) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

BUTTAZZO, G.; SPURI, M. Scheduling Aperiodic Tasks in Dynamic Priority Systems.**The Journal of Real-Time Systems**, USA, v.10, n.2, p. 179–210, 1996.

C, LIU.; J, LAYLAND. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. **Journal of the Association for Computing Machinery**, New york, v.20, n.1, p. 44-61, Jan.1973.

CARSTEN, D. Dreams - Concepts of a Distributed Real-Time Management System. In: WORKSHOP ON REALTIME PROGRAMMING,20., 1995, Fort Lauderdale, Florida. **Proceedings...** IFIP/IFAC, Amsterdam: Elsiver, 1995.

CHENYANG, LU; STANKOVIC, J. The Design and Evaluation of a Feedback Control EDF Scheduling Algorithm. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM, 20., 1999, Phoenix, AZ, USA. **Proceedings…** New york:IEEE, dec. 1999. p. 56.

CHETTO, H.; CHETTO, M. Some Results of the Earliest Deadline Scheduling Algorithm. **IEEE Transactions on Software Engineering**, USA, v. 15, n.10, p. 466-473, oct. 1989.

COOLING, J.; TWEEDALE, P. Task Scheduler Co-Processor for Hard Real-Time Systems. **Microprocessors and Microsystems**, Guildford, England, v. 20, n. 9,. p 553-566, may. 1997. ISSN: 0141 9331

CRESPO, A.; SÁEZ, S.; VILA, J. Firm Aperiodic Task Scheduling in Hard Real-Time Multiprocessor Systems. In: WORKSHOP ON REAL-TIME PROGRAMMING, 27., 2003, Lagow. **Proceedings...** Lagow, Poland: IFAC/IFIP/IEEE, may. 2003, p. 51-56.

ENGBLOM, J. On Hardware and Hardware Models for Embedded Real-Time Systems. In: IEEE EMBEDDED REAL-TIME SYSTEMS WORKSHOP HELD IN CONJUNCTION WITH THE IEEE, 22., REAL-TIME SYSTEMS SYMPOSIUM (RTSS), 2001, London. **Proceedings...** London, UK: IEEE/RTSS, dec. 2001.

ENGBLOM, J. et al. Worst-Case Execution-Time Analysis for Embedded Real-Time Systems. **International Journal on Software Tools for Technology**, Heidelberg: Springer-Verlag GmbH, v. 4, n. 4, p. 437-455, aug. 2001. ISSN: 1433-2779.

GALLMEISTER, B. **POSIX.4:** programming for the real world. Sebastopol, California: O'Reilly and Associates, 1995. 548p. ISBN, 1565920740.

GERGELEIT, M.; NETT, E. Scheduling Transient Overload with theTAFT Scheduler. In: FACHGRUPPE BETRIEBSSYSTEME – HERBSTTREFFEN. 2002, Berlin. **Proceedings...** Berlin: GI / ITG, nov. 2002.

GERGELEIT, M. **A Monitoring-based Approach to Object-Oriented Real-Time Computing**, 2001. Dissertation – Institut fur Verteilte Systeme, Otto-Von-Cuericke-Universitat Magdeburg, Magdeburg, 2001.

GOTZ, M. **Proposta de Arquitetura de Hardware e Software para Sistemas Tempo Real Distribuídos**, 2001. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

HALANG, A.; COLNARIC, M. Architectural Support for Predictability in Hard Real Time Systems. **Control Engineering Practice**, In: Real Time Programming. Oxford: Pergamon, v. 1, n.1, p. 281-286, 1992.

HUSEMANN, R.; PEREIRA, C. E.; SCHMIDT, R. L. Sistema Monitorador para Aplicações Baseadas em Comunicação por Barramentos Industriais. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 14., 2002, Natal, RN: **Anais …** Natal, RN: UFRN, p. 2780-2785, 2002.

HÜSEMANNN, R. **Sistema de Validação Temporal para Redes de Barramento de Campo**,2003. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

IMMICH, P.; BHAGAVATULA, R.; PENDSE, R. Performance Analysis of Five Interprocess Communication Mechanisms across UNIX Operating Systems. **Elsevier Science**, New York, v. 68, p. 27-43, oct. 2003.

CAI, L.; GAJSKI, D. **Codesign Visualizer.** Irvine, California : Department of ICS Universityof California, 92697-3425. Final Report 1998-99.

MILLIGAN, M.; CRAGON, H. The Use of Cache Memory in Real-Time Systems. **Control Engineering Practice**, Great Britain: Elsevier Science Ltd. v. 4, n. 10, p. 1435-1442, oct. 1996.

MANIMARAN, G.; MURTHY, C. A New Scheduling Approach Supporting Differentfault-Tolerant Techniques for Real-Time Multiprocessor Systems. **Microprocessors and Microsystems**, Guildford, England, v.21, n.3, p. 163-173, dec. 1997.

MILENKOVIC, M. **Operating Systems:** concepts and Design**.** New York: McGraw-Hill, 1992. 755 p. ISBN:1-56592-074-0.

MINK, A. et al. Performance Measurement using Low Perturbation and High Precision Hardware Assist. In: REAL-TIME SYSTEM SYMPOSIUM, 19., 1998, Madri. **Proceeding...** Madri: IEEE, Madri, dec. 1998. p. 379-388.

NITSCH, C. et al. Embedded System Architecture Design Based on Real-Time Emulation Rapid System Prototyping. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING (RSP),11., 2000. Paris, France. **Proceeding...** New York, USA: IEEE, june 2000. p. 228-233.

OLIVE, V.; MARTIN, S.; VARSEILLE, A. Os for Embedded Systems:state of the art and prospects. In: MIGAS FOURTH SESSION ON MICROELECTRONICS FOR TELECOMMUNICATIONS and MANAGING HIGH COMPLEXITY AND MOBILITY, 2000, Austrans, France. **Proceedings...** Autrans, France: MIGAS, 2000. p. 113-121

OLIVEIRA, R. Aspectos Construtivos dos Sistemas Operacionais de Tempo Real. In: WORKSHOP DE TEMPO REAL, (WTR), 5., Natal, RN. **Anais...** Natal: WTR, 22 de maio de 2003.

PILLAI, P.; SHIN, K. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES,18., 2001, Banff, Canada. **Proceeding...** Banff, Canada: ACM, oct. 2001. p. 89-102.

PONTREMOLI, M. **Arquitetura de Hardware de Baixo Custo para Sistemas Tempo real Distribuídos.** 1998. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Mecânica e Materiais, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1998.

RUBINI, A. **Linux Device Drivers**. 2. ed. São Paulo: Market Books, 1999. 317 p.

SANTOS, J.; OROZCO, J. Rate Monotonic Scheduling in Hard Real-Time Systems. **Information Processing Letters**, Amsterdam, v.48 n.1, p. 39-45, oct. 1993.

STANKOVIC, J. et al. The Spring System: integrated support for complex real-time systems. **Journal Real-Time Systems**, USA, v.16, n. 2/3, p. 97-125, may. 1999a.

STANKOVIC, J. et al. The Spring Scheduling Co-Processor. **IEEE Transactions on (VLSI)**, USA, v. 7, n. 1, p. 38-47, mar. 1999b.

STANKOVIC, J. Misconceptions about Realtime Computing: a serious problem for next generation systems. **IEEE Computer**, USA, v.21, n.10, p. 10-19, oct. 1988.

STANKOVIC, J. Real-Time and Embedded Systems. **ACM Computing Surveys**, New York, v. 28, n. 1, p. 205-208, mar. 1996a.

STANKOVIC, J. Strategic Directions: Real-Time and Embedded Systems. **ACM Computing Surveys**, New York, v. 28, n. 4, p. 751-763, dec. 1996b.

STÄRNER, J. Increasing Predictability of Real-Time Operating Systems. In: SWEDISH WORKSHOP ON COMPUTER SYSTEM ARCHITECTURE,7., 1998, Gothenburg, Sweden. **Proceedings...** Gothenburg, Sweden: IEEE, june. 1998.

SWAMINATHAN, V.; CHAKRABARTY, K. Real-Time Task Scheduling for Energy-Aware Embedded Systems. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM, 21., 2000, Florida, USA. **Proceedings...** Florida, USA: IEEE, nov. 2000.

TANENBAUM, A; WOODHULL, A. **Sistemas Operacionais:** projeto e implementação. 2.ed. Porto alegre: Bookman, 2000. 759p.

VOIGT, P.; MADSEN, J. Integrating Communication Protocol Selection with Hardware/Software Codesign. In: IEEE TRANSACTIONS ON COMPUTERAIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 1999, **Proceedings...** [S.i]: IEEE transactions on, aug 1999, v.18, n.8, p. 1077-1095.

WEIß, K. et al. Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System. In: IEEE SYMPOSIOM OF FIELD PROGRAMMABLE GATE ARRAY, 1999, Monterey, Califórnia. **Proceedings...** Monterey: IEEE, 1999, p. 235-242.

WEIß, K.; STECKSTOR, T.; ROSENSTIEL, W. Performance Analysis of a RTOS by Emulation of an Embedded System. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, 1999, Florida, USA. **Proceedings...** Clearwater, Florida, USA: IEEE, june. 1999, p. 146.

WEIß, K. et al. Analysis of the XC6000 Architecture for Embedded System Design. In: IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, 1998, California. **Proceedings...** Califórnia, USA: FCCM, 1998, p. 245.

WEIß, K.; STECKSTOR, T.; ROSENSTIEL, W. Emulation of a Fast Reactive Embedded System Using a Real Time Operating System. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 1999, München, März. **Proceedings...** München, März, jan. 1999, p. 764-765.

WELSH, M.; KAUFMAN, L. **Dominando o linux.** 2. ed.. Rio de Janeiro: Ciência Moderna, 1997. 603p. ISBN: 85-7993-018-7.

ZELENOVSKY, R.; MENDONÇA, A. **PC**: um guia prático de hardware e interfaceamento. 3. ed. Rio de Janeiro: MZ, 2002. 760 p.

APÊNDICE: CÓDIGO DO ALGORITMO TAFT

```
#define TASK_CREATE  0x0034
#define TASK_START   0x0010
#define TASK_STOP    0x0001
#define TASK_DELETE  0x0057

#define CREATE_ID           0x30
#define START_ID    0x15
#define STOP_ID     0x40
#define DELETE_ID           0x12
#define SCHED_MP_ID         0x52
#define SCHED_EP_ID         0x25
#define FIN_TASK_ID         0x93

// Status de tarefas
#define DESATIVADA          0
#define ESPERA_ATIVACAO         1
#define EXECUTANDO_MP           2
#define EXECUTANDO_EP 3
#define EXECUCAO_OK     4

#define CF_PAR   *(short*) 0x200000ca
#define CF_PADDR *(char*) 0x200001c5
#define CF_PADAT *(char *) 0x200001c9

#define CF_TMR1   *(short*) 0x20000100
#define CF_TRR1   *(short*) 0x20000104
#define CF_TER1   *(char*) 0x20000111

#define DUAL_PORT_TX (short*) 0xE0010100
#define DUAL_PORT_RX (short*) 0xE0010200

#define NUM_MAX_TAREFAS   20
/*struct tarefas_TAFT
{
unsigned short id_tarefa;
unsigned int   deadline;
unsigned int   periodo;
unsigned short tempo_mp;
unsigned short tempo_ep;
unsigned int ti_absoluto;
unsigned int di_absoluto;
unsigned short status;
unsigned char  prioridade;
}; */

struct tarefas_TAFT
{
```

```
unsigned shortid_tarefa;
unsigned long   deadline;
unsigned long   periodo;
unsigned long           tempo_mp;
unsigned long           tempo_ep;
unsigned long           ti_absoluto;
unsigned long           di_absoluto;
unsigned short          status;
unsigned char           prioridade;
};

struct tarefas_TAFT tarefa[NUM_MAX_TAREFAS];
char tarefa_atual=1;

unsigned char indice_DPRAM=1;

void le_tabela();
void verifica_tarefa();

unsigned long tempo_abs_sup;
unsigned long tempo_abs;
unsigned long tempo_rel;

void main()
 {
 short teste=0,x=0;
 int i=0;

 static char num_tar=0;
unsigned short * end_memo;
short tarefa_rec = 0;
unsigned long temp;

for (i=0; i<NUM_MAX_TAREFAS; i++)
    tarefa[i].status = DESATIVADA;

tarefa[0].id_tarefa = 100;
tarefa[0].deadline = 1000000*2;//000;
tarefa[0].periodo  = 1000000*2;//000;
tarefa[0].tempo_ep = 600000*2;
tarefa[0].tempo_mp = 127486*2;//80000;//000;
//tarefa[0].tempo_ep = 90000*2;//20000;//000;
tarefa[0].ti_absoluto = 0;
tarefa[0].di_absoluto = tempo_abs+ tarefa[0].tempo_mp;
tarefa[0].status = ESPERA_ATIVACAO;
tarefa[0].prioridade = 1;

tarefa[1].id_tarefa = 101;
tarefa[1].deadline = 2000000*2;//000;
tarefa[1].periodo  = 2000000*2;//000;
```

```
//tarefa[1].tempo_mp = 900000*2;
tarefa[1].tempo_mp = 509847*2;//50000;//000;
tarefa[1].tempo_ep = 100000*2;//000;
tarefa[1].ti_absoluto = 0;
tarefa[1].di_absoluto = tempo_abs+ tarefa[1].tempo_mp;
tarefa[1].status = ESPERA_ATIVACAO;
tarefa[1].prioridade = 2;
//tarefa[1].status= ESPERA_ATIVACAO;

tarefa[2].id_tarefa = 100;
tarefa[2].deadline = 2000000*1.5;//000;
tarefa[2].periodo  = 2000000*1.5;//000;
tarefa[2].tempo_mp = 600000*2;
tarefa[2].tempo_mp = 127486*2;//80000;//000;
//tarefa[0].tempo_ep = 90000*2;//20000;//000;
tarefa[2].ti_absoluto = 0;
tarefa[2].di_absoluto = tempo_abs+ tarefa[2].tempo_mp;
tarefa[2].status = ESPERA_ATIVACAO;
tarefa[2].prioridade = 1;

/*
tarefa[3].id_tarefa = 100;
tarefa[3].deadline = 2000000*0.75;//000;
tarefa[3].periodo  = 2000000*0.75;//000;
tarefa[3].tempo_mp = 127486*3;//80000;//000;
tarefa[3].tempo_ep = 16000*2;//20000;//000;
tarefa[3].ti_absoluto = 0;
tarefa[3].di_absoluto = tempo_abs+ tarefa[3].tempo_mp;
tarefa[3].status = ESPERA_ATIVACAO;
tarefa[3].prioridade = 1;
*/
/*
tarefa[4].id_tarefa = 101;
tarefa[4].deadline = 2000000*0.125;//000;
tarefa[4].periodo  = 2000000*0.125;//000;
tarefa[4].tempo_mp = 598470*2;//50000;//000;
tarefa[4].tempo_ep = 40000*2;//000;
tarefa[4].ti_absoluto = 0;
tarefa[4].di_absoluto = tempo_abs+ tarefa[4].tempo_mp;
tarefa[4].status = ESPERA_ATIVACAO;
tarefa[4].prioridade = 2;
//tarefa[1].status= ESPERA_ATIVACAO;

*/

/*
tarefa[1].id_tarefa = 101;
tarefa[1].deadline = 500000*2;//000;
tarefa[1].periodo  = 500000*2;//000;
tarefa[1].tempo_mp = 59847*2;//50000;//000;
```

```
tarefa[1].tempo_ep = 4000*2;//000;
tarefa[1].ti_absoluto = 0;
tarefa[1].di_absoluto = tempo_abs+ tarefa[1].tempo_mp;
tarefa[1].status = ESPERA_ATIVACAO;
tarefa[1].prioridade = 2;

*/


 tempo_abs_sup = tempo_abs = 0;


 CF_PAR   =   0x20;
 CF_PADDR = 0xf;
 CF_TMR1  = 0x000B;
 CF_TRR1  =  2000;    //100 us

 tempo_rel =0;

// *(DUAL_PORT_TX) = 0x15;

 *(DUAL_PORT_TX) = indice_DPRAM;



// le_tabela();

// le_tabela();

// le_tabela();

// le_tabela();


i=0;
 while(1)
 {


// le_tabela();


  if(CF_TER1&2)
          {
          CF_TER1&=2;

/*        i = !i;
          CF_PADAT = i; */

          verifica_tarefa();
```

```
   temp = tempo_abs+100;//000;

   if (temp<tempo_abs)
                tempo_abs_sup++;

   tempo_abs = temp;


        }
 }

}

void le_tabela()
{
short c=0;
static char num_tar=0;
unsigned short * end_memo;
short tarefa_rec = 0;



switch (*(DUAL_PORT_RX))
   {
    case TASK_CREATE:
                                end_memo = (unsigned short *) &tarefa[num_tar].periodo;
                                *(end_memo+1) = *(DUAL_PORT_RX+1);
                                *(end_memo) = *(DUAL_PORT_RX+2);

   tarefa[num_tar].periodo/=500;

                        tarefa[num_tar].deadline = tarefa[num_tar].periodo;

                                end_memo = (unsigned short *) &tarefa[num_tar].tempo_mp;
                                *(end_memo+1) = *(DUAL_PORT_RX+3);
                                *(end_memo) = *(DUAL_PORT_RX+4);

  tarefa[num_tar].tempo_mp/=500;

                                end_memo = (unsigned short *) &tarefa[num_tar].tempo_ep;
                                *(end_memo+1) = *(DUAL_PORT_RX+5);
                                *(end_memo) = *(DUAL_PORT_RX+6);

  tarefa[num_tar].tempo_mp/=500;

                                tarefa[num_tar].status = DESATIVADA;
                                tarefa[num_tar].prioridade = 2;
```

```
                                tarefa[num_tar].id_tarefa = num_tar+100;

                                   *(DUAL_PORT_TX+1) = num_tar+100;

                                   *(DUAL_PORT_TX) = CREATE_ID;

                                        num_tar++;

                                        break;

    case TASK_START:

                                tarefa_rec = *(DUAL_PORT_RX+1);

                                tarefa[(tarefa_rec-100)].ti_absoluto = 0;
//tempo_abs+tarefa[1].periodo;
                                tarefa[(tarefa_rec-100)].di_absoluto = tempo_abs+
tarefa[1].tempo_mp; //+tarefa[1].deadline;

                                tarefa[(tarefa_rec-100)].status = ESPERA_ATIVACAO;

                                   *(DUAL_PORT_TX+1) = tarefa_rec;
                                *(DUAL_PORT_TX) = START_ID;
                                break;

     case TASK_STOP:
    case TASK_DELETE:

                                tarefa_rec = *(DUAL_PORT_RX+1);

                                tarefa[(tarefa_rec-100)].status = DESATIVADA;

                                   *(DUAL_PORT_TX+1) = tarefa_rec;
                                *(DUAL_PORT_TX) = DELETE_ID;
                                break;

         default:
                                        break;
}



}



void escreve_tabela()
{
```

```
}

/*
void insere_tarefa(struct tarefas_TAFT tarefa1)
{
char i;
for (i=0; i<NUM_MAX_TAREFAS; i++)
   {

    if (tarefa[i].status == DESATIVADA)
          {
           tarefa[i].id_tarefa = tarefa1.id_tarefa;
           tarefa[i].deadline =  tarefa1.deadline;
           tarefa[i].periodo  =  tarefa1.periodo;
           tarefa[i].tempo_mp =  tarefa1.tempo_mp;
           tarefa[i].tempo_ep =  tarefa1.tempo_ep;
           tarefa[i].ti_absoluto = tempo_abs;
           tarefa[i].di_absoluto = tempo_abs+ tarefa1.tempo_mp;
           tarefa[i].status = ESPERA_ATIVACAO;
           return;
          }
   }

}
*/


void verifica_tarefa()
{
unsigned char i;
unsigned char tarefa_ativa = 255;
unsigned long deadline_tarefa=0xffffffff;
unsigned char prioridade_tarefa = 0;
unsigned long temp, temp2;



if ((tarefa[tarefa_atual].status == EXECUTANDO_MP)||(tarefa[tarefa_atual].status ==
EXECUTANDO_EP))
  {
       if (tempo_abs>0xF0000000)
          {
          if (tarefa[tarefa_atual].di_absoluto<0x10000000)
             {
             return;
             }
          else
              if (tempo_abs<tarefa[tarefa_atual].di_absoluto)
                 return;
          }
```

```
        else
           {
                  if (tempo_abs<0x10000000)
                    {
                     if (tarefa[tarefa_atual].di_absoluto<0xF0000000)
                  {
                  if (tempo_abs<tarefa[tarefa_atual].di_absoluto)
                        return;
                  }
              }
             else
                   if (tempo_abs<tarefa[tarefa_atual].di_absoluto)
                        return;
             }
/*         (tempo_abs<tarefa[tarefa_atual].di_absoluto)||(&&))
       {
       //CF_PADAT = 1;
       return;
       }
          else {*/

                    CF_PADAT = 0;

//
//                  *(DUAL_PORT_TX) = 0;
//                  *(DUAL_PORT_TX+3) = tarefa_atual+100;
//                  *(DUAL_PORT_TX+2) = FIN_TASK_ID;
//

            *(DUAL_PORT_TX + indice_DPRAM + 1) = tarefa_atual+100;

            *(DUAL_PORT_TX + indice_DPRAM) = FIN_TASK_ID;
            indice_DPRAM += 2;
            if (indice_DPRAM >= 100)//NUM_MAX_TAREFAS*2)
               indice_DPRAM = 1;
            *(DUAL_PORT_TX) = indice_DPRAM;


                    tarefa[tarefa_atual].status = EXECUCAO_OK;

/*             }*/
   }


for (i=0; i<NUM_MAX_TAREFAS; i++)
   {

    if (tarefa[i].status == EXECUCAO_OK)
```

```
        if
((tempo_abs>=tarefa[i].ti_absoluto)||((tarefa[i].ti_absoluto>0xF0000000)&&(tempo_abs<0x1
0000000)))
            {
             tarefa[i].status = ESPERA_ATIVACAO;
            }

   if (tarefa[i].status == ESPERA_ATIVACAO)
        if
((tempo_abs>=tarefa[i].ti_absoluto)||((tarefa[i].ti_absoluto>0xF0000000)&&(tempo_abs<0x1
0000000)))

            {
             temp =tempo_abs+tarefa[i].tempo_mp;
             temp2 = tarefa[i].ti_absoluto+tarefa[i].deadline;
             if (temp>0xF0000000)
                  if (temp2>0xF0000000)
                    {
                          if
(temp>temp2)//(tempo_abs+tarefa[i].tempo_mp)>(tarefa[i].ti_absoluto+tarefa[i].deadline))
                                {
                                      deadline_tarefa = 0;
                                }
                          }
                  else
                          if (temp2>0x10000000)
                                deadline_tarefa = 0;

             if (temp<0x10000000)
                  if (temp2<0x10000000)
                    {
                          if
(temp>temp2)//(tempo_abs+tarefa[i].tempo_mp)>(tarefa[i].ti_absoluto+tarefa[i].deadline))
                                {
                                      deadline_tarefa = 0;
                                }
                          }
                  else deadline_tarefa = 0;


             temp = tempo_abs+tarefa[i].deadline;
             if (deadline_tarefa)
               {
/*             if (temp<0xff000000)
                  if (temp<deadline_tarefa)*/
            if
(((temp<0xf0000000)&&(temp<deadline_tarefa))||((temp>0xf0000000)&&((temp+0x100000
00)<(deadline_tarefa+0x10000000))))

//          if ((deadline_tarefa)&&(((tempo_abs+tarefa[i].deadline)<deadline_tarefa))
```

```
                {
                        deadline_tarefa = tempo_abs+tarefa[i].deadline;
//              deadline_tarefa_h = tempo_abs+tarefa[i].deadline_h;
                        tarefa_ativa = i;
//              CF_PADAT = (tarefa_ativa+1)*2;
                        }

/*      if (temp>0xff000000)
        if ((temp+0x1000000)<(deadline_tarefa+0x1000000))
                {
                        deadline_tarefa = tempo_abs+tarefa[i].deadline;
                        tarefa_ativa = i;
                        switch (tarefa_ativa)
                                {
                                case 0:
                                        CF_PADAT=1;
                                        break;
                                case 1:
                                                CF_PADAT=2;
                                        break;
                                case 2:
                                                CF_PADAT=4;
                                        break;
                                }


                }*/

                  };
    }


       }
  if (deadline_tarefa)
    {
    if (tarefa_ativa !=255)
        {

      switch  (tarefa_ativa)
              {
              case 0:
          CF_PADAT=1;
          break;
              case 1:
            CF_PADAT=2;
          break;
              case 2:
            CF_PADAT=4;
          break;
              }
```

```
//
//                *(DUAL_PORT_TX+1) = tarefa_ativa+100;
//                *(DUAL_PORT_TX) = SCHED_MP_ID;
//        *(DUAL_PORT_TX+2) = 0;
//

          *(DUAL_PORT_TX + indice_DPRAM + 1) = tarefa_ativa+100;

          *(DUAL_PORT_TX + indice_DPRAM) = SCHED_MP_ID;
          indice_DPRAM += 2;
          if (indice_DPRAM >= 100)//NUM_MAX_TAREFAS*2)
             indice_DPRAM = 1;
          *(DUAL_PORT_TX) = indice_DPRAM;


          tarefa_atual = tarefa_ativa;
          tarefa[tarefa_atual].di_absoluto=tempo_abs+tarefa[tarefa_atual].tempo_mp;
          tarefa[tarefa_atual].ti_absoluto+=tarefa[tarefa_atual].periodo;
//      tarefa[tarefa_atual].ti_absoluto=tempo_abs+tarefa[tarefa_atual].periodo;
          tarefa[tarefa_atual].status = EXECUTANDO_MP;
         }
     }
     else
          {


           for (i=0; i<NUM_MAX_TAREFAS; i++)
                {
                 if (tarefa[i].status == ESPERA_ATIVACAO)
                   if (tempo_abs>=tarefa[i].ti_absoluto)
                        {
                         if
((tempo_abs+tarefa[i].tempo_mp)>(tarefa[i].ti_absoluto+tarefa[i].deadline))
                              if (tarefa[i].prioridade>prioridade_tarefa)
                                   {
                                    prioridade_tarefa = tarefa[i].prioridade;
                                    tarefa_ativa = i;
                                   }
                        }
                 }
//        switch  (tarefa_ativa)
//                {
//                case 1:
//            CF_PADAT=1;
//            break;
//                case 2:
//                  CF_PADAT=2;
//            break;
//                case 3:
```

```
//                  CF_PADAT=4;
//             break;
//                 }

      if (tarefa_ativa !=255)
        {
            switch (tarefa_ativa)
                            {
                                    case 0:
                                            CF_PADAT=1;
                                            break;
                                    case 1:
                                                    CF_PADAT=2;
                                            break;
                                    case 2:
                                                    CF_PADAT=4;
                                            break;
                            }
//
//                  *(DUAL_PORT_TX+1) = tarefa_ativa+100;
//                  *(DUAL_PORT_TX) = SCHED_EP_ID;
//       *(DUAL_PORT_TX+2) = 0;

            *(DUAL_PORT_TX + indice_DPRAM + 1) = tarefa_ativa+100;

            *(DUAL_PORT_TX + indice_DPRAM) = SCHED_EP_ID;
            indice_DPRAM += 2;
            if (indice_DPRAM >= 100)//NUM_MAX_TAREFAS*2)
               indice_DPRAM = 1;
            *(DUAL_PORT_TX) = indice_DPRAM;
             tarefa_atual = tarefa_ativa;
             tarefa[tarefa_atual].di_absoluto=tempo_abs+tarefa[tarefa_atual].tempo_ep;
             tarefa[tarefa_atual].ti_absoluto+=tarefa[tarefa_atual].periodo;
//                  tarefa[tarefa_atual].ti_absoluto=tempo_abs+tarefa[tarefa_atual].periodo;
             tarefa[tarefa_atual].status = EXECUTANDO_EP;
                            }
            }

}
```