Introdução aos Sistemas Embarcados

Edna Barros

Sérgio Cavalcante

ensb@cin.ufpe.br, svc@cin.ufpe.br

Grupo e Engenharia da Computação – GRECO
Centro de Informática – CIn
Universidade Federal de Pernambuco – UFPE

1. Introdução

A indústria eletrônica tem crescido nos últimos anos a uma taxa impressionante e um dos principais motivos para tal crescimento é a incorporação de sistemas eletrônicos numa grande variedade de produtos tais como automóveis, eletrodomésticos e equipamentos de comunicação pessoal. Sistemas de computação estão presentes em todo lugar e não é supresa que anualmente são produzidos milhões de sistemas destinados a computadores pessoais (desktop), estações de trabalho, servidores e computadores de grande porte. O que pode surpreender, no entanto, é que **bilhões** de sistemas são produzidos anualmente para as mais diferentes propostas; tais sistemas estão embutidos em equipamentos eletrônicos maiores e executam repetidamente uma função específica de forma transparente para o usuário do equipamento. Como resultado da introdução de sistemas eletrônicos em aplicações tradicionais temos produtos mais eficientes, de melhor qualidade e mais baratos. Dentre os componentes eletrônicos mais utilizados temos os componentes digitais que permitem algum tipo de computação tais como microprocessadores e microcontroladores.

Apesar da melhoria na qualidade do produto final, o projeto de tais sistemas tornou-se bem mais complexo, principalmente por envolver uma série de componentes distintos e de natureza heterogênea. O projeto de uma mesma aplicação pode incluir, por exemplo, transistores e instruções de máquina de um processador.

A maioria das funções dos sistemas eletrônicos atuais, em geral, envolvem algum tipo de computação e controle e são realizadas por componentes digitais. Atualmente é uma tendência que sinais analógicos sejam processados como sinais digitais, de forma que componentes para processamento digital são dominantes nos sistemas eletrônicos e terão o principal enfoque deste texto.

Segundo De Micheli existem três classes básicas de sistemas digitais: emulação e sistemas de prototipação, sistemas de computação de propósito geral e sistemas embarcados (embedded systems). Sistemas de emulação e prototipação são baseados em tecnologias de *hardware* reprogramáveis, onde o *hardware* pode ser reconfigurado pela utilização de ferramentas de síntese. Tais sistemas requerem usuários especialistas e são utilizados para a validação de sistemas digitais.

Sistemas de computação de propósito geral incluem computadores tradicionais abrangendo desde laptops até supercomputadores. Tais sistemas são caracterizados pelo fato de que usuários finais podem programar o sistema. Diferentes aplicações são suportadas dependendo do tipo do *software* utilizado pelo usuário.

Segundo a Webopedia (www.webopedia.com), um sistema embarcado ou embutido (embedded system) pode ser definido como um sistema computacional especializado que faz parte de uma máquina ou sistema maior. Sistemas embarcados são encontrados numa variedade de equipamentos eletrônicos do nosso dia a dia:

(a) produtos de consumo: telefones celulares, pagers, câmeras digitais, video-cassete, vídeo games portáteis, calculadores, etc;

(b) eletrodomésticos: forno de microondas, secretárias eletrônicas, equipamentos de segurança, termostados, máquinas de lavar e sistemas de iluminação;

(c) automação de escritório: máquinas de fax, copiadoras, impressoras e scanners;

(d) automóveis: controle de transmissão, injeção eletrônica, suspensão ativa, freio ABS.

Poderíamos dizer que praticamente quase todos os equipamentos a eletricidade já possuem (ou possuirão em breve) algum sistema de computação embutido no mesmo. Enquanto que cerca de 40% das residências americanas possuiam um computador em 1994, cada residência possui em média 30 computadores embarcados, com a prespectiva deste número crescer para mais de 100 no ano 2000. O custo médio da eletrônica embarcada de um automóvel era de U$1237 em 1995 e hoje chega a U$ 2125. **Bilhões** de microprocessadores embarcados foram vendidos nos últimos anos, enquanto que e venda de microprocessadores para computadores pessoais é da ordem de milhões. O mercado de sistemas embarcados apresenta-se como um nicho extremamente atrativo, porém bastante crítico com relação a alguns aspectos de projeto tais como: custo e tempo de desenvolvimento, bem como o desempenho do produto final.

Sistemas embarcados possuem algumas características que são comuns:

1. Funcionalidade única: usualmente um sistema embarcado executa somente um programa repetidamente. Por exemplo, um pager é sempre um pager, enquanto que um computador pessoal pode executar uma variedade de programas;
2. Restrições de projeto mais rígidas: todos os sistemas de computação possuem em geral alguma restrição de projeto a ser satisfeita, como por exemplo custo, tamanho, desempenho, potência dissipada, etc. Nos sistemas embarcados, no entanto, estas restrições são normalmente mais rígidas, por exemplo o custo de um sistema não pode ser muito alto para não onerar o custo do equipamento, o tempo de resposta deve permitir em várias aplicações processamento em tempo real e devem dissipar pouca potência para permitir uma maior duração da bateria ou não necessitar de um sistema de refrigeração;
3. Sistemas reativos de tempo real: muitos sistemas embarcados devem reagir a mudanças no ambiente e devem fornecer resultados em tempo real. Por exemplo um piloto automático (cruise controller) continuamente monitora e reage a velocidade e aos sensores de freio. Ele deve computar a aceleração e desaceleração repetidamente num intervalo de tempo. Caso haja um retardo o controle do carro pode ser perdido

Vamos analisar agora um sistema embarcado com mais detalhe considerando como exemplo uma câmara digital mostrada na Figura 1. Os circuitos A/D e D/A fazem a conversão de um sinal analógico para digital e vice-versa. O circuito JPEG codec realiza a compressão de descompressão de uma imagem no formato JPEG permitindo um armazenamento compacto de informações na reduzida memória da câmara.

O coprocessador de pixels permite a rápida visualização das imagens. O controlador de memória e o DMA controlam o acesso a memória sem a necessidade de se utilizar o microprocessador. A UART permite a comunicação com um PC via porta serial para carregamento dos frames enquanto que interface ISA permite uma conexão mais rápida com um PC via barramento ISA. Os circuitos controladores de display e LCD controlam a visualização das imagens no display de cristal líquido. O circuito Multiplicador/Acumulador permite permitem o alguns tipos de processamento no sinal e o microcontrolador controla a ativação de todos os outros circuitos. Neste sistema cada módulo mencionado tem uma aplicação específica enquanto que o microcontrolador é um circuito para aplicações mais genéricas.

Vamos analisar agora as características mencionadas considerando o exemplo como base: o sistema possui uma única função como câmara digital e executa repetidamente uma série de ações: captura, comprime e armazena frames, descomprime e permite a visualização de frames e carrega frames. Algumas restrições de projeto são bastante rígidas: o sistema deve ter um baixo custo para permitir que uma câmara seja acessível a maioria das pessoas, deve ser pequeno para caber numa câmara convencional, deve ser rápido para processar várias imagens em milisegundos e deve consumir pouca potência para uma maior duração da bateria. Este exemplo, particularmente, não possui um alto grau de reatividade de tempo real, basta responder ao clique do botão.

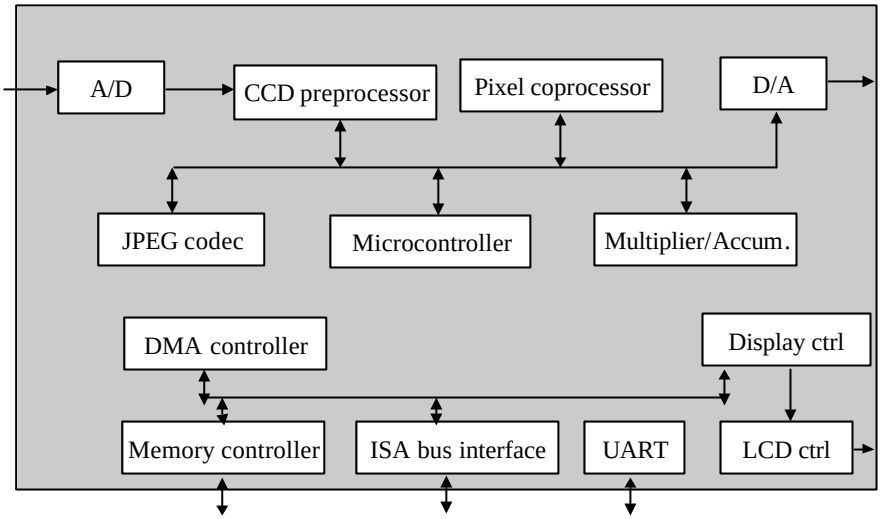


Figura 1 – Câmara Digital

Como visto no exemplo apresentado, a grande maioria dos sistemas embarcados incluem componentes de *hardware*, isto é, de aplicação específica, e de *software*, ou seja, componentes programáveis de propósito geral. As principais tecnologias utilizadas na implementação de sistemas embarcados, incluindo processadores, circuitos de aplicação específica e de interface, estão descritas na seção 0. A qualidade de tais sistemas é, em geral, medida por critérios específicos à aplicação incluindo desempenho, custo de desenvolvimento e de produção, flexibilidade, tolerância a falhas, etc., os quais serão atingidos dependendo de como *hardware* e *software* são projetados.

Um desafio no projeto de sistemas embarcados é a obtenção de um resultado final que implemente a funcionalidade desejada e que ao mesmo tempo satisfaça à todas as restrições de projeto. A seção 2.1 apresenta alguns tópicos de fundamental importância no projeto de sistemas embarcados. O projeto de sistemas digitais envolve basicamente 3 fases distintas: análise, *design* e implementação. Na fase de análise os requisitos funcionais e não funcionais devem ser capturados para que uma primeira especificação do sistema possa ser delineada. Na fase de *design,* as várias alternativas de projeto devem ser análisadas de forma a se ter a solução mais adequada para a aplicação. Esta solução deverá ser implementada posteriormente. Com a crescente complexidade dos sistemas digitais a escolha da alternativa adequada passou a ser uma tarefa não trivial. Para suportar as fases de análise e implementação de forma efetiva, uma nova metodologia de projeto vem sendo desenvolvida denominada *hardware/software co-design*.

Hardware/software co-design é um novo paradigma de projeto, cujo principal objetivo consiste em projetar sistemas digitais que satisfaçam às restrições de projeto através da utilização de componentes de prateleira (*common-off-the-shelf* COTS) e componentes de aplicação específica. Esta heterogeneidade dos componentes implica que em tais sistemas *hardware* e *software* devem ser desenvolvidos de forma integrada e eficiente.

O uso de técnicas de hardware-software co-design permite o projeto do sistema como um todo, e não como partes separadas de hardware e software. Várias áreas de ciência da computação são envolvidas no processo incluindo mecanismos de especificação, técnicas de síntese e compilação, simulação e verificação formal, etc. Além disso, co-design permite a unificação e integração de esforços envolvendo outras áreas como a mecânica, elétrica, matemática e estatística. Isto favorece uma aproximação e melhor visão das semelhanças existentes entre elas.

Na elaboração de projetos de sistemas envolvendo hardware e software, esta unificação permite uma melhor coordenação das diversos equipes de desenvolvimento e também uma visão global do sistema mesmo antes de se obter a implementação final. Esta visão global é de fundamental importância nas tomadas de decisão relativas à implementação, tecnologia empregada, etc., contribuindo para a redução de custos e de tempo de projeto. A capacitação de pessoal nesta área vai promover um melhor entendimento do sistema como um todo e de todas as áreas de conhecimento envolvidas no processo de desenvolvimento de tais sistemas, podendo inclusive culminar com o surgimento de um novo perfil de profissional, o de Engenheiro de Sistemas.

No Brasil, diversas são as equipes que trabalham com *co-design*, dentre as quais destacam-se: UFPE, PUC-RS, UFRGS, UFMG, e UFRN. Devido à grande abrangência e inter-disciplinaridade desta área de pesquisa, têm surgido projetos cooperativos entre universidades e centros de pesquisa nacionais e internacionais de modo a solucionar a vasta gama de problemas existentes. Vale ressaltar que o conceito de *co-design* tem alcançado um nível de maturidade tal que ele não somente capta o interesse do setor acadêmico, mas também do setor industrial. A tendência é a integração de esforços no desenvolvimento de metodologias e ferramentas segundo um conhecimento claro dos conceitos e tendências atuais. Uma descrição mais detalhadas da metodologia de hardware/software co-design será dada na seção 2.1. A seção 4 finaliza este texto com algumas conclusões e tendências futuras na utilização e projetos de sistemas embarcados.

2. Tecnologia Empregada

Diferente dos projetos de software para desktops, onde a plataforma a ser usada na implementação do sistema já é conhecida a priori, os projetos de sistemas embarcados apresentam uma grande flexibilidade não apenas do ponto de vista do software mas também do hardware. Sistemas embarcados podem ser implementados sobre plataformas de uso geral mas também podem ser completamente projetados com hardware dedicado. Além disso, sua interação com o ambiente geralmente requer o uso de sensores, atuadores e outros dispositivos que normalmente não estão presentes em sistemas de propósito geral.

As subseções, a seguir, apresentam uma panorâmica das tecnologias empregadas na implementação de sistemas embarcados, principalmente do ponto de vista de hardware.

2.1 Dispositivos Processadores

Geralmente chamamos de processadores apenas aqueles dispositivos cuja função é definida por meio de linguagens de programação. Entretanto, podemos utilizar uma definição mais abrangente e dizer que processadores são dispositivos usados para transformar ou mover dados e/ou tomar decisões sobre ações a serem executadas.

Neste sentido, vários dispositivos podem ser considerados processadores. Alguns seguem a linha mais tradicional, sendo programados via software e portanto apresentando grande flexibilidade para alterações de sua funcionalidade e comportamento. Estes serão chamados por nós de *processadores de software.* Por outro lado podemos ter dispositivos de hardware desenvolvidos especificamente para desempenhar uma determinada função. Estes dispositivos são extremamente rápidos mas não são flexíveis e caso seja necessária alguma alteração de seu comportamento deverão ser substituídos por outro dispositivo. Estes serão chamados de *processadores de hardware.* Entre estes extremos existem várias abordagens intermediárias. Nas seções seguintes serão apresentados alguns dos dispositivos processadores mais usados para sistemas embarcados no momento.

É importante fazer uma distinção entre o processador que está sendo usado para desenvolver o projeto e aquele(s) onde o sistema embarcado será implementado. Chamaremos o primeiro de *processador de desenvolvimento*, onde temos o ambiente de desenvolvimento, como editores de texto, compiladores, depuradores, etc. O segundo é o *processador alvo*.

2.1.1 Processadores de propósito geral

Embora seja possível utilizar processadores de propósito geral, como Pentiums e PowerPCs, na implementação de sistemas embarcados, isto geralmente não é feito na prática. De modo a tornar os projetos de computadores baseados nestes processadores mais flexíveis, os fabricantes deixam muitas tarefas para serem implementadas externamente, por componentes auxiliares. Esta flexibilidade tem vantagens do ponto de vista de desempenho mas adiciona um custo extra que pode inviabilizar o projeto de sistemas embarcados baseados nestes processadores.

Uma solução paliativa encontrada pelos fabricantes é criar “processadores embarcados” (*embedded processors*) baseados em processadores de uso gera que incorporam vários dispositivos que facilitam e barateiam o projeto de sistemas embarcados.

A grande vantagem em utilizar este tipo de processador está na fase de desenvolvimento do projeto. Pode-se usar um desktop que utilize um processador da mesma família do processador que se deseja usar no sistema embarcado. Neste caso, o processador de desenvolvimento e o processador alvo são o mesmo ou pelo menos pertencem à mesma família. Assim, ferramentas como compiladores, depuradores e ambientes completos de desenvolvimento para softwares de desktop podem ser utilizados no desenvolvimento do sistema dedicado com facilidades de visualização dos testes na tela do desktop. Para verificar a interação do sistema com os dispositivos externos de entrada e saída, o projetista pode optar por instalar estes dispositivos diretamente no desktop ou simular seu comportamento por meio de software.

2.1.2 Microcontroladores

Microcontroladores são processadores de software com uma filosofia muito semelhante à dos processadores embarcados mencionados na seção anterior, ou seja, incorporam muitas funções num único chip. Embora não exista um consenso quanto a distinção entre microcontroladores e processadores embarcados, consideraremos aqui que, diferente dos anteriores, microcontroladores não são derivados de famílias de processadores de propósito geral usados em desktops e tem, normalmente, um poder de processamento menor. Como os microcontroladores são projetados especificamente para sistemas embarcados, é comum que apresentem um repertório de instruções melhor adaptado para este fim, como por exemplo instruções de manipulação de bits ou acesso a pinos específicos do processador para facilitar a implementação de interfaces com dispositivos externos.

Um microcontrolador pode incorporar uma grande variedade de dispositivos como: conversores analógico-digitais (ADC) e digital-analógicos (DAC), temporizadores, contadores, interfaces seriais, memória de instruções e/ou dados, controladores de interrupção, geradores de clock, controladores de DMA, etc. Por isso, é comum que não seja desenvolvido apenas um mas sim uma família de microcontroladores cada um apresentando um conjunto diferente de dispositivos, freqüência de clock, potência consumida, faixa de temperatura suportada, encapsulamento e preços compatíveis com estas facilidades. Desse modo o projetista pode escolher o modelo que melhor se adapte aos seus requisitos técnicos e de custo. Além disso, quanto maior a família do microcontrolador mais vida útil o projeto terá, visto que é mais provável encontrar processadores da mesma família que incorporem mudanças futuras do projeto.

Família Nitron

A família Nitron inclui seis dispositivos baseados na arquitetura 68HC08. Cada dispositivo possui uma unidade central de processamento (CPU) de alto desempenho largamente aceita, com memória Flash (1,5 K a 4 Kbytes) reprogramável na aplicação, e periféricos - incluindo um "timer" de dois canais de 16 *bits*, com comparação e PWM, proteção do sistema do tipo LVI (Low Voltage Inhibit) com ponto de excursão selecionável e *"auto-wakeup"* a partir do STOP COP (Computer Operating Properly), e um conversor A/D de 8 *bits* e quatro canais (nas versões QT2/QT4/QY2/QY4). Os encapsulamentos disponíveis de 8 e 16 pinos.

Os dispositivos 68HC908 incorporam memória Flash, o que não só permite reprogramar os sistemas durante o desenvolvimento, mas também realizar programações futuras na produção e atualizações em campo. Em síntese, significa que os clientes passam a contar com novos recursos, melhor desempenho, proteção e segurança com aumentos mínimos nos custos finais dos produtos.

Família 8051

Esta é, atualmente, a família mais conhecida de microcontroladores. Inicialmente lançada pela Intel, é atualmente fabricada por várias companhias como Philips, Atmel, Dallas Semiconductors, entre outros, o que garante preços baixos e uma enorme variedade de opções. Estes processadores apresentam, de um modo geral, baixo desempenho e consumo, embora já existam membros da família com performance considerável. Mais informações podem ser encontradas na Internet em http://www.8052.com.

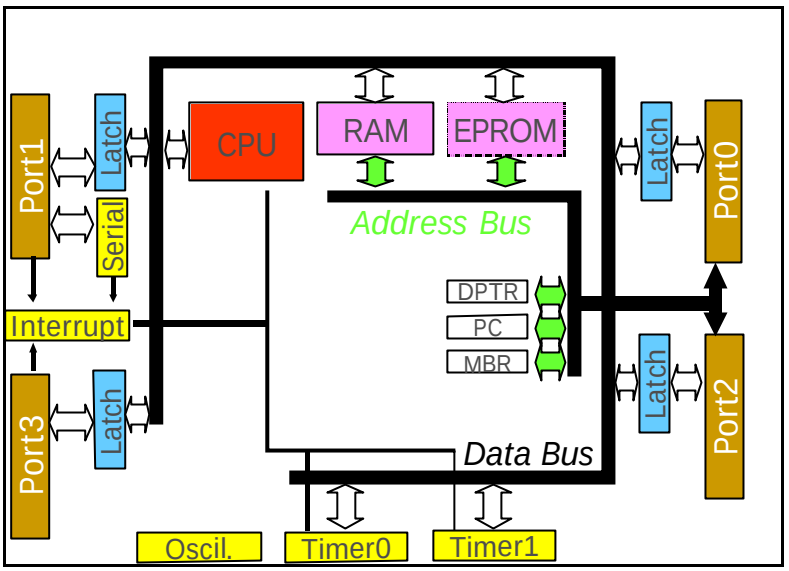


Figura 2. Processador 8051 típico

Outras famílias

Existem várias outras famílias de microcontroladores. A escolha da família a adotar vai depender não apenas dos fatores técnicos mais comuns, como velocidade, potência, tamanho, mas também da facilidade de uso, ambientes de desenvolvimento existentes, conhecimento prévio do time de desenvolvimento, facilidade de compra, número de fornecedores, etc. Dentre as várias famílias podemos citar: ARM da Intel, PIC da Microchip, Série HC da Motorola e Transputers da SGS-Thomson.

2.1.3 ASIPs

Application-Specific Instruction-Set Processors são processadores especialmente desenvolvidos para uma determinada função, seja de controle, processamento de sinais, comunicação, etc. Seu conjunto de instruções e os periféricos incorporados são escolhidos de modo a se obter a melhor relação custo-benefício para o projeto em questão.

De um modo geral, o ASIP e seu compilador são projetados em paralelo de modo a garantir a melhor escolha de implementação de instruções, seja em hardware ou em software. Por exemplo, no projeto de um ASIP para um determinado sistema de controle o projetista pode chegar a conclusão que o número de instruções de ponto flutuante é tão pequeno que não justifica incorporar uma unidade especial ao processador para fazer seu tratamento diretamente em hardware. Neste caso, as instruções de ponto flutuante são tratadas por rotinas de software incorporadas pelo compilador. Atualmente existem muitas pesquisas em andamento para permitir o desenvolvimento rápido de ASIPs e seu compiladores simultaneamente.

ASIPs estão no meio do caminho entre processadores de uso geral e dispositivos de hardware especialmente desenvolvidos para uma determinada aplicação. Eles apresentam melhor performance que os primeiros mas menor que os segundos. Em contrapartida são mais flexíveis a mudanças que os segundos, já que sua programação é feita por software.

2.1.4 Hardware Específico

A opção mais radical com relação ao projeto de dispositivos específicos para uma dada função é construir um hardware dedicado à execução de um determinado algoritmo ou ASIC (Application-Specific Integrated Circuit). Neste caso o projetista tem total controle sobre a implementação, podendo escolher entre diversas opções no espaço de soluções de acordo com seus requisitos de performance, consumo, tamanho, preço, etc. A opção por usar tais dispositivos só se justifica quando os requisitos mencionados não são possíveis de se obter por soluções de software executados sobre

processadores de propósito geral. Isso se deve ao elevado preço para a fabricação de dispositivos sob encomenda e à perda de flexibilidade com relação a mudanças futuras do projeto que se façam necessárias.

Uma opção menos radical está no uso de FPGAs (Field-Programable Gate Arrays), que são dispositivos de hardware programáveis (veja Figura 3). Internamente os FPGAs são compostos de blocos básicos, que por sua vez são compostos de uma lógica combinacional que pode ser programada para implementar qualquer função booleana de 4 ou 5 variáveis, conforme o modelo usado. Cada bloco básico dispõe também de elementos de memória (flip-flops) que podem armazenar os resultados obtidos pela função booleana. A conexão entre blocos básicos também é configurável, formando complexas estruturas combinacionais com armazenamento de estado nos blocos configurados como memória. Em resumo, é possível configurar o FPGA para executar qualquer tipo de algoritmo.

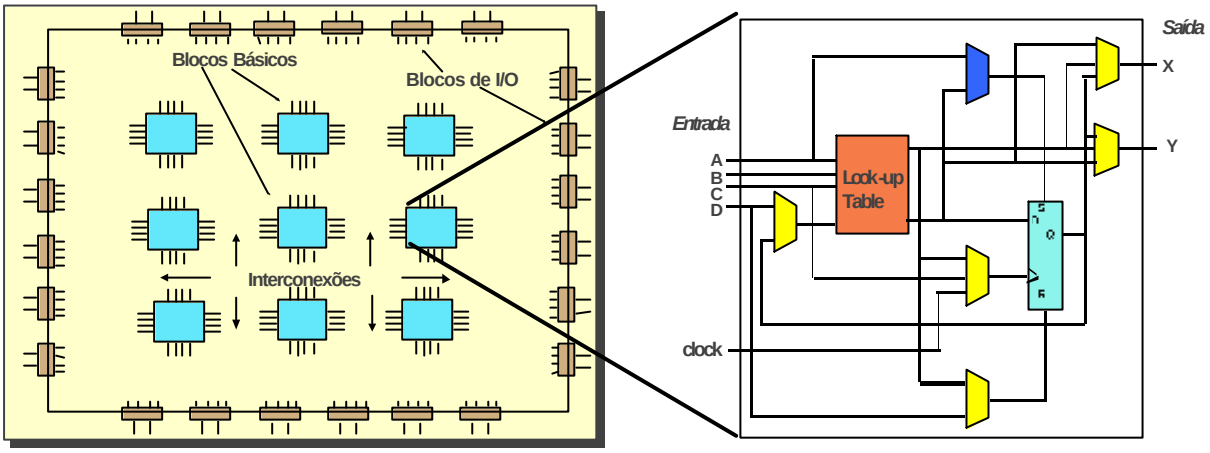


Figura 3. Exemplo de FPGA e bloco básico

Existem vários tipos de FPGA, alguns com programação com tecnologia PROM, que só podem ser programados uma única vez, outros com EPROM, que podem ser reprogramados em laboratório, e ainda outros em RAM, onde é possível até reconfigurar durante a execução. Isso dá uma grande flexibilidade ao projetista aliado a uma performance em geral bem maior que a obtida com o uso de processadores de propósito geral. Embora seu custo por unidade seja em geral maior que o custo unitário de um ASIC, o FPGA não tem custo relacionado à fabricação da primeira unidade, como ocorre com os ASICs. Concluímos então que, caso a performance seja um fator que determine o emprego de FPGA ou ASIC, é melhor adotar o uso de FPGAs para volumes de produção da ordem de alguns milhares de unidades. Se o volume passar para dezenas de milhares de unidades, o uso de ASICs passa, então, a ser competitivo (veja a Figura 4).

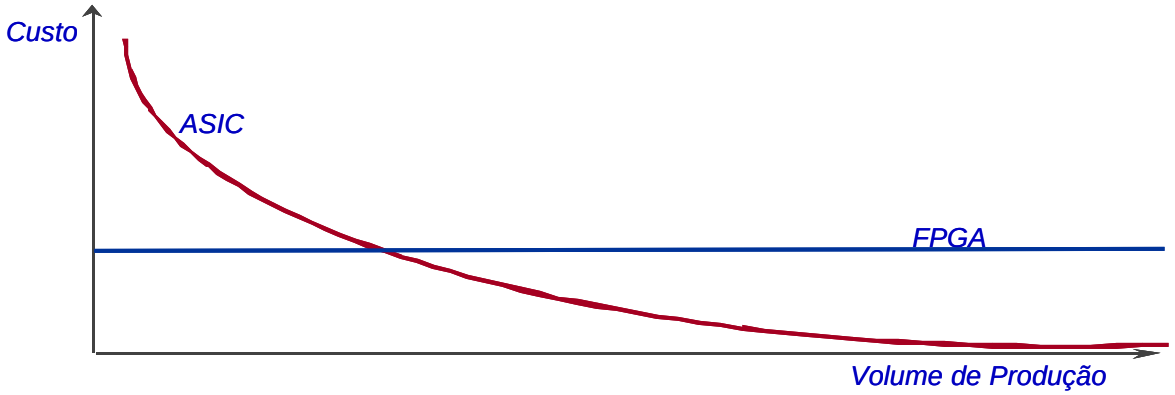


Figura 4. Custo x Volume de Produção para ASICs e FPGAs

2.1.5 System-On-A-Chip

De forma a obter projetos dedicados com reutilização de componentes, uma nova técnica vem sendo usada para permitir o encapsulamento de todo o projeto num único chip, mesmo que o projeto seja composto por uma mistura de processadores de software e de hardware. Por exemplo, vamos supor que o projeto seja composto por um processador 8051 e um ASIC. Para implementá-los como um único chip, o projetista deve dispor de *processor core* (núcleo de processador), que corresponde a uma biblioteca contendo todo o projeto de um 8051 que pode ser incorporada ao chip em desenvolvimento por meio de ferramentas de CAD (*Computer-aided design* ou projeto assistido por computador).

A vantagem em fazer implementações tipo *system-on-a-chip* está no custo da fabricação em série, na qualidade, diminuição de defeitos de montagem e fabricação em geral, potência consumida, tamanho, velocidade, etc. Entretanto, este tipo de implementação só se justifica para grandes volumes de fabricação ou requisitos técnicos muito restritivos.

2.2 Memória

Os tipos de memória usados em sistemas embarcados podem ser bastante diferentes daqueles usados em sistemas de uso geral. Em primeiro lugar, não é comum usar discos rígidos ou outros tipos de armazenamento externo. Depois, dependendo da aplicação, o tamanho da memória pode ser muito pequeno. Para aplicações de controle, por exemplo, a memória de dados não passa, em geral, de algumas centenas ou milhares de bytes.

Além disso, os sistemas embarcados são projetados para trabalhar ininterruptamente sem falhas e estar operacionais o mais rápido possível após serem ligados. Seria inadmissível, por exemplo, que o motorista de um carro fosse obrigado a esperar alguns minutos até que o controle do freio ABS (Anti-Break System) ou da injeção eletrônica estivessem operacionais após o carro ser ligado.

Assim, é muito comum que memórias não voláteis[1] sejam usadas para guardar dados de configuração do sistema, de modo que os dados possam ser recuperados mesmo se o sistema for desligado e religado. Imagine se você fosse obrigado a re-inserir os números de telefone na agenda do seu telefone celular toda vez que ligasse o aparelho! As memórias não voláteis normalmente usadas são do tipo EEPROM (Electrically-Erasable Programable Read-Only Memory), que são memórias cujos dados podem ser apagados byte por byte de forma independente, ou Flash EPROMs, cujos dados são apagados em blocos. A escolha do tipo a ser usado dependerá da aplicação, já que as EEPROMs são mais flexíveis porém mais lentas no apagamento.

Mesmo as memórias voláteis são, normalmente, do tipo SRAM (Static Random-Access Memory), visto que os processadores empregados não dispõem de unidades de gerenciamento de memória sofisticadas o bastante para prover o *refresh* necessário à manutenção de dados nas memórias DRAM (Dynamic Random-Access Memory). É comum, também, utilizar memórias SRAM alimentadas por baterias de modo a manter os dados em caso de falta de energia. Existem chips dessas memórias que já vêm com baterias de Lítio internas capazes de manter os dados por até 10 anos.

2.3 Interfaces externas

O próprio nome *sistemas embarcados* pressupõe que estes são sistemas de computação embutidos em sistemas maiores com os quais interagem por meio de interfaces. Estes sistemas podem ainda interagir com outros sistemas e com o ambiente. Vários tipos de interfaces são utilizados e alguns serão apresentados nesta seção

2.3.1 Interfaces Seriais

São aquelas em que os dados são enviados bit a bit. Existem váriso tipos, como poderão ser vistos abaixo.

---

[1] Memórias não voláteis são aquelas que mantêm os dados mesmo se houver falta de energia.

RS232/422/485

Existem vários tipos de interfaces seriais. A mais comum é o padrão RS232C (*Recommended Standard-232*) da *Electronic Industries Association* (EIA) que suporta apenas comunicações entre dois dispositivos, e é bastante usado por modems, mouses e algumas impressoras. Embora o padrão estabeleça o limite de transmissão de 20kbps e distância máxima entre dispositivos de 15 metros, na prática pode-se transmitir até cerca de 200kbps e com bom cabeamento pode-se atingir facilmente os 30 metros.

Devido à baixa imunidade a ruídos este padrão não é adequado para locais onde haja condições adversas de operação, como fábricas por exemplo. Nestes locais, é aconselhável utilizar um padrão mais robusto, como é o caso dos padrões RS422 e RS485. Estes modelos fazem a comunicação por pares de fio trançados (como os fios telefônicos) o que possibilita comunicações com velocidades superiores a 100Mbps e distâncias de vários quilômetros. Além disso permitem a conexão de vários dispositivos na mesma linha.

Em geral as famílias de microcontroladores existentes no mercado dispõem de modelos com uma ou duas interfaces seriais compatíveis com RS232/422/485.

Universal Serial Bus – USB

É um padrão serial que começou a ser usado em 1998 que suporta taxas de até 12Mbps e pode conectar até 127 periféricos numa única linha. USB permite também que os dispositivos possam ser conectados durante a execução (*hot plugging*) e sejam reconhecidos automaticamente pelo sistema hospedeiro (*Plug-and-Play*). Existem no mercado microcontroladores já dotados de interfaces USB.

Infravermelho

Infravermelho é utilizado como meio de comunicação em vários sistemas embarcados, como videocassetes, televisores, aparelhos de som, celulares, etc. De forma a padronizar seu uso foi criada, em 1993, a Infrared Data Association (IrDA – http://www.irda.org) que oferece 3 padrões de comunicação por infravermelho para vários tipos de aplicações, com várias faixas de velocidade, variações de distância, ponto a ponto ou em pequenas redes. Os padrões são: IrDA-Data, IrDA-Control, e o AIr. Como exemplo, a comunicação do padrão IrDA-Data é do tipo ponto-a-ponto com um cone de incidência estreito (30°), distância máxima de 1 metro e velocidades de 9600 bps a 16 Mbps.

Vários dispositivos já usam IrDA, como notebooks, desktops, palmtops, impressoras, telefones, pagers, modems, câmeras, dispositivos de acesso a redes locais, equipamentos médicos, relógios, etc. Atualmente a base instalada de equipamentos usando IrDA é de cerca de 50 milhões e cresce 40% anualmente.

Bluetooth

Bluetooth é um padrão de comunicação de voz e dados por ondas de rádio para pequenas distâncias, ponto-a-multiponto (*broadcast*). Por usar rádio, Bluetooth pode transmitir através de ojetos não-metálicos entre distâncias que vão de 10 cm a 100 m, embora o normal vá até 10 m. A faixa de frequência usada é de 2.4 GHz. Suporta uma pequena rede de até 8 dispositivos e tem fácil integração com protocolo TCP/IP.

Os dispositivos que estão usando Bluetooth são semelhantes àqueles mencionados na seção de infra-vermelho. Várias companhias estão adotando esta tecnologia, como Motorola, Ericsson, Nokia, IBM, Toshiba, Intel, entre outras.

2.3.2 Teclados e Visores de cristal líquido

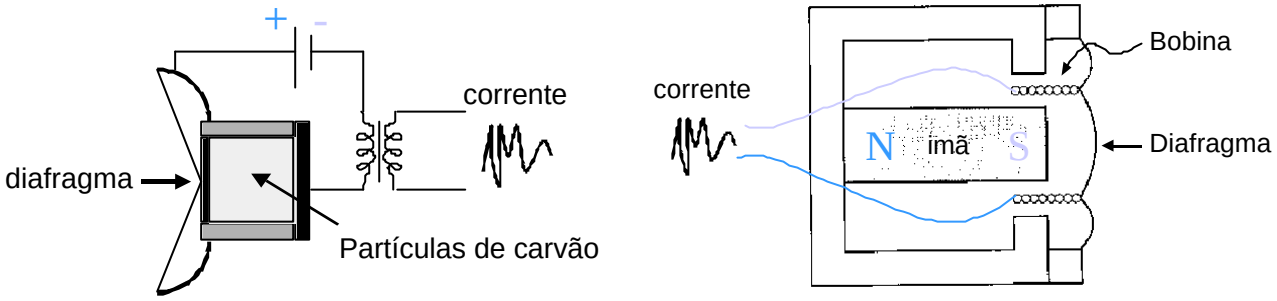
A maioria dos sistemas embarcados requer poucas teclas na interface homem-máquina. Sendo assim, não são utilizados dispositivos especiais para implementar teclados. Geralmente as teclas são conectadas às portas paralelas as quais são verificadas periodicamente pelo microcontrolador (a essa verificação dá-se o nome de *varredura*). Caso haja necessidade pode-se usar controladores de teclado que são facilmente encontrados no mercado. Estes dispositivos fazem a varredura automaticamente e informam o processador o código das teclas pressionadas.

No caso dos visores de cristal líquido, existem vários modelos como os alfanuméricos, que só apresentam letras, números e caracteres de pontuação, e os gráficos que permitem o acesso a cada ponto independentemente, possibilitando desenhar as mais diversas formas. Vários tamanhos estão disponíveis no mercado com ou sem iluminação (*backlight*). Sua interface é, em geral, paralela.

2.3.3 Transdutores

O funcionamento de um microfone é bastante simples. Vejamos o exemplo do microfone de carvão na Figura 5.a. A pressão do ar desloca o diafragma, que muda a densidade de partículas de carvão, variando a resistência elétrica e consequentemente a corrente, que pode ser medida por um conversor analógico-digital. No caso do microfone de bobina móvel da Figura 5.b, a variação do diafragma faz variar o campo elétrico que induz uma corrente elétrica na bobina proporcional a esta variação.

Como os microfones acima, muitos dispositivos são capazes de converter um tipo de energia em outro, e por isso são chamados de *transdutores.* Continuando nosso exemplo, os microfones transformam a energia mecânica das ondas sonoras em energia elétrica. A mesma idéia do microfone de bobina móvel pode ser aplicada de forma inversa para gerar som, como em um alto-falante, que é outro tipo de transdutor.



a) Microfone de carvão **b) Microfone de bobina móvel**

Figura 5

Alguns sensores utilizam materiais especiais para realizar esta transformação de energia. Os materiais piezzo-elétricos, por exemplo, são capazes de se expandir ou contrair de acordo com a voltagem aplicada a eles, e podem também gerar uma voltagem de acordo com a pressão ou expansão que lhe seja aplicada. Por isso eles vêm sendo usados em sensores de pressão, peso, e em microfones e tweeters (alto-falantes para sons agudos). Outros materiais, como os termopares e células fotoelétricas, são capazes de gerar essa voltagem de acordo com a sua temperatura ou iluminação, respectivamente.

2.3.4 Conversores Digital-Analógicos e Analógico-Digitais

Os sinais oriundos dos transdutores são chamados de *analógicos* e oscilam numa escala contínua de valores. Para que o processador possa interagir com esses dispositivos, é necessário que a informação contida na energia elétrica que eles geram seja convertida para números binários, que possam ser interpretados pelo sistema embarcado. Da mesma forma, os sinais digitais gerados pelos processadores devem ser convertidos para analógicos para poderem ser tratados pelos transdutores. Os dispositivos que fazem esta conversão são chamados de *Conversores Analógico-Digitais* (ADC) e *Conversores Digital-Analógicos* (DAC), respectivamente.

Está fora do escopo deste trabalho mostrar como funcionam estes conversores. O importante saber aqui são as características que devem ser observadas na escolha destes dispositivos. Entre elas temos o número de bits, que define a precisão da conversão, e a velocidade de conversão. Pelo Critério de Nyquist, para que a representação digital de um sinal contínuo seja perfeita, a taxa de conversão deve ser pelo menos duas vezes maior que a maior freqüência presente no sinal original. Por exemplo, a taxa de conversão de um CD de música é de 44,1KHz.

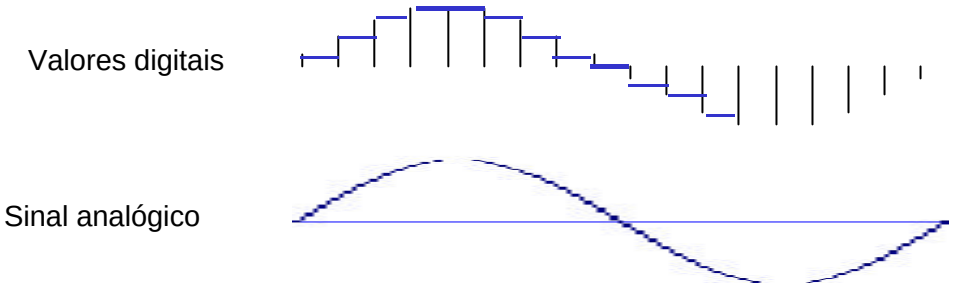


Figura 6. Conversão analógico-digital

2.3.5 Pulse-Width Modulation

Modulação de largura de pulso é muito útil para controlar motores de corrente contínua e solenóides. Neste tipo de modulação, uma onda quadrada é gerada onde podemos definir o período e o percentual do tempo em que o sinal fica em 1 (chamado *duty cycle*), como no exemplo abaixo.

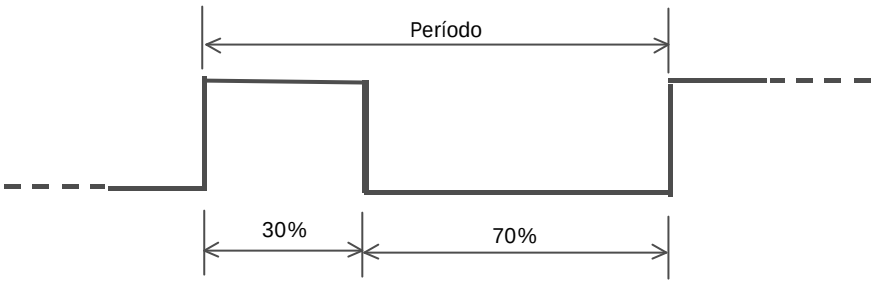


Figura 7. PWM com *duty cycle* de 30%

Ao aplicar um sinal PWM para um motor de corrente contínua, podemos regular sua velocidade por meio do duty cycle. Um duty cycle de 100% faz com que o motor gire na velocidade máxima e com 0% o motor fica parado. Valores intermediários determinam a velocidade entre estes dois extremos sem a necessidade de usar conversores digital-analógicos. O mesmo se aplica aos solenóides, que são dispositivos compostos de um imã permanente e de um eletroimã (veja Figura 8). Dependendo da corrente que circule pelo eletroimã, o solenóide terá um deslocamento maior ou menor. Se a freqüência do PWM for alta o suficiente, o solenóide ficará parado na posição definida pelo duty cycle.

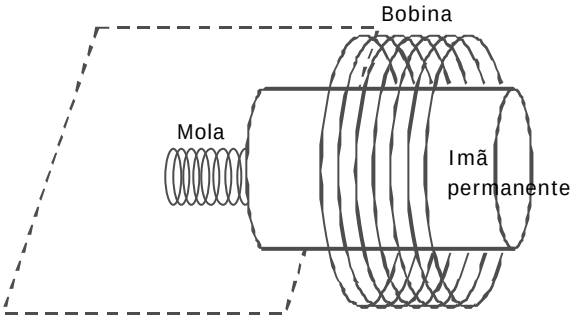


Figura 8. Solenóide

2.3.6 Motores de passo

Motores de passo são tipos especiais de motores que giram alguns graus quando aplicamos o sinal correspondente a um passo. Eles são construídos de forma que o imã permanente do rotor e os eletroimãs do passo subsequente estão desalinhados de alguns graus. No exemplo da Figura 9 o desalinhamento é de 15°. Assim, se desligarmos as bobinas B e B' e ligarmos A e A', o rotor irá girar

exatamente 15°. Para fazer com que o rotor continue a girar é necessário continuar aplicando a seqüência de passos. Estes motores não necessitam de conversores D/A e são muito usados quando precisão de posicionamento é um fator relevante. Por exemplo, motores de passo são usados para movimentar cabeças de impressão e o papel de impressoras, e também para mover a cabeça de leitura/gravação de discos flexíveis.

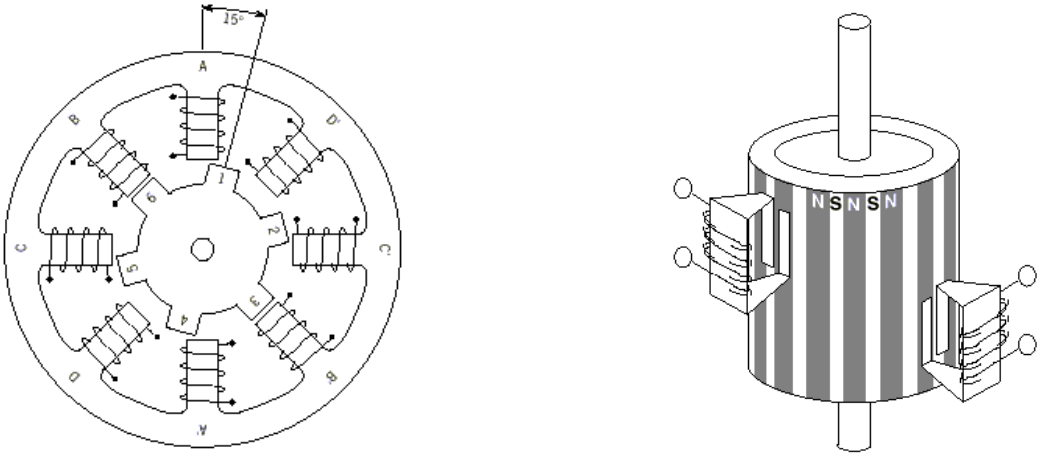


Figura 9. Motor de passo

2.4 Linguagens empregadas

2.4.1 Software

Várias linguagens podem ser usadas para o desenvolvimento de software de sistemas embarcados. Algumas vezes é interessante usar mais de uma. Por exemplo, o projetista pode usar C na maior parte do sistema e usar assembly em regiões críticas, onde o tempo de execução deve ser controlado em detalhes.

Segundo uma pesquisa realizada recentemente pela www.8052.com, 49% dos usuários de processadores compatíveis com o 8051 usam assembly, 33% usam C, 5% usam Basic, 3% usam Pascal e 9% usam outras linguagens. O interesse por linguagens orientadas a objeto vem crescendo dia a dia. Estas linguagens permitem um ciclo de desenvolvimento mais rápido com o uso de melhores métodos de estruturação e modularização, bem como a reutilização de objetos. Entretanto, há ainda alguns obstáculos a serem vencidos para que sejam mais utilizadas em sistemas embarcados. Linguagens orientadas a objetos criam e destróem objetos dinamicamente, dificultando o controle do tamanho necessário de memória e tempo de execução dos programas.

Linguagens como Java, que usam *garbage collectors* automáticos pioram ainda mais o controle do tempo de execução, uma vez que não podemos saber a priori quando o garbage collector entrará em execução e por quanto tempo. Além disso, o escalonamento das *threads* não pode ser definido pelo usuário. Estes problemas estão sendo abordados pelos desenvolvedores da linguagem e por pesquisadores de um modo geral. Recentemente foi lançado uma especificação para Java voltada para tempo real. Resta esperar e ver se esta versão atende os requisitos dos sistemas embarcados.

2.4.2 Hardware

Do ponto de vista de hardware, as linguagens mais empregadas hoje em dia para implementação de dispositivos são VHDL e Verilog. VHDL permite descrições em termos estruturais, com implementações por meio de blocos, e em termos comportamentais, onde as descrições são vistas como processos. A apresentação destas linguagens está fora do escopo deste trabalho.

3. Metodologia de Projeto

O projeto de um sistema embarcado consiste basicamente de três fases:

- Análise
- *Design*
- Implementação

Cada fase consiste numa sequência de atividades. Durante análise os principais objetivos do sistema devem ser capturados e documentados assim como as restrições de projeto. Na etapa de *design* , é decidido como os objetivos serão alcançados e as restrições serão satisfeitas. Na fase de implementação o sistema é construído e testado. É bastente comum alguns projetistas iniciarem pela implementação, porém esta não é uma boa estratégia. As etapas de análise e *design* são necessárias independente da complexidade do projeto.

Em geral as etapas de análise, *design* e implementação não são executadas linearmente, uma abordagem iterativa é a mais conveniente. Ou seja, quando se inicia um projeto as informações sobre os objetivos e funcionalidades são, em geral, incompletas. Estas informações, no entanto, permitirão a obtenção de uma primeira arquitetura do sistema. Após esta fase, os objetivos e restrições obtidos na fase de análise deverão ser revistos para complementação das informações e um consequente refinamento da arquitetura. As fases de análise e design deverão ser repetidas até que a arquitetura obtida satisfaça aos objetivos e requisitos. A partir deste ponto pode se partir para a implementação, porém mesmo durante a implementação pode ser necessário voltar para as etapas de análise e design para complementação da funcionalidade ou satisfação de algum requisito. Um retorno neste caso pode representar um aumento significativo no custo do projeto. Quanto maior o número de iterações envolvendo as etapas de análise e design menor será a probabilidade de um retorno a estas fases durante ou após a implementação.

3.1 Análise - Engenharia de Requisitos

Esta fase inclui a captura dos requisitos funcionais e não funcionais do sistema e pode ser dividida em cinco etapas: especificação do problema, especificação das restrições de projeto, especificação dos requerimentos do usuário, especificação do hardware e do software e verificação da análise. Na etapa de especificação são definidos apenas os objetivos do sistema sem nenhum tipo de solução. Esta especificação deve ser analisada pelo cliente e refinada até que se chegue a uma especificação que mais se aproxime dos objetivos idealizados. Durante a especificação das restrições de projeto todas as restrições que podem comprometer o projeto devem ser analizadas incluindo restrições internas e externas. Uma lista não exaustiva de tais restrições seria:

- Quais os prazos a serem cumpridos?
- projeto possui orçamento limitado?
- Qual o número máximo de pessoas que poderão atuar no projeto?
- Qual a disponibilidade de cada um (horas/sem)?
- Qual a experiência e conhecimento prévio da equipe?
- Existe algum hardware/software pre-alocado ao projeto?
- Existe dependência de alguns fornecedores?

Uma característica desta classe de restrições é que elas odem aparecer em qualquer etapa do projeto. Adicionalmente elas afetam a solução escolhida e podem induzir outras restrições. Algumas destas restrições podem inviabilizar o projeto.

Durante a especificação das restrições do projeto pelo cliente, o projetista deve ter sempre em mente que o cliente não é um engenheiro de forma que lagumas iterações cliente/projetista devem existir até se ter uma lista completa. Exemplos de restrições do projeto a serem fornecidas pelo cliente incluem funcionalidade do sistema, as entradas/saídas do sistema (usuário/outras fontes), o tipo de interface física com o usuário, o peso e tamanho do produto, os periféricos a serem conectados ao produto, se sistema necessita de algum software pré-existente, o tipo de dado a ser processado, se existe comunicação com outros sistemas, se sistema vai estar em rede, o tempo de resposta do sistema, se existe necessidade de algum mecanismo de segurança, quais as condições ambientais de operação, qual a capacidade de armazenamento do sistema, qual o grau de robustês

do sistema de ser robusto, qual o grau de disponibilidade do sistema, se o sistema deve ser escalável, qual a fonte de energia a ser utilizada e como deve se notificar mal funcionamento.

A próxima etapa consiste numa previsão inicial do hardware e do software incluindo velocidade do processador, tamanho do barramento, sistema operacional a ser usado, a linguagem de programação, se alguma biblioteca de software será necessária e quais os componentes de hardware.

Ao fim desta fase a etapa de análise deve ser validada a partir da clareza e completude do resultado obtido. Deve se verificas se alguma informação é irrelevante (ruído), se alguma informação foi omitida, se houve super-especificação, se existe contradição nas restrições listadas e se há ambiguidade na informação capturada.

Uma vez que os resultados da análise foram validados e revistos inicia-se a fase de *design* onde alternativas de projetos são avaliadas de forma a se guarantir que o sistema vai satisfazer requerimentos do usuário dadas as restrições de projeto. Esta fase inclui a especificação dos componentes de hardware, a definição da interface de hardware a especificação dos subsistemas de software, a definição das interfaces de software a especificação dos processos de início e final, bem como das rotinas de tratamento de erros.

Devido a crescente complexidade dos sistemas a análise de todas as alternativas de projeto pode Ter um custo muito alto além de aumentar consideravelmente o tempo de projeto. Neste sentido metodologias que suportam a análise do espaço de projeto vem sendo desenvolvidas de forma a se reduzir o tempo de projeto para achar a solução que melhor se adeque às restrições de projeto e do cliente. A utilização de tais metodologias necessitam que o sistema (funcionalidade + restrições) sejam especificadas em algum mecanismo de especificação. Na próxima seção serão descritos os modelos de especificação mais comumente utilizados na especificação de sistemas embarcados.

3.2 Modelos para Especificação de Sistemas Embarcados

Este seção trata dos vários modelos adotados de arquitetura e especificação de sistemas embarcados, em especial considerando que tais sistemas serão analisados por ferramentas de suporte a hardware/software co-design.

Os modelos de arquitetura adotados em *co-design* podem ser classificados em uma das seguintes categorias: Arquitetura Mono-Processador, a qual possui um processador principal e um ou mais componentes de *hardware* (ASICs, FPGAs, etc.) e Arquitetura Multi-Processador, a qual possui vários processadores trabalhando em paralelo, com um controle distribuído de processamento. Estes processadores compreendem desde processadores de *software* até componentes de *hardware* como ASICs, FPGAs, etc.

É importante perceber que esta classificação está relacionada com a distribuição do controle entre os diversos componentes, sejam de *hardware* ou de *software*, e não com a forma final de implementação. Um sistema pode ser implementado num único *chip* e ainda assim ser considerado como multi-processador, por exemplo, (usando *processor cores* e vários dispositivos de *hardware* com controles independentes implementados sobre a mesma pastilha de silício).

O modelo mono-processador é empregado pela maioria das abordagens de *co-design*. Esta escolha pela arquitetura mono-processador se deve, principalmente, pela simplicidade na implementação do controle, geração de interfaces, e produção de estimativas de qualidade do projeto. Em contrapartida, embora esta arquitetura facilite a metodologia de *co-design*, o modelo multi-processador oferece uma flexibilidade maior, com mais opções de soluções, além de um melhor aproveitamento do paralelismo dos componentes. Entre os sistemas que utilizam esta abordagem temos o COSMOS, LYCOS e *Co-design Studio*.

Com relação aos modelos de especificação, existem duas formas básicas de modelagem usadas em *co-design*: homogênea e heterogênea. A diferença principal é que na abordagem homogênea um único modelo é usado para especificar a funcionalidade do sistema desde a especificação inicial até mesmo após o particionamento entre *hardware* e *software*. Isto permite um tratamento uniforme e, dependendo do modelo usado, não tendencioso durante o particionamento do sistema em *hardware* e *software*. Esta é a abordagem usada pela maioria dos sistemas de *co-design*.

Vários aspectos são importantes para a descrição de sistemas para *co-design*, dentre os quais destacam-se: modelagem de estrutura, comportamento, paralelismo, sincronização, aspectos temporais e requisitos gerais (área, potência, etc.), modularidade, hierarquia, existência de formalismo e métodos de análise. Dada a dificuldade de encontrar todas estas características num único modelo, é comum a combinação de vários modelos, o que caracteriza a abordagem heterogênea. Exemplos desta abordagem podem ser encontrados em no trabalho e Kalavade e Lee.

Dentre os vários modelos usados não só em *co-design* mas também em sistemas embarcados de um modo geral, destacam-se modelos orientados a estados (FSM, e Redes de Petri), modelos orientados a atividades (grafos de fluxo de dados), modelos orientados a estrutura (conexão entre componentes) e modelos heterogêneos (linguagens de programação: Occam, C, Java, etc., linguagens de descrição de *hardware*: VHDL, HardwareC, SpecCharts e Redes de Petri Temporizadas de Alto Nível).

3.3 Sistemas de Tempo Real e Tolerância a Falhas

Como mencionado, a maioria dos sistemas embarcados reagem com o ambiente e em alguns casos devem prover uma resposta num determinado intervalo de tempo. Sistemas cujo tempo de resposta é importante são denominados sistemas de tempo real. Dependendo de quão crítico seja a restrição de tempo, sistemas de tempo real podem ser classificados com *hard real-time* ou *soft real-time*. No primeiro grupo as restrições temporais devem ser satisfeitas de forma rígida. No segundo grupo de sistemas há uma certa flexibilidade quaanto aos intervalos de tempo suportados pelo sistema. Além de todas as restrições já mencionadas, sistemas embarcados de tempo real devem ser projetados de forma que todos os *deadlines* sejam obedecidos, para tal torna-se necessário incluir escalonadores ou núcleo de sistemas operacionais que garantam a execução concorrente de tarefas de acordo com as restrições temporais.

Assim como os computadores pessoais, os sistemas embarcados também têm se tornado mais utilizados nos últimos anos, afetando cada vez mais pessoas. Desde terminais bancários de caixas eletrônicos a aparelhos eletrodomésticos, diariamente as pessoas são beneficiadas pelos serviços que esse tipo de sistema oferece. Desse modo, os sistemas embarcados devem oferecer confiabilidade no seu funcionamento, evitando o prejuízo das pessoas que utilizam os sistemas e dependem deles.

Dessa forma, torna-se necessária a utilização de mecanismos para lidar com os problemas que potencialmente possam afetar os sistemas. Tolerância a falhas é um desses mecanismos. Diferente da prevenção de falhas, tolerar as falhas do sistema, implica em reconhecer que as falhas são inevitáveis; tendo origem em erros de projeto ou de implementação, desgaste do material ou colapsos na fonte de energia; e oferecer alternativas que permitam ao sistema manter o funcionamento desejado mesmo na ocorrência de falhas. Ainda que todo cuidado tenha sido empregado, utilizando técnicas formais de especificação e refinamento dos projetos e verificações de que a implementação dos algoritmos é correta, o software depende do hardware para executar suas funções, estando este sujeito ao desgaste físico do material, que é inevitável.

Portanto para sistemas críticos, onde uma falha acarreta grandes prejuízos, um bom sistema de tolerância a falhas deve ser empregado.

Para se adquirir tolerância a falhas, faz-se necessário o uso de redundância, seja ela de componentes de software ou hardware, informações ou tempo. E no caso dos sistemas embarcados, onde não só o custo e o desempenho, mas atributos como volume, peso e consumo de energia são cruciais para o viabilidade de seu desenvolvimento e utilização, a aplicação de técnicas de tolerância a falhas deve ser bem dosada.

Um estudo das técnicas existentes para aplicação de tolerância a falhas e uma proposta de modelos para utilização de tais técnicas para o desenvolvimento de sistemas embarcados confiáveis, analisando os aspectos de complexidade e recursos envolvidos em cada técnica pode ser encontrado em http://www.di.ufpe.br/~acos/tg/documentos.html.

3.4 Metodologias de Hardware/Software Co-design

O mercado de sistemas embarcados tem crescido numa taxa extremamente alta não só em volume de produção mas também em diversidade de aplicações. Esta demanda crescente de mercado necessita de novas ferramentas e métodos para um suporte efetivo no projeto de tais sistemas. Adicionalmente, os produtos deste mercado possuem um tempo de vida relativamente curto em relação à outras aplicações. Esta peculiaridade exige que o "*time-to-market*" seja o menor possível para que o produto possa ser competitivo no mercado.

A redução do *time-to-market* é um fator extremamente crítico no projeto de sistemas embarcados. Quanto maior for o atraso no lançamento do produto no mercado maior será a perda nos lucros. Estudos mostram que um atraso de 6 meses implica numa queda de 33% nos lucros.

A utilização da técnica de *co-design* tem permitido um significativa redução de custo e do tempo de projeto de sistemas digitais. Durante as ultimas décadas, o aumento em complexidade dos sistemas computacionais tem diversificado as áreas de desenvolvimento em informática e microeletrônica. Inúmeras são as ferramentas e linguagens disponíveis para o desenvolvimento de sistemas digitais. O processo de desenvolvimento tradicional de sistemas mistos *hardware*/*software* baseado no desenvolvimento do *hardware* primeiro e do *software* depois, tem se mostrado cada vez mais difícil e custoso. A razão que leva a esta justificativa baseia-se no fato da existência de disfunções na interface *hardware*/*software* que são descobertas durante o processo de prototipagem ou até mesmo durante o processo de integração e teste do sistema. Assim, o descobrimento tardio destes problemas resulta em projetos extremamente caros e completamente fora do cronograma inicial, visto que etapas de re-projeto e ajustes entre *hardware* e *software,* de custo extremamente alto, são necessárias para implementar efetivamente o sistema.

O crescente interesse em *hardware/software co-design* pode ser justificado pelo avanço tecnológico e pela crescente complexidade das aplicações. A disponibilidade de ambientes de projeto suportando desde a especificação até a prototipação de sistemas digitais complexos tem permitido o projeto de uma variada gama de aplicações cada vez mais complexas.

Além dos fatores tecnológicos e econômicos que motivam cada vez mais pesquisas em *hardware*/*software co-design*, este texto apresentará o algumas técnicas e métodos que suportam o projeto de sistemas digitais de forma automática ou semi-automática.

Uma metodologia genérica de *hardware*/*software co-design* pode ser visualizada na Figura 10. As principais etapas de projeto segundo as publicações mais recentes na área estão representadas nesta figura. Dentre estas etapas destacam-se: a análise de restrições e requisitos, a especificação do sistema, o particionamento em *hardware* e *software*, a etapa de co-síntese incluindo síntese do *hardware*, do *software* e das interfaces, a integração do *hardware* e do *software* e sua co-simulação e, finalmente, a validação e verificação do projeto.

As etapas de análise e especificação já foram discutidas nas seções anteriores, as etapas de particionamento, co-síntese e prototipação serão detalhadas a seguir. Convém lembrar ao leitor que este texto apenas apresentará alguns conceitos básicos de cada etapa. Um maior detalhamento pode ser encontrado nos trabalhos de Barros.

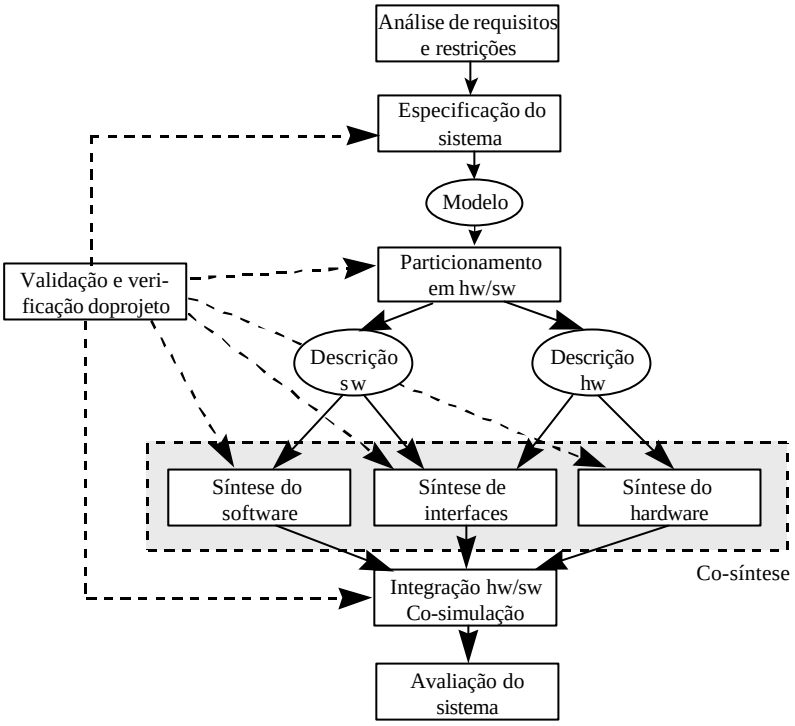


Figura 10 - Metodologia de *Hardware/software co-design*.

3.4.1 Particionamento em Hardware/Software

Particionamento em *hardware* e *software* consiste em decidir como será o mapeamento de uma descrição da funcionalidade de um sistema digital em uma arquitetura composta de componentes programáveis de propósito geral (microprocessadores e microcontroladores) e componentes de *hardware* de aplicação específica (ASIC's, FPGA's). Esta tarefa envolve duas atividades básicas: seleção dos componentes da arquitetura alvo e o particionamento da funcionalidade do sistema entre estes componentes, as quais devem ser realizadas de forma a resultar numa implementação que satisfaça às restrições de projeto tais como: custo, desempenho, área e consumo de potência. A definição das várias possibilidades de implementação e seu mapeamento em *hardware* ou *software* são problemas NP-completos.

Na maioria dos sistemas, a arquitetura alvo é, geralmente, pré-definida e restrições são impostas na mesma de forma a diminuir a complexidade do problema de particionamento. Por exemplo a arquitetura pode estar restrita a uma biblioteca de componentes pré-definidos, o número e a topologia dos componentes de *software* (microprocessadores e microcontroladores) pode ser fixo, bem como o protocolo de comunicação entre *hardware* e *software* pode ser pré-estabelecido.

De um modo geral, os métodos de particionamento dividem a especificação inicial em objetos funcionais, os quais são particionados entre os componentes do sistema para posterior implementação em *hardware* ou *software*. Tais técnicas de particionamento iniciam com uma especificação executável formal da funcionalidade do sistema em um linguagem de programação ou de especificação, a partir da qual os objetos funcionais são derivados.

Muitas técnicas de particionamento têm sido propostas, porém devido a complexidade do problema, a maioria delas focaliza um pequeno subconjunto do problema mais geral do particionamento.

Neste capítulo são discutidas, inicialmente, algumas características das técnicas de particionamento, as quais serão utilizadas posteriormente no estudo comparativo entre as principais e mais recentes técnicas. Tais características foram propostas por Gajski, quais sejam: nível de abstração da especificação, granularidade, métricas e estimadores, função objetivo, algoritmo de particionamento e formato do resultado do particionamento.

Considerando o problema genérico de mapear um conjunto de objetos funcionais em um sistema composto de vários componentes, o principal objetivo de um algoritmo de particionamento é realizar este mapeamento de forma que a partição resultante minimize alguma função objetivo. O problema de particionamento em *hardware* e *software* pode ser visto como um caso especial do problema genérico, onde uma das partições é mapeada para um componente de *software* (microprocessador, microcontrolador). Dentre os algoritmos usados para o particionamento temos o algoritmo de *clustering* hierárquico, o algoritmo *Min Cut*, o algoritmo de *Simulated annealing* e os algoritmos baseados em programação linear inteira (ILP).

Os sistemas de particionamento podem ser agrupados em duas classes: sistemas onde o particionamento é automático e sistemas onde o particionamento é manual ou semi-automático.

Dentre os sistemas de particionamento automático temos o sistema COSYMA, o sistema VULCAN, o sistema LYCOS, o sistema PISH. Uma descrição detalhada do sistema PISH é dada na seção 3.5.

Além dos sistemas citados, existem outros sistemas de *co-design* onde o particionamento é realizado de forma automática ou semi-automática. Dentre tais sistemas temos o sistema SpecSynth, o sistema *Co-design Studio*, o sistema CAMAD e alguns algoritmos de particionamento.

Dentre os sistema que realizam o particionamento a partir de uma interação com o usuário destacam-se os sistemas POLIS, COSMOS e CASTLE.

3.4.2 Estimadores de Qualidade em Hardware/Software Co-design

Uma das grandes vantagens de *hardware*/*software co-design* é permitir que o projetista conheça melhor o projeto e as opções existentes de implementação antes de tomar decisões de particionamento do mesmo. Decisões de projeto devem ser tomadas com base na qualidade esperada para cada opção de implementação. Assim, é crucial a existência de bons estimadores que possam ser empregados para avaliar e comparar a qualidade de cada opção em relação aos objetivos estabelecidos para o projeto.

De modo a entender melhor as técnicas usadas na estimativa da qualidade do projeto é necessário entender, primeiro, que resultados obtemos quando implementamos *hardware* e *software* de modo a saber que parâmetros devemos estimar. Sendo assim, precisamos introduzir as técnicas usadas na implementação das partições de *hardware* e de *software*.

Do ponto de vista de *software*, as técnicas de implementação são bastante conhecidas, sendo baseadas no uso de compiladores. No caso do *hardware*, são utilizadas técnicas de síntese de alto nível, visto que estas são as mais empregadas em *co-design*.

Podemos usar as estimativas de duas formas básicas: como uma comparação entre as diversas opções de implementação, ou como uma forma de estimar se um determinado requisito inicial será atendido numa opção de implementação específica. Para que isso seja possível, é importante que o estimador tenha uma alta *fidelidade*, no primeiro caso, e uma alta *precisão*, no segundo. Os parâmetros de qualidade mais usados são *área*, *velocidade*, e *taxa de comunicação*. Além disso, existem outras medidas também consideradas, como potência dissipada, testabilidade (controlabilidade e observabilidade, no caso de *hardware*), flexibilidade para manutenção, tempo de projeto.

Do ponto de vista de *hardware*, os parâmetros de qualidade mais gerais relacionados com a velocidade são o tamanho do ciclo de relógio, número de passos de controle, e, finalmente, o tempo de execução total .

Dado o número e tipo de componentes que serão usados numa implementação, a área final do projeto pode ser estimada baseada na tecnologia que será empregada (FPGA, *gate arrays*, etc.) e o conhecimento prévio do tamanho de cada componente naquela tecnologia. A estimativa da área depende então da estimativa do tipo e número destes componentes. No caso da Unidade de Execução isto se divide em unidades funcionais (adicionadores, ALUS, etc.), unidades de armazenamento (registradores), e unidades de interconexão (multiplexadores, barramentos, dispositivos *tri-state*, etc.). A área da Unidade de Controle tem uma estimativa diferente, como será detalhadamente neste capítulo.

A forma mais simples de se estimar a velocidade do *software* é compilando o mesmo para o processador desejado e analisando o código executável, o que pode ser feito de forma dinâmica, através de simulação, ou de forma estática, semelhante a técnica anteriormente. Uma outra forma de se estimar este parâmetro é fazer a compilação do *software* para um conjunto genérico de instruções, ou seja, não associado a nenhum processador específico. Após esta compilação são usadas informações sobre o tempo de execução destas operações para processadores específicos de modo a se obter a estimativa de tempo desejada. O segundo método é mais aplicado quando o processador a ser utilizado ainda não foi definido e, por isso, se requer uma comparação entre vários deles. Entretanto, a precisão do primeiro método é maior por utilizar uma compilação específica para o processador desejado. Um dos principais problemas associados às técnicas estáticas é o problema do caminho falso.

Visto que em laços dependentes de dados não é possível se determinar o número de iterações sem impor limites (similar ao problema da parada), é necessário que, para estimar a velocidade de execução do sistema, sejam impostas restrições no uso de tais construtores, bem como no uso de ponteiros, e limitação da recursão.

Um outro problema bastante complexo é como considerar os efeitos de processadores com *pipeline* e memória cache no cálculo do tempo de execução de *software*.

Com relação à área para *software*, a forma mais simples de se obter este parâmetro é por meio de compilação e verificação do código gerado de forma a se obter o espaço alocado para variáveis e instruções. Um outro método é usar um banco de dados com o tamanho do código para cada instrução de alto nível, associado com técnicas de estimativa de espaço para alocação semelhante à técnica discutida anteriormente. Este segundo método, embora mais rápido, não considera muitas das otimizações realizadas pelos compiladores atuais.

3.4.3 Co-síntese

Conforme mencionado anteriormente, o particionamento da especificação de um sistema, a qual é dada, em geral, por um conjunto de módulos que interagem, resulta uma especificação de módulos a serem implementados em *software* e módulos a serem implementados como circuitos específicos, isto é, em *hardware*.

Esta descrição é denominada protótipo virtual e consiste num modelo simulável do sistema. Neste protótipo, a funcionalidade dos módulos de *hardware* e de *software* pode ser descrita utilizando-se um mesmo formalismo ou linguagens distintas, onde os módulos de *hardware* são descritos em uma linguagem de descrição de *hardware*, por exemplo: VHDL, enquanto que os módulos de *software* são especificados numa linguagem de programação convencional (por exemplo: C ou occam). O próximo passo consiste na etapa de prototipação, onde é realizado o mapeamento do protótipo virtual para a arquitetura alvo, que implementa a especificação inicial, ou protótipo real.

Co-síntese consiste, então, num método que permita o mapeamento automático do protótipo virtual no protótipo real de forma a satisfazer às restrições de projeto e às restrições temporais dos componentes da arquitetura alvo.

Uma das principais dificuldades, porém, em se ter este fluxo de forma automática é se conseguir o link entre os dois últimos estágios: prototipação virtual e prototipação real, principalmente quando se considera um amplo espectro de arquiteturas incluindo diversos tipos de processadores, de redes de interconexão e de protocolos de comunicação.

Esta dificuldade torna-se mais evidente se considerarmos a natureza específica dos sistemas embarcados, os quais devem ser otimizados para uma determinada funcionalidade. Neste caso, a tarefa de mapear uma especificação em uma arquitetura alvo de forma a se ter uma implementação ótima demanda muito mais tempo para a sua realização, além de ser bastante suscetível a erros. O alto consumo de tempo tem obrigado a maioria dos projetistas a fixar a arquitetura e os componentes de forma a realizar apenas um mapeamento. Este tipo de conduta pode acarretar, no entanto, um custo de implementação muito maior com processadores mais rápidos e lógicas mais complexas que o necessário. Adicionalmente, sistemas embarcados devem ser projetados como uma família de produtos que possam satisfazer diferentes critérios de custo, desempenho e funcionalidades para cada cliente.

Além do mapeamento do protótipo virtual no protótipo real de maneira eficiente e satisfazendo às restrições de projeto, as características intrínsecas dos sistemas embarcados exigem que técnicas de co-síntese permitam a integração rápida de novos componentes à arquitetura e que garantam a portabilidade de uma dada especificação para mais de uma implementação. A rápida integração se faz necessária para acompanhar o avanço tecnológico e permitir a geração rápida de protótipos, enquanto que a portabilidade permitirá o desenvolvimento de famílias de um mesmo produto.

As principais tarefas a serem realizadas durante a etapa de co-síntese incluem a síntese do *hardware*, a síntese do *software* e a síntese da interface de comunicação entre os diversos componentes.

A síntese de interface constitui um tópico de pesquisa bastante recente em função da complexidade em se gerar interfaces genéricas automaticamente. As principais atividades no processo de geração de interfaces são discutidas neste capítulo, entre as quais destacam-se a geração de *device-drivers*, a geração de *hardware* de interface e a geração de hardware e software para a comunicação entre processadores.

As etapas restantes do processo de co-síntese consiste na síntese dos componentes de *hardware* e de *software* que implementam a funcionalidade do sistema. Os blocos de *hardware* podem ser sintetizados a partir da utilização de ferramentas de síntese de alto nível. Tais ferramentas permitem o mapeamento de descrições funcionais em estruturas, as quais podem ser implementadas em FPGA's ou como ASIC's (ver próxima seção). A síntese de *software* inclui a compilação dos módulos de *software* para a arquitetura alvo, assim como a geração de um escalonador que permita a interação correta de todos os módulos do sistema. Após a fase de co-síntese, a especificação estrutural do sistema pode ser simulada com os módulos de *software* sendo executados nos modelos dos processadores. O sistema final pode ser avaliado a partir da utilização de depuradores e *profilers*.

Os principais componentes de uma interface são os *device-drivers, I/O units* e *hardware* específico. Existem atualmente dois sistemas de síntese de interface: o sistema Chinook e o sistema Symphony. Alguns sistemas de *hardware*/*software co-design* mais populares, entre os quais o sistema COSYMA, o sistema VULCAN, o sistema LYCOS, o sistema PISH e o sistema COSMOS suportam síntese de interface.

A síntese de *software* envolve a tradução da funcionalidade em alguma linguagem de máquina por um compilador, bem como o escalonamento de funções críticas no tempo e concorrentes, quando for o caso. Dependendo se o escalonamento é estático ou dinâmico, um pequeno núcleo de sistema operacional deve ser gerado. Algumas técnicas de escalonamento, incluindo serialização, *multi-threading* e interrupção, serão discutidas neste capítulo. Tais técnicas são utilizadas para que restrições temporais sejam satisfeitas.

3.4.4 Co-simulação

Como mencionado anteriormente, o fluxo de desenvolvimento de sistemas em geral, e de sistemas eletrônicos em particular, passa por três etapas principais: análise, *design* e prototipagem e implementação. Em cada uma destas etapas as ferramentas de verificação ou de validação permitem a constatação do bom funcionamento do sistema antes de passar à próxima etapa.

Com o incremento na complexidade dos sistemas, o tempo investido em cada uma destas etapas resulta um fator que afeta o valor final do produto. Existem diversas alternativas que permitem a redução deste tempo: aumentar o nível de abstração, dividir o projeto em subsistemas e reduzir os erros de projeto. A primeira opção, aumento no nível de abstração, permite trabalhar com um número menor de elementos o que reduz a complexidade e por conseqüência facilita a detecção de erros. Mas esta opção requer o uso de ferramentas que permitam a verificação de que não foram introduzidos erros durante a passagem de um nível de abstração para um outro, onde se tem uma visão mais precisa do projeto. Neste caso é importante dispor de ferramentas que permitam a verificação em cada um destes níveis de abstração assim como também entre níveis diferentes. A segunda opção requer a subdivisão ou particionamento do projeto, o que permite a formação de equipes de trabalho para o desenvolvimento dos diversos componentes do sistema. É importante ressaltar que a validação de cada um desses componentes não é suficiente para garantir o bom funcionamento do sistema completo, pois muitas vezes é preciso verificar a interação entre eles. A ultima opção consiste em garantir que as restrições e requerimentos da especificação sejam

satisfeitas desde as primeiras etapas do projeto a fim de se evitar modificações de custo elevado nas ultimas etapas. Isto implica na utilização de modelos formais e de protótipos virtuais que permitam avaliar custos sem se conhecer detalhes da implementação.

O inconveniente apresentado no desenvolvimento de sistemas embarcados resulta da necessidade de desenvolver três elementos que são intimamente relacionados: o ASIC, o processador e o programa a ser executado pelo processador. Estes são os elementos fundamentais do circuito integrado. Nesta área, a co-simulação se apresenta como um suporte indispensável para permitir o desenvolvimento conjunto desses componentes do sistema. Cada componente é desenvolvido por uma equipe de trabalho, com ferramentas próprias e linguagens adaptadas ao tipo de desenvolvimento a ser efetuado. Por exemplo, os dispositivos ASIC são projetados e desenvolvidos utilizando a linguagem VHDL por uma equipe de projeto. A arquitetura do processador (ASIP) é desenvolvida por uma outra equipe encarregada da definição do conjunto de instruções, unidades funcionais, e do desenvolvimento de ferramentas tais como o compilador, depurador, etc. E, finalmente, tem-se a equipe encarregada de criar os programas a serem executados por esses processadores, geralmente em linguagem C. A validação destes sistemas durante todo o ciclo de desenvolvimento requer não somente a interação de ambientes de simulação diferentes, do ponto de vista do tipo de simulação e a linguagem utilizada, mas também deve permitir que essa validação aconteça antes de serem desenvolvidas algumas das partes, o que significa um ganho importante no tempo de desenvolvimento. Assim, a co-simulação permite por exemplo a execução do programa junto ao ASIC sem se ter ainda o processador ou depurador dedicado, como também a possibilidade de utilizar um modelo funcional do ASIC junto ao resto do sistema.

A técnica de co-simulação vem ultrapassando as fronteiras e hoje ela é utilizada no desenvolvimento de sistemas mecatrônicos, comuns na indústria automobilística, onde combina-se o uso de elementos mecânicos a componentes eletrônicos. As ferramentas de co-simulação permitem a validação de tais sistemas em ambientes integrados onde interagem ferramentas heterogêneas, algumas delas comerciais, utilizadas no desenvolvimentos dos diferentes e diversos componentes destes sistemas.

O interesse e importância da co-simulação tem propiciado uma rápida evolução desta técnica para permitir a interação entre os centros de desenvolvimento. Por este motivo, hoje se fala de co-simulação distribuída como mecanismo de interação e sincronização de ferramentas de simulação diferentes e localizadas em lugares geograficamente distantes.

3.4.5 Prototipação

Quando se fala em prototipar um sistema deve se considerar, também, a validação deste, o que em geral representa de 50% a 70% do tempo de projeto. Dois fatores tem pesado enormemente no desenvolvimento de projetos atuais: a complexidade do projeto do sistema VLSI e o *time-to-market*. Idealmente os projetos deveriam ser desenvolvidos mais rapidamente, lançados no mercado o quanto antes, e devidamente validados. Técnicas de validação de projetos de hardware seguem em geral dois caminhos: a verificação formal e a simulação. Na verificação formal a corretude funcional de um sistema é realizada através de provas matemáticas. Este é ainda um tema de pesquisa e atualmente limitado pelo alto custo computacional para circuitos complexos. A simulação, por sua vez, pode ser usada para verificar a funcionalidade e sua execução requer muitos recursos computacionais. Infelizmente, o tempo de simulação cresce com o quadrado da complexidade do sistema, o que torna este processo também inviável para grandes sistemas.

Uma possibilidade de se acelerar o processo de validação de projetos é o uso de aceleradores em *hardware*, capazes de emular grandes sistemas em tempos próximos de suas implementações reais, e que nos permitisse observar aspectos funcionais e temporais dos mesmos. Técnicas de prototipação rápida baseadas em dispositivos reconfiguráveis permitem compensar este gargalo de simulação proporcionando um método de validação rápida durante a fase de desenvolvimento de sistemas, com menor custo computacional e em menos tempo. Ambientes para a prototipação rápida de sistemas digitais tem se tornado uma realidade e estão sendo cada vez mais utilizados graças ao desenvolvimento de dispositivos reconfiguáveis (FPGA´s) e das ferramentas de CAD que permitem a síntese de sistemas digitais a partir de sua descrição comportamental.

Os FPGAs surgiram em meados de 1980 como uma nova tecnologia para implementação de circuitos digitais. Estes dispositivos programáveis no campo, eram capazes de implementar uma significante quantidade a mais de *hardware* que os tradicionais PLDs além de lógica multi-níveis. Na sua versão baseada em SRAM, os FPGAs passaram a ser a base para a computação reconfigurável, que tem se tornado uma poderosa metodologia para alcançarmos alta performance na implementação de sistemas digitais. Hoje, dispositivos reconfiguráveis são capazes de comportar 500.000 portas lógicas, e expectativas conservadoras calculam algo em torno do 4.000.000 portas lógicas disponíveis até 2010.

Como foi mencionado, anteriormente, o desenvolvimento de metodologias e ferramentas CAD foram de importância fundamental, para o projeto de sistemas digitais e para a disponibilidade de ambientes de prototipação rápida, em particular. A evolução das metodologias e ferramentas foi acompanhada pela evolução de modelos de representação dos sistemas. O desenvolvimento de modelos gráficos, tais como Redes de Petri, StateCharts, Grafos de Fluxo de Dados, etc., e linguagens de descrição de *hardware* representou um importante fator para a automação do projeto de sistemas digitais; visto que ferramentas CAD trabalham em cima destas representações de dados. À medida que modelos novos de representação de sistemas, que conseguem capturar vários aspectos do sistema , vão sendo desenvolvidos, novas ferramentas são criadas e assim mais etapas de projeto de sistemas digitais vão sendo automatizadas. Enquanto que as primeiras ferramentas CAD desenvolvidas cobriam apenas a fase de verificação de sistemas, hoje em dia, tem-se ferramentas que, a partir de uma especificação de alto nível de um sistema, conseguem especificar quais subsistemas devem ser implementados em *hardware* e quais em *software*.

Um projeto de sistemas digitais, normalmente, envolve uma equipe de pessoas, dividida em grupos onde cada um destes grupos executa uma tarefa diferente no projeto. Um grupo estará encarregado de obter os requisitos funcionais e temporais do projeto, outro será responsável por definir a arquitetura do sistema, dados os requisitos; outro, ainda, será encarregado de escolher que tecnologia será utilizada. Portanto, é claro que cada grupo tem uma visão diferente do mesmo sistema. Enquanto o primeiro grupo enxerga, apenas, o comportamento do sistema, o segundo grupo vê o sistema como um conjunto de elementos funcionais que devem ser interconectados de forma que o comportamento do sistema seja preservado, e o terceiro grupo se concentra na disposição dos elementos da arquitetura proposta, assim como qual tecnologia deverá ser utilizada. Assim, são necessárias várias representações de um mesmo sistema, para as diferentes partes envolvidas no projeto. Estas representações são, comumente, chamadas de domínios. Sistemas digitais podem ser descritos em três domínios: comportamental, estrutural e físico.

O domínio comportamental, como o próprio nome diz, descreve o comportamento do sistema. A descrição pode ser vista como uma caixa preta, onde se é especificado, apenas, o resultado esperado diante dos valores de entrada. Neste domínio, nenhum detalhe de implementação é dado. Ou seja, este domínio informa o que deve ser feito e não como. O domínio estrutural, ao contrário, define o que está dentro da caixa preta, isto é, a implementação. Neste domínio, descreve-se que elementos funcionais devem ser utilizados, assim como, também, como estes elementos devem estar interconectados. Embora, a partir de uma descrição estrutural, pode-se obter o comportamento do sistema, neste domínio a funcionalidade do sistema não é expressa explicitamente. O domínio físico, descreve as características físicas dos componentes descritos no domínio estrutural. Uma descrição física expressa a dimensão e a localização de cada componente no chip. Enquanto uma descrição estrutural estabelece a interconexão entre os componentes, uma descrição física define a relação espacial entre os componentes interconectados, estabelecendo o peso, tamanho, dissipação de calor, consumo de potência de cada componente e a posição de cada pino do chip.

Cada domínio pode ter vários níveis de abstração. O nível de abstração de um domínio estabelece qual a complexidade do elemento que se irá trabalhar dentro do domínio. Por exemplo, no domínio estrutural pode-se trabalhar a nível de transistor ou a nível de flip-flops. Neste exemplo, no segundo caso trabalha-se com um nível de abstração maior do que no primeiro caso. Portanto, trabalhar em um determinado nível de abstração corresponde a granularidade com que se deseja trabalhar. Quanto maior a granularidade, menor o nível de abstração, e vice-versa. A Figura 11, conhecida como diagrama Y, mostra a relação entre os diferentes domínios e seus vários níveis de abstração. Os eixos representam os três domínios que convergem para um ponto comum. Dentro de cada domínio, os elementos do sistema são descritos em diferentes níveis de abstração, Estes níveis são representados por pontos ao longo dos três eixos, de tal forma que o crescimento do nível de

abstração se dá do centro para a periferia dos eixos. Assim, no domínio estrutural o sistema pode ser descrito em vários níveis, o nível mais baixo seria o de circuito, onde se usam transistores, enquanto que o nível mais alto de abstração seria o de sistema, onde se usam processadores e memórias.

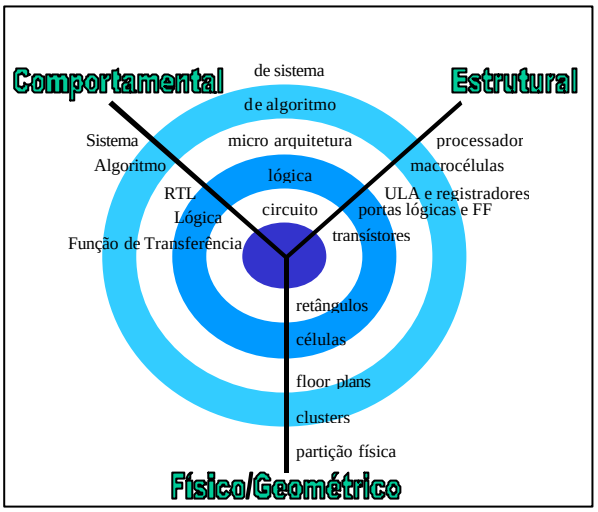


Figura 11 - Diagrama Y

O diagrama Y, também, é bastante útil para representar as diferentes tarefas durante projetos de sistemas digitais. Transições ao longo dos eixos ou de um eixo para outro representam tarefas e têm denominações específicas. A Figura 12 ilustra tais transições. Uma transição do domínio estrutural para o domínio físico é chamada geração, e extração se for na direção oposta. Do domínio comportamental para o estrutural, tem-se uma transição denominada de síntese, e na direção oposta, análise. Normalmente, esta última é utilizada na fase de verificação do sistema. As transições que ocorrem no mesmo eixo são: abstração, quando move-se para a periferia do eixo; refinamento, que é o movimento em direção ao centro do eixo; e, por último, otimização, que é um loop para o mesmo ponto.

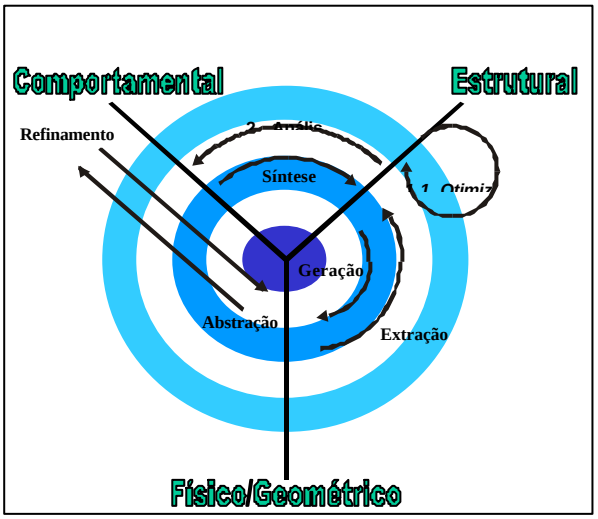


Figura 12- Transições no diagrama Y

O aumento espantoso da complexidade de sistemas digitais, onde o número de transistores por chip passa de um milhão atualmente, tornou o projeto de sistemas digitais a nível de transistor ou portas lógicas muito difícil. Assim, a necessidade de níveis de abstração maiores surgiu para facilitar o trabalho do projetista e também permitir uma maior compreensão sobre o sistema a ser projetado. Isto empurrou o desenvolvimento de novas ferramentas que dessem suporte a esses níveis de

abstração, razão pela qual, hoje em dia, tem-se ferramentas que permitem pessoas não familiarizadas com transistores e portas lógicas a desenvolverem sistemas digitais.

O estado da arte em ferramentas de CAD encontra-se a nível de algoritmo. O processo de síntese inclui uma série de passos apresentados na Figura 13. A síntese algorítmica (high-level synthesis) recebe com entrada uma descrição algorítmica e gera uma arquitetura que o implementa, a qual é composta por uma unidade de controle e uma unidade de processamento. A unidade de controle é descrita como uma máquina de estados finito (FSM) e o próximo passo é a síntese à nível RT, a qual realiza a minimização do número de estados e a codificação dos mesmos. A minimização lógica das unidades de controle e processamento e o mapeamento tecnológico, que consiste na escolha de componentes de uma biblioteca para a implementação das duas unidades, são atividades realizadas durante a síntese lógica. A saída gerada neste passo é uma descrição a nível de portas lógicas e flip-flops, que implementa a descrição inicial. A descrição inicial do sistema digital pode ser validada com ferramentas de simulação funcional. Tais ferramentas também são utilizadas durante as várias etapas do processo de síntese de forma a garantir que a funcionalidade não foi alterada

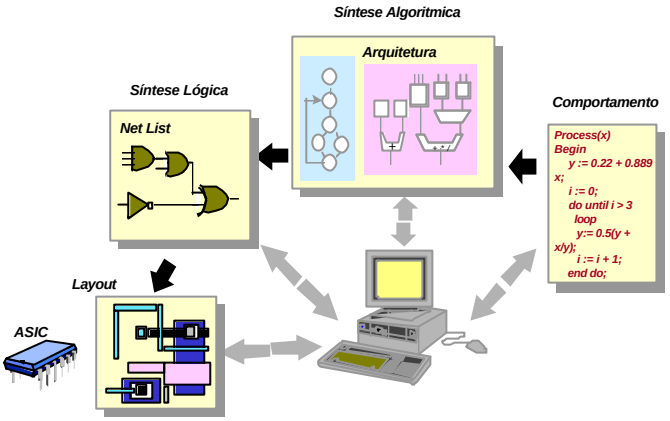


Figura 13. O Processo de Síntese

A seguir será apresentado o sistema PISH de co-design desenvolvido no Centro de Informática.

3.5 A metodologia PISH de hardware/software co-design

O sistema PISH usa occam como mecanismo de especificação e compreende todas as fases de projeto como ilustrado na Figura 14.

A especificação inicial é transformada num conjunto de processos, os quais são agrupados em subconjuntos. Um destes subconjuntos deverá ser implementado em software enquanto os outros serão implementados em hardware. Adicionalmente, o algoritmo de particionamento resulta processos responsáveis pela comunicação entre estes subconjuntos. O particionamento é realizado segundo uma estratégia que permite a verificação formal de que a descrição do sistema particionado possui a mesma semântica que a especificação inicial.

Após a etapa de particionamento, processos a serem implementados em hardware são sintetizados, enquanto processos de software são compilados. Os processos de comunicação também são implementados em hardware e software, conforme um protocolo pré-estabelecido de comunicação. A geração de um protótipo do sistema em hardware e software é obtida usando o ambiente Chameleon. Durante a validação do protótipo, o particionamento pode ser executado novamente, caso as restrições de projeto não estejam sendo satisfeitas. Uma vez que o sistema particionado foi validado o mesmo poderá ser implementado usando vários circuitos (CPU + ASIC´s) ou como um único ASIC.

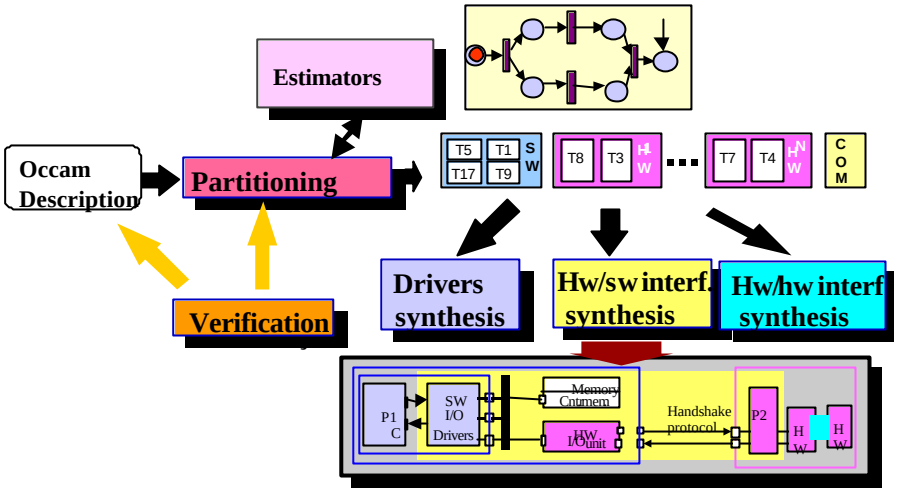


Figura 14– O Sistema PISH de Co-design

3.5.1 Particionamento no sistema PISH

Diferentemente dos métodos mencionados, o método de particionamento do sistema PISH enfatiza a corretude do sistema particionado e gera os processos de comunicação durante o particionamento do sistema segundo uma estratégia que pode ser provada correta por construção.

O algoritmo de particionamento recebe como entrada uma descrição à nível de processos em occam, uma linguagem de especificação que permite a descrição de concorrência de forma explícita e que é baseada em leis algébricas bem definidas. De forma a preservar a semântica da descrição original, o particionamento consiste em transformar a descrição inicial numa descrição particionada, que reflete a arquitetura alvo e que é semanticamente equivalente à descrição inicial. A transformação é obtida pela aplicação de regras de re-escrita, as quais garantem a preservação da semântica. O processo de transformação agrupa processos de forma a serializar os mesmos ou não, segundo o resultado da aplicação de um algoritmo de particionamento baseado em técnicas de clustering. Atualmente, o algoritmo de particionamento considera como arquitetura alvo uma arquitetura contendo um componente de software (microprocessador ou microcontrolador) e mais de um componente de hardware. A estratégia utilizada no método de particionamento pode ser vista na Figura 15.

Inicialmente a descrição original é transformada em uma outra descrição, onde todos os processos são concorrentes e simples[2]. A granularidade do particionamento pode, no entanto, ser definida pelo usuário através da utilização do construtor box. Processos inseridos neste tipo de construtor são considerados objetos funcionais. Esta transformação inicial, realizada na etapa de splitting, é obtida pela aplicação de um conjunto finito de regras de re-escrita definidas nos trabalhos de Silva. O objetivo básico desta transformação inicial é “separar” todos os processos de forma a se ter mais flexibilidade no agrupamento dos mesmos em partições. Como todos os processos passam a ser concorrentes, comunicação é introduzida entre processos com dependência de dados.

Após a etapa do splitting, o conjunto de processos é dividido em partições através de um algoritmo baseado em clustering hierárquico. A função de closeness considera como critérios a similaridade funcional e de concorrência entre processos bem como o custo de comunicação entre os mesmos. A função objetivo considerada na formação dos clusters (isto é, durante o posicionamento da linha de corte na árvore) incorpora as métricas de área de hardware, área de armazenamento para programas e dados, retardo, custo de comunicação e reutilização de recursos. Estimadores baseados em redes de Petri hierárquicas foram desenvolvidos de forma a permitir estimativas mais precisas.

---

[2] Um processo simples contem apenas um if, atribuição, os quais podem estar inseridos num replicador sequencial ou paralelo.

Tais técnicas permitem tanto a estimativa de métricas dependentes da tecnologia de implementação tais como retardo, número de unidades funcionais e área, bem como o cálculo de métricas independentes da tecnologia de implementação tais como custo de comunicação, exclusão mútua, balanceamento de carga entre processadores. O algoritmo de clustering é realizado em dois estágios: no primeiro estágio agrupa-se processos segundo suas similaridades enquanto que no segundo estágio considera-se compartilhamento de recursos e implementações em pipeline.

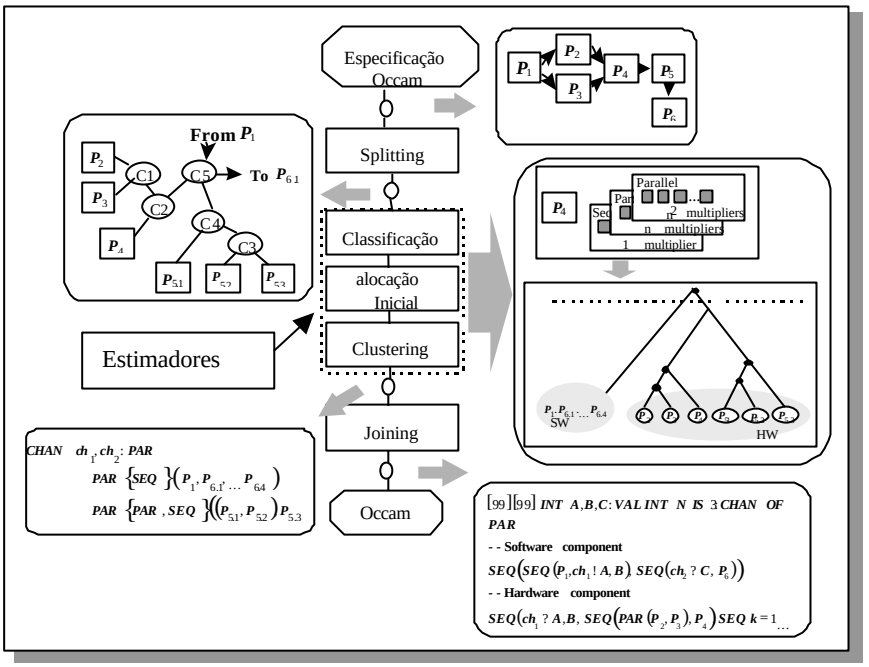


Figura 15– A estratégia de particionamento do sistema PISH

O resultado do algoritmo de clustering é uma descrição em occam, que reflete a seqüência de clusters com informação adicional sobre a serialização ou não de processos pertencentes a um mesmo cluster. A descrição correta final do sistema particionado é obtida após a etapa de joining, a descrição obtida na etapa do splitting é transformada mais uma vez através de aplicação de regras, de forma a refletir o resultado do particionamento. As transformações realizadas nesta fase incluem a serialização de processos, a reconstrução de estruturas aninhadas e a eliminação de comunicação. Além de processos a serem implementados em hardware e processos de software, a descrição obtida após a fase de joining inclui os processos de comunicação, o que facilita a geração automática de interfaces na etapa de síntese. Nesta abordagem processos de software e de hardware podem ser executados em paralelo. Como mencionado, vários ferramentas e metodologias para hardware/software co-design tem sido publicadas recentemente. Os vários ambientes diferem basicamente no mecanismo de descrição utilizado, da técnica empregada para realização do particionamento em hardware e software e nas técnicas para implementação e avaliação do sistema particionado. No sistema VULCAN, um sistema de particionamento orientado para hardware, a linguagem de entrada é HardwareC e o sistema tenta mover gradualmente hardware para software. O sistema COSYMA considera uma especificação do sistema em C*, um super-conjunto da linguagem C. O método utilizado é orientado para software e a especificação inicial é traduzida num formato interno baseado em grafos, o qual será utilizado durante o particionamento. Ambas as ferramentas necessitam ser acrescidas com um sistema de prototipação e de co-simulação. Na técnica proposta e SpecSyn, o particionamento é considerado como três suproblemas: particionamento da funcionalidade nos componentes do sistema, mapeamento das variáveis na memória e o mapeamento da comunicação para os barramentos disponíveis.

A metodologia proposta no sistema PISH suporta o particionamento com base nas diversas alternativas de implementação de cada processo.

. Estas alternativas incluem não apenas implementação por software ou hardware, mas também implementações distintas em hardware com diferentes graus de paralelismo. Outra vantagem da metodologia proposta consiste na disponibilidade de um protótipo do sistema particionado logo após a etapa de particionamento. Este protótipo virtual já inclui os módulos de comunicação, os quais são gerados automaticamente durante o processo de clustering. O uso de técnicas de reescrita na transformação da descrição original até a obtenção da descrição particionada garantem a corretude do particionamento o que consiste um outro diferencial da metodologia proposta.

3.5.2 Co-síntese no sistema PISH

Uma visão geral da metodologia de co-síntese, desenvolvida no contexto do sistema PISH pode ser vista na Figura 16. Esta metodologia considera uma arquitetura pré-definida composta de um único componente de hardware porém com o fluxo de controle distribuído. A partir das descrições em occam dos processos a serem implementados em hardware, dos processos a serem implementados em software e dos processos de comunicação são geradas descrições de FSM´s concorrentes, que representam os processos acima citados. As FSM´s são descritas usando Redes de Petri, o formalismo usado no contexto do sistema PISH. A representação de processos paralelos em occam como FSM´s concorrentes facilita a geração de código C e VHDL.

A partir da descrição como FSM´s concorrentes código C e VHDL é gerado automaticamente. Adicionalmente código C e VHDL, que implementa a interface entre hardware e software, bem como a interface entre processos em hardware, pode ser gerada automaticamente. A geração automática de VHDL é realizada utilizando a plataforma HardWWWired, a qual suporta projetos em VHDL na WEB.

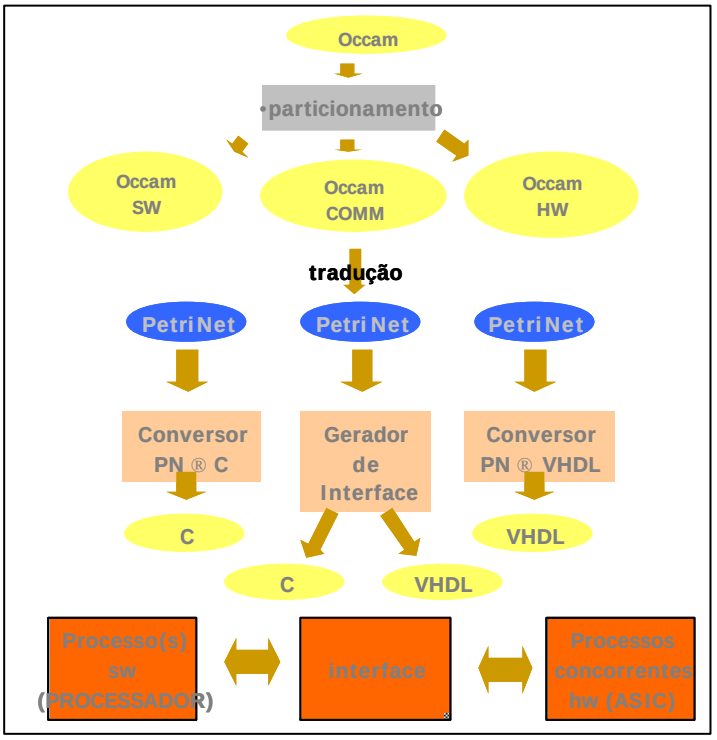


Figura 16- Uma metodologia para geração automática de Interfaces

Para a geração automática de interfaces foi desenvolvido um modelo de interface que inclui blocos a serem implementados em hardware e blocos implementados em software. Para facilitar a geração automática do mesmo, os blocos de software e de hardware possuem a mesma interface e funcionalidade, a única diferença é que os blocos de software são funções em C, enquanto que os blocos de hardware são componentes descritos em VHDL. O modelo de interface desenvolvido é

baseado em níveis de forma que somente os níveis mais inferiores são dependentes do processador utilizado. Para implementar a semântica de comunicação síncrona baseada em canais foi definido um modelo de canal, cuja interface pode ser vista na Figura 17. A implementação em hardware de um canal consiste em componentes descritos em VHDL e a implementação em software consiste de funções em C.

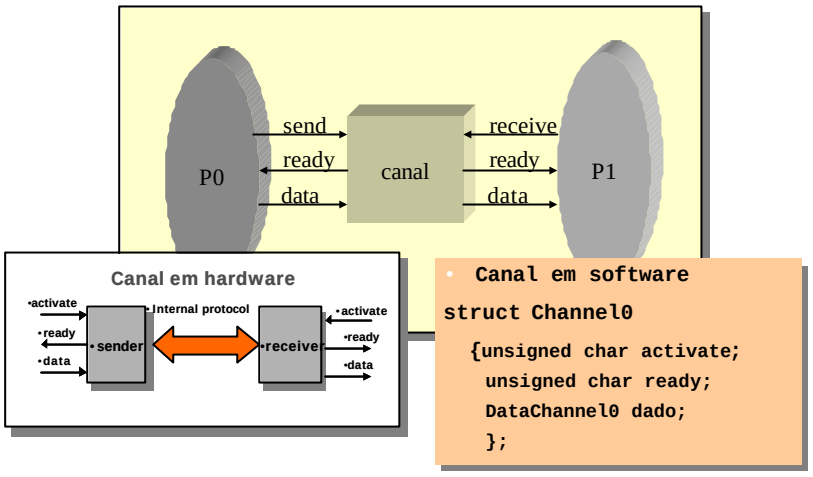


Figura 17 – Modelo de canal e sua implementação em hardware e software respectivamente

A interface entre o hardware e software consiste basicamente de dois blocos, o prcs_unit e a i/o_unit, conforme mostrado na Figura 18.

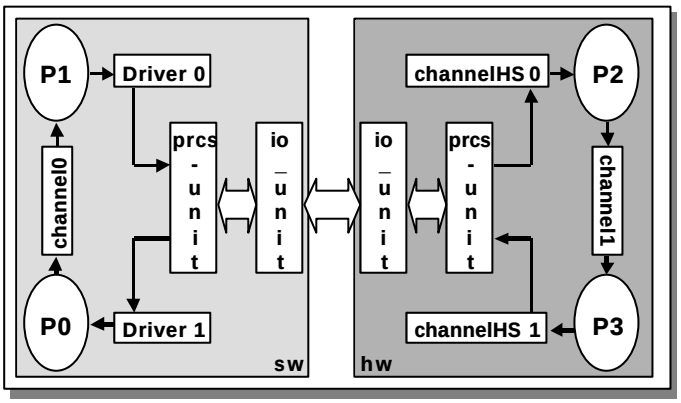


Figura 18 – Interface hardware/software

O primeiro bloco é responsável pelo escalonamento das operações de envio e recebimento nos processos uma vez que somente uma operação pode ocorrer por vez. Adicionalmente, este módulo compatibiliza o tamanho físico do meio de interconexão entre processador e componente de hardware e o tamanho do dado a ser transmitido, através da utilização de buffers. O bloco denominado I/O_unit é o responsável pela transmissão dos dados no meio físico e sua implementação depende do processador sendo utilizado. Neste projeto implementamos a interface para o processador 8051. Ambos os blocos possuem implementações em hardware e em software, dadas por descrições em VHDL e C, respectivamente. A metodologia de geração de interfaces desenvolvida suporta, adicionalmente, a geração de interfaces entre processos em hardware. Neste caso são introduzidos estados nas FSM´s representando os processos que ativam os canais em VHDL.

4. O futuro dos sistemas embarcados

A interconectividade e interoperabilididade são dois fatores-chave que vêm ganhando importância em relação aos sistemas embarcados. Com o advento dos computadores pessoais e das redes de interconexão, tanto locais quanto globais, como a Internet, os sistemas embarcados estão deixando de serem projetados como dispositivos isolados para se tornarem parte dessas redes de modo a facilitar a troca de informações para atingir objetivos comuns ou simplesmente facilitar seu controle à distância por seus usuários.

Hoje existem inúmeras pesquisas acadêmicas e empresariais tanto de dispositivos como padrões que garantam essa interoperabilidade. O que se deseja é que aparelhos de diversos fabricantes possam conversar entre si. Por exemplo, é interessante que o aparelho de som ou a televisão recebam informações do telefone para baixar o volume automaticamente caso o último esteja recebendo uma ligação. Ou ainda que todos os aparelhos domésticos possam ser controlados por um único controle remoto.

Alguns padrões já começaram a serem estabelecidos, como o IrDA e Bluetooth, que mencionamos na Seção 2.3.1. Entretanto, muito ainda precisa ser feito para que os equipamentos possam realmente funcionar em conjunto. Técnicas de inteligência artificial terão que ser utilizadas nestes equipamentos de modo a que estes possam tomar decisões e não fiquem passivos, restritos apenas a ações de controle do usuário. No aspecto de facilitar o controle destes sistemas por seus usuários, alguns sistemas embarcados estão sendo desenvolvidos especificamente para adaptar melhor os computadores aos meios de comunicação dos usuários, como voz, imagem, etc, e não vice-versa, ou seja, o usuário se adaptar ao computador. Afinal, o teclado, por exemplo, é uma interface completamente artificial entre o homem e a máquina. Já é possível encontrar no mercado dispositivos de reconhecimento limitado de voz (até 32 palavras) por menos de US$5,00.

Do ponto de vista de tecnologia, os desafios também são grandes. Os objetivos de projeto estão ficando cada vez mais restritos em termos de tamanho, velocidade de processamento, potência consumida (afinal, muitos dispositivos trabalham com bateria), tempo de desenvolvimento, custo, flexibilidade para mudanças, tolerância a falhas, entre outros.

As seções a seguir apresentam algumas aplicações atuais e futuras dos sistemas embarcados.

4.1 Automóveis

Em 1998, os sistemas embarcados representavam cerca de 10 a 15% do valor de um veículo. A previsão é que chegue a 40% em menos de 5 anos, ou ainda mais para carros elétricos ou de luxo. Carros comuns em 1990 tinham 14 microprocessadores e este ano terão 35. Esta tendência também é comprovada em outras áreas. Por exemplo, helicópteros e aviões militares têm cerca de 60% de seu valor nos sistemas embarcados e apenas 40% em mecânica.

Existe também uma rede de computadores especial para carros, chamada Computer Automotive Network – CAN. Já existem propostas para controlar os dispositivos elétricos, como lâmpadas, por meio de uma rede como esta, de modo a diminuir a quantidade de fios e, consequentemente, o peso dos carros.

Imagine que os carros hoje têm freios ABS (Anti-Break System), ignição eletrônica, injeção eletrônica, suspensão ativa, computador de bordo, painel digital (com conta-giros, velocímetro, marcador de combustível, etc), aparelho de som digital, ar-condicionado inteligente (que desliga se o motor for requisitado para uma ultrapassagem, por exemplo), controle de tração, air-bag, alarme contra roubo, entre outros, e todos estes equipamentos são computadorizados.

Equipamentos mais sofisticados já vêm sendo implantados, como sistemas GPS (*Global Positioning System* ou Sistema de Posicionamento Global) que, integrados com mapas digitais, mostram ao motorista a sua localização exata e propo~em rotas para o destino desejado. Existem estudos no sentido de fazer com que estes equipamentos possam receber informações sobre o trânsito no percurso desejado de modo a que possam propor rotas alternativas caso haja algum engarrafamento no caminho original.

Além disso, estuda-se o uso de carros autônomos capazes de se auto-guiar, os quais aliados a rodovias inteligentes que trocam informações entre os diversos carros, ajuda-os a controlar a

distância entre eles e a velocidade, de modo a que se comportem efetivamente como um trem. Cada carro se comporta como um vagão e são controlados automaticamente por computador. Espera-se com isso poder aumentar o número de veículos nas estradas.

4.2 Wearable Computers ou Ectocomputadores

Wearable computers são aqueles que levamos conosco de uma forma natural, sem causar incômodos quer pelo peso, forma, ou posição de uso. Para isso eles devem poder ser usados quando estamos em movimento e com pelo menos uma das mãos livres. Eles devem estar sempre disponíveis, ou seja, devem estar num estado que garanta pronta resposta a qualquer instante. Além disso, embora a tendência seja fazer com estes equipamentos tenham algum poder de decisão (inteligência artificial), eles devem sempre permitir o controle do usuário.

No Brasil estes sistemas vêm sendo chamados de computadores vestíveis. Entretanto a palavra vestível não determina bem esta classe de computadores. Vestível lembra roupa e não inclui implantes de computadores na pele e no corpo, óculos especiais, ou agendas eletrônicas. Um termo mais apropriado poderia ser ectocomputador, ou seja, computador do corpo. Neste artigo utilizaremos este termo.

A interação do usuário com o ectocomputador é diferente da que temos com um desktop: usuários de desktops interagem diretamente com o computador enquanto os ectocomputadores auxiliam seus usuários na interação com o ambiente. O fato de carregarmos estes computadores conosco, permite uma interação muito maior destes com o contexto em que estamos inseridos. Conhecendo melhor este contexto, o ectocomputador pode definir o que fazer sem ordem explícita do usuário. Para permitir esta interação é comum que estes computadores tenham sensores como câmeras, microfones, GPS, e dispositivos para comunicação (acesso a telefone e Internet), fones de ouvido e displays miniatura.

Os ectocomputadores já estão sendo testados. Na guerra da Bósnia, por exemplo, pára-quedistas da OTAN usaram computadores acoplados aos cintos compostos de uma placa-mãe dobrável com um processador 586, um display sensível ao toque, fones de ouvido e um microfone, além de um software que fazia a tradução automática de inglês para croata, francês, russo, etc. Também já foi feito um computador que traduz a linguagem dos sinais, usada pelos surdos-mudos, para texto, com uma taxa de acerto de 98%.

Várias companhias estão desenvolvendo ou utilizando os ectocomputadores. A Boeing, por exemplo, é uma grande usuária deste computadores. O Navigator 2, desenvolvido pela Carnegie Mellon University, está sendo usado pela Boeing e pela força aérea americana para agilizar a inspeção de defeitos em aeronaves. O inspetor pode examinar a aeronave e comparar com esquemas e informações que lhe são apresentadas visualmente no micro-display. Além disso, informações sobre defeitos são armazenadas diretamente no ectocomputador. Tudo isto reduziu em 20% o tempo necessário para a inspeção e entrada de dados no computador.

Figura 19. Wearable computers na linha de montagem e manutenção da Boeing

Na área de saúde existem muitas pesquisas sendo feitas mundialmente. No Centro de Informática da UFPE estamos desenvolvendo, juntamente com o Laboratório de Imunopatologia Keiso Asami (LIKA - UFPE), um sistema com biosensores, os quais serão acoplados a microcomputadores a bateria, num projeto financiado pela FACEPE. Assim o paciente poderá ficar em casa ou no trabalho tendo suas condições clínicas constantemente monitoradas, com uma qualidade de vida muito melhor e, possivelmente, menor custo para os hospitais, em vez de ficar numa cama de hospital coberto de equipamentos. Os dados obtidos pelos biosensores serão transmitidos sem fio pelo ectocomputador para um computador conectado à Internet de modo que o médico responsável possa acompanhar a evolução do quadro clínico ou ser avisado automaticamente em caso de urgência. Evidentemente este monitoramento remoto não se aplica a qualquer caso mas achamos que muitos pacientes poderão se beneficiar de tal tecnologia.

Um outro exemplo temos a NASA, que está interessada em cirurgias vírtuo/reais para atender possíveis problemas médicos na estação espacial que está sendo construída. Neste caso, o cirurgião usaria óculos especiais que podem mostrar imagens geradas por computador sobre a imagem real do paciente, permitindo que este realize procedimentos cirúrgicos os quais não dominasse totalmente. Adesivos inteligentes estão sendo desenvolvidos para injetar medicamentos com base nas necessidades do paciente no momento, as quais são monitoradas com biosensores.

Os maiores desafios tecnológicos no desenvolvimento destes computadores está justamente na percepção do contexto e na tomada de decisões, que envolvem as áreas de processamento de sinais (som, imagem, cheiro, sinais biométricos, etc.) e de inteligência artificial. Também se está estudando como estabelecer a comunicação dos vários ectocomputadores que uma pessoa estiver usando. Para isso está sendo pesquisada a criação da *Personal Area Network* (PAN), que utiliza o próprio corpo do usuário como meio de comunicação. Existe também o problema de fornecimento de energia para estes computadores. Já foram criados sapatos capazes de gerar energia ao andar. Esta energia seria distribuída pelo corpo do usuário de forma semelhante à comunicação de dados usada na PAN.

Os ectocomputadores nos permitirão acesso imediato e constante a informações e outros recursos, como edição de texto, transferência de dados, etc que poderão ser usados para lazer e trabalho a qualquer instante. Esses recursos poderão ser usados para a melhoria de nossa qualidade de vida mas também podem torná-la um inferno. Nossas vidas privadas podem passar a ser menos privadas, nossos momentos de lazer podem ser extensões do trabalho. Como toda nova tecnologia devemos tomar cuidado com sua utilização para que seja usada para nos auxiliar.

5. Referências

[1] Altenbernd, P. "On the false path problem in hard real-time programs". Proceedings of the Euromicro Workshop on Real-Time Systems. Los Alamitos: IEEE Computer Society Press, June 1996, pp. 102-107.

[2] Araújo, C. C. and Barros, E. An Approach for Interface Generation in the PISH Co-design System. In Proceedings of XII Brazilian Symposium on Integrated Circuits and System Design, Natal – Brasil, 10/1999, Páginas: 66 à 69

[3] Araújo, C. C. and Barros, E. Automatic Interface Generation among VHDL Processes in Hardware/Software Co-Design. In Proceedings of the FDL99 Forum on Design Languages, Lion – Franca, 8/1999

[4] Barros E., Cavalcante S. – Uma Introdução a Hardware/Software Co-design – Anais dos Cursos da Jornada de Atualização Científica JAI da SBC, 1998

[5] Barros, E.; and Rosenstiel, W. "A Clustering Approach to Support Hardware/Software Partitioning". In: K. Buchenrieder, and J. Rozenblit (eds.), Computer Aided Software/Hardware Engineering. IEEE Press, 1994.

[6] Barros, E.; and Sampaio, A. "Towards provably correct hardware/software partitioning using occam". Proceedings of the 3rd International Workshop on Hardware/Software Codesign. Los Alamitos: IEEE, September 1994, pp. 210-217.

[7] Bass, T.A. "Dress code". Wired Magazine, 6.04 Abril 1998, pp.162-187.

[8] Bergamaschi, R. "Tutorial: Behavioral synthesis - an overview". Proceedings of the VLSI'97 - IX IFIP International Conference on Very Large Scale Integration. Porto Alegre: UFRGS, 1997.

[9] Borriello, G.; Chou, P.; and Ortega, R. "Embedded System Co-design: Towards Portability and Rapid Integration". In: M. Sami, and G. DeMicheli (eds.), Hardware/Software Co-design. Kluwer Academic Publishers, 1996, pp. 243-264.

[10] Camposano, R.; and Brayton, R.K. "Partitioning before logic synthesis". Proceedings of the International Conference on Computer Aided Design. 1987.

[11] Camposano, R.; and Wilberg, J. "Embedded System Design". Design Automation for Embedded Systems, 1(1/2): 5-50, 1996.

[12] Cavalcante, S.V. "A hardware-software codesign environment for real-time embedded applications". In: C. Muller-Schloer, F. Geerinckx, B. Stanford-Smith, and R. van Riet (eds.), Embedded microprocessor systems. Amsterdam: IOS Press, 1996, pp. 253-262.

[13] Cavalcante, S.V. "A hardware-software co-design system for embedded real-time applications". Ph.D. Thesis, University of Newcastle upon Tyne, 1997.

[14] Cavalcante, S.V.; and Kinniment, D.J. "Time Wizard: a design and assessment tool for real-time applications". Proceedings of the Euromicro Workshop on Real-Time Systems. Los Alamitos: IEEE Computer Society Press, June 1996, pp. 22-27.

[15] Chang, W.T.; Kalavade, A.; and Lee, E.A. "Effective heterogeneous design and co-simulation". In: G. DeMicheli, and M. Sami (eds.), Hardware/software co-design. Dordrecht: Kluwer Academic Publishers, 1996, pp. 187-212.

[16] Chiodo, M.; Engels, D.; Hsieh, H.C.; Giusto, P.; Jurecska, A.; Lavagno, L.; Suzuki, K.; and SangiovanniVincentelli, A. "A case study in computer-aided co-design of embedded controllers". Design Automation for Embedded Systems, 1(1/2): 51-67, 1996.

[17] Chiodo, M.; Giusto, P.; Jurecska, A.; Hsieh, H.C.; SangiovanniVincentelli, A.; and Lavagno, L. "Hardware-software codesign of embedded systems". IEEE Micro, 14(4): 26-36, 1994.

[18] Chou, P.; Ortega, R.B.; and Borriello, G. "The Chinook Hardware/Software Co-synthesis System". Proceedings of the 8th International Symposium on System Synthesis. 1995.

[19] DeMicheli, G. "Hardware/software co-design: application domains and design technologies". In: G. DeMicheli, and M. Sami (eds.), Hardware/software co-design. Dordrecht: Kluwer Academic Publishers, 1996, pp. 1-28.

[20] Dewey, A. "VHSIC Hardware Description Language (VHDL) development program". Proceedings of the 20th Design Automation Conference. IEEE, 1983, pp. 625-628.

[21] Drusinsky,D. and D. Harel, “Using Statecharts for hardware description and synthesis”. IEEE Transactions on Computer-Aided Design, vol. 8, no. 7, pp. 798-807, 1989.

[22] Eles, P.; Peng, Z.; Kuchcinski, K.; and Doboli, A. "Hardware/software partitioning of VHDL system specifications". Proceedings of the European Design Automation Conference with EURO-VHDL'96. Los Alamitos: IEEE, September 1996, pp. 434-439.

[23] Eles, P.; Peng, Z.; Kuchcinski, K.; and Doboli, A. "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search". Design Automation for Embedded Systems, 2(1): 5-32, 1997.

[24] Ericsson Microelectronics. “Industrial Circuits Application Notes: Stepper motor basics”, http://www.ericsson.se/microe/apn_ind.html

[25] Ernst, R.; Henkel, J.; and Benner, T. "Hardware/software cosynthesis for microcontrollers". IEEE Design & Test of Computers, 10(4): 64-75, 1993.

[26] Ernst, R.; Henkel, J.; Benner, T.; Ye, W.; Holtmann, U.; Herrmann, D.; and Trawny, M. "COSYMA environment for hardware/software cosynthesis of small embedded systems". Microprocessors and Microsystems, 20(3): 159-166, 1996.

[27] Gajski, D.D.; Vahid, F.; Narayan, S.; and Gong, J., Specification and Design of Embedded Systems, first ed. Prentice Hall, 1994.

[28] Garey, M.R.; and Johnson, D.S., Computers and intractability: a guide to the theory of NP-completeness. San Francisco, USA: Freeman, 1979.

[29] Grehan, R., Moote, R., Cyliac I., Real-Time Programming: A Guide to 32-Bit Embedded Development. Addison Wesley, 1998.

[30] Gupta, R.; and DeMicheli, G. "Hardware-software co-synthesis for digital systems". IEEE Design&test of Computers: 29-41, 1993.

[31] Gupta, R.; and DeMicheli, G. "System-level Synthesis using re-programmable components". Proceedings of the European Conference on Design Automation. 1992, pp. 2-7.

[32] Gupta, R.; Coelho, C.N.; and DeMicheli, G. "Synthesis and simulation of digital systems containing interacting hardware and software components". Proceedings of the 29th Design Automation Conference. 1992.

[33] Gupta, R.K.; and DeMicheli, G. "Co-synthesis approach to embedded system design automation". Design Automation for Embedded Systems, 1(1-2): 69-120, 1996.

[34] Harel, D. "Statecharts: A visual formalism for complex systems". Science of Computer Programming. North Holland, 1987.

[35] Hu, X.; and Amborsia, J.G. "Hardware/Software Partitioning for Real-Time Embedded Systems". Design Automation for Embedded Systems, 2(3/4): 339-358, 1997.

[36] Ismail, T.B.; Abid, M.; and Jerraya, A. "COSMOS: a CoDesign approach for communicating systems". Proceedings of the 3rd International Workshop on Hardware/Software Codesign. Los Alamitos: IEEE, September 1994, pp. 17-24.

[37] Jantsch, A.; Ellervee, P.; Oberg, J.; Hemani, A.; and Tenhunen, H. "Hardware/software partitioning and minimizing memory interface traffic". Proceedings of the European Design Automation Conference. Los Alamitos: IEEE, September 1994, pp. 226-231.

[38] Jerraya, A.; Daveau, J.M.; Marchioro, G.; Valderrama, C.; Romdhani, M.; and Ismail, T.B. "Tutorial: Hardware software codesign". Proceedings of the VLSI'97 - IX IFIP International Conference on Very Large Scale Integration. Porto Alegre: UFRGS, 1997.

[39] Johnson,S. C., “Hierarchical clustering schemes”. Psychometrika, vol. pp. 241-254, 1967.

[40] Kalavade, A.; and Lee, E.A. "Manifestations of heterogeneity in hardware/software codesign". Proceedings of the 31st Design Automation Conference. Piscataway: IEEE, June 1994, pp. 437-438.

[41] Kalavade, A.; and Lee, E.A. "The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling and Implementation-bin Selection". Design Automation for Embedded Systems, 2(2): 125-164, 1997.

[42] Kernighan, B.W.; and Lin, S. "An efficient heuristic procedure for partitioning graphs". Bell System Technical Journal, 1970.

[43] Kirkpatrick,S. C.; D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing”. Science, vol. 220, no. 4589, pp. 671-680, 1983.

[44] Knudsen, P.V.; and Madsen, J. "Communication Estimation for Hardware/Software Codesign". Proceedings of the International Workshop in Hardware/Sotware Co-design - CODES 1998. 1998.

[45] Krishnamurthy,B., “An improved min-cut algorithm for partitioning VLSI networks”. IEEE Transactions on Computer, vol. 1984.

[46] Ku, D.; and DeMicheli, G. "HardwareC: a language for hardware design". Technical Report CSL-TR-90-419, Computer System Laboratory, Stanford University, Stanford, USA: 1990.

[47] Lagnese, E.D. "Architectural Partitioning for System Level Design of Integrated Circuits". PhD Thesis, Carnegie Mellon University, 1989.

[48] Lengauer, T., Combinatorial Algorithms for Integrated Circuit Layout. England: John Wiley and Sons, 1990.

[49] Liem, C.; and Paulin, P. "Compilation Techniques and Tools for Embedded processor Architectures". In: J. Staunstrup, and W. Wolf (eds.), Hardware/Software Co-design: Principles and Practice. Amsterdan: Kluwer Academic Publishers, 1997, pp. 149-192.

[50] Liem,C.; F. Nacabal; C. Valderrama; P. G. Paulin; A. Jerraya. Co-simulation and Software Compilation Methodologies for the System-on-a-Chip in Multimedia. IEEE Design & Test of Computers, special issue on "Design, Test & ECAD in Europe", Summer 1997.

[51] Maciel ,P. R; Barros, E. and Rosenstiel, W. A Petri Net Model for Hardware/Software Co-design, in Journal "Design Automation of Embedded Systems", Kluwer Academic USA, Volume: 4, Número: 4, Páginas: 243 à 310

[52] Maciel, P. , Barros, E. and Rosenstiel, W. A Petri Net Based Approach for Estimating Area in Hardware/Software Codesign. In Proceedings of High Performance Computing '99, San Diego, California, USA, 4/1999, Páginas: 425 à 432

[53] Maciel, P. Barros, E. and Rosenstiel, W . In Proceedings of International Conference on CAD/CAM Robotics & Factories of the Future, Aguas de Lindoia – SP, 8/1999, Páginas: 17 à 22

[54] Maciel, P. Barros, E. and Rosenstiel, W. A Petri Net Approach to Compute Load Balance in Hardware/Software Codesign. In Proceedings of Advanced Simulation Technologies Conference - High Performance Computing, Boston – USA, 1998, Páginas:360 à 368

[55] Maciel, P.; and Barros, E. "Capturing Time Constraints by Using Petri-nets in the Context of Hardware/Software Codesign". Proceedings of the Proceedings of the 6th International Workshop on Rapid Systems Prototyping, RSP97. IEEE Press, 1996.

[56] Maciel, P.; Barros, E.; and Rosenstiel, W. "A Petri Net Approach to Compute Load Balance in Hardware/Software Codesign". Proceedings of the Conference for Performance Computing. 1998.

[57] Maciel, P.; Barros, E.; and Rosenstiel, W. "A Petri Net-based Approach for Performing the Initial Allocation in Hardware/Software Co-design". Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. 1998.

[58] Maciel, P.; Barros, E.; and Rosenstiel, W. "Computing Communication Cost by Petri Nets for Hardware/Software Codesign". Proceedings of the 7th International Workshop on Rapid Systems Prototyping, RSP97. IEEE, 1997.

[59] Madisetti,V. K. "Rapid Prototyping of Digital Systems: Current Practice, Future Challenges". IEEE Design and Test on Computers, vol. 1996.

[60] Madsen, J.; Grode, J.; and Knudsen, P.V. "Hardware/Software Partitioning using the LYCOS System". In: J. Staunstrup, and W. Wolf (eds.), Hardware/Software Co-design: Principles and Practice. Amsterdan: Kluwer Academic Publishers, 1997, pp. 283-306.

[61] Madsen, J.; Grode, J.; Knudsen, P.V.; Peterson, M.E.; and Haxthausen, A. "LYCOS: the Lyngby Co-Synthesis System". Design Automation for Embedded Systems, 2(2): 195-236, 1997.

[62] Malik, S.; Wolf, W.; Wolfe, A.; Li, Y.; and Yen, T.Y. "Performance analysis of embedded systems". In: G. DeMicheli, and M. Sami (eds.), Hardware/software co-design. Dordrecht: Kluwer Academic Publishers, 1996, pp. 45-74.

[63] Malley,J. H. M., "RASSP: changing paradigm of electronic system design". IEEE Design and Test on Computers, vol. 3, no. 3, pp. 23-31, 1996. IEEE Computer Society.

[64] McFarland, M.C.; Parker, A.C.; and Camposano, R. "Tutorial on high-level synthesis". Proceedings of the 25th ACM/IEEE Design Automation Conference. Piscataway: IEEE, June 1988, pp. 330-336.

[65] McFarland,M. C. and T. J. Kowalski, "Incorporating bottom-up design into hardware synthesis". IEEE Transaction on Computer Aided Design, vol. 1990.

[66] Michel, P.; Lauther, U.; and Duzy, P., The Synthesis Approach to Digital System Design. Kluwer Academic Publishers, 1992.

[67] Murata, T. "Petri nets: properties, analysis and applications". Proceedings of the IEEE, 77(4): 541-580, 1989.

[68] Narayan, S.; and Gajski, D.D. "System clock estimation based on clock slack minimization". Proceedings of the European Design Automation Conference - EURO-VHDL '92. Los Alamitos: IEEE, September 1992, pp. 66-71.

[69] Niemann, R.; and Marwedel, P. "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming". Design Automation for Embedded Systems, 2(2): 165-194, 1997.

[70] Oesterling, A.; Benner, T.; Ernst, R.; Herrmann, D.; Scholz, T.; and Ye, W. "The Cosyma System". In: J. Staunstrup, and W. Wolf (eds.), Hardware/Software Co-design: Principles and Practice. Amsterdan: Kluwer Academic Publishers, 1997, pp. 263-281.

[71] Peng, Z.; and Kuchcinski, K. "An algorithm for partitioning of application specific systems". Proceedings of the European Design Automation Conference. Los Alamitos: IEEE, March 1993, pp. 316-321.

[72] Post, E.R.; Reynolds, M.; Gray, M.; Paradiso, J.; and Gershenfeld, N. “Intrabody Buses for Data and Power”. 1st International Symposium on Wearable Computers (ISWC '97) Cambridge, Massachusetts, USA. October 13-14, 1997

[73] Pountain, D.; and May, D., A Tutorial Introduction to OCCAM Programming. Inmos, 1987.

[74] Rim, M.; and Jain, R. "Representing conditional branches for high-level synthesis applications". Proceedings of the 29th ACM/IEEE Design Automation Conference. Piscataway: IEEE, June 1992, pp. 106-111.

[75] Rosenstiel, W. "Prototyping and Emulation". In: J. Staunstrup, and W. Wolf (eds.), Hardware/Software Co-design: Principles and Practice. Amsterdan: Kluwer Academic Publishers, 1997, pp. 75-111.

[76] Sangoma Technologies Inc. “RS232, RS422 and V.35 interfaces”. http://www.sangoma.com/signal.htm

[77] Sarmento A. A. M. , Fernandes, J.H. and Barros, E. HardWWWired: Projetando Sistemas Digitais em VHDL na WEB. Anais do Congresso da Sociedade Brasileira de Computação – SBC, Rio de Janeiro, 8/1999, Páginas: 265 à 279

[78] Sarmento A. A. M., Fernandes, J.H. and Barros, E. HardWWWired: Using the Web as Repository of VHDL Components. In Proceedings of FDL99 - Forum on Design Languages, Lion – Franca, 8/1999, Páginas: 271 à 280

[79] Sarmento A. A. M., Fernandes, J.H. and Barros, E. Hardwwwired: Designing VHDL Systems in the WEB. In Proceedings of International Conference on Microeletronics and Packaging, Campinas - SP – Brazil, 8/1999, Páginas: 139 à 144

[80] Silva, L. Sampaio, A., Barros, E. and Yoda, J. An Algebraic Approach to Combining Processes in a Hardware/Software Partitioning Environment. In Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology (AMAST) 98, Amazonia – Brasil, 1/1999, Páginas: 308 à 324

[81] Silva, L.M.; Sampaio, A.; and Barros, E. "A normal form reduction strategy for hardware/software partitioning". Proceedings of the 4th International Symposium Formal Methods Europe FME'97. Sringer Verlag, 1997.

[82] Silva, L.M.; Sampaio, A.; and Barros, E. "Projeto integrado de hardware e software com ênfase em corretude: um estudo de caso". Proceedings of the XII Congresso da Sociedade Brasileira de Computação - SBC. 1997.

[83] Stallings, W., Computer organization and architecture: designing for performance, 4th ed. Prentice Hall, 1996.

[84] Sundance, Product Overview no. SMT219, 1997. Sundance Multiprocessor Technology Ltda. Bucks, UK.

[85] Vahid, F. and Givargis, T, Embedded System Design: A Unified Hardware/Software Approach, Wisley&Sons, 1999.

[86] Vahid, F.; and Gajski, D.D. "Clustering for improved system-level functional partitioning". Proceedings of the 8th International Symposium on System Synthesis. 1995.

[87] Vahid, F.; Narayan, S.; and Gajski, D.D. "SpecCharts: a VHDL front-end for embedded systems". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 14(6): 694-706, 1995.

[88] Valderrama, C.; Changuel, A.; and Jerraya, A.A. "Virtual Prototyping for Modular and Flexible Hardware/Software Systems". Design Automation for Embedded Systems, 2(3/4): 267-282, 1997.

[89] Valderrama, C.; Romdhani, G.; Changuel, A.; and Jerraya, A.A. "Cosmos: A Transformational Co-design Tool for Multiprocessor Architecture". In: J. Staunstrup, and W. Wolf (eds.), Hardware/Software Co-design: Principles and Practice. Amsterdan: Kluwer Academic Publishers, 1997, pp. 307-358.

[90] Vercauteren, S.; and Lin, B. "Hardware/Software Communication and System Integration for Embedded Architectures". Design Automation for Embedded Systems, 2(3/4): 359-382, 1997.

[91] Walker,R. A., “Behavioral transformation for algorithmic level IC design”. IEEE Transactions on Computer-Aided Design, vol. 8, no. 10, pp. 1115-1128, 1989. 0278-0070.

[92] Walkup,E. A; G. Boriello. Automatic Synthesis of Device Drivers for Hardware / Software. Int'l Workshop on Harsdware Software Codesign. Cambridge, IEEE CS Press, October 1993.

[93] Wolf,W. Hardware-Software Codesign of Embedded Systems. Proc. IEEE, Vol. 82, n. 3, pp. 5, September 1993.

[94] Yakovlev, A.V.; Koelmans, A.M.; Semenov, A.; and Kinniment, D.J. "Modelling, analysis and synthesis of asynchronous control circuits using Petri nets". Integration, the VLSI Journal, 21(3): 143-170, 1996.