Uma Implementação de Hiper-Realismo Baseada em Sistema Embarcado

Yuri G. G. da Costa[1], Alexandre S. G. Vianna[2], José A. G. de Lima[1], Liliane S. Machado[2], Ronei M. Moraes[2]

{yuriggc,strapacao}@gmail.com, {jose,liliane}@di.ufpb.br, ronei@de.ufpb.br

Resumo

Este artigo apresenta uma proposta de arquitetura de um sistema embarcado para hiper-realismo que prove uma interface entre ambientes virtuais e ambientes reais, proporcionando o acionamento e controle dos mais variados elementos que compõem um ambiente físico. O sistema é implementado através do processador Nios® II e FPGA (*Field Programmable Gate Array*) *que permitem o uso de módulos especialmente projetados para a aplicação específica, produzindo um sistema flexível e compacto.*

1. Introdução

Nos tempos atuais, as novas tecnologias aplicadas às interfaces de sistemas computacionais permitem conceber sistemas chamados de hiper-realidade, em que ambientes virtuais oferecem interação intuitiva em ambientes virtuais que podem estar conectados remotamente a ambientes reais. Desde a década de 70, essas tecnologias fizeram com que o acesso a serviços e variáveis de um ambiente real não necessite da presença humana no local onde estes se encontram [1].

As condições de um ambiente real são descritas por uma coleção de variáveis físicas como temperatura, luminosidade, presença de pessoas, estado de portas e janelas e, etc. Em hiper-realidade, alterações nas variáveis de ambiente correspondem em tempo real entre os ambientes virtual e físico. O controle da interface é possível através de sistemas inteligentes, que oferecem uma camada de abstração entre o ambiente virtual e o ambiente físico. Sistemas embarcados são capazes de responder em tempo real a alterações em ambos os lados, possibilitando a hiper-realidade.

Com o avanço no desenvolvimento de circuitos integrados e, tecnologias como FPGA (*Field Programmable Gate Array*) e novas ferramentas de desenvolvimento CAD (*Computer Aided Design*), sistemas embarcados complexos podem ser desenvolvidos em linguagens de alto nível e mapeados para *chips* com eficiência. Atualmente processadores podem ser sintetizados em um *chip* FPGA. A vantagem dessa implementação é que o sistema não está amarrado a especificações do *hardware*; logo, há a possibilidade de adaptação do processador ao projeto. Essa flexibilidade permite também a criação de módulos personalizados que podem ser adicionados à arquitetura já existente. A liberdade de personalizar o sistema embarcado conforme as necessidades do projeto significa economia de tempo e *hardware*. Assim, o sistema gerado segue os principais conceitos de sistemas embarcados, um sistema compacto e, ideal para a aplicação e sem excesso ou falta de componentes [2].

Nesse artigo apresenta-se um sistema embarcado para hiper-realidade, que utiliza um *chip* FPGA em que é sintetizado o processador Nios® II da Altera [3], um processador configurável de 32 bits para propósitos gerais. O sistema aborda uma camada de interface do ambiente físico para o ambiente virtual.

O artigo é organizado da seguinte forma. Na seção 2, apresenta-se a visão geral do sistema, mostrando a organização e as principais características. A seção 3 indica a arquitetura proposta para o sistema detalhando as camadas de *software* e *hardware*. A seção 4 apresenta o fluxo de desenvolvimento do sistema. A seção 5 mostra resultados obtidos e uma discussão.

2. Visão geral do sistema

Geralmente, em um sistema de hiper-realidade a função do sistema embarcado é prover um gerenciamento do ambiente físico, enquanto o ambiente virtual é gerenciado por um servidor. Aqui apresentamos uma arquitetura de *software* para o sistema embarcado que é capaz de gerenciar de forma inteligente os acionamentos de vários dispositivos como lâmpada, ar-condicionado, portas e, janelas, e sistemas de incêndio em um ambiente real. O sistema aciona os dispositivos através de atuadores (dispositivos de potência) e os sensores verificam o real estado dos dispositivos.

[1] Universidade Federal da Paraíba - Laboratório de Sistemas Digitais
[2] Universidade Federal da Paraíba - Laboratório de Tecnologias para o Ensino Virtual e Estatística

O *software* de controle de acionamentos é responsável por uma lógica de acionamento inteligente que siga regras predefinidas. O sistema deve ser capaz de gerenciar instruções de acionamento de dispositivos recebidas do servidor, executar os acionamentos, verificar possíveis falhas no acionamento, checar variáveis de ambiente e informar ao servidor as modificações. Todas essas funções são feitas conforme especificações do dispositivo a ser acionado.

Entre servidor e sistema embarcado há uma comunicação, que pode ser feita através de uma interface Ethernet que utiliza programação de *sockets* ou USB (*Universal Serial Bus*). A comunicação é bidirecional. Ambos informam alterações em variáveis do ambiente para que o outro acompanhe as mudanças. Para ocorrer um gerenciamento eficiente foi elaborado um protocolo de comunicação, em que o servidor envia uma instrução de acionamento ao ambiente físico e só altera o banco de dados quando houver uma confirmação do acionamento. Uma instrução é uma mensagem de 2 bytes com formato mostrado na Figura 1, 2 bits para acionamentos e 6 bits parâmetros. Esse formato suporta 256 dispositivos, 4 acionamentos e 64 parâmetros possíveis.

Código do dispositivo 8 bits	Acionamento 2 bits	Parâmetro 6 bits

Figura 1. Formato de instrução

A arquitetura do *software* dentro do sistema embarcado funciona com processos concorrentes, um processo executa as instruções da fila de acionamentos recebida do servidor. Quando não se consegue acionar um dispositivo em várias tentativas, uma mensagem que indica defeito é enviada ao servidor indicando defeito. O outro processo checa as variáveis do ambiente através dos sensores e as possíveis alterações no meio físico, são enviadas ao servidor. Além de verificar os dispositivos do ambiente físico o sistema deve também checar as variáveis como luminosidade, presença de pessoas, portas, fumaça e etc. A Figura 2 mostra o funcionamento desse sistema.

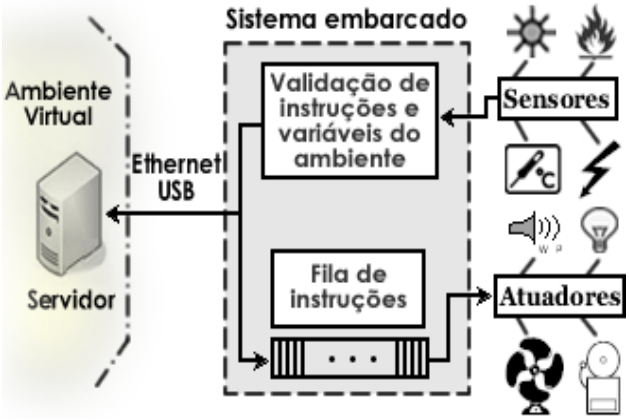

Figura 2. Visão geral do sistema

3. Arquitetura Proposta

Segundo Noergaard [4], os sistemas embarcados geralmente possuem uma estrutura de arquitetura semelhante ao modelo de referência formado por uma camada de *hardware*, uma camada de sistema de *software* e uma camada de aplicação.

O sistema embarcado proposto baseia-se, então, em uma arquitetura física que implementa a camada de *hardware*, a qual suporta a existência das camadas de sistema de *software* e de aplicação.

3.1. Camada de *Hardware*

A camada de *hardware* é formada por um *chip* FPGA, uma memória SDRAM (*Synchronous Dynamic Random Access Memory*), uma interface UART (*Universal Asynchronous Receiver/Transmiter*), uma interface Ethernet física, um dispositivo USB e um ADC (*analog-to-digital converter*), além dos sensores (analógicos e digitais) e atuadores (digitais) que relacionam-se com o ambiente.

No interior do *chip*, FPGA, a lógica digital é dividida em duas regiões: subsistema Nios® II e outros componentes. A primeira região é composta pelos seguintes elementos: processador embarcado Nios® II, barramento interno Avalon®, módulo JTAG (*Join Test Action Group*) UART, módulo PIO (*Parallel Input/Output*), controlador de SDRAM e módulo Ethernet MAC (*Media Access Control*). No exterior do subsistema Nios® II, a segunda região é formada pelos seguintes componentes: controlador USB e controlador ADC.

Da união de todos os elementos físicos listados acima, forma-se a camada de *hardware*, conforme ilustrado na Figura 3.

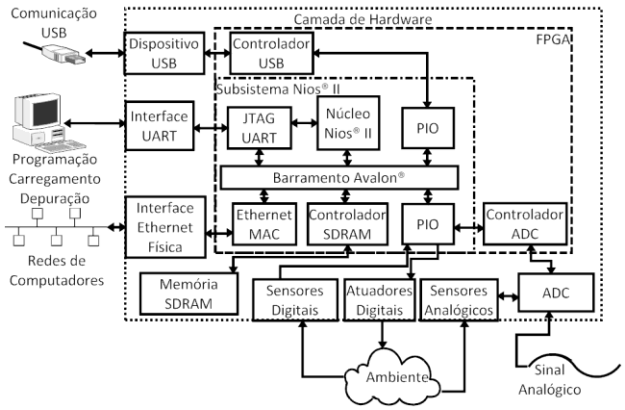

Figura 3. Camada de *hardware* da arquitetura proposta.

Devido à necessidade de armazenamento de dados e instruções em vários níveis de abstração, foi incluída à

camada de *hardware* uma memória SDRAM, já que a memória interna ao *chip* FPGA não é satisfatória para essa aplicação. Dessa forma, o subsistema Nios® II possui um controlador de SDRAM, que realiza a mediação entre o barramento Avalon® e a memória externa.

A interface UART permite a programação do *chip* FPGA conectado a um *host*, através da tecnologia JTAG e o carregamento do *software* no sistema final. Além disso, ela fornece, à etapa de desenvolvimento das camadas de *software*, meios de depuração do código a ser executado pelo processador Nios® II.

O sistema embarcado pode comunicar-se com outros sistemas utilizando redes de computadores. Tal comunicação se dá através da interface física e do módulo MAC, os quais implementam o padrão Ethernet.

Através do módulo PIO, o subsistema Nios® II pode comunicar-se com elementos externos, pois ele oferece múltiplas portas de entrada e saída. É assim que o sistema oferece comunicação USB, conversão analógico/digital, interpretação dos sensores digitais e manipulação dos atuadores digitais. Já os sensores analógicos precisam primeiro passar pelo ADC para que sejam utilizados dentro do sistema embarcado.

Os blocos do controlador USB e do controlador ADC foram desenvolvidos em VHDL (*VHSIC Hardware Description Language*) e validados com o uso de ferramentas CAD no Laboratório de Sistemas Digitais como parte de um trabalho anterior [5].

3.2. Camada de Sistema de *Software*

Para a arquitetura proposta, a camada de sistema de *software* é subdividida em *drivers* de dispositivos, sistema operacional e *middleware*.

Os *drivers* de dispositivos são fornecidos pelos fabricantes de cada módulo e integrados ao sistema embarcado. Para o subsistema Nios® II, a Altera® fornece uma biblioteca de funções da linguagem de programação C, denominada HAL (*Hardware Abstracion Layer*), que permite o acesso aos *drivers* através de funções padrão ANSI C [6].

O sistema operacional utilizado é o µC/OS-II™, o qual é fornecido em uma versão especializada ao processador Nios® II. Trata-se de sistema operacional em tempo real, que possui um *kernel* de tempo real e um ambiente multitarefa [7].

Para que ocorra a comunicação via redes de computadores, mais especificamente a Internet, é utilizada como *middleware* uma implementação simplificada da pilha de protocolos TCP/IP denominada lwIP [8]. Ela fornece uma API (*Application Programming Interface*) baseada no modelo Berkeley de *sockets*.

A Figura 4 contém todos os elementos da camada de sistema de *software*.

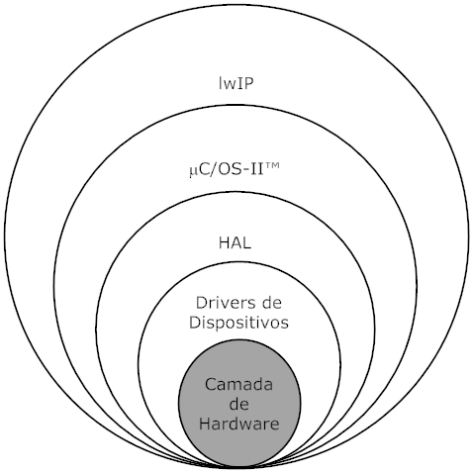

Figura 4. Camada de sistema de *software*.

3.3. Camada de Aplicação

Para execução da aplicação proposta, o sistema embarcado, inicialmente, recebe ordens de acionamento de dispositivos através dos meios de comunicação *Ethernet* e USB. Assim, os primeiros processos de *software* no que tange à de aplicação tratam do recebimento de ordens de acionamento através das interfaces *Ethernet* e USB. Uma ordem de acionamento é recebida e passada a um processo que trata da execução dos acionamentos através dos atuadores.

Cada vez, então, que o estado de um dispositivo for alterado, seja pelo processo acionador ou por um fator externo, os sensores irão ativar novos processos. Eles recebem as alterações percebidas e as transmitem de volta através dos processos de comunicação, só que, desta vez, no sentido contrário. Quando um sensor analógico é utilizado, um processo diverso é ativado e os dados digitados, recém convertidos pelo ADC, são ativados.

O conjunto de todos esses processos, executado concorrentemente, forma a camada de aplicação, a qual está posicionada logo acima da camada de sistema de *software*. Do relacionamento entre os processos, pode-se observar seu Diagrama de Fluxo de Dados (DFD) – Figura 5.

4. Metodologia Utilizada

Para o desenvolvimento do presente sistema embarcado, segue-se um fluxo de desenvolvimento adaptado de [9] o que é apresentado na Figura 6. Para tanto, utiliza-se um conjunto de ferramentas fornecido pela Altera® para desenvolvimento de projetos de sistemas embarcados que envolvam o processador Nios® II.

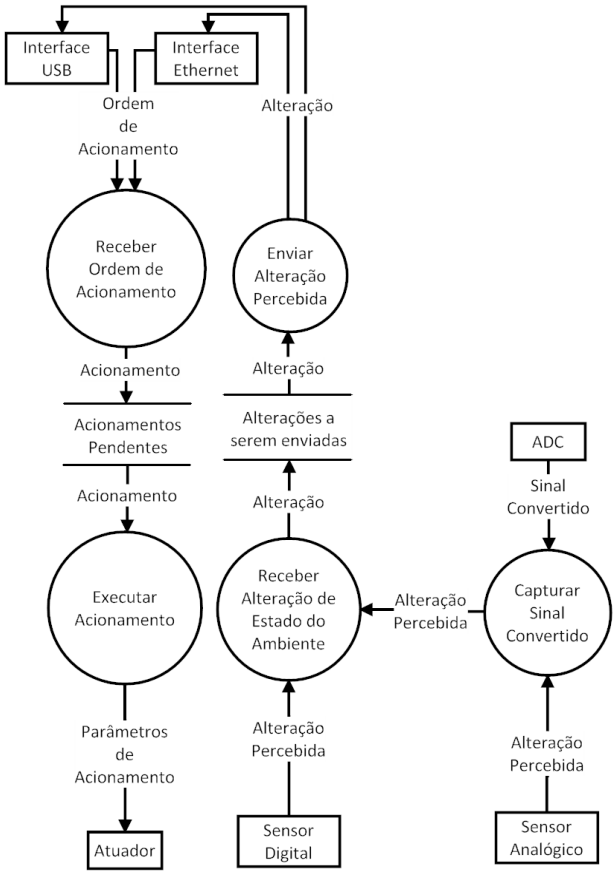

Figura 5. DFD dos processos que compõe a camada de aplicação.

A análise dos requisitos corresponde à detecção de todos os detalhes de funcionamento da camada de aplicação, retrata as funcionalidades que o sistema oferece. É daí que alguns elementos de *hardware* fundamentais, tais como sensores e atuadores, são identificados.

A partir das funcionalidades identificadas, deve ser definido e gerado um subsistema Nios® II que dá suporte às necessidades detectadas, através da ferramenta SOPC (*System On a Programmable Chip*) *Builder*. Ela fornece a possibilidade de configuração do núcleo do processador Nios® II de modo flexível. Além disso, também oferece instâncias a periféricos padrões que podem ser adicionados ao subsistema.

Nesse ponto, o fluxo divide-se entre o desenvolvimento de *hardware* e o de *software*. Posteriormente os fluxos se unem novamente para a finalização do sistema.

Dentro do fluxo de desenvolvimento de *hardware*, o subsistema gerado é fundido aos demais elementos de *hardware* que serão compostos no *chip* FPGA, dentro do projeto da ferramenta Quartus II. Ela provê, em seguida, a compilação do projeto de *hardware*, obedecendo às especificações do *chip* FPGA a ser utilizado.

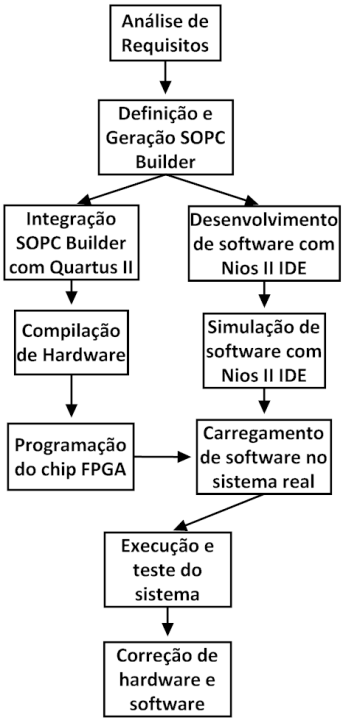

Figura 6. Fluxo de desenvolvimento do sistema.

Para o fluxo de desenvolvimento de *software*, surge um ambiente chamado Nios® II IDE (*Integrated Development Environment*), que permite a codificação do software em C/C++ com o uso da biblioteca ANSI C através da HAL. Em seguida, o código compilado gera o programa a ser executado pela camada de *hardware*. Contudo, antes da execução real, a ferramenta fornece um simulador denominado ISS (*Instruction Set Simulator*).

Quando os artefatos de *hardware* e *software* estão prontos, o sistema pode ser executado e testado à nível real. Com o *hardware* compilado, o *chip* FPGA é programado para que passe a conter os módulos projetados. Seqüencialmente, o *sofware* é carregado na memória do sistema a fim de ser executado.

Observa-se, então, a execução real do sistema, incluindo *software* e *hardware*. Nesse momento, o Nios® II IDE fornece a possibilidade de depuração do *software*. Esse processo só é permitido graças a interface UART que foi descrita na camada de *hardware*.

Da execução, teste e depuração, tanto de *hardware* quanto de *software*, deve-se, de acordo com a maneira como o sistema se comportou, fazer uma avaliação do que pode ser modificado, na busca por uma eficiência maior de *hardware* e *software*. A última etapa do fluxo pode fazê-lo voltar para qualquer um dos pontos anteriores.

5. Resultados e Discussões

A camada de *hardware* foi sintetizada para o FPGA EPS1SF10780C6ES (família Stratix) utilizando o Quartus

II da Altera®. A Tabela 1 apresenta os resultados em termos de número de células lógicas, quantidade de pinos utilizados, blocos DSP (Digital Signal Processing) e PLL (Phase-Locked Loop), além da frequência de operação.

Tabela 1. Características de hardware.

Total de células lógicas	3.742
Total de pinos	156
Blocos DSP	8
Blocos PLL	1
Frequência de clock	58,43 MHz

Os resultados apresentados na Tabela 1 mostram que a arquitetura de *hardware* proposta utiliza apenas 35% dos elementos lógicos, 37% dos pinos, 17% dos blocos DSP e PLL do dispositivo escolhido. Da freqüência de operação alcançada podemos afirmar que o sistema proposto produz resultados eficiências, principalmente para o grau de confiabilidade e característica de tempo real da aplicação a qual está submetido.

Falar do que pode ser posto neste espaço, mostra o potencial de ampliar

6. Conclusão e Trabalhos Futuros

Da junção entre características do sistema proposto, obtidas com os resultados apresentados, e a flexibilidade oriunda da utilização de dispositivos lógicos programáveis, como FPGAs, pode-se afirmar que a implementação de hiper-realidade baseada em sistemas embarcados é viável, principalmente levando-se em conta a facilidade e rapidez no desenvolvimento do projeto. O sistema apresentado corresponde em eficiência ao esperado em termos de confiabilidade e tempos de resposta, além de um baixo custo de implementação, manutenção e grande flexibilidade para incorporação de novos tipos de acionamentos.

Os trabalhos futuros prevêem o desenvolvimento do software embarcado (camada de aplicação) para vários novos tipos de acionamentos, além dos já existentes no sistema, e a construção de módulos IP (*Intelectual Property*) específicos que se fizerem necessários para controle desses acionamentos. Esses módulos serão desenvolvidos através da linguagem de descrição de *hardware* VHDL e incorporados ao sistema.

7. Agradecimentos

Agradecemos à UFPB, ao Conselho Nacional de Pesquisa, CNPq – (Processo 485437/2007-4), e ao Programa de Educação Tutorial, Departamento de Modernização e Programas da Educação Superior, Secretaria de Educação Superior do Ministério da Educação - PET/DEPEM/SESu/MEC pelo auxílio financeiro.

8. Referências

[1] L.S. Machado, J.A.G. Lima, R.M. Morais, B.R.A. Sales, and T.K.L. Costa, "Real Environments Management Through Virtual Campus", Proceedings of International Conference on Engineering and Technology Education 2008 (Intertech'2008). , Santos Brasil, março 2008.

[2] Raghavan, P., Lad, A., and Neelakandan, S., Embedded Linux system design and development, Taylor & Francis Group, Washington USA, 2006.

[3] Altera Corporation, Nios II Processor Reference Handbook. Página web www.altera.com acessada em 10 de março de 2008, San Jose: Altera Corporation, 2007.

[4] Noergaard, T., Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers. Oxford: Elsevier, 2005.

[5] A. R. C. Souza, L.V. Batista, J. A. G. Lima, "Compressão sem Perdas de Sinais Eletrocardiográficos", In: CLAIB 2007 - IV Congreso Latinoamericano de Engenharia Biomedica, 2007, Margarita. CLAIB 2007 - IV Congreso Latinoamericano de Medicina Biomedica, 2007.

[6] Altera Corporation, Nios II Software Developer's Handbook. Página web www.altera.com acessada em 10 de março de 2008, San Jose: Altera Corporation, 2007. 620 p.

[7] Labrosse, J.J., MicroC/OS-II: The Real-Time Kernel, St. Louis: Focal Press, 2002.

[8] Dunkels, A., Minimal TCP/IP implementation with proxy support. Relatório Técnico T2001:20, SICS - Swedish Institute of Computer Science, Tese de mestrado, <http://www.sics.se/~adam/publications.html> acessado em 10 de março de 2008, Fevereiro de 2001.

[9] Altera Corporation, Nios II Hardware Development Tutorial. Página web <http://www.altera.com> acessada. em 10 de março de 2008, San Jose: Altera Corporation, 2007.