Projeto de Sistemas Embarcados

Article

1 author:

Marcos Zurita
Universidade Federal do Piauí
6 PUBLICATIONS **24** CITATIONS

SEE PROFILE

Projeto de Sistemas Embarcados

Marcos E. P. V. Zurita
Universidade Federal do Piauí, Curso de Engenharia Elétrica
Campus Universitário Ministro Petrônio Portela - 64049-550 - Teresina – PI
zurita@ufpi.edu.br, www.ufpi.edu.br/zurita

Resumo

A vasta gama de aplicações nas quais os sistemas embarcados estão presentes atualmente requer diferentes técnicas e abordagens para sua elaboração. O conhecimento prévio dessas metodologias e sua correta adoção é de fundamental importância para o sucesso do projeto em condições limitadas de tempo e recursos. Nesse sentido, são abordados neste documento alguns conhecimentos introdutórios fundamentais a respeito de sistemas embarcados e seu projeto, no intuito de guiar os passos iniciais para o aprofundamento no tema.

1 Introdução

A popularização dos microprocessadores, iniciada em meados da década de 80, aliada à contínua redução dos custos da tecnologia, deu início a uma verdadeira revolução na vida cotidiana. Os processadores, antes quase totalmente restritos a computadores corporativos e algumas grandes máquinas, hoje podem ser encontrados praticamente por toda a parte. Automóveis, televisores, brinquedos, máquinas de lavar, telefones, aparelhos de som, relógios de pulso, cartões de banco e até mesmo etiquetas de identificação descartáveis [1] são hoje objetos permeados por microprocessadores. A adoção de microprocessadores fora do universo dos computadores pessoais e corporativos, deu origem aos chamados Sistemas Computacionais Embarcados, ou simplesmente "Sistemas Embarcados".

Segundo uma recente pesquisa realizada pela IDC [2], cerca de 19% de todos os sistemas eletrônicos vendidos hoje no mundo são sistemas embarcados, e este número deverá crescer para 33% até 2015. Eles são atualmente responsáveis por uma receita anual de 1 trilhão de dólares, o que deve dobrar nos próximos 4 anos, quando então serão responsáveis pelo consumo de cerca de 14,5 bilhões de processadores por ano, mais de 2 para cada habitante no mundo.

O rápido crescimento desse setor implica numa demanda igualmente crescente de recursos humanos adequados para poder suplanta-lo. Em outras palavras, há uma imensa oportunidade para projetistas e desenvolvedores na área de projetos de sistemas embarcados. Por outro lado, a complexidade cada vez maior das novas aplicações e o aumento da dinâmica do mercado consumidor tem exigido a melhoria das técnicas e metodologias empregadas em seu projeto de forma e executa-lo no menor espaço de tempo possível. Isto significa que os profissionais que pretendem atuar nessa área precisam estar familiarizados com essas metodologias e técnicas de projeto, bem como as particularidades desse tipo de sistema.

Neste artigo são apresentados alguns conceitos básicos a respeito de sistemas embarcados e uma breve introdução respeito do seu projeto.

2 Sistemas Embarcados

Um Sistema Embarcado (SE), é definido pela IEEE [3] como *"um sistema computacional que faz parte de um sistema maior e implementa alguns dos requerimentos deste sistema"*. Esta definição, estabelecida há mais de duas décadas continua válida, embora a revolução experimentada por esse segmento durante estes últimos anos tenha impulsionado alguns autores a complementarem-na. Nesta linha, Steve Heath o define como sendo *"um sistema baseado em um microprocessador, que é projetado para controlar uma função ou uma gama de funções, e não para ser programado pelo usuário final como ocorre com os PCs"* [4].

Por se tratarem de sistemas computacionais, Arnold Berger [5] prefere descreve-los enumerando as características que os distinguem dos computadores pessoais. São elas:

1. *São dedicados a tarefas específicas enquanto PCs são plataformas genéricas de computação* – Esta característica tem impacto principalmente no poder computacional da máquina. PCs devem poder rodar um grande número de aplicativos, com diferentes exigências de processamento, mantendo um bom desempenho, enquanto SEs precisam realizar apenas uma ou poucas tarefas bem específicas;

2. *São suportados por uma vasta gama de processadores e arquiteturas de processadores* – Atualmente, mais de 40 empresas de semicondutores disputam o mercado de microprocessadores e

microcontroladores [6]. Cada uma delas por sua vez oferece várias soluções distintas. Apenas a Microchip [7], uma das líderes deste segmento, tem hoje uma lista de mais de 500 diferentes microcontroladores. Naturalmente, um grande leque de opções aumenta o grau de liberdade de escolha dos projetistas mas também requer deles conhecimento adequado para encontrar as soluções mais apropriadas a cada projeto;

3. *São geralmente sensíveis aos custos* – a maior parte dos sistemas embarcados possuem bem menos componentes e custam bem menos do que um PC. Evidentemente o impacto nos custos da adição de meia dúzia de componentes é bem mais significativo num sistema que possui uma dúzia deles do que num que possui mais de uma centena. Em alguns sistemas de identificação sem contato (RFId), um simples capacitor pode representar até 20% dos custos totais [8].

4. *Possuem requisitos de tempo-real* – neste aspecto, SEs geralmente podem ser divididos em dois grupos: os que possuem requisitos de "tempo sensível" e os que possuem requisitos de "tempo crítico". As tarefas de tempo crítico são intolerantes a atrasos, devem ser realizadas dentro de um intervalo preciso de tempo ou a tarefa falha. O sistema de acionamento dos *airbags* de um carro são um bom exemplo disso. Nele, alguns milissegundos podem fazer a diferença entre salvar ou não a vida do motorista [9]. Tarefas de tempo sensível, por outro lado, são mais tolerantes. Se a tarefa responsável pelo fechamento da válvula de água de uma máquina de lavar atrasar um ou dois segundos, a roupa será lavada com um pouco mais de água além do necessário, perdendo um pouco da eficiência almejada;

5. *Quando utilizam um sistema operacional, este é quase sempre um RTOS* – A principal diferença entre um Sistema Operacional de Tempo Real (RTOS – *Real Time Operating System*) e um sistema operacional convencional é que num RTOS a importância para o sistema da finalização de uma tarefa é um valor que varia com o tempo [10][11]. Este valor geralmente é descrito em termos de deadlines para finalização de cada tarefa. Uma vez que o tempo de resposta é visto como parte crucial da exatidão do *software* (SW), ele deve poder ser comprovado sem argumentos estatísticos [12]. Assim como os microcontroladores, os RTOSs existem em grande variedade. A indústria automotiva, por exemplo, possuí um padrão chamado OSEK, no qual estão definidas uma série de especificações a serem seguidas pelos seus RTOSs [13].

6. *Neles, as implicações de uma falha de software são muito mais severas do que num desktop* – muitos SEs interagem com o ambiente ou seres humanos, como é o caso de sistemas de vida-críticos (*life-critical systems*) [14][15], por exemplo. Falhas podem ter consequências no ambiente, no próprio sistema ou até mesmo nas pessoas em torno dele. Um *bug* no *software* do sistema de injeção eletrônica de um automóvel, por exemplo, pode reduzir a vida útil de componentes do motor e aumentar o consumo e a emissão de gases poluentes. Um *bug* no *software* de controle da dosagem de radiação de uma máquina de raios-X poderia ter consequências sérias na saúde do paciente. Por essa razão, muitos SEs possuem mecanismos de segurança para detectar e contornar falhas, tal como o *watchdog timer* [16]. Neste aspecto, Peter Marwedel [17] estabelece 5 parâmetros de caraterização de SEs:

    - Confiabilidade: é a probabilidade do sistema de não falhar;
    - Manutenibilidade: é a probabilidade que uma falha no sistema possa ser corrigida em um certo intervalo de tempo;
    - Disponibilidade: é a probabilidade do sistema estar disponível. Será tanto maior quanto maior for sua confiabilidade e manutenibilidade;
    - Segurança: descreve a probabilidade do sistema de não causar algum tipo de dano;
    - Confidencialidade: descreve quão capaz é o sistema de manter dados confidenciais e de garantir uma comunicação autenticada;

7. *Costumam ter restrições no consumo de energia* – Ao contrario dos PCs, grande parte dos sistemas embarcados são alimentados unicamente por pequenas baterias. Neles, a redução de alguns miliwatts/hora no consumo pode estender a duração das baterias por dias ou meses. Frequentemente, a responsabilidade de poupar energia é atribuída unicamente aos engenheiros do

hardware (HW), mas em SEs essa responsabilidade deve ser compartilhada também pelos desenvolvedores do *software*. Em um PDA, por exemplo, o consumo pode ser reduzido em até 60% através de alterações na forma como o *software* é escrito [18]. Evidentemente, as escolhas feitas no projeto do *hardware* tem impacto crucial no consumo do sistema. Em muitos SEs o principal vilão do consumo é o processador, mas esse nem sempre é o caso. A Pathfinder, sonda espacial enviada para Marte, por exemplo, teve o consumo como uma das principais restrições de projeto. Todos os módulos da sonda foram projetados para poderem ser ligados e desligados individualmente e a sonda inteira permanecia em "*sleep mode*" durante a noite, quando as células fotovoltaicas deixavam de captar energia e a eficiência da bateria caía devido à queda de temperatura. Mesmo quando em modo operacional, o *software* não podia manter ligado mais de um módulo ao mesmo tempo devido à escassez de energia disponível [19];

8. *Devem poder operar em condições ambientais extremas* – A natureza portátil de muitos sistemas embarcados implica que eles devem ser capazes de suportar as mesmas condições que seus usuários ou sistemas receptores. Um PDA deve poder funcionar à 40ºC na praia de Copacabana, a -15ºC nos Alpes ou a 11.000 metros de altitude na cabine de um avião. Da mesma forma, o sistema de frenagem ABS deve funcionar no calor de 45ºC de Teresina, no frio de -5ºC de Porto Alegre, em dias secos e dias chuvosos, e em todas essas mesmas condições após 5.000 km de estrada de terra trepidante. Os requisitos ambientais de operação geralmente têm consequências diretas no *hardware* e as vezes até mesmo no *software*. Um processador que precise ser selado em borracha de silicone para poder suportar elevados índices de umidade, terá sua capacidade de dissipação térmica comprometida e precisará ter seu *clock* reduzido, afetando o desempenho do *software*;

9. *Seus recursos de sistema são extremamente menores do que os de um desktop* – Processadores com múltiplos núcleos rodando à frequências superiores a 2 GHz, discos rígidos com capacidade da ordem de *tera-bytes*, barramentos de alto desempenho, portas de comunicação de alta vazão, memórias RAM de alta capacidade, processadores gráficos e monitores de alta resolução, são recursos comuns dos *desktops* atuais. A capacidade de armazenamento da memória RAM desses computadores pessoais é maior do que todo o disco rígido de um típico PC do final da década de 90. Com isso, a maior parte dos aplicativos podem ser escritos assumindo a memória como um recurso infinito. Para um sistema embarcado a realidade é bem diferente. Na maior parte deles a RAM não chega a quinhentos *bytes,* todo o SW deve caber em alguns poucos kB e rodar em um processador cujo *clock* mal chega a 20 MHz. A quantidade de teclas é bastante limitada fazendo com que precisem acumular funções. Muitos não tem sequer um *display* e a interface com o usuário se dá através de LEDs e sons. Evidentemente, o código escrito para essas aplicações precisa observar uma série de cuidados que os voltados para PCs não precisam. Uma simples operação matemática, pode consumir 1% ou 70% da memória em um dado microcontrolador, dependendo apenas da forma como é escrita;

10. *Geralmente todo seu programa fica armazenado em uma ROM* – Esta característica pode parecer insignificante mas não é. Ter seu código armazenado em uma ROM impõe severas limitações ao sistema. A primeira, discutida anteriormente, diz respeito ao tamanho do código. A outra está ligada aos métodos usados para projeta-lo. Um código armazenado numa ROM não pode ser depurado da mesma maneira como ocorre nos PCs. Para inserir um *breakpoint,* o depurador precisa remover uma instrução do programa e substitui-la por outra responsável por desviar a execução para algum ponto do depurador, tarefa impossível numa memória que não pode ser alterada aleatoriamente;

11. *Requerem ferramentas e métodos especializados para serem eficientemente projetados* – O fato dos sistemas embarcados serem compostos por *hardware* e *software* integrados exige mudanças na sua forma de concepção, teste e depuração, em relação a um projeto de *hardware* e um projeto de *software* feitos isoladamente. Depurar um sistema de processamento de vídeo no qual parte dos blocos são implementados em *software* e outra parte em *hardware* não é uma tarefa nada trivial e necessita bem mais do que um simples aplicativo de depuração. Muitas vezes, é necessário projetar uma

plataforma que emule o sistema no qual o SE será integrado, paralelamente ao projeto deste, para ser usado nas etapas de teste e depuração. Depurar um sistema de controle de altitude usando o próprio avião não é muito prático, tampouco uma boa ideia;

12. *Microprocessadores embarcados geralmente possuem circuitos dedicados para depuração* – Conforme explicado anteriormente, a depuração do código programado em uma ROM encontra sérias dificuldades. Por essa razão, microprocessadores voltados para aplicações embarcadas costumam incluir circuitos especialmente dedicados à depuração. Alguns fabricantes oferecem duas versões dos seus microprocessadores, uma "de desenvolvimento", contendo os recursos de depuração, necessários ao projeto, e uma versão "de produção", sem tais recursos, a fim de poupar área de silício e reduzir os custos do componente.

Uma vez expostas, em linhas gerais, as características típicas de um sistema embarcado, torna-se evidente a necessidade do estudo e domínio de metodologias de projeto que permitam implementa-lo de forma eficiente no menor tempo possível.

3 Projeto de Sistemas Embarcados

O projeto de sistemas embarcados deve seguir uma metodologia precisa. Segundo Waine Wolf [20], três razões principais justificam a importância de se seguir uma metodologia de projeto: 1 - permitir que o cumprimento dos requisitos de projeto possam ser assegurados de maneira formal à medida que o projeto avança; 2 - potencializar o uso de ferramentas de automatização das tarefas (tais como ferramentas de CAD), uma vez que cada etapa é conhecida e pode ser dividida em pequenas tarefas automatizáveis; 3 - facilitar a comunicação entre as equipes de desenvolvimento, já que o estabelecimento da metodologia também auxilia a repartição de responsabilidades, permitindo que cada equipe tome ciência das atribuições das demais e da maneira como estão vinculadas no projeto.

Seja qual for a metodologia adotada, ela deve ser capaz de realizar três ações comuns a todo projeto [21]: especificação/modelagem, validação e síntese.

O processo de especificação e modelagem de um sistema, sub-sistema ou componente, pode ser resumido como o processo que inicia com a descrição de uma especificação e termina com a descrição de um modelo. Segundo Edwards [21], um modelo formal deve conter:

- Uma especificação funcional, sob a forma de um conjunto de relações explícitas ou implícitas que envolvem entradas, saídas, e possíveis estados internos;
- Um conjunto de propriedades que o projeto deve satisfazer, dadas na forma de um conjunto de relações entre entradas, saídas e estados, que possam ser comparados com a especificação funcional;
- Um conjunto de índices de desempenho que avaliem a qualidade do projeto em termos de custo, confiabilidade, velocidade, tamanho, etc., dados como um conjunto de equações que envolvam, entre outras coisas, entradas e saídas;
- Um conjunto de restrições sobre os índices de desempenho.

Dessa forma, o projeto durante o processo de especificação e modelagem pode ser visto como a sequência de passos que implementam o modelo especificado cumprindo as formalidades descritas acima.

Uma vez estabelecido o modelo, ele deve ser validado. Segundo o IEEE [22], a verificação e validação de *software* tem como principal objetivo, determinar durante o ciclo de vida do *software*, se os requisitos do próprio *software* e do sistema estão corretos, completos, precisos e consistentes, assegurando assim a qualidade de um produto de *software*. Este conceito pode ser estendido também para componentes de *hardware*, até porque eles também podem ser modelados em linguagens semelhantes às de programação (linguagens de descrição de *hardware*). As duas principais ferramentas utilizadas nessa etapa são a simulação e a verificação formal [21]. Se um erro ou inconformidade com os requisitos é detectado, o processo se modelagem deve ser retomado para corrigi-lo, repetindo o ciclo, tantas vezes quantas forem necessárias até que o modelo seja validado.

Se um modelo foi validado ele pode ser sintetizado. Por síntese entende-se, de forma abrangente, *o processo de se transformar uma especificação mais abstrata em uma menos abstrata*, ou seja, o processo de se aumentar o nível de detalhamento de uma especificação. Este conceito é frequentemente confundido com o conceito de compilação, entretanto, é conveniente estabelecer uma diferença entre eles. A compilação tem como entrada um *software* e como saída um *software* em um nível de abstração mais baixo. A síntese, por outro lado, é mais abrangente. Ela pode ter como entrada um

software e como saída uma descrição de *hardware*, ou mesmo uma descrição de *hardware* e como saída uma descrição de *hardware* em um nível de abstração mais baixo.

Neste ponto, é interessante introduzir outro importante conceito, o de níveis de abstração. Para isso, é oportuno apresentar o *diagrama em Y* proposto por Daniel Gajski [23], bastante popular entre projetistas de sistemas embarcados. Segundo Gajski, toda a informação sobre um sistema pode ser representada por meio de um diagrama em Y, conforme esboçado na Fig. 1.

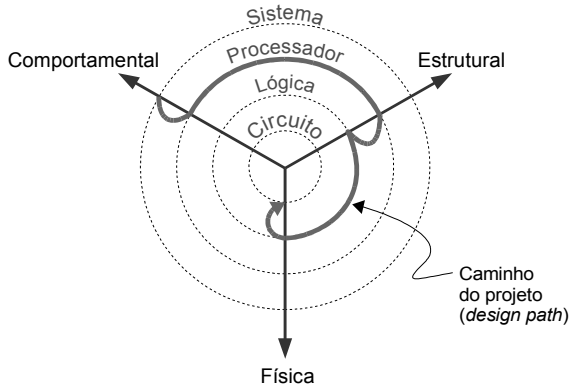

Fig. 1: Diagrama em Y de Gajski

Neste diagrama, toda a informação a respeito do sistema é repartida em três dimensões: comportamental, estrutural e física. Na dimensão comportamental cada componente é visto como uma caixa preta com saídas descritas em termos das entradas aplicadas em função do tempo. Nela não há qualquer informação de como ela é construída, ou seja, ela descreve o que é o componente, mas não como ele é. Na dimensão estrutural a caixa preta é representada como um conjunto de componentes e conexões. Na dimensão física informações de geometria são adicionadas, tais como altura, largura, peso, posição dos componentes, comprimento e forma das conexões, etc.

Cada um dos círculos concêntricos (ou camadas) do diagrama descreve o sistema alvo com diferentes níveis de detalhamento. Percorrendo-o de fora para dentro, parte-se do menor ao maior nível de detalhamento, o que implica numa elevação da complexidade. Quanto menor o nível de detalhamento na descrição do sistema, menor é a complexidade e maior é o nível de abstração dessa descrição, ou seja, a descrição do sistema vai do nível mais abstrato (e menos complexo) ao menos abstrato (e mais complexo) nas camadas mais internas. Ao mesmo tempo, quanto mais se desce nas camadas do diagrama, mais a descrição do projeto se assemelha ao seu nível final: o produto fisicamente implementado com componentes de silício.

Neste exemplo o projeto foi divido em 4 níveis de abstração: sistema, processador, lógica e circuito. No nível do sistema o projeto é descrito em termos de componentes tais como processadores, barramentos e memórias. No nível do processador os componentes do nível de sistema são detalhados em componentes da macro-arquitetura como processadores específicos, controladores de memória, árbitros de barramento, etc. No nível da lógica os componentes da macroarquitetura são detalhados em componentes como portas lógicas, *flip-flops* e registradores, enquanto no nível de circuito eles passam a ser descritos em termos de pequenas células formadas por transistores interconectados (pMOS e nMOS no caso da tecnologia CMOS).

Naturalmente, o número de níveis de abstração de um projeto não é rígido, podendo variar de acordo com o tipo de projeto e a metodologia empregada. Entretanto, na maior parte das vezes, os níveis de abstração são definidos pelas ferramentas de projeto adotadas, que por sua vez seguem um certa padronização, o que acaba por criar conjuntos bem definidos de níveis de abstração para as metodologias mais populares.

É importante observar que uma vez que a diferença de detalhamento entre os modelos de níveis consecutivos de abstração é gradual, a passagem de um nível para outro pode ser feita muitas vezes de maneira automática ou semi-automática através de ferramentas apropriadas. Da mesma forma, ferramentas de verificação podem ser utilizadas para analisar se os modelos de um mesmo sistema, ou sub-sistema, são coerentes entre si em diferentes níveis de abstração, a fim de detectar possíveis erros de implementação.

De maneira geral, o produto final do projeto situa-se na camada mais interna do eixo de descrição física. A maneira como o diagrama "é percorrido" até chegar a esse ponto é definida pela metodologia de projeto adotada. No exemplo esboçado na Fig. 1, a metodologia parte da descrição comportamental em nível de sistema, refinando-a para uma descrição comportamental em nível de processador, depois para uma descrição estrutural no mesmo nível, passando para uma descrição estrutural em nível de lógica, em seguida para uma descrição física, até finalmente sintetizar o projeto em uma descrição física em nível de circuitos.

Diversas metodologias de projeto de sistemas embarcados têm sido propostas ao longo dos últimos anos. De maneira geral, podemos enquadra-

las em três principais categorias, conforme a sequência em que dão evolução ao projeto. São elas:

- ***Bottom-up*** - o projeto inicia a partir da descrição detalhada dos componentes elementares do sistema. Esses componentes são então conectados para formar subsistemas, que por sua vez são conectados para formarem subsistemas maiores e assim, sucessivamente até compor o sistema completo. Numa metodologia *botton-up* típica, os projetistas desenvolvem uma série de componentes e os armazenam em uma biblioteca para serem utilizadas no próximo nível de abstração. A vantagem desta metodologia é de separar claramente cada nível de abstração em sua própria biblioteca. Isto permite a repartição de todo o projeto em pequenas partes a cada nível de abstração e facilitando o gerenciamento do projeto, já que cada grupo fornece uma biblioteca de componentes para o próximo nível de abstração. Por outro lado, a desvantagem dessa metodologia é que as bibliotecas devem conter todos os possíveis componentes com todos os possíveis parâmetros, devendo ser otimizados para os requisitos do nível de abstração seguinte, algo difícil de se prever nos baixos níveis de abstração.

- ***Top-down*** - o projeto inicia com uma formulação geral das características finais do sistema desejado, feita de maneira abstrata, ou seja, sem detalhes de como será implementado. À medida que o projeto avança, o sistema vai sendo refinado, dividido em subsistemas, os subsistemas em sub-subsistemas, e assim por diante, até que seja descrito em termos de componentes elementares. Além de ser bastante intuitiva, esta metodologia tem a grande vantagem de permitir um bom grau de automatização, o que permite reduzir o tempo de projeto e minimizar a introdução de falhas humanas. Por outro lado, tem a desvantagem de não permitir conhecer com precisão as reais métricas do sistema até que o último passo tenha sido completado. Quando elas não atendem aos requisitos, o projeto precisa recuar e ser refeito repetidas vezes até que eles sejam alcançados. Para minimizar esse inconveniente, são empregados os chamados estimadores, ferramentas capazes de estimar características de um nível mais baixo de descrição, tais como consumo, desempenho, área de silício ocupada, latência, vazão, etc., a partir de descrições em mais alto nível. Infelizmente, estimativas não são perfeitas, o que acaba por não eliminar o risco de reiterações no projeto.

- ***Meet-in-the-middle*** – é uma combinação das metodologias *bottom-up* e *top-down*, visando aproveitar as vantagens de cada uma delas e ao mesmo tempo evitar suas inconveniências. Em geral, essa metodologia emprega a abordagem *top-down* nos níveis de abstração mais altos e a *bottom-up* nos níveis mais baixos [24]. Nela o projeto inicia com uma descrição em alto nível de abstração que é refinada sucessivamente para níveis cada vez mais baixos até atingir um nível intermediário quando então é descrito em termos de componentes de uma biblioteca. Cada um dos componentes dessa biblioteca, por sua vez, tem as descrições dos níveis de abstração inferiores feitas através de alguma metodologia *botton-up*. Um exemplo típico desta metodologia é a *Metodologia de Projeto Baseada em Plataforma*, proposta por Sangiovanni-Vincentelli [25].

Atualmente, a maior parte dos projetistas utilizam algum tipo de metodologia *meet-in-the-middle* [24]. P:or essa razão, estes últimos parágrafos serão dedicados a expor um pouco mais sobre ela.

Um fluxo de projeto bastante popular, compatível com metodologias *meet-in-the-middle,* é apresentado por Arnold Berger [5], conforme exposto na Fig. 2.

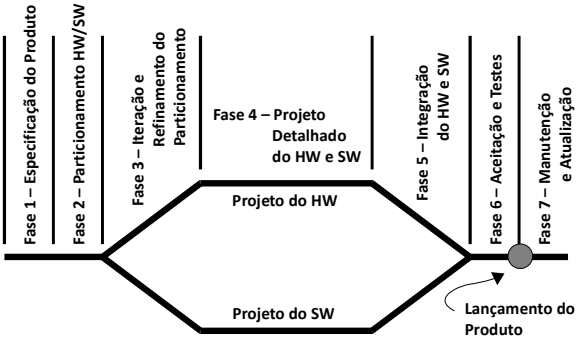

Fig. 2: Fluxo de projeto de sistemas embarcados

Neste fluxo, o ciclo de vida do projeto de sistemas embarcados é divido em 7 fases:

1. Especificação do produto;
2. Particionamento do projeto em componentes de *hardware* e *software*;
3. Iteração e refinamento do particionamento;
4. Tarefas independentes de projeto do *hardware* e do *software*;

5. Integração dos componentes de *hardware* e *software*;
6. Testes, aceitação e lançamento do produto;
7. Manutenção e atualização contínua.

Na fase de especificação o sistema é inicialmente descrito de maneira informal através de uma série de especificações. Durante a especificação deve-se destacar de forma clara quais são os requisitos do projeto que se deseja realizar. Os requisitos são as guias-mestre do projeto e estabelecem critérios importantes à tomada de decisões durante as etapas futuras. Pontos de otimização subjetiva, se existirem, devem ser colocados de forma hierárquica, ou seja, devem ser estabelecidos em ordem decrescente de relevância, já que muitas vezes sua implementação é conflitante, fazendo com que a otimização de um deprecie as características de outro. Um exemplo tipico é o projeto de um *smartphone*, no qual o desempenho de processamento e o consumo da bateria são duas características que comumente se deseja otimizar. O problema é que, geralmente, o aumento do desempenho implica no aumento do consumo. Se, ao final da fase 5, as duas características atenderem aos requisitos (com algum grau de liberdade), os projetistas deverão saber qual delas deverá ser priorizada na otimização. Ao final da fase de especificação, as especificações, inicialmente informais, devem ter sido formalizadas (frequentemente através de alguma linguagem de alto nível, como UML, por exemplo), de tal sorte que um diagrama de blocos do sistema embarcado completo possa ser visualizado, e cada bloco contenha especificações funcionais e comportamentais.

Na fase de particionamento HW/SW, os projetistas deverão decidir quais partes do sistema serão implementadas em *hardware* e quais serão implementadas em *software*. Essas partes podem tanto ser blocos definidos na fase de especificação, como também frações deles (sub-blocos). Dependendo da metodologia utilizada, esse particionamento pode ser guiado em função dos componente de HW e SW pré-existentes nas bibliotecas de projeto em uso. Deve-se notar no entanto que essa decisão deve atender prioritariamente os requisitos estabelecidos na fase de especificação. A decisão de implementar um dado bloco em *hardware* implica, na maioria das vezes, num aumento da complexidade, tempo e custos do projeto, mas permite um ganho de desempenho bem maior que sua implementação em *software*. Os aficionados por jogos de PC sabem bem a diferença entre ter uma placa dedicada à aceleração gráfica e emular OpenGL na própria CPU.

Durante a fase de iteração e refinamento do particionamento, o particionamento definido na fase anterior é posto à prova por meio de ferramentas de alto nível. Projetistas de *hardware* poderão se valer de ferramentas de simulação arquitetural, por exemplo, enquanto projetistas de *software* poderão rodar ferramentas de *benchmark* em placas de avaliação do processador ou microcontrolador escolhido. Em função dos resultados obtidos nesta fase, o particionamento pode ser revisto, repetindo-se o processo até que as simulações forneçam resultados satisfatórios, indicando que o projeto atenderá aos requisitos estabelecidos.

Uma vez concluída a fase de iteração e refinamento do particionamento, tem início a fase seguinte, na qual os componentes de *hardware* e *software* devem ser implementados de fato. Cada um deles modelado separadamente. Durante essa fase, ferramentas e técnicas de verificação são empregadas tanto nos componentes de HW quanto nos de SW e os erros encontrados, corrigidos. Após essa fase os componentes de HW e SW são reunidos e integrados (fase 5) de forma a compor o sistema embarcado completo. Idealmente, ao se reunir os componentes tudo deveria funcionar perfeitamente, mas a realidade costuma ser diferente e erros aparecem. Por se tratarem de sistemas complexos, a tarefa de se localizar e corrigir as fontes dos problemas não costuma ser algo evidente. Além disso, alguns problemas podem ter seu mecanismo de ativação bastante complexo, dependendo, por exemplo, de uma sequência específica de pressionamento de teclas quando o sistema se encontra em um dado estado interno e a temperatura ultrapassa um valor limite. Neste ponto, ferramentas de co-simulação e verificação, em conjunto com instrumentos como osciloscópios e analisadores lógicos costumam ser necessários.

Após a integração do sistema e depuração dos erros encontrados, vem a fase de testes e lançamento do produto. Nesta fase o sistema é testado exaustivamente em condições idênticas às que ele será submetido após o lançamento do produto. Durante esses testes, o sistema deverá responder e comportar-se conforme os requisitos estabelecidos no início do projeto.

Finalmente, após o lançamento do produto no mercado, vem a fase de manutenção e atualização. Manter e atualizar o produto concebido costuma ser bem mais lucrativo do que refazer todo um projeto novamente. Faz parte dessa fase também a busca por otimizar as características do produto lançado,

bem como o enriquecimento da documentação externa (para o público) e interna (para os atuais e futuros projetistas) a respeito dele.

4 Conclusão

Foram apresentados alguns conceitos básicos sobre sistemas computacionais embarcados, e as linhas gerais das metodologias de projeto mais aplicadas, bem como uma exposição sucinta de alguns dos formalismos importantes. Embora pontos relevantes tenham sido citados, uma quantidade imensa de informação foi omitida pela brevidade deste documento. Por essa razão, literaturas que considero imprescindíveis foram citadas ao longo do texto, a fim de permitir aos seus leitores aprofundar seus conhecimentos a respeito do tema.

5 Referências

[1] Klaus Finkenzeller, ***"RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification"***, 2ª ed., Wiley, 2003.

[2] Morales, M., Rau, S., Palma, M.J., Venkatesan, M., Pulskamp, F., Dugar, A., ***"Worldwide Intelligent Systems 2011–2015 Forecast: The Next Big Opportunity"***, International Data Corporation - IDC, Setembro de 2011.

[3] ***"IEEE Standard Glossary of Software Engineering Terminology"***, Version 610.12-1990, Standards Coordinating Committee of the IEEE Computer Society, pp. 30, USA, 1990.

[4] Heath, Steve, ***"Embedded System Design"***, 2ª ed., Elsevier, 2003.

[5] Berger, A.S., ***"Embedded Systems Design – An Introduction to Process, Tools, & Techniques"***, CMP Books, USA, 2002.

[6] Levy, Marcus, ***"EDN Microprocessor/Microcontroller Directory"***, EDN, 14 de setembro de 2000.

[7] Microchip Technology Inc., www.microchip.com, outubro de 2011.

[8] Swamy, G., Sarma S., ***"Manufacturing Cost Simulations for Low Cost RFID Systems"***, Auto-ID Center, MIT, USA, fevereiro de 2003.

[9] Jung, C. R.; Osório, F. S.; Kelber, C.; Heinen, F., ***"Computação Embarcada: Projeto e Implementação de Veículos Autônomos Inteligentes"***, Anais do CSBC'05 XXIV Jornada de Atualização em Informática (JAI), v. 1, p. 1358–1406, São Leopoldo, RS: SBC, julho de 2005.

[10] Stankovic, J.A., Rajkumar, R., ***"Real-Time Operating Systems"***, Real-Time Systems Journal, Vol. 28, Issue 2, pp. 237-253, Springer, Novembro de 2004.

[11] Jensen, E.D., Locke, C.D., Tokuda, H., ***"A Time-Driven Scheduling Model For Real-Time Operating Systems"***, IEEE Real-Time Systems Symposium, pp 112-122, 1985.

[12] Kopetz, Hermann., ***"Real-Time Systems – Design Principles for Distributed Embedded Applications"***, 1ª ed., Kluwer Academic Publishers, Boston, MA, USA, 1997.

[13] OSEK/VDX Steering Committe, ***"OSEK/VDX Operating System"***, Ver. 2.0, rev. 1, outubro de 1997.

[14] Bowen, J.P, Stavridou, V., ***"Safety-Critical Systems, Formal Methods and Standards"***, Software Engineering Journal, vol. 8, no. 4, pp: 189-209, Julho de 1993.

[15] Butler, R.W., Finelli, G.B., ***"The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software"***, IEEE Transactions on Software Engineering, vol. 19, no. 1, pp. 3-12, 1993.

[16] Fumihide Kitamura et al., ***"Watch Dog timer"***, United States Patent, Patent number: 4752930, 21 de Junho de 1988.

[17] Marwedel, P., ***"Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems"***, 2ª ed., Springer, 2011.

[18] Ellis, C.S., ***"The Case for Higher-Level Power Management"***, 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII), pp. 162–167, Rio Rico, AZ, Março de 1999.

[19] H. W. Stone, ***"Mars Pathfinder Microrover: A Low-Cost, Low-Power Spacecraft"***, 1996 AIAA Forum on Advanced Developments in Space Robotics, Madison WI, 1996.

[20] Wolf, W., ***"Computers as Components - Principles of Embedded Computing System Design"***, Morgan Kaufmann Publishers, San Francisco, 2ª ed., Junho de 2008.

[21] Edwards, S., Lavagno, L., Lee, E.A., Sangiovanni-Vincentelli, A., ***"Design of Embedded Systems: Formal Models, Validation and Synthesis"***, IEEE, vol. 85, No 3, pp. 366–390, Março de 1997.

[22] IEEE, ***"IEEE Std 1012-2004 - IEEE Standard for Software Verification and Validation"***, Revisão do padrão IEEE Std 1012-1998, 2004.

[23] Gajski D., Kuhn, R., ***"New VLSI Tools"***, Computer Magazine, pp 11–14, Dezembro de 1983.

[24] Gajski, D.D., Abdi, S., Gerstlauer, A., Schirner, G., ***"Embedded System Design - Modeling, Synthesis and Verification"***, Springer, 2009.

[25] Sangiovanni-Vincentelli, A., Martin, G.,

"Platform-based Design and Software Design Methodology for Embedded Systems", IEEE Design and Test for Computer, vol. 18, n. 6, p. 23-33, 2001.