Modelo de Arquitetura para Construção de Plataformas de Software Embarcado

Gustavo A. F. B. Melo, Sérgio V. Cavalcante

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
CEP. 50740-540 – Recife – PE – Brazil

{gafbm,svc}@cin.ufpe.br

Abstract. *The development of embedded software is still very difficult. A few tools assist the embedded application design with visual modeling and automatic code generation, but they fail to support the HAL. This paper proposes an open architecture for reusable and easily portable software platform design. The architecture is composed of nine layers which encapsulate and separate features and functionalities of the microcontroller, peripherals, electronic components, kernel and middleware. This work aims standardization for embedded software development which allows you to automate the process of elaborating software platforms.*

Resumo. *O desenvolvimento de software embarcado ainda é muito deficiente. Algumas ferramentas auxiliam o projeto de aplicações embarcadas com modelagem visual e geração automática de código, mas falham no suporte a HAL. Este artigo propõe uma arquitetura para projetar plataformas de software reusável e de fácil portabilidade. A arquitetura é composta por nove camadas que encapsulam e separam características e funcionalidades do microcontrolador, periféricos, componentes eletrônicos da placa, kernel e middleware. Este trabalho visa uma padronização no desenvolvimento de software embarcado que permite automatizar o processo de construção de plataformas de software.*

1. Introdução

O desenvolvimento de sistemas embarcados é um processo que envolve duas abordagens contrastantes. Por uma lado o desenvolvimento de hardware é suportado por métodos analíticos e ferramentas de síntese automática (EDA) [1], como IDEs para construção de ASICs e PCBs. Por outro lado o desenvolvimento de software embarcado não tem acompanhado a mesma velocidade e eficiência, principalmente no que diz respeito à automatização. Algumas poucas ferramentas auxiliam o projeto de software embarcado com modelagem visual e geração automática de código [2][3], mas têm uma deficiência no suporte às interfaces de hardware. Boa parte disso deve-se ao fato de que as plataformas de hardware utilizadas por sistemas embarcados apresentam arquiteturas bastante distintas [4]. Além disso, esses dois paradigmas (projeto de hardware e software) têm sido tratados de forma ortogonal. Grande parte das metodologias de co-design [5][6] baseiam-se em dividir o projeto em hardware e software, e depois desenvolvê-los separadamente. No entanto, alguns componentes de software podem ser modelados de modo que eles sejam diretamente relacionados a componentes de hardware.

AUTOSAR [7] é uma parceria de empresas a fim de estabelecer uma padronização aberta à indústria para arquiteturas elétrico-eletrônicas automotivas. O principal conceito de AUTOSAR é separação entre aplicação e infra-estrutura. Entretanto, esta abordagem é muito direcionada a aplicações automotivas, além de ser bastante restritiva. EPOS [4] é um sistema operacional baseado em componentes desenvolvido para permitir portabilidade da aplicação. Contudo, as abstrações de hardware designadas por esta abordagem não separam interface de acesso e controle de acesso, o que acarreta em menos modularidade.

Este artigo propõe uma arquitetura para construção da plataforma de software no nível de componentes. Ele descreve uma estrutura de software embarcado reusável e de fácil portabilidade, que é composta por nove camadas que encapsulam e separam características e funcionalidades do microcontrolador, periféricos, componentes eletrônicos da placa, kernel e middleware.

A seção 2 apresenta a arquitetura proposta para o desenvolvimento de software embarcado, bem como descreve cada uma das suas nove camadas. Um exemplo real de utilização da arquitetura é apresentado na seção 3. As conclusões do trabalho são expostas na seção 4.

2. Arquitetura

A arquitetura proposta utiliza os padrões de projeto Layered Pattern e Five-Layer Architecture Pattern [8]. Ela organiza o software embarcado em camadas, de modo a facilitar o reuso e a manutenabilidade dos componentes do sistema (Figura 1). Esta estrutura permite ao projetista abstrair a plataforma em termos de hardware e sistema operacional, proporcionando um alto grau de portabilidade da aplicação embarcada.

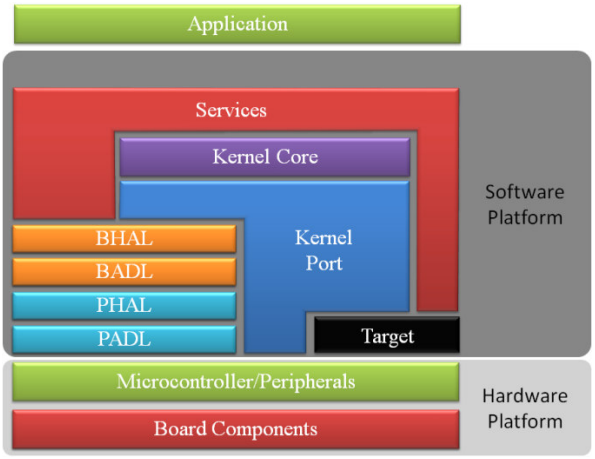

Figura 1. Embedded Software Architeture

A plataforma de hardware constitui-se dos componentes eletrônicos os quais o software é capaz de acessar. Exemplo: chip com o core do processador e seus periféricos, memória flash, LCD, LED ou FPGA. A plataforma de software provê as funcionalidades necessárias para lançar uma aplicação, dando suporte a escalonamento, gerenciamento de memória, acesso ao hardware e outros. Os componentes da plataforma de software são divididos em nove camadas: Target, PADL, PHDL, BADL, BHAL, Kernel Core, Kernel Port, Services e Application. As subseções seguintes descrevem com maiores detalhes cada uma das camadas da plataforma de software.

2.1 Target

Existem diferentes fabricantes que fornecem diferentes famílias de processadores. Cada uma dessas famílias possui suas peculiaridades. Além disso, vários compiladores distintos dão suporte a uma família. O código necessário para o startup do microcontrolador, definição dos vetores de interrupção e os tipos básicos são parcelas do software que normalmente diferem entre os compiladores e/ou processadores. Toda esta parte do software dedicada à combinação processador/compilador é encapsulada na camada Target. Dessa forma é possível mudar o compilador usado no projeto trocando apenas o componente Target. A abstração que esta camada proporciona também permite a portabilidade do resto do software para o caso de troca do microcontrolador por outro da mesma família e com diferente capacidade de memória.

2.2 PADL

Camada Dependente da Arquitetura do Periférico (*Peripheral Architecture Dependent Layer*) inclui os componentes de software que permitem o acesso direto aos periféricos do microcontrolador por meio de seus registradores. Cada família de microcontrolador possui um conjunto de periféricos com uma arquitetura particular. É função dos componentes desta camada definir a interface de acesso, independente do compilador, aos registradores de cada um dos periféricos. Assim podemos desenvolver as interfaces para cara periférico do microcontrolador sem se preocupar com as peculiaridades do compilador.

2.3 PHAL

Camada de Abstração de Hardware do Periférico (*Peripheral Hardware Abstraction Layer*), como o próprio nome diz, abstrai especificidades da arquitetura dos periféricos do microcontrolador e permite que as camadas superiores sejam independentes do hardware. Este conceito já é bem difundido no campo de sistemas operacionais [9][10][11].

O motivo de a camada PHAL ser separada da camada PADL é que um mesmo periférico pode ter mais de uma abstração de hardware. Um exemplo é a UART1 do microcontrolador LPC2368 [12]. Esta UART (*Universal Asynchronous Receiver Transmitter*) possui todas as funcionalidades das outras UARTs e mais a funcionalidade de controle de fluxo. Um componente *phal_uart* pode implementar todas as UARTs deste chip, inclusive a UART1, enquanto o componente *phal_modem_uart* pode contemplar as características específicas de controle de fluxo da UART1.

2.4 BADL

Camada Dependente da Arquitetura da Placa (*Board Architecture Dependent Layer*) possui os módulos que permitem o acesso, através dos periféricos, aos elementos da placa externos ao microcontrolador. Os componentes desta camada implementam o protocolo específico para acessar determinado hardware, como por exemplo uma determinada memória flash serial da ou um simples LED, por meio dos componentes PHAL.

2.5 BHAL

Camada de Abstração de Hardware da Placa (*Board Hardware Abstraction Layer*) é análoga a PHAL, abstrai as características específicas dos componentes eletrônicos da

placa, discutidos na seção 2.4. Uma memória flash serial Atmel família AT45 *badl_atmel_45_flash* é encapsulada por um módulo *bhal_flash_memory*, que possui uma interface genérica para todas as memórias flash.

2.6 Kernel Port

Esta camada tem o papel de garantir a portabilidade do kernel. Várias funções do kernel utilizam rotinas que dependem da arquitetura do microcontrolador. Como muitas vezes estas rotinas são escritas em *assembly*, elas dependem também do compilador. O *port* do kernel tem o objetivo de abstrair esta porção que é específica a microcontrolador/compilador [9]. Os componentes desta camada podem utilizar módulos da camada PHAL, Target ou ainda acessar diretamente a CPU.

2.7 Kernel Core

É a essência do kernel. Todas as funções do kernel, independentes do hardware e compilador, são implementadas nesta camada. Rotinas de criação de tarefas, troca de contexto, entrada e saída de seção crítica e incrementação do tempo são exemplos da infra-estrutura que o kernel fornece para os componentes da camada Service (seção 2.8), como, por exemplo, serviço de controle de tempo (*delay*), serviço de semáforo e serviço de fila, além do serviço de escalonamento.

Já que cada kernel apresenta funcionalidades distintas, fica difícil determinar interfaces genéricas para todos os kernels. Cada componente possui uma API diferente. Por outro lado alguns kernels suportam interfaces padronizadas para RTOS, como POSIX [13] e µITRON [14], o que pode aumentar a reusabilidade.

2.8 Services

Esta camada encapsula todo o acesso da aplicação à plataforma de software. Funcionalidades como escalonamento, gerenciamento de memória, acesso a dispositivos periféricos, storage, dentre outros, são disponibilizadas para as aplicações por meio de componentes chamados serviços. Esta idéia baseia-se no conceito de middleware [15], na qual um conjunto de serviços permite que múltiplos processos rodando em uma ou mais plataformas interajam independentemente da arquitetura (interoperabilidade).

Como foi dito, os dispositivos do sistema, sejam eles IO, comunicação, armazenamento de dados ou outros, têm interface com a aplicação por meio de serviços, que gerenciam o recurso de uma determinada forma (fila, semáforo,...). Se existem duas formas diferentes de gerenciar um recurso, como uma UART, deve haver dois componentes de serviços diferentes, um para cada tipo de gerenciamento. O projetista que deve decidir qual serviço deve ser instanciado para gerenciar a UART da forma mais apropriada às necessidades da aplicação.

2.9 Application

Esta camada designa os componentes que implementam as funcionalidades do sistema embarcado. Os módulos da aplicação dependem das interfaces fornecidas (serviços instanciados) pela camada Service, e.g. escalonamento de tarefas, mas são totalmente independentes da arquitetura de hardware da plataforma. Para rodar um componente da aplicação em outra plataforma basta que esta outra plataforma possua os mesmos serviços que este componente utiliza.

3. Exemplo

Esta seção apresenta um exemplo de utilização da arquitetura proposta para o projeto de um painel de LEDs. A plataforma de hardware é a placa MCB2300 da Keil [12] (microcontrolador LPC2368 da NXP) com uma memória flash serial da Atmel, família AT45 [16]. O RTOS aplicado foi o FreeRTOS [17]. O projeto de software possui 73 arquivos contendo 6553 linhas úteis de código.

Primeiro, nós mapeamos os componentes de hardware da placa em componentes de software. Como visto, os recursos da placa utilizados são o microcontrolador LPC2368, uma flash e um LED. Para o primeiro, um componente da camada Target foi instanciado. Para os outros, dois componentes da camada BHAL são instanciados. Os módulos BHAL proveêm abstração de hardware, mas eles precisam acessar funcionalidades específicas dos respectivos componentes eletrônicos. Portanto, os módulos *bhal_led* e *bhal_flash_memory* são implementados, respectivamente, sobre os módulos *badl_led* e *badl_atmel_45_flash* (implementação BADL para a memória AT45). Os componentes da placa são conectados ao microcontrolador através de pinos que são controlados pelos periféricos. Cada módulo BADL depende dos componentes PHAL relacionadas a cada um dos periféricos utilizados. O módulo *badl_led* depende do módulo *phal_io_port*, enquanto o módulo *badl_flash_memory* depende de *phal_io_port*, bem como de *phal_spi*. O componente *phal_spi* acessa *padl_spi* (SPI), *padl_pincon* (Pin Connect Block), *padl_sysctrl* (System Control Block) e *padl_vic* (Vectored Interrupt Controller), todos periféricos do LPC2368. O componente *phal_io_port* também depende de *padl_pincon* e *padl_sysctrl*, além de usar o módulo *padl_gpio* (General Purpose Input/Output).

O *port* do FreeRTOS, além de outros detalhes, utiliza o timer. Portanto, o componente *kernel_port_FreeRTOS* torna-se dependente de *phal_timer*. Este, por sua vez, faz uso das interfaces implementadas pelos componentes *padl_timer*, *padl_sysctrl* e *padl_vic*. O componente *kernel_FreeRTOS*, a essência do kernel, usou a implementação original do núcleo do FreeRTOS. Após instanciar os componentes do kernel nós criamos os serviços necessários à aplicação. O primeiro serviço, *service_led* fornece um suporte simples para acessar o LED. O acesso à memória flash é garantida por *service_lightdot_storage*. O componente *service_frt_panel* fornece acesso ao painel de LEDs. O serviço *service_scheduling* dá suporte a escalonamento baseado em prioridade para tarefas periódicas, usando a infra-estrutura de *kernel_FreeRTOS*. O serviço de tempo (*service_time*) é essencial para tarefas que exigem a função de *delay*. Daí então criou-se a aplicação. O módulo *LedTest* é responsável por fazer o LED piscar. O componente *DrawPanel* destina-se a exibir rolando uma mensagem lida da unidade de armazenamento.

4. Conclusão

Atualmente há uma deficiência de modelos de arquitetura de software embarcado apropriados para a composição de plataformas de software por meio da conexão de componentes. Este artigo apresentou uma arquitetura para o desenvolvimento de software embarcado visando maximizar a reusabilidade e portabilidade dos componentes de software. Esta arquitetura proposta foi exemplificada por um projeto de um painel de LEDs, que demonstra a implementação de componentes para cada camada.

Pretendemos evoluir este trabalho a fim de elaborar uma padronização para o projeto de software embarcado que permita a criação de repositórios de componentes, como acontece com plataformas de software desktop (Java, C #,...). Dessa forma, a construção de uma plataforma exige apenas instanciar os componentes de cada camada do repositório correspondente.

5. Referências

[1] Hazinger, T.A. and J. Sifakis. (2006) "The Embedded Systems Design Challenge", In: *Proc. of the 14 International Symposium on Formal Methods*.

[2] The MathWorks, "Simulink - Simulation and Model-Based Design", www.mathworks.com/products/simulink/, 1994.

[3] Ha, S., Kim, S., Lee, C., Yi. Y., Kwon, S. and Joo, Y.-P. (2007) "PeaCE: A hardware-software codesign environment for multimedia embedded systems", In: *ACM Transactions on Design Automation of Electronic Systems*, Vol. 12, No. 3, pp. 1-25.

[4] Marcondes, H., Junior, A., Wanner, L., Cancian, R., Santos, D., and Fröhlich, A. (2006). "EPOS: Um Sistema Operacional Portável para Sistemas Profundamente Embarcados". *Workshop de Sistemas Operacionais*.

[5] Cavalcante, S. (1997) "A Hardware-Software Codesign System for Embedded Real-Time Applications", PhD Thesis, Department of Electrical and Electronic Engineering, University of Newcastle.

[6] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabbara. (1997) "Hardware-software co-design of embedded systems: the POLIS approach". Norwell, MA: *Kluwer Academic Publishers*.

[7] AUTOSAR GbR. (2006) AUTOSAR – Technical Overview V2.0.1.

[8] B. P. Douglass, (2002), "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", *Addison-Wesley.*

[9] A. J. Massa, (2002), "Embedded Software Development with eCos"*, Prentice Hall*.

[10] A. Tanenbaum, (1992), "Modern Operating Systems", *Prentice Hall*, Englewood Cliffs, NJ.

[11] "Windows NT Hardware Abstraction Layer (HAL)", Microsoft, http://support.microsoft.com/kb/99588, (2006).

[12] Keil, "MCB2300 Evaluation Board", www.keil.com/mcb2300/, abril/2010.

[13] D. R. Butenhof, (1997), "Programming with POSIX threads"*, Addison-Wesley*.

[14] µITRON4.0 Specification, (1999), ITRON Committee, TRON Association, Japan.

[15] J. M. Myerson, (2002), "The Complete Book of Middleware", *Auerbach Publications*.

[16] Atmel, "Atmel DataFlash", www.atmel.com/products/DataFlash/, abril/2010.

[17] Real Time Engineers Ltd., "FreeRTOS", www.freertos.org, acessado em abril/2010.