



# EPOS: Um Sistema Operacional Portável para Sistemas Profundamente Embarcados\*

Hugo Marcondes<sup>1</sup>, Arliones Stevert Hoeller Junior<sup>1</sup>,  
Lucas Francisco Wanner<sup>1</sup>, Rafael Luiz Cancian<sup>1</sup>,  
Danillo Moura Santos<sup>1</sup> e Antônio Augusto M. Fröhlich<sup>1</sup>

<sup>1</sup>Laboratório de Integração de Software e Hardware  
Universidade Federal de Santa Catarina  
Caixa Postal 476 – 88049-900 – Florianópolis – SC – Brasil

{hugom, arliones, lucas, cancian, danillo, guto}@lisha.ufsc.br

**Abstract.** *Several reasons may cause the change of a embedded application hardware platform, like the need for using less expensive hardware or the need for extra resources. The use of a architecture-independent software/hardware interface brings several benefits to the process of embedded system development. However, the definition of this interface in the realm of embedded systems is not a straight-forward task, since these platform present a huge architectural variability. This work shows how an application-oriented component-based operating system was developed to allow application portability. Case studies present two embedded applications running in different platforms, showing that application source code is totally free of architecture-dependencies. Finally, an ongoing work of Co-Design on proposed system is presented.*

**Resumo.** *Diversos motivos, tais como a necessidade do uso de um hardware mais barato ou pela necessidade de recursos adicionais podem ocasionar a mudança da plataforma de hardware de um sistema embarcado. O uso de uma interface software/hardware independente traz diversos benefícios para o processo de desenvolvimento destes sistemas, contudo definir tal interface no domínio de sistemas embarcados não é uma tarefa trivial, já que as plataformas utilizadas por esses sistemas apresentam uma grande variabilidade arquitetural. Este trabalho demonstra como um sistema operacional orientado à aplicação e baseado em componentes foi desenvolvido facilitando a portabilidade do sistema. Os estudos de caso apresentam dois sistemas embarcados executando em plataformas diferentes, evidenciando que o código-fonte da aplicação é livre de dependências arquiteturais da plataforma. Por fim, um trabalho em andamento de Co-Design no sistema proposto é apresentado.*

## 1. Introdução

No desenvolvimento de sistemas embarcados, é comum que a aplicação seja migrada de um sistema para outro. Isto pode ocorrer por diversos motivos, entre eles a necessidade do uso de hardware de menor custo ou de recursos adicionais tais como memória e componentes presentes em plataformas específicas. Modificar a aplicação para que esta seja

\*Este trabalho foi parcialmente financiado pela FINEP - Financiadora de Estudos e Projetos

executada em uma nova plataforma de hardware pode ocasionar em um indesejável e elevado custo de engenharia recorrente. O uso de uma interface software/hardware altamente portátil é essencial para se reduzir tais custos, sendo uma importante ferramenta para o desenvolvimento de sistemas embarcados. A especificação e implementação desta interface não é uma tarefa trivial no contexto de sistemas embarcados uma vez que as plataformas de hardware presentes neste domínio possuem características bem específicas.

Diversas estratégias tem sido adotadas para permitir a portabilidade da aplicação, diminuindo desta forma os custos de engenharia recorrente. O uso de interfaces de chamadas de sistema (e.g POSIX, WIN32, MOSI) [Mooney 1990] é um exemplo. Seu uso permite que aplicações executem em sistemas operacionais que a contemplem. Para ser efetivamente utilizada, uma interface padrão deve ser amplamente aceita pela indústria. Contudo nota-se que no domínio de sistemas profundamente embarcado, o uso de tais padrões é quase inexistente, restringindo-se apenas a sistemas de propósitos bem específicos, e que geralmente implementam apenas um subconjunto da interface. Isto ocorre principalmente porque que as interfaces aceitas pela indústria são definidas no domínio de sistemas de propósito geral, agregando desta forma, características que dificilmente podem ser acomodadas nos pequenos microcontroladores utilizados em sistemas profundamente embarcados.

Máquinas virtuais (VM) e camadas de abstração de hardware (HAL) são duas outras estratégias utilizadas para se obter portabilidade em sistemas operacionais e consequentemente em suas aplicações [Mooney 1990]. Embora máquinas virtuais ofereçam um bom nível de portabilidade, esta estratégia acarreta em um excessivo *overhead* de processamento e memória, restringido seu uso em sistemas embarcados. Uma outra forma de se obter portabilidade em sistemas operacionais é através do uso de HALs (ex. ECOS, LINUX, WINDOWS). No entanto, nota-se que as implementações atuais sofrem a tendência de incorporarem características arquiteturais presentes na plataforma da qual foram concebidas, limitando desta forma a sua portabilidade. Isto não é desejável em sistemas embarcados, visto que estes apresentam uma grande variabilidade arquitetural.

Este trabalho demonstra como um sistema operacional baseado em componentes e orientado a aplicação foi desenvolvido para permitir a portabilidade da aplicação. Isto foi possível devido ao uso de um cuidadoso processo de engenharia de domínio aliado ao uso de avançadas técnicas de programação, como programação orientada a aspectos e meta-programação estática, em um ambiente orientado a objetos [Fröhlich 2001]. O uso de *mediadores de hardware* [Polpeta e Fröhlich 2004] permitiu que o mesmo sistema executasse em arquiteturas bem distintas (ex. H8, AVR, ARM, POWERPC, SPARCV8, IA32).

Mediadores de hardware também possibilitaram a expansão do EPOS para a geração de SoCs. Componentes de hardware sintetizáveis podem ser inferidos e parcialmente configurados a partir do código da aplicação, pois estes são diretamente relacionados com os mediadores de hardware [Polpeta e Fröhlich 2005]. A engenharia de software realmente eficiente utilizada no projeto deste sistema operacional e sua expansão para a geração de SoCs, fornecem base para que este contemple aspectos de co-design e particionamento, permitindo maior exploração do espaço de projeto quando a plataforma de hardware for configurável.

Estudo de casos apresentam duas aplicações embarcadas executando em diferentes plataformas, demonstrando como o código-fonte da aplicação é livre de dependências arquiteturais, facilitando desta forma a portabilidade da mesma. O primeiro estudo de caso apresenta a implementação de um sistema de controle de acesso utilizando um microcontrolador AVR de 8-bits da Atmel. O segundo estudo de caso apresenta um multiplexador MPEG executando em uma plataforma baseada no processador de 32-bits POWERPC da IBM e em uma plataforma baseada na arquitetura IA32.

A seção 2 apresenta as principais variações arquiteturais identificadas no domínio de sistemas embarcados, assim como, uma análise das técnicas utilizadas para a portabilidade de sistemas operacionais. A seção 3 apresenta nossa proposta de interface software/hardware para permitir a portabilidade de sistemas embarcados. A seção 4 apresenta os estudos de casos realizados e o trabalho de Co-Design em andamento, finalmente a seção 5 apresenta um resumo do trabalho e discute seus resultados.

## **2. Portabilidade em Sistemas Embarcados**

Sistemas embarcados utilizam uma variada gama de arquiteturas de hardware, desde simples microprocessadores de 8-bits a sofisticados processadores de 32, 64 ou 128-bits. A escolha por uma determinada arquitetura é baseada nos requisitos da aplicação-alvo com o intuito de reduzir os custos de produção.

Aplicações profundamente embarcadas geralmente não necessitam de mecanismos complexos de proteção de memória e podem ser construídas em arquiteturas que não provém uma unidade de gerenciamento de memória (MMU). Muitos microcontroladores são baseados em uma arquitetura Harvard, com barramentos de instruções e dados separados, contrastando desta forma com a arquitetura de von Neumann, utilizada na maioria dos microprocessadores de propósito geral. Microcontroladores embarcados são baseados em arquiteturas RISC ou CISC, trocando eficiência de pipeline por densidade de código.

A compilação do software também é fortemente influenciada pela arquitetura-alvo. Quando uma determinada arquitetura provém uma grande quantidade de registradores, é comum que o compilador utilize parte deles para efetuar a passagem de parâmetros na chamada de funções e para retornar seus valores. Algumas arquiteturas podem utilizar estruturas mais complexas, como a janela de registradores presentes no SPARC. Arquiteturas POWERPC por exemplo, utilizam um registrador dedicado para armazenar o endereço de retorno de uma função, postergando a decisão de mover tal registrador para pilha, apenas quando um nova chamada for efetuada.

Atualmente, a abordagem de System-on-a-Chip (SoC) surge como uma solução de compromisso entre complexidade de desenvolvimento e custo da plataforma. Dispositivos Lógicos Programáveis (PLD) permitem aos desenvolvedores prototipar projetos complexos num pequeno espaço de tempo usando técnicas que estão mais próximas do desenvolvimento de software do que do projeto tradicional de hardware, tornando o uso de FPGAs (*Field Programmable Gate Array*) uma alternativa viável no desenvolvimento de sistemas embarcados. Do ponto de vista do hardware, muito esforço tem sido feito para desenvolver ferramentas que auxiliam projetistas na seleção e configuração de componentes de hardware e também na geração de lógica de cola (*glue logic*), permitindo que o hardware seja definido e gerado de acordo com os requisitos da aplicação. Desta forma, se faz necessário o uso de um sistema operacional altamente portátil e flexível para se

adaptar ao hardware gerado.

Permitir a portabilidade da aplicação através de uma interface software/hardware comum de forma eficiente neste cenário não é trivial e requer a utilização de técnicas de programação eficientes. A utilização de interfaces-padrão de chamadas de sistema (ex. POSIX, WIN32) tem permitido a portabilidade entre sistemas operacionais e arquiteturas de hardware. No entanto, para um padrão ser efetivamente adotado, este deve ser baseado em tecnologias amplamente utilizadas e estabelecidas. Este pode ser o principal motivo pelo qual alguns padrões de interfaces para sistemas embarcados (ex. MOSI - Microcontroller Operating System Interface) não tenham sido adotadas pela indústria.

O modelo de negócio da indústria de sistemas embarcados também contribui para a não aceitação de interfaces-padrão para chamadas de sistema. Na indústria de computação de propósito geral, hardware e software constituem produtos distintos e são geralmente vendidos separadamente. Sistemas embarcados são constituídos de um único produto, que geralmente faz parte de um sistema maior. Desta maneira a portabilidade entre diversos sistemas operacionais adquire uma importância muito menor, já que este é definido pelo próprio desenvolvedor do sistema, e não pelo consumidor.

Camadas de abstração do hardware (HALS) e máquinas virtuais (VMS) também são estratégias para alcançar a portabilidade da aplicação. Máquinas virtuais permitem a portabilidade através da definição de uma arquitetura de hardware hipotética. Dessa forma, aplicações são desenvolvidas para esta arquitetura e traduzidas para o ambiente atual de execução. Esta abordagem permite a portabilidade binária da aplicação, permitindo que os programas executem em diversas plataformas de hardware sem qualquer tipo de modificação. No entanto, a maioria dos sistemas embarcados não podem suportar o *overhead* gerado por uma máquina virtual. Portabilidade binária da aplicação nem sempre é um requisito da aplicação embarcada, e os custos associados aos recursos computacionais necessários para suportar uma máquina virtual (ex. JAVA VM) tornaria o projeto destes sistemas economicamente inviável.

A utilização de uma camada de abstração de hardware (HAL) para abstrair o hardware da plataforma, parece ser a melhor opção para atingir a portabilidade em sistemas operacionais, e é de fato a maneira mais comum entre estes (LINUX, WINDOWS, ECOS). No entanto tais sistemas, geralmente acabam incorporando peculiaridades da arquitetura original para a qual foram concebidos, permitindo apenas uma portabilidade parcial a sistemas similares. Para exemplificar tais dependências analisaremos o já bem conhecido esquema de gerência de memória [Bach 1987] utilizado pelo sistema UNIX.

Neste sistema, a chamada de sistema *brk* é utilizada por processos afim de modificar o tamanho de seu segmento de dados, mais especificamente, utilizada pelas funções *malloc* e *free* da biblioteca *libc* para gerenciar o *heap* de um processo. A implementação desta chamada presume a existência de uma MMU em hardware como suporte a sua estratégia de gerenciamento de memória baseada em paginação, tornando impraticável sua implementação sem uma MMU, pois implicaria em um processo de relocação dinâmica. Desta forma, a HAL do sistema UNIX inclui um mecanismo de alocação paginada, o que torna este sistema atraente quando pensamos em um ambiente multi-tarefa, mas compromete severamente sua portabilidade para uma plataforma que não dispõe de uma MMU.

### 3. Modelando a Interface Software/Hardware Portável

Diversas técnicas podem ser utilizadas afim de se obter um elevado nível de portabilidade em uma HAL. A seguir iremos apresentar as principais técnicas utilizadas na concepção da interface proposta neste trabalho, assim como a modelagem desta interface.

As atuais HALS não exploram satisfatoriamente a variabilidade arquitetural presente no domínio de sistemas embarcados. Isso ocorre por que no momento em que são projetadas, estas não passam por um profundo processo de análise de domínio. O uso de uma cuidadosa engenharia de domínio é fundamental para se atingir o nível de portabilidade necessário em sistemas embarcados. Engenharia de domínio consiste no desenvolvimento sistemático de um modelo de domínio, e sua implementação. Um modelo de domínio, segundo Czarnecki [Czarnecki 1997], é uma representação dos aspectos comuns e variantes de um número representativo de sistemas em um domínio e uma explicação para as suas variantes.

Neste contexto, diversas variações e semelhanças foram identificadas no domínio de sistemas operacionais para sistemas embarcados, tais como, políticas de escalonamento, sincronizadores (mutex, semáforos e variáveis de condição), temporização, mecanismos de gerenciamento de memória (paginação, segmentação, ambas ou nenhuma), tratamento de interrupções, tratamento de rotinas de entrada e saída (mapeamento em memória, portas de I/O, DMA, PIO). Para garantir uma ampla portabilidade do sistema, é desejável que tais características sejam configuráveis no sistema operacional, permitindo desta forma que o sistema operacional se adapte ao hardware e a aplicação.

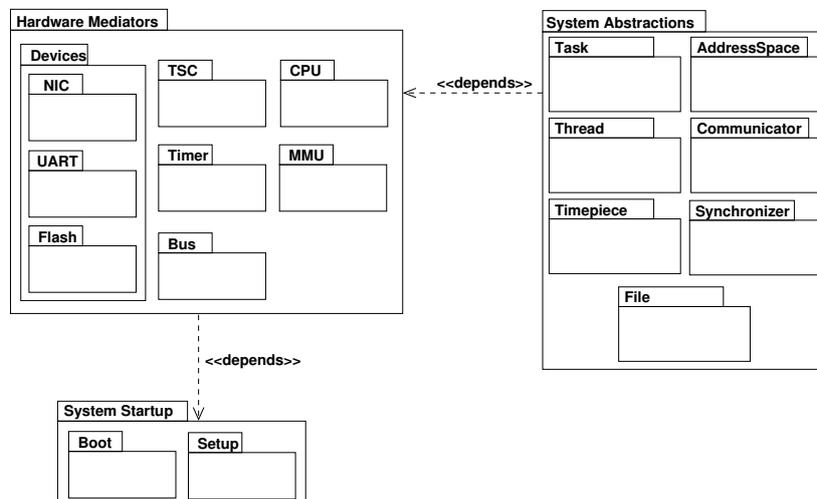


Figura 1. Visão geral dos componentes

Afim de desenvolver um sistema operacional com tais características, um processo de análise de domínio foi utilizado, guiado pela metodologia denominada *Projeto de Sistemas Orientados à Aplicação* (AOSD - *Application-Oriented System Design*) [Fröhlich 2001]. Esta metodologia propõe o uso de diversas técnicas de modelagem e programação para atingir um alto nível de configurabilidade com um overhead de processamento e memória mínimo. Para isso, técnicas como *Modelagem Orientada a Objetos* [Booch 1994], *Modelagem Baseada em Famílias* [Parnas 1976] e *Programação Orientada a Aspectos* [Kiczales et al. 1997] foram utilizadas.

A figura 1 apresenta a organização geral das famílias de componentes do EPOS, um sistema operacional orientado a aplicação. Todas as unidades de hardware dependentes da arquitetura foram abstraídas através de artefatos denominados *mediadores de hardware* [Polpeta e Fröhlich 2004]. Tais artefatos são responsáveis em exportar, através de uma interface software/hardware, toda a funcionalidade necessária as abstrações de mais alto nível do sistema operacional. Essas abstrações correspondem aos serviços tradicionais de sistemas operacionais tais como gerenciamento de memória, gerenciamento de processos, comunicação entre processos, etc. Esses componentes são discutidos em detalhes nas próximas seções, que apresentam os principais sub-sistemas do EPOS.

### 3.1. Gerenciamento de Processos

Processos são gerenciados pelas abstrações *Thread* e *Task*. *Task* corresponde às atividades especificadas na aplicação, enquanto *Threads* são entidades que efetuam tais atividades. Os principais requisitos e dependências de tais abstrações estão relacionadas com a unidade central de processamento (CPU). O contexto de execução, por exemplo, é definido de acordo com os registradores presentes na CPU, e a manipulação da pilha é determinada pelo padrão de interface binária da aplicação, definida de acordo com a arquitetura da CPU.

Muitas das dependências arquiteturais no gerenciamento de processos são tratados pelo *mediador* de CPU (fig. 2). A classe interna *Context* define todos os dados que devem ser armazenados e que caracterizam um determinado fluxo de execução, desta forma, cada arquitetura define o seu próprio *Contexto*. O objeto *Context* é sempre armazenado na pilha da *Thread*.

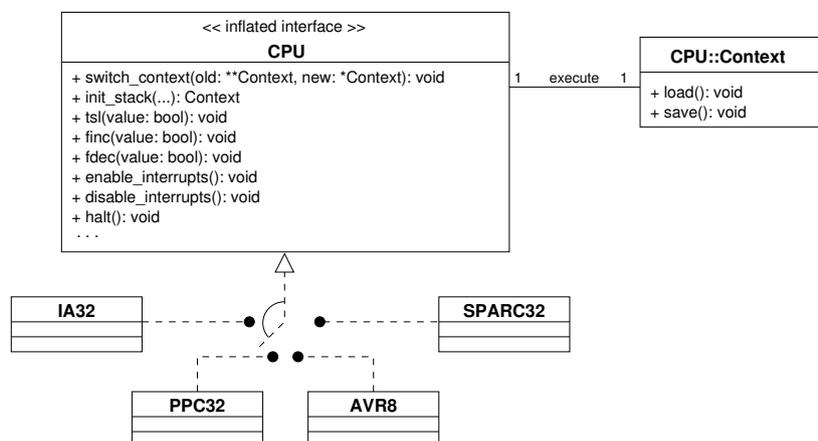
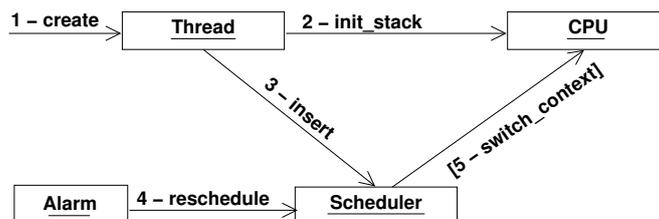


Figura 2. Mediador de Hardware CPU

Outra dependência arquitetural no gerenciamento de processos está relacionado a inicialização da pilha de execução. O uso das estruturas de *template* da linguagem C++ foi essencial para permitir a criação de pontos de entrada flexíveis para as *Threads*, sem implicar em overhead desnecessários. O construtor do objeto *Thread* recebe um ponteiro para uma função com um número arbitrário de argumentos de qualquer tipo. A resolução do tipo e quantidade de argumentos da função é efetuada pelo meta-programa em tempo de compilação. Como compiladores para diferentes arquiteturas manipulam a passagem de argumentos de formas diferentes e os pontos de entrada das *Threads* são compilados

como funções, a inicialização da pilha da *Thread* precisa ser efetuada pelo mediador de CPU, através do método *CPU::init\_stack*, garantindo desta forma que os argumentos da função de entrada são manipulados de acordo com cada padrão de interface binária de aplicação. A figura 3 ilustra a criação (passos 1, 2 e 3) e escalonamento (passos 4 e 5) de *Threads* no sistema. Note que caso um escalonamento preemptivo esteja configurado no sistema, o passo 5 ocorre durante a criação de uma *Thread* caso esta possua uma prioridade maior que a da *Thread* em execução.



**Figura 3. Criação e escalonamento de Threads**

Os mediadores de CPU também implementam algumas funcionalidades para outras abstrações de sistema como transações com travamento do barramento (*Test and Set Lock*) necessárias para a família de abstrações *Synchronizer* e funções de conversão do ordenamento de bytes (ex. Host to Network e CPU to Little Endian) utilizadas pelos *Communicators* e dispositivos de E/S (Dispositivos PCI). O algoritmo de escalonamento de processos é manipulado pela família de abstrações *Timepiece*.

### 3.2. Sincronizadores

Processos e *Threads* geralmente cooperam entre si e compartilham recursos durante a execução da aplicação. Essa cooperação é efetuada através de mecanismos de comunicação entre processos ou através de dados compartilhados. Acesso concorrente a dados/recursos compartilhados pode resultar em inconsistência de dados. Sincronizadores (fig 4) são mecanismos responsáveis por garantir a consistência de dados em um ambiente de processos compartilhados.

O membro *Mutex* implementa um mecanismo simples de exclusão mútua através de duas operações atômicas: *lock* e *unlock*. O membro *Semaphore* realiza a implementação de uma variável de semáforo, que é uma variável do tipo *integer* cujo valor pode ser manipulado indiretamente pelas operações atômicas *p* e *v*. O membro *Condition* realiza uma abstração de sistema inspirada no conceito de linguagem com variáveis de condição, que permite uma determinada *Thread* esperar um determinado predicado em uma variável compartilhada se tornar verdadeiro.

Afim de implementar tais mecanismos, o hardware deve prover meios para manipular dados na memória através de operações atômicas. Algumas arquiteturas provêm instruções específicas para essas operações (ex. instrução *tsl* na arquitetura IA32). Quando a arquitetura não provê tais operações, sua atomicidade é realizada através da desativação de ocorrência de interrupções na CPU. Como visto anteriormente, essas operações atômicas são implementadas no mediador de hardware CPU.

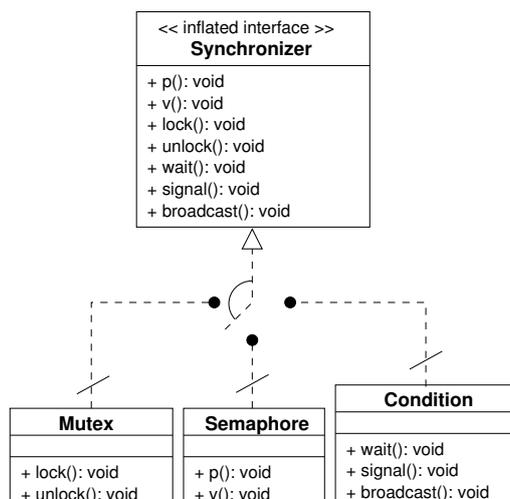


Figura 4. Família de Sincronizadores

### 3.3. Temporização

A noção de passagem de tempo é essencial em qualquer sistema multi-processado. O tempo no EPOS é manipulado através da família de abstração denominada *Timepiece*. A abstração *Timepiece* é suportada pelos mediadores de hardware como o *Timer*, *Timestamp Counters* (TSC) e *Real-Time Clocks*.

A abstração *Clock* é responsável por armazenar o tempo corrente e está disponível apenas em sistemas que possuem dispositivos de relógios de tempo-real. A abstração *Alarm* pode ser utilizada para gerar eventos, para acordar uma *Thread* ou chamar uma função. *Alarms* também possuem um evento principal com alta prioridade que ocorre a uma determinada frequência. Este evento principal é utilizado para invocar o algoritmo de escalonamento de processos a cada *quantum* de tempo, quando um escalonador está configurado no sistema. A abstração *Chronometer* é utilizada para realizar medições de tempo com alta precisão.

Temporizadores em hardware apresentam diversas funções distintas, e podem ser configurados de diversas formas diferentes. Um temporizador pode atuar como um modulador de largura de pulsos controlando um circuito analógico, um temporizador Watchdog, um temporizador de intervalo programável ou um simples temporizador de intervalo fixo. Cada um desses possíveis tipos de temporizadores possuem a suas peculiaridades de configuração. Para preservar um contrato de interface com as abstrações *Alarm*, o mediador de *Timer* apresenta uma visão simplificada do hardware, abstraindo este a um gerador de interrupções periódicas. O programador pode apenas habilitar ou desabilitar a ocorrência de interrupções do timer, assim como, definir uma frequência para a sua ocorrência. Outras abstrações de alto nível podem ser criadas para satisfazer funcionalidades específicas de temporizadores (ex. abstração *Pulse-Width Modulator*), criando desta forma novos contratos de interfaces com propósitos específicos.

Geralmente, arquiteturas profundamente embarcadas não dispõem de um contador de ciclos da CPU (*timestamp*). Quando isto ocorre, o mediador *Timestamp Counter* (TSC) utiliza um temporizador em hardware para contar o tempo. Geralmente, esta abordagem implica em medições de baixa precisão, mas não inviabilizam o uso da abstração *Chro-*

nometer.

### 3.4. Gerenciamento de Memória

Para a maioria dos sistemas operacionais, a presença de uma unidade de gerenciamento de memória (MMU) representa uma barreira que força o sistema a ser portátil apenas para plataformas que possuem um tipo específico de MMU (ex. Paginação). No entanto, através de uma cuidadosa modelagem de abstrações e mediadores de hardware, é possível o desenvolvimento de componentes portáteis para praticamente qualquer plataforma.

O encapsulamento dos detalhes pertinentes a proteção do espaço de endereçamento, tradução de endereços e alocação de memória na família de mediadores de MMU, foi essencial para atingir o alto grau de portabilidade do EPOS. A abstração *Address\_Space* é um *container* para regiões físicas de memória denominada *Segments*. Esse não implementa nenhum mecanismo de proteção, tradução ou alocação, delegando essas responsabilidades ao mediador de MMU. Esta modelagem é retratada na figura 5, que adicionalmente ilustra o fluxo de mensagens para a criação de um segmento (1 e 2) e sua agregação ao espaço de endereçamento (3 e 4).

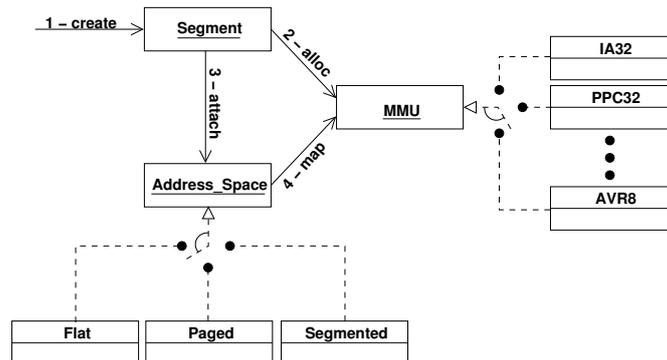


Figura 5. Gerenciamento de Memória

O espaço de endereçamento *Flat* (fig 5) define um modelo de memória no qual os endereços físicos e lógicos são iguais, eliminando desta forma a necessidade do hardware de MMU. Isto garante o cumprimento de um contrato de interface entre os componentes do subsistema de memória em plataformas que não provém uma MMU. O mediador de MMU para uma plataforma que não dispõe de tal hardware é de fato um artefato em software simplificado. Regras de configuração garantem que este artefato simples não pode ser utilizado sem um modelo de espaço de endereçamento *Flat*. Métodos referentes a agregação de segmentos em um espaço de endereçamento *Flat* acaba se tornando vazio, com segmentos sendo agregados ao seu próprio endereço físico. Métodos referentes a alocação de memória opera sobre bytes de uma forma similiar a tradicional função *malloc* da *libc*.

Conceitualmente, o modelo de memória definido pelo membro *Flat* pode ser visto como uma degeneração do modelo de memória paginado, onde o tamanho da página é igual a um byte e a tabela de páginas mapeia endereços físicos como endereços lógicos.

### 3.5. Dispositivos de Entrada e Saída

Controlar os dispositivos de entrada e saída (E/S) de um sistema de computador é uma das principais funções de um sistema operacional. Sistemas profundamente embarcados geralmente não provêm interfaces tradicionais de E/S (ex. teclados, monitores e mouses). Geralmente tais sistemas interagem com o ambiente em que estão inseridos através de sensores e atuadores [Marwedel 2003]. Tais dispositivos apresentam uma grande variabilidade de interface de acesso podendo variar de acordo com arquiteturas utilizadas.

Um fenômeno típico da programação de baixo nível ocorre em relação interface de um mesmo hardware em diferentes arquiteturas. Supondo que um determinado dispositivo faz parte de duas plataformas de hardware, sendo que uma plataforma utiliza I/O programado e a outra utiliza I/O mapeado em memória é bem provável que os procedimentos de acesso, configuração e interação com o hardware sejam idênticos em ambas plataformas, tornando possível que o driver do dispositivo seja um componente portátil. Contudo, as diferenças em relação ao modo de acesso aos dados do dispositivo poderá guiar sistemas operacionais tradicionais a configurar e utilizar dois drivers distintos e não portáteis. Um mediador de hardware meta-programado pode solucionar esse tipo de problema no modo de acesso, introduzindo um componente de *IO\_Register* capaz de definir o modo de acesso ao registrador do dispositivo em tempo de compilação (utilizando especialização de templates e sobrecarga de operadores da linguagem C++). Dispositivos mapeado em memória podem ainda trazer outros problemas de portabilidade. Dispositivos conectados ao barramento PCI operam palavras com ordenamento de bytes *little-endian* e desta forma seu respectivo mediador deve levar em consideração o ordenamento de bytes utilizados pela arquitetura em uso. Isto é tratado pelo mediador de CPU através de métodos específicos para a troca da ordenação dos bytes, caso necessário.

O tratamento de interrupções é outro aspecto importante no gerenciamento de E/S, já que este evita o *polling* de registradores de hardware pela CPU. O gerenciamento de interrupções é feito pelos mediadores *Machine* e controladora de interrupção (*IC*). O *IC* abstrai o processo de habilitar e desabilitar a ocorrência de interrupções, enquanto o vetor de interrupções é gerenciado pelo mediador *Machine*.

A tabela de vetores de interrupções pode ser manipulada de diversas maneiras, de acordo com a arquitetura. Máquinas baseadas na arquitetura POWERPC por exemplo, implementam dois níveis de tabelas, uma para exceções internas da CPU com tamanhos distintos para cada entrada da tabela e uma segunda tabela com entradas de tamanho fixo para as interrupções externas. Microcontroladores AVR geralmente implementam uma única tabela de interrupção, com tamanho e localização pré definidos. O sistema EPOS manipula a tabela de interrupções de uma maneira uniforme. Isto é possível graças ao uso de um componente especial de inicialização do sistema denominado *Setup*. O *Setup* é uma ferramenta não portátil que executa antes do sistema operacional criar seu contexto de execução inicial. Desta maneira, estruturas complementares são criadas quando necessário pelo *Setup* afim de garantir uma manipulação uniforme de interrupções nos mediadores de hardware *Machine* e *IC*.

## 4. Estudos de caso

Esta seção apresenta dois estudos de caso utilizando a interface software/hardware proposta: um sistema de controle de acesso e um multiplexador MPEG-2. Adicionalmente,

esta seção apresenta uma breve descrição do trabalho sendo realizado para o Co-Design de componentes do sistema EPOS.

#### 4.1. Sistema de Controle de Acesso

Esse estudo de caso consiste em um sistema de controle de acesso desenvolvido utilizando um leitor de *smart cards* por aproximação. O sistema embarcado lê os dados do leitor de *smart cards* e realiza uma busca em um banco de dados presente na memória EEPROM interna. Se o dado é encontrado no banco de dados, o sistema libera a trava (por exemplo, ativa um *triac* conectado a uma tranca eletromagnética) através de uma interface de GPIO (*General Purpose Input/Output*).

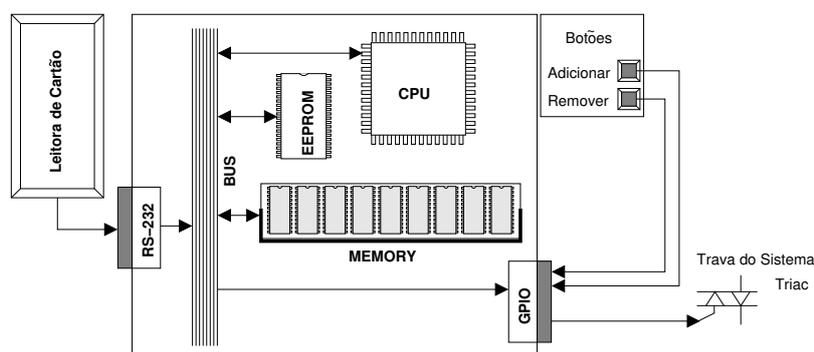


Figura 6. Sistema de controle de acesso

O leitor de *smart cards* por aproximação lê os dados do cartão e os envia através de uma interface serial. Utilizamos um microcontrolador AVR ATmega16 no gerenciamento do sistema. O ATmega16 possui um processador AVR de 8 bits com espaço de endereçamento de 16 bits. Ele possui 16Kb de memória de programa e 1Kb de memória RAM. O banco de dados foi implementado em uma memória EEPROM com capacidade de 512 bytes. O ATmega16 possui um conjunto de dispositivos, como temporizadores programáveis, controladores seriais, ADCs (*Analog Digital Converters*) e interfaces de interação com dispositivos externos (GPIO).

As abstrações de sistema utilizadas neste estudo de caso foram o comunicador serial, para a recepção dos dados recebidos do leitor de *smart cards*, abstrações de armazenamento, para construir o banco de dados na memória (EEPROM), e GPIO para interação do sistema com sistemas externos e usuários. Uma *thread* representa o fluxo principal de execução. Uma segunda *thread* manipula o processo de inserção e remoção de dados no banco de dados em memória EEPROM. Como as duas *threads* dividem o recurso EEPROM, sincronizadores são utilizados para manter a integridade dos dados.

A aplicação foi compilada e ligada com o sistema operacional EPOS utilizando GCC 4.0.2 para a arquitetura AVR. O código objeto resultante utilizou 241 bytes da memória de dados (segmentos *.data* e *.bss*) e 13.484 bytes da memória de programa (segmento *.text*).

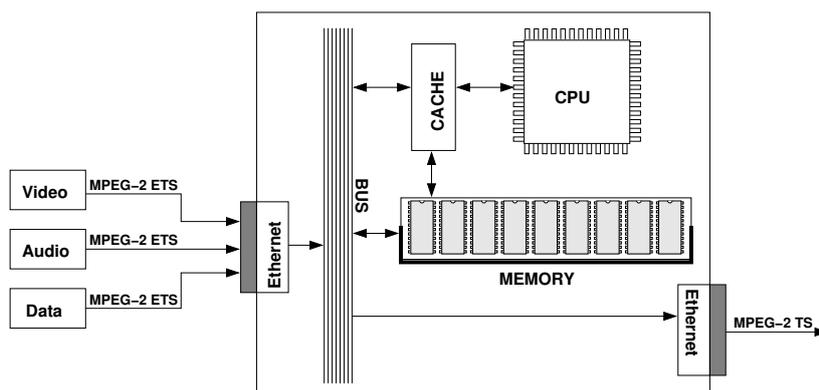
#### 4.2. Multiplexador MPEG-2

O Multiplexador MPEG-2 foi desenvolvido no contexto do sistema brasileiro de televisão digital (SBTVD) e consiste em um sistema embarcado responsável por criar um fluxo de

dados MPEG-2, recebendo um fluxo de dados de áudio, um de vídeo e um de dados. O fluxo de dados MPEG-2 é enviado para o sistema de modulação, sendo depois transmitido para o receptor do usuário.

A sincronização do áudio e do vídeo é fundamental para esta aplicação multimídia, por isso o sistema possui requisitos de tempo real. O uso de imagens com alta definição resulta em um grande fluxo de dados, tornando o uso de microcontroladores simples impossível.

Para a implementação do sistema foi utilizado uma placa de desenvolvimento ML310 da Xilinx que contém uma FPGA (*Field Programmable Gate Array*) modelo VirtexII-Pro XC2V30. Essa FPGA possui dois processadores *hardcores* PowerPC 405 de 32 bits da IBM, sendo que apenas um destes foi necessário a aplicação. Todas as outras funcionalidades necessárias pelo sistema foram sintetizadas na FPGA, de acordo com os requisitos da aplicação. Uma porta UART foi utilizada para efetuar a depuração do sistema, uma ponte PCI e uma controladora de interrupções também foram instanciadas na FPGA para conectar o processador aos dispositivos de I/O externos (ex. placas de rede), responsáveis por receber e enviar os dados MPEG.



**Figura 7. Plataforma do Multiplexador MPEG-2**

A aplicação utiliza um número arbitrário de *threads* para manipular o fluxo de recepção (ES) de dados. Essas *threads* executam de maneira a prevenir estouros dos *buffers* em hardware. Duas *threads* de controle fornecem informações de tempo (T) e sincronização de pacotes (S). A *thread* multiplexadora recebe dados da ES, T e S e envia um fluxo de dados de transporte para saída utilizando uma *thread* de saída (OUT). Para que seja garantida consistência de áreas de dados compartilhadas, sincronizadores são utilizados nas 3 *threads*.

A aplicação do MUX foi desenvolvida também em uma plataforma de PC industrial Geode GX1 com um processador Geode SC2100. A principal razão para isso foi a restrição do tempo de execução do projeto SBTVD. Sendo essa plataforma baseada na arquitetura IA32, uma arquitetura com porte estável do sistema EPOS, os desenvolvedores puderam testar a aplicação em um PC comum enquanto o EPOS era portado para a plataforma ML310.

Isso demonstra a portabilidade da aplicação obtida utilizando a metodologia de projeto proposta. A aplicação executa em ambas plataformas, sem a necessidade de

modificação de seu código-fonte, e apesar de ambas plataformas serem arquiteturas de 32 bits, diversas diferenças podem ser identificadas entre estas. A arquitetura IA32 é uma máquina *little-endian*, possui um conjunto de registradores restrito e uma MMU imposta pelo hardware. A arquitetura PowerPC é uma máquina *big-endian* com 32 registradores (alguns utilizados para passagem de argumentos de funções), e a MMU em hardware foi desabilitada pois a aplicação não necessita de um ambiente multi-tarefa.

A aplicação foi compilada e ligada com o sistema operacional EPOS utilizando GCC 4.0.2 para a arquitetura PowerPC. O código objeto resultante utilizou 1.055.708 bytes da memória de dados (segmentos *.data* e *.bss*) e 66.820 bytes da memória de programa (segmento *.text*).

### 4.3. Trabalho em andamento: Co-Design de Sistemas Operacionais

Esse trabalho visa permitir que a composição de SoCs com o EPOS considere aspectos da aplicação e do espaço de projeto (frequência, latência, área de chip, energia) para a escolha do domínio (hardware ou software) no qual seus componentes serão mapeados (particionamento). Como plataforma de hardware para esse trabalho utilizamos uma FPGA Virtex 2 da Xilinx na qual são sintetizados os componentes de hardware compostos pelo EPOS, incluindo o LEON2, um modelo VHDL de um SoC que possui um processador compatível com a arquitetura SPARC V8. As características do EPOS, já mencionadas, permitiram que ele fosse portado também para essa arquitetura e que diversas funcionalidades do sistema possam ser especificadas de forma a serem particionadas em ambos os domínios.

Uma funcionalidade do sistema que pode ser mapeada em hardware ou software pode ser descrita numa linguagem única, como *SystemC*, mas algumas grandes limitações dessa abordagem, como o suporte de compiladores apenas a um sub-conjunto sintetizável da linguagem, a grande diferença entre os algoritmos implementados em hardware e software que em muitos casos [Grattan et al. 2002] ocasiona a geração de código ineficiente [Cote e Zilic 2002], levaram à escolha de outra alternativa. No desenvolvimento deste projeto, as versões para software e hardware das funcionalidades dos mediadores de hardware estão sendo especificadas em C++ e em VHDL, respectivamente, mas compostas no mesmo arquivo-fonte, de forma que possam ser facilmente comparadas, mantidas e selecionadas, alternativamente, utilizando características do pré-processador do GCC.

Com essa abordagem, a interface software/hardware passa a ser determinada pela ferramenta de configuração e algoritmos de particionamento (atualmente bastante simples), responsáveis por determinar qual das versões implementadas (sw ou hw) será efetivamente incorporada ao sistema final. Quando a versão de software é selecionada, o código correspondente é compilado, e características da linguagem C++ utilizadas pela metodologia AOSD, como metaprogramação estática, garantem *overhead* mínimo ou nulo, transformando chamadas polimórficas em chamadas simples ou mesmo incluindo diretamente o código associado (*inline*).

Entretanto, linguagens de descrição de hardware (como VHDL) não possuem essa e outras características necessárias à obtenção de código eficiente. Assim, neste projeto, o pré-processador do gcc é utilizado para sanar algumas dessas limitações. Utiliza-se o pré-processador inicialmente para selecionar, através de diretivas de compilação (*#if-def*, *#ifndef*, etc), trechos de código VHDL e não C++, quando a funcionalidade asso-

ciada foi mapeada ao hardware. Do mesmo modo, o pré-processador é utilizado para, dependendo do cenário de execução, incluir trechos VHDL de outros arquivos em pontos específicos do código. Com isso, consegue-se selecionar trechos alternativos de código C++ ou VHDL e ainda adaptar o código VHDL para que ele possua apenas as estruturas de programação necessárias e adaptadas. Note-se que essa composição corresponde à aplicação de aspectos ao VHDL. Uma vez gerado o arquivo VHDL puro com o código assim composto, características da linguagem VHDL, como os *generic ports* e os mapeamentos abstratos são usados normalmente para gerar outras adaptações mais simples, como a largura de bits dos componentes ou a escolha correta da arquitetura (*architecture of entity*).

Por fim, o código VHDL completo, incluindo o modelo LEON2, os IPs selecionados e aqueles gerados a partir do particionamento das funções da aplicação são sintetizados com ferramentas disponibilizadas pela Xilinx (como Synplify e o ISE) para geração do *NetList* e do *BitStream*, que contém a configuração do hardware a ser utilizado na FPGA escolhida (via ferramenta de configuração). Na seqüência outras ferramentas realizam a integração dessa configuração com o software (ACE) e finalmente o download (síntese) na FPGA (iMPACT). Seguindo esse processo de desenvolvimento conseguimos gerar sistemas embarcados (hardware e software) em um único chip, sintetizados e funcionais.

## 5. Conclusão

Esse trabalho apresentou o projeto de um sistema operacional altamente flexível que pode ser executado em uma grande variedade de arquiteturas de hardware, desde simples microcontroladores até processadores sofisticados, de acordo com os requisitos da aplicação. Isso é possível utilizando uma refinada engenharia de domínio, projeto baseado em componentes e famílias e técnicas de programação modernas como meta-programação e programação orientada à aspectos. Essas técnicas possibilitaram o projeto de um *contrato de interface* entre abstrações de sistema portáveis e *mediadores de hardware*, possibilitando a implementação de requisitos em software quando este não é disponibilizado pelo hardware.

O uso de uma interface hardware/software única facilita o processo de desenvolvimento. O reuso de abstrações de sistema reduz o *time-to-market* e minimiza gastos com engenharia recorrente quando a aplicação deve ser portada para outra arquitetura de hardware.

A arquitetura desse sistema operacional também favorece o particionamento de funções em componentes de hardware sintetizável e software, viabilizando o trabalho com Co-Design de componentes de sistemas operacionais, já iniciado e que deve trazer novos resultados em breve.

## Referências

- Bach, M. J. (1987). *The Design of the UNIX Operating System*. Prentice-Hall.
- Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2 edition.

- Cote, C. e Zilic, Z. (2002). Automated systemc to vhdl translation in hardware/software codesign. In *Proceeding of 9th International Conference on Electronics, Circuits and Systems*.
- Czarnecki, K. (1997). Beyond objects: Generative programming.
- Denys, G., Piessens, F., e Matthijs, F. (2002). A survey of customizability in operating systems research. *ACM Comput. Surv.*, 34(4):450–468.
- Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, 1 edition.
- Grattan, B., Stitt, G., e Vahid, F. (2002). Codesign extended applications. In *Proceeding of International Workshop on Hardware/Software Codesign*.
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., e Irwin, J. (1997). Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-oriented Programming'97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä  $\frac{1}{2}$ kyliä  $\frac{1}{2}$  Finland. Springer.
- Marwedel, P. (2003). *Embedded System Design*. Kluwer Academic Publishers.
- Mooney, J. D. (1990). Strategies for supporting application portability. *IEEE Computer*, 23(11):59–70.
- Parnas, D. L. (1976). On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9.
- Polpetta, F. V. e Fröhlich, A. A. (2004). Hardware mediators: A portability artifact for component-based systems. In Yang, L. T., Guo, M., Gao, G. R., e Jha, N. K., editors, *EUC*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280. Springer.
- Polpetta, F. V. e Fröhlich, A. A. (2005). On the automatic generation of soc-based embedded systems. Catania, Italy.
- Teng, Q., Wang, H., e Chen, X. (2005). A hal for component-based embedded operating systems. In *COMPSAC (2)*, pages 23–24. IEEE Computer Society.