Universidade do Estado do Rio Grande do Norte
Universidade Federal Rural do Semi-Árido
Programa de Pós-Graduação em Ciência da Computação

Uma Arquitetura Parametrizável para Sistemas Embarcados: um Estudo de Caso

Dissertação de Mestrado

Lenardo Chaves e Silva

Orientador: Pedro Fernandes Ribeiro Neto, D.Sc.

Mossoró - RN
Fevereiro - 2011

Lenardo Chaves e Silva

Uma Arquitetura Parametrizável para Sistemas Embarcados: um Estudo de Caso

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-Graduação em Ciência da Computação da Universidade do Estado do Rio Grande do Norte em Associação Ampla com a Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Pedro Fernandes Ribeiro Neto, D.Sc.

Mossoró - RN
Fevereiro - 2011

Lenardo Chaves e Silva

Uma Arquitetura Parametrizável para Sistemas Embarcados: um Estudo de Caso

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade do Estado do Rio Grande do Norte (UERN) em Associação Ampla com a Universidade Federal Rural do Semi-Árido (UFERSA).

Lenardo Chaves e Silva

Banca Examinadora:

Prof. Pedro Fernandes Ribeiro Neto, D.Sc. (Orientador)
Universidade do Estado do Rio Grandes do Norte – UERN

Profa. Rossana Maria de Castro Andrade, D.Sc.
Universidade Federal do Ceará – UFC

Profa. Karla Darlene Nepomuceno Ramos, D.Sc.
Universidade do Estado do Rio Grande do Norte – UERN

Mossoró-RN, 23 de Fevereiro de 2011

Dedico esta dissertação a todos àqueles
que diante da possibilidade do fracasso
continuaram seguindo em frente.
E, à minha avó Anauta e ao meu primo
Lucas (*in memorian*), que não soube
superar às dificuldades da vida.

Agradecimentos

Esta Dissertação é fruto da união de várias famílias que, de uma forma ou de outra, contribuíram para o desenvolvimento desta pesquisa.

Assim, inicío meus agradecimentos a um ser humano único, minha noiva Monalisa Lima. Falar dela é muito fácil, porque ela é autêntica, irreverente e representa o significado da palavra "sentimento". Sua personalidade forte me conquistou e fez-me ver os outros rumos da vida. Obrigado minha linda por ser o que você é, e, apesar dos desencontros, sempre me segurar quando pareço estar caindo. Te AMO Muito. Zim, sua família é a minha família também, pessoas como D. Rita e Sr. Jorge são especiais e raras de encontrar... obrigado.

Aos meus pais, Marlene e José Neuton, a minha irmã, Patrícia, e a minha madrinha Raimundinha, sou grato pela pessoa que sou, pois são vocês a nascente do meu rio, mesmo ele correndo por caminhos próprios e seguindo seu destino. À minha família, meu humilde obrigado por apoiar todas as minhas decisões e me dar força quando preciso, amo vocês.

Ao Grupo de Engenharia de Software (GES) - UERN, por me acolher quando ainda era apenas um simples estudante, e hoje sou um Pesquisador. Esta família é guerreira e escreve a sua própria história. À todos os membros que fizeram, fazem e irão fazer parte desta família, meu sincero abraço e parabéns. Aproveitem o que muitos grupos não compartilham, isto é, a nossa união.

Ao meu amigo, irmão, às vezes confidente, mas acima de tudo meu mestre, opa..., é Doutor, Pedro Fernandes Ribeiro Neto. Obrigado por seus ensinamentos científicos e profissionais, sua vivência e sua paciência. Seu jeito às vezes manipulador, ou melhor articulado, mas sempre dedicado e atencioso, nos faz mostrar as estradas que nos levam à conquista. Por tudo isto, o máximo que posso fazer é agradecer... Obrigado!!!

Aos meus amigos de infância e adolescência que, apesar de distantes ou ausentes,

continuam sempre em minhas lembranças. Seja nos momentos de alegria ou de tristeza, vocês fazem parte desta conquista. Como também sou grato aos meus colegas e amigos que conquistei durante toda esta jornada, dentre estes Marcos Tullyo, Gracon, Cleone, Luiz Cláudio, Eduardo, João Borges, Fernando, Anchieta, João Paulo, Luís Carlos, Kléber Jacinto, Alexandre Ádames, Sebastião, Abraão, Welbb, Jomar, Ronnisson e Marlon. As coleguinhas Aislânia, Welliana, Luana, Natalyany e Jocileide. E aos professores e amigos Milton Mendes, Marcelino Pereira, Cicília Maia e Cláudia Ribeiro.

Agradeço, em especial, ao CNPq, pelo fomento a esta pesquisa, permitindo minha total dedicação ao programa de mestrado e possibilitando o desenvolvimento de uma pesquisa de qualidade.

E, por fim, mas certamente em primeiro lugar, agradeço a DEUS e seus discípulos, que pra mim, são todas as pessoas e famílias supracitadas. O Senhor realmente foi generoso comigo, pois um time como este não se encontra em nenhum lugar. Em nome do Senhor, obrigado à todos!

A sabedoria da vida não estar em fazer aquilo que se gosta, mas gostar daquilo que se faz.

Leonardo da Vinci

Resumo

A computação reconfigurável tem facilitado o desenvolvimento de projetos de sistemas embarcados complexos, uma vez que sua capacidade flexível possibilita a realização de mudanças nos projetos de hardware e software destes sistemas. Esta característica de permitir a reprogramação do hardware pelo usuário juntamente com o alto desempenho de suas plataformas fez despertar o interesse da comunidade científica e da indústria, como uma das principais alternativas à prototipagem rápida de sistemas embarcados.

Em virtude da complexidade destes sistemas, torna-se fundamental a utilização de uma metodologia consistente de desenvolvimento que possibilite especificar, desenvolver e depurar ambos os projetos de hardware e software. Para facilitar a visualização e a compreensão de suas propriedades estáticas e dinâmicas, tais sistemas devem fazer uso de diagramas e modelos executáveis baseados em técnicas formais que permitam a análise de seus comportamentos por meio de simulações.

Tendo em vista a demanda do mercado por soluções computacionais mais robustas e que possam agregar qualidade aos produtos e serviços fornecidos, esta dissertação propõe, através da integração e exploração do potencial das plataformas reconfiguráveis, uma arquitetura que permita o desenvolvimento de projetos de aplicações embarcadas e sua adaptação à outros domínios por meio do reuso. Para verificá-la e validá-la, um modelo em redes de petri coloridas hierárquicas foi elaborado e um estudo de caso foi desenvolvido no domínio de aplicação dos sistemas de segurança patrimonial.

Palavras-chave: Computação Reconfigurável; Hardware/Software *Codesign*; Arquitetura Parametrizável; Abordagem Formal; HCPN

Abstract

Reconfigurable computing has facilitated the development of complex Embedded Systems design, since its flexible capacity allows the realization of changes in hardware design and software of these systems. This feature that permits its user to reprogramming of hardware along with the high performance of their platforms aroused the interest of the scientific community and industry, as a major alternative to rapid prototyping of embedded systems.

Due to the complexity of these systems, it becomes essential to use a consistent development methodology that allows to specify, develop and debug both hardware and software designs. To facilitate the visualization and understanding of their static and dynamic properties, such systems should make use of diagrams and executable models based on formal techniques that allow analysis of their behavior by simulations. Therefore, this research suggests, by integrating and exploiting of the potential of reconfigurable platforms, providing an architecture that allow the development of the embedded project applications and their adaptation to other domains by reuse. To verify it and validate it, a model on Hierarchical Colored Petri Nets has been prepared and a study case has been developed in domain application of the patrimonial surveillance systems.

Given the market's demand for more robust computing solutions that can add quality to products and services, this research suggests, by integrating and exploiting of the potential of reconfigurable platforms, an architecture that allows the development of embedded applications and projects and their adaptation to other domains by reuse. To verify it and validate it, a model in hierarchical colored Petri nets was prepared and a case study was developed within the scope of asset security systems.

Keywords: Reconfigurable Computing; Hardware/Software *Codesign*; Parameterizable Architecture; Formal Approach; HCPN

Sumário

1 Introdução **1**

1.1 Contexto 1

1.2 Motivação 3

1.3 Objetivos 3

1.3.1 Objetivo Geral 3

1.3.2 Objetivos Específicos 4

1.4 Metodologia 4

1.5 Estrutura do Documento 6

2 *Computação Reconfigurável* **8**

2.1 Contextualização 9

2.2 FPGA 11

2.2.1 Componentes básicos 12

2.2.2 Tecnologias de Programação 14

2.3 Fluxo de Desenvolvimento de Projeto 16

2.4 *Trade-offs* 18

2.5 Aplicações 20

2.6 Conclusão 21

3 Hardware/Software *Codesign* **22**

3.1 Contextualização 23
3.2 Definição 24
3.3 Abordagens 25
3.4 Fluxo Geral de Desenvolvimento 26
3.5 Hardware/Software *Codesign* para Arquiteturas Reconfiguráveis 28
3.5.1 Desafios 29
3.6 Conclusão 30

4 Arquitetura Parametrizável para projetos de Sistemas Embarcados 31
4.1 Concepção 32
4.2 Arquitetura Proposta e Estudo de Caso 34
4.3 Plataforma Reconfigurável 37
4.3.1 FPGA *Cyclone II* 38
4.4 Ferramentas CAD 39
4.5 Fluxo de Desenvolvimento do Sistema 41
4.5.1 Especificação do Sistema 42
4.5.2 Particionamento 42
4.5.3 Co-Síntese 47
4.5.4 Integração 49
4.5.5 Avaliação e Testes Incrementais 51
4.6 Aplicação de Controle de Mensagens SMS 52
4.6.1 Terminal TC65 52
4.6.2 Comandos AT 53
4.6.3 J2ME 56
4.6.4 *MIDlet* 58
4.6.5 Arquitetura, Configuração e Execução 59
4.7 Conclusão 63

5 Uma abordagem de verificação e validação formal **65**

5.1 Introdução 66

5.2 Diagrama de Implantação 67

5.3 Modelo Formal 69

5.3.1 HCPN 69

5.3.2 *CPN Tools* 70

5.3.3 Modelo HCPN 71

5.4 Diagramas de Sequência de Mensagens 79

5.4.1 Diagrama de Sequência de Mensagens - CAP 80

5.4.2 Diagrama de Sequência de Mensagens - DE2 82

5.4.3 Diagrama de Sequência de Mensagens - TC65 84

5.5 Conclusão 85

6 Conclusões e Trabalhos Futuros **87**

6.1 Contribuições 88

6.2 Trabalhos Futuros 89

Referências Bibliográficas **90**

Lista de Figuras

2.1 Comparativo das plataformas computacionais 11

2.2 Estrutura interna de um FPGA 13

2.3 Estrutura dos Elementos Programáveis 15

2.4 Projeto de Hardware para FPGA 17

3.1 Fluxo Geral de Hardware/Software *Codesign* 27

4.1 Modelo Convencional de uma Central de Alarme 33

4.2 Modelo Proposto de uma Central de Alarme Parametrizável 35

4.3 Arquitetura Parametrizável aplicada à Central de Alarme Parametrizável . 36

4.4 Plataforma DE2 37

4.5 Arquitetura do FPGA *Cyclone II* 39

4.6 Fluxo de projeto *HW/SW Codesign* para a DE2 41

4.7 Configuração dos componentes de hardware definidos no *SOPC Builder* . 43

4.8 Esquemático do módulo de hardware construído no *Quartus II* 45

4.9 Esquema de pinagem do FPGA atribuído com a ferramenta *Pin Planner* integrada ao *Quartus II* 47

4.10 Resumo do relatório de síntese do hardware executado no *Quartus II* . . . 48

4.11 Esquema de *download* dos Projetos de Hardware e Software 50

4.12 Relação entre os Módulos de Hardware e Software 51

4.13 Terminal TC65 Siemens 52

4.14 Estrutura de uma linha de comando AT 54
4.15 Resposta à linha de comando AT 55
4.16 Ciclo de Vida de uma *MIDlet* 59
4.17 Arquitetura da Aplicação de Controle SMS 60
4.18 Fluxo de estados do Terminal TC65 com aplicação Java embarcada 63

5.1 Abordagem de verificação e validação da CAP 65
5.2 Diagrama de Implantação - Aplicação: Central de Alarme Parametrizável 68
5.3 Níveis de abstração de uma HCPN 70
5.4 Modelo HCPN da CAP 75
5.5 Sub-página da transição de substituição Sensoriar 76
5.6 Sub-página da transição de substituição Processar 77
5.7 Sub-página da transição de substituição Atuar 78
5.8 Diagrama de Sequência de Mensagens - Central de Alarme Parametrizável 81
5.9 Diagrama de Sequência de Mensagens - DE2 82
5.10 Diagrama de Sequência de Mensagens - TC65 84

Lista de Tabelas

2.1 *Trade-offs* das principais tecnologias de programação FPGA 16

2.2 Trabalhos Relacionados 20

Lista de Abreviaturas e Siglas

ASIC	*Application Specific Integrated Circuit*
API	*Application Programming Interface*
AT	*ATtention*
BSF	*Block Symbol File*
BDF	*Block Diagram Files*
BSP	*Board Support Packages*
CSD	*Circuit Switched Data*
CFTV	Circuito Fechado de TV
CAD	*Computer-Aided Design*
CPLD	*Complex PLD*
CR	Computação Reconfigurável
CFI	*Common Flash Interface*
CDC	*Connected Device Configuration*
CLDC	*Connected Limited Device Configuration*
CPN	*Colored Petri Nets*
DTE	*Data Terminal Equipment*
DE2	*Development and Education Board*
DSM	Diagrama de Sequência de Mensagens
DSP	*Digital Signal Processor*
DMA	*Direct Memory Access*
EDS	*Embedded Design Suite*
FPD	*Field-Programmable Devices*
FPGA	*Field-Programmable Gate Array*
GPRS	*General Packet Radio Service*
GPIO	*General Purpose Input/Output*
GPP	*General-Purpose Processor*
GUI	*Graphical User Interface*
GSM	*Global System for Mobile Communications*

HAL	*Hardware Abstraction Layer*
HDL	*Hardware Description Language*
HCPN	*Hierarchical Colored Petri Nets*
IMP-NP	*IMP Next Generation*
IrDA	*Infrared Data Association*
IMP	*Information Module Profile*
IDE	*Integrated Development Environment*
IRQ	*Interrupt Request Line*
ISR	*Interrupt Service Routine*
IP cores	*Intellectual Property cores*
JVM	*Java Virtual Machine*
J2ME	*Java 2 Micro Edition*
LE	*Logic Elements*
LUT	*LookUp Tables*
SML	*Standard ML*
MID	*Mobile Information Device*
MIDP	*MID Profile*
OEM	*Original Equipment Manufacturer*
PLL	*Phase-Locked Loop*
PLD	*Programmable Logic Device*
RTR	*Run-Time Reconfiguration*
SPLD	*Simple Programmable Logic Device*
SE	Sistemas Embarcados
SMS	*Short Message Service*
SOF	*SRAM Object File*
SoC	*System-on-Chip*
SOPC	*System-On-a-Programmable-Chip*
UML	*Unified Modeling Language*
URC	*Unsolicited Result Codes*
VHDL	*Very High Speed Integrated Circuits HDL*

Capítulo 1

Introdução

Nesta dissertação o foco é disponibilizar uma arquitetura parametrizável para as plataformas reconfiguráveis embarcadas. O processo de desenvolvimento de suas aplicações é realizado por meio da Metodologia de Hardware/Software *Codesign*. Buscando melhorar o projeto de tais aplicações, a abordagem utilizada desta metodologia teve algumas de suas etapas adaptadas. Com o objetivo de compreender o funcionamento e explorar o potencial dos dispositivos FPGA (*Field-Programmable Gate Array*) foi desenvolvido um estudo de caso. Para permitir a verificação e a análise do comportamento de tal arquitetura quando aplicada a um domínio, um modelo formal foi elaborado.

Assim, na Seção 1.1 é feita a contextualização deste trabalho. Na Seção 1.2 discute-se sobre a principal motivação que influenciou o desenvolvimento desta pesquisa. Em seguida, na Seção 1.3, são expostos os principais objetivos a serem alcançados com este trabalho, com sua metodologia sendo discutida na Seção 1.4. Por fim, na Seção 1.5 apresenta-se a estrutura organizacional deste documento.

1.1 Contexto

A crescente demanda do mercado tecnológico por soluções automatizadas e eficientes, principalmente em se tratando de dispositivos miniaturizados e sistemas com as mais diversas funcionalidades, fez despertar cada vez mais o interesse da comunidade científica por novas descobertas para os denominados Sistemas Embarcados (SE) (LI; YAO, 2003).

A forma como as pessoas interagem com e por meio destes dispositivos eletrônicos e computacionais muda conforme o avanço nas pesquisas relacionadas a este tipo de sis-

tema. As tecnologias de redes e comunicação, interface homem-máquina, otimização de recursos e mecanismos e técnicas de auto-aprendizagem por parte dos próprios dispositivos, são exemplos típicos de tais avanços.

Por meio destes avanços, uma gama de novos serviços computacionais tem sido disponibilizada nos mais diversos domínios de aplicação. Na medicina e na indústria, cirurgias controladas e linhas de produtos montadas com auxílio de braços robóticos; no processamento multimídia e no reconhecimento de padrões, técnicas aplicadas à imagem e vídeo utilizadas para extrair características em tempo-real; e, na automação e na domótica, dispositivos e equipamentos integrados controlados por uma central de monitoramento *on-line* são exemplos típicos de domínios beneficiados com a adoção de SE.

Almejando atender, facilitar e agilizar o processo de desenvolvimento destas aplicações surge, em consequência de tais avanços, uma nova categoria de dispositivos computacionais, os Dispositivos Lógicos Programáveis (SVENSSON, 2008). Estes, além de agregarem a alta velocidade de processamento proporcionado pelo hardware, possuem a flexibilidade do software quando há a necessidade de realizar mudanças no projeto da aplicação. Sua capacidade de ser reprogramado em campo pelo próprio usuário minimiza o tempo e o custo associado aos projetos de suas aplicações, uma vez que estes dispositivos não precisam passar por um processo de reestruturação física do hardware pelos seus fabricantes.

A área da computação destinada a realizar pesquisas para estes dispositivos é a Computação Reconfigurável (PLAKS et al., 2008), estando esta totalmente inter-relacionada com os Sistemas Embarcados. Uma de suas principais características é a reusabilidade, visto que a mesma estrutura de hardware pode ser configurada e adaptada para provê diferentes soluções computacionais, desde que atendam as exigências do usuário e o mesmo esteja apto a realizar tais alterações (HAUCK; DEHON, 2008).

Um dos dispositivos bastante utilizados na literatura para o desenvolvimento de protótipos de Sistemas Embarcados baseados na Computação Reconfigurável são os FPGA. Estes consistem de um arranjo de blocos lógicos configuráveis capazes de executar operações lógicas complexas (KHARAT; RAHANGADALE, 2009).

Pelo fato dos FPGA possuírem um bom custo-benefício em termos de capacidades fornecidas à elaboração de projetos de SE, a integração e a exploração de seu potencial juntamente com outros dispositivos de hardware convencionais, tais como sensores digitais e equipamento terminal de dados (*Data Terminal Equipment* - DTE), são, portanto, o foco desta dissertação.

1.2 Motivação

A Computação Reconfigurável, apesar do crescimento acentuado nas pesquisas, é um campo ainda em estágio de amadurecimento. Inúmeras questões em aberto estão sendo melhor investigadas, dentre elas como garantir o equilíbrio entre os parâmetros típicos de projetos (área, desempenho, flexibilidade e consumo de energia), bem como em relação a dependência de ferramentas avançadas para modelagem, simulação e análise de seus projetos, em virtude da complexidade dos mesmos (SVENSSON, 2008).

Em meio às mudanças periódicas de paradigmas computacionais, pesquisadores da área têm se interessado em buscar novos desafios. Exemplos recentes de temáticas estão relacionados à capacidade de adaptação dinâmica destas plataformas ao contexto do ambiente no qual estas são inseridas (PHILLIPS et al., 2009), como também novas abordagens de metodologias para o desenvolvimento de suas arquiteturas, tal como em três dimensões (3D) (DE PAULO; ABABEI, 2010).

Em se tratando de arquiteturas reconfiguráveis, a literatura demonstra ser uma tendência o uso destas como propostas de solução para vários domínios de aplicação que englobam projetos complexos de Sistemas Embarcados. Isto devido principalmente a capacidade flexível de mudanças no projeto de hardware de tais plataformas, uma vez que este pode ser adaptado pelo próprio usuário para diferentes tipos de aplicações, não comprometendo seu desempenho (IQBAL-CH et al., 2010).

Neste contexto, com esta dissertação busca-se explorar o potencial das plataformas reconfiguráveis por meio dos FPGA e disponibilizar uma arquitetura parametrizável. Para a validação, aplicou-se esta arquitetura ao domínio dos sistemas de segurança patrimonial. No entanto, esta pode ser adaptada a outros domínios que façam uso dos seus componentes de hardware, sendo permitido também integrar outros dispositivos compatíveis com suas interfaces de comunicação disponíveis.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo deste trabalho é disponilizar uma arquitetura parametrizável como alternativa de solução para vários domínios de aplicação que necessitem de uma plataforma embarcada robusta e de fácil uso. Esta permitirá que os próprios usuários desenvolvam

suas aplicações conforme suas necessidades, sem necessitar alterar sua estrutura física de hardware.

O desenvolvimento de tais aplicações seguirão a metodologia Hardware/Software *Codesign*, uma metodologia consistente e que se adequa às propriedades inerentes a um sistema embarcado complexo. Com esta, é possível especificar, desenvolver e depurar, simultaneamente, ambos os projetos de hardware e software.

Para validação da arquitetura parametrizável proposta, considera-se um estudo no contexto de uma Central de Alarme, como as comumente empregadas como solução para segurança de patrimônio. As ferramentas matemáticas usadas para desenvolver o modelo formal da arquitetura são as Redes de Petri Coloridas Hierárquicas (*Hierarchical Colored Petri Nets* - HCPN) e por meio do pacote de ferramentas computacionais *CPN Tools* serão realizadas simulações desse.

1.3.2 Objetivos Específicos

Para atingir o objetivo deste trabalho, as atividades a seguir devem ser realizadas:

- definir uma abordagem para a Metodologia Hardware/Software *Codesign* que será utilizada para guiar o processo de desenvolvimento de aplicações embarcadas que façam uso da arquitetura proposta;
- propôr adaptações na metodologia adotada para agregar artefatos importantes a algumas de suas etapas de forma a permitir uma melhor visualização e compreenção das estruturas dos projetos de sistemas embarcados a serem desenvolvidos com base na arquitetura proposta, em relação aos componentes de hardware e software empregados;
- disponibilizar um modelo formal, baseado na arquitetura definida, que possa refletir e servir como ferramenta de análise do comportamento dinâmico de tais sistemas.

1.4 Metodologia

A metodologia científica utilizada nesta dissertação pode ser resumida nos seguintes itens:

- Revisão Bibliográfica

  Inicialmente foi realizada uma revisão bibliográfica sobre computação reconfigurável no que diz respeito a arquitetura geral de suas plataformas, representada neste trabalho por um FPGA integrado a uma variedades de componentes de hardware. Além desta necessidade, foi preciso estudar uma metodologia que possibilitasse o rápido desenvolvimento de aplicações utilizando tais plataformas, onde se pudesse agregar características de melhor desempenho, maior flexibilidade e melhor custo benefício na elaboração e implementação de seus projetos.

- Adaptação da Metodologia Hardware/Software *Codesign*

  Nesta etapa, realizou-se o estudo da metodologia e dois artefatos foram incorporados às suas etapas de *Integração* e *Avaliação e Testes Incrementais*. Estes tiveram como objetivo facilitar, respectivamente, a visualização das propriedades estáticas e dinâmicas dos sistemas embarcados a serem desenvolvidos com base nesta metodologia. Isto possibilitará a identificação de erros nestes sistemas, complementando, portanto, a etapa de *Especificação*.

- Definição do Estudo de Caso

  A etapa de definição do estudo de caso foi aplicada ao domínio dos sistemas de segurança patrimonial. Com a utilização de dispositivos de baixo custo, fácil uso e comumente empregados nos sistemas tradicionais associados a este domínio pôde-se verificar a compatibilidade destes em relação à arquitetura parametrizável proposta nesta pesquisa. Com isto, foi definido que a aplicação para validação deste trabalho seria a criação de uma Central de Alarme Parametrizável. Com a integração de vários dispositivos, tal aplicação é controlada por FPGA. Isto possibilita realizar a qualquer momento adaptações nos projetos de hardware e software, para adequar a arquitetura proposta a outras aplicações.

- Sistematização do Modelo Formal do Estudo de Caso

  Por fim, uma abordagem formal foi definida para verificar a corretude da arquitetura parametrizável proposta. Tal modelagem foi adaptada ao estudo de caso desenvolvido e possibilitará que o usuário faça a análise do comportamento dos sistemas a serem desenvolvidos por meio de simulações. Este método formal para a arquitetura proposta pode ter sua estrutura personalizada conforme as necessidades do usuário.

1.5 Estrutura do Documento

Esta dissertação está organizada conforme os seguintes Capítulos:

- Capítulo 2

  No Capítulo 2 é realizada uma pesquisa acerca da Computação Reconfigurável, em especial de seu principal representante, os FPGA. Neste, são apresentados os principais conceitos e características, enfatizando a estrutura física e os mecanismos de funcionamento, bem como o fluxo de programação de tais dispositivos. Questões relacionadas a alguns de seus *trade-offs* são discutidas, além de elencados os mais importantes domínios de aplicação.

- Capítulo 3

  No Capítulo 3 são apresentadas todas as etapas do fluxo de desenvolvimento da Metodologia Hardware/Software *Codesign*. Esta metodologia é indispensável para o desenvolvimento conjunto dos projetos de hardware e software e lidar com a natureza complexa inerente aos Sistemas Embarcados.

- Capítulo 4

  No Capítulo 4 enfatiza-se o foco desta pesquisa, ou seja, o desenvolvimento de uma arquitetura parametrizável que possibilite a criação de sistemas embarcados robustos e personalizáveis. Logo, neste Capítulo é disponibilizada uma arquitetura parametrizável, tendo seu potencial demonstrado por meio do estudo de caso desenvolvido, quando guiada por uma metodologia adequada para o desenvolvimento de seus projetos. O estudo de caso mencionado trata-se do projeto de uma Central de Alarme Reconfigurável no domínio dos sistemas de segurança patrimonial.

- Capítulo 5

  No Capítulo 5 é discutida a abordagem de verificação e validação formal para a arquitetura proposta, a fim de assegurar que a mesma está correta e atende as necessidades do usuário. O modelo formal foi desenvolvido utilizando Redes de Petri Coloridas Hierárquicas (HCPN). O esquema para a modelagem dos diagramas que descrevem os artefatos construídos para a metodologia Hardware/Software *Codesign*, bem como para o modelo HCPN é apresentado e detalhado.

- Capítulo 6

  E, por fim, no Capítulo 6 são apresentadas as conclusões sobre esta dissertação, sendo discutida a experiência adquirida com o seu desenvolvimento e que servirá como alicerce para os prováveis trabalhos futuros decorrentes da mesma.

Capítulo 2

Computação Reconfigurável

Os Sistemas Embarcados (SE), imperceptivelmente, vêm sendo utilizados todos os dias para realizar uma ampla variedade de tarefas específicas. Dentre suas aplicações encontra-se na literatura, por exemplo, trabalhos relacionados às Redes de Sensores Sem Fio (BORGES NETO, 2009), Biomedicina (SHIH et al., 2010), Algoritmos e Reconhecimento de Padrões (LU et al., 2010), Sistemas em Tempo-Real(ANDERSON; KHOO, 2009) e Processamento Multimídia (ZLOKOLICA et al., 2009). Nesta categoria de sistemas há uma estreita correlação entre hardware e software, visto que os projetistas podem usufruir da velocidade de processamento inerente ao hardware e caráter altamente flexível do software. Dentre algumas alternativas de dispositivos que faz uso deste tipo de solução, destaca-se o FPGA.

Logo, este Capítulo visa compreender detalhadamente os FPGA, onde na Seção 2.1 é feita uma contextualização destes, buscando classificá-los com base em suas particularidades. Em seguida, na Seção 2.2, além da concepção de FPGA descreve-se sobre sua composição e a forma como estes armazenam e executam seus programas, característa esta mencionada na literatura como tecnologias de programação. Na Seção 2.3 aborda-se sobre o fluxo de projeto básico a ser seguido para o desenvolvimento de aplicações. A Seção 2.4 realiza um levantamento acerca dos principais *trade-offs* ao adotar-se FPGA como uma alternativa de solução para um determinado problema. No que diz respeito aos vários domínios de aplicações, a Seção 2.5 faz um breve levantamento. Finalizando este Capítulo com a Seção 2.6, é discutida a situação do mercado mundial destes dispositivos em relação aos seus principais fabricantes, bem como a justificativa do real crescimento no uso dos FPGA como solução para automatização de inúmeras tarefas diárias. Tarefas estas executadas manualmente ou mesmo de forma automática, mas sem nenhum

dinamismo e consideração do contexto do ambiente de atuação.

2.1 Contextualização

Em meio as diversas áreas de pesquisa da Computação destacam-se os Sistemas Embarcados, devido sua participação na evolução tecnológica. Uma de suas sub-áreas que vem ganhando bastante espaço na literatura é a Computação Reconfigurável (CR). A característica marcante da CR é a habilidade de unir o poder de realizar computação em hardware de forma a aumentar o desempenho, mantendo consideravelmente a flexibilidade semelhante a uma solução em software (HAUCK; DEHON, 2008).

Com esta particularidade, a CR permite aos projetistas desenvolver soluções computacionais automatizadas, mais robustas, adaptáveis e que agreguem questões relativas à reusabilidade. A reusabilidade é proporcionada pela capacidade de reprogramação do hardware, uma vez que uma mesma arquitetura de hardware pode ser utilizada para implementar várias soluções de propósito geral (HAUCK; DEHON, 2008). Em consequência, surgem outros benefícios como a redução dos custos de desenvolvimento e manutenção, bem como do prazo de entrega do produto ao mercado consumidor (BECKER et al., 2008).

Com a Computação Reconfigurável é possível também realizar alterações nas instruções do software, reduzindo a necessidade de reestruturação e/ou reprogramação do hardware (FERLIN, 2008). Havendo tal necessidade de mudanças no projeto de hardware, a CR possibilita realizar este procedimento em tempo de execução. Porém, isto pode acarretar uma sobrecarga de tempo bastante considerável em virtude da funcionalidade e complexidade do projeto, o que torna-se necessário um melhor gerenciamento do mesmo (HAUCK; DEHON, 2008).

Há um pouco mais de duas décadas haviam duas plataformas tradicionais computacionais de implementação e execução de algoritmos: uma cuja solução é baseada em nível de hardware, denominada ASIC (*Application Specific Integrated Circuit*); e, uma segunda, com ênfase em microprocessadores programados em software, conhecida como GPP (*General-Purpose Processor*). Com a necessidade de atender a demanda de mercado por novas soluções que agregassem um maior potencial aos seus produtos em termos de desempenho e flexibilidade e com um menor tempo de produção, acrescentou-se a tal taxonomia os Dispositivos Lógico Programável (*Programmable Logic Devices* - PLD) (BECKER et al., 2008)(HAUCK; DEHON, 2008).

Neste Capítulo é realizado um breve paralelo entre as plataformas ASIC, GPP e PLD, sendo os GPP representados pelo DSP (*Digital Signal Processor*) e os PLD pelo FPGA.

Os FPGA são uma especificação avançada ou evolução dos tradicionais PLD, pertencendo a uma classe de dispositivos conhecida como Dispositivos Programáveis em Campo (*Field-Programmable Devices* - FPD) (ALSOLAIM, 2002).

Em relação aos ASIC, pode-se dizer que estes são chips projetados para executar computação específica em hardware de forma rápida e eficiente, mas com a limitação de tornar-se inalterável após sua fabricação, pois modificar o circuito significa re-projetar e re-fabricar o chip (COMPTON, 2003)(IQBAL-CH et al., 2010). Tal solução, em muitos casos, é financeiramente inviável visto que o custo do projeto é muito alto e exige um esforço maior em seu processo de produção (HAUCK; DEHON, 2008). Por este motivo, os ASIC são mais adequados para projetos cuja produção seja em grande escala, como em uma linha de produtos comercializada mundialmente. Normalmente, a implementação dos ASIC da-se por meio do uso de uma Linguagem de Descrição de Hardware (*Hardware Description Language* - HDL).

Em se tratando dos DSP, como dito anteriormente, estes são uma especificação dos Processadores de Propósito Geral, sendo disponíveis nos mais diversos tamanhos e características. Estes são baseados no modelo de arquitetura sequencial, onde se busca, processa e salva informações, sendo sua implementação diretamente em linguagem Assembly (ORACLE, 2010b) ou de alto nível, como C (ANSI, 1999), que deve ser compilada e convertida em instruções de máquina. Os DSP, no entanto, determinam o poder funcional de uma aplicação, já que seus recursos estão limitados ao conjunto de instruções codificadas (ALSOLAIM, 2002). Alterações nas instruções de software podem modificar a funcionalidade de uma operação sem a necessidade de alterar qualquer parte ou módulo dos recursos de hardware (IQBAL-CH et al., 2010). Apesar dessa maior flexibilidade em relação aos ASIC, os DSP perdem desempenho em suas operações já que torna-se necessário executar um conjunto de instruções adicionais para realizar cada operação, aumentando a sobrecarga e o consumo de energia (COMPTON, 2003).

Quanto aos FPGA, estes possuem os benefícios de um melhor rendimento e maior flexibilidade no projeto como um todo, quando comparado, respectivamente, aos DSP e ASIC. Isto os torna uma interessante alternativa de solução para projetos de caráter computacional dinâmico e sistemas digitais (HARTENSTEIN, 2001). Os FPGA preenchem a lacuna entre hardware e software, com maior desempenho em relação aos DSP, e, ao mesmo instante, com a facilidade de serem reprogramados para implementar uma gama

de tarefas e a um baixo custo, quando comparado aos ASIC (COMPTON, 2003)(IQBAL-CH et al., 2010).

Na Figura 2.1 é mostrada a comparação das plataformas ASIC, DSP e FPGA, no que diz respeito às suas capacidades de desempenho e flexibilidade. Enquanto os ASIC possuem desempenho acentuado em relação aos DSP, estes são bem mais flexíveis às mudanças de projeto. Já os FPGA são um misto de flexibilidade e desempenho, o que os tornam dispositivos ideais para projetos de aplicações que exigem tais propriedades.

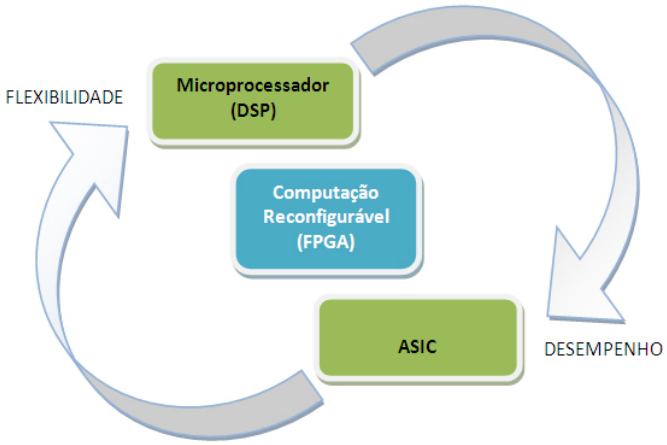


Figura 2.1: Comparativo das plataformas computacionais. Fonte: Adaptada de (HARTENSTEIN, 2001)

Os algoritmos em FPGA podem ser codificados tanto em linguagem de baixo nível (como Assembly) quanto em alto nível (como C ou C++) similarmente aos DSP. Estes também podem ser programados em HDL, tal como VHDL, caracterizando ainda mais a flexibilidade de tais dispositivos. Como elemento indispensável desta pesquisa, este será minusciosamente detalhado na Seção 2.2.

2.2 FPGA

Em DRIMER (2008) é estabelecido, informalmente, o conceito de FPGA como sendo dispositivos semicondutores genéricos compostos por blocos funcionais interconectados que podem ser programados, e reprogramados, desenvolvidos com o intuito de executar funções lógicas descritas pelo usuário.

Os PLD, como uma generalização destes dispositivos, consistem de planos lógi-

cos com portas lógicas *AND* e *OR* e *flip-flops*, que se utilizam de lógica sequencial para, por meio da combinação de tais elementos, realizarem inúmeras operações lógicas (SVENSSON, 2008). Estes têm sido largamente utilizados por questões como rápido tempo de manufatura, baixo custo e, em especial, flexibilidade e facilidade de mudanças em seus projetos (CHEN et al., 2003). Os FPD, como um avanço dos PLD, mudaram o processo de projetar hardware digital, já que – diferentemente das tecnologias anteriores que utilizavam grandes quantidades de chips com integração em pequena escala (*Small-Scale Integration* - SSI) – a partir da década de 1990 praticamente todos os dispositivos passaram a ter alta densidade, como os processadores, a memória, os contadores, os registradores e os decodificadores (BROWN; ROSE, 1996)(HAUCK; DEHON, 2008).

Portanto, FPGA é um FPD moderno, no qual substitui os planos lógicos *AND* e *OR* por um arranjo de blocos lógicos configuráveis, com capacidade de executar um conjunto de operações lógicas complexas (KHARAT; RAHANGADALE, 2009).

2.2.1 Componentes básicos

De acordo com a Figura 2.2, um FPGA é tipicamente composto por: blocos lógicos, representados pelos quadriláteros localizados na região central do substrato do FPGA; interconexões, representadas pelos losangos; e, blocos de entrada e saída (E/S), representados pelos quadriláteros localizados na região periférica, estes atuando como interfaces de comunicação.

Os blocos lógicos, comumente chamados de elementos lógicos (*Logic Elements* - LE) são interligados por meio de interconexões programáveis pelo usuário, permitindo a comunicação com hardware externo utilizando os blocos de E/S (BROWN; ROSE, 1996)(BARR, 1999).

Os LE que formam a matriz no FPGA são de tipos potencialmente diferentes, incluindo, por exemplo, unidades lógicas aritméticas, memória e multiplicadores. Alguns FPGA utilizam como fonte de armazenamento de dados uma espécie de memória denominada de *LookUp Tables* (LUT) (KUON et al., 2008). Uma LUT em um FPGA é construída com o objetivo de implementar funções lógicas, utilizando para isto, algumas entradas, em geral quatro ou cinco, para gerar uma única saída. Assim, o tamanho das LUT é inversamente proporcional a quantidade necessária destes elementos para a implementação de circuitos lógicos (HAMBLEN; FURMAN, 2001).

O tamanho e a complexidade associada aos LE definem a granularidade dos FPGA.

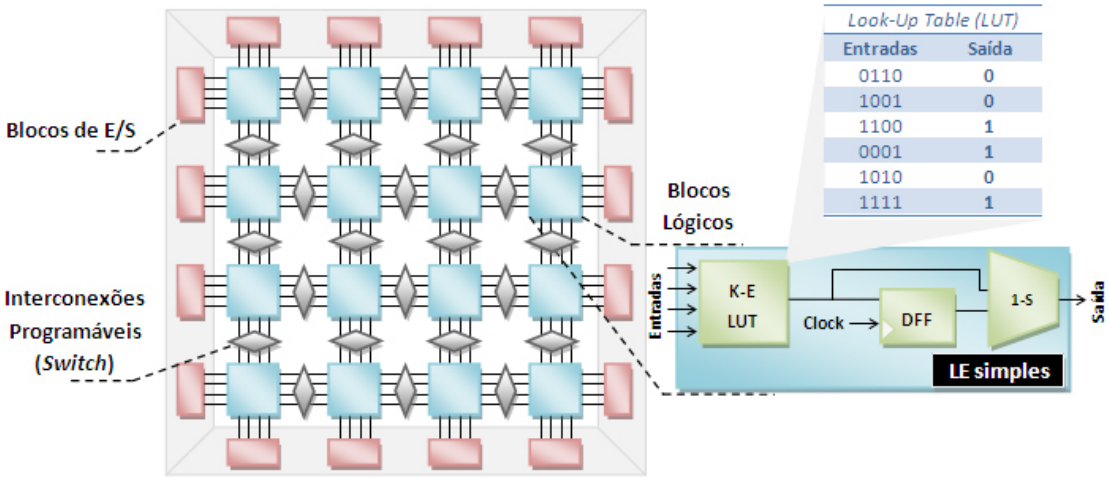


Figura 2.2: Estrutura interna de um FPGA. Fonte: Adaptada de (KUON et al., 2008)

Um FPGA contendo LE pequenos e simples com a implementação de lógica aritimética básica, caracteriza-o como de granularidade fina, e com LE grandes e complexos implementando megafunçoes lógicas, de granularidade grossa (BARR, 1999)(AOUDNI et al., 2006).

As interconexões entre os LE e os blocos E/S representam a programação em um FPGA, sendo basicamente compostas por fios e *switches* programáveis. O processo de estabelecer as interconexões é denominado de Roteamento. No roteamento é definida a posição relativa dos canais em relação ao posicionamento dos LE, considerando a conexão entre estes, bem como a quantidade de fios necessários para cada canal, formando o circuito projetado pelo usuário. A forma como se dá o roteamento pode influenciar em dois fatores importantes no projeto de um FPGA, que são o desempenho e o consumo de energia (KUON et al., 2008).

Os blocos E/S são componentes de extrema importância em um FPGA, pois são estes que determinam, por meio dos padrões de interfaces de comunicação, seu nível de conectividade e integração com outros dispositivos periféricos externos. Para cada componente periférico de E/S definido no projeto, é fundamental especificar os pinos que devem estar habilitados para realizar tal comunicação. Estes componentes podem ser especificados como sendo somente de entrada ou saída, bem como bidirecionais, atuando por meio de um *buffer* (CASTRO, 2007). Em alguns casos, estes elementos ocupam um espaço considerável em uma placa de circuito impresso, como no caso da placa DE2 (*Development and Education Board*), cujo fabricante é a *Altera Corporation* (ALTERA, 2010a).

2.2.2 Tecnologias de Programação

A programação de um FPGA baseia-se no controle das interconexões programáveis, sendo tal programação fundamentada em algumas tecnologias básicas, que dependendo de suas características podem influenciar em sua arquitetura lógica programável (KUON et al., 2008). Quanto as abordagens tecnológicas, descritas na literatura, utilizadas na programação dos FPGA pode-se referir a EPROM, EEPROM (OHSAKI et al., 1994), SRAM (VIOLANTE et al., 2003) (REDDY et al., 2005), *FLASH* (SPEERS et al., 1999) (STERPONE et al., 2010) e *Anti-fuse* (WANG et al., 2000) (KANG et al., 2007). Em KUON et al. (2008) é feita uma breve descrição das três principais tecnologias, que são:

- *SRAM*: tendo como componente básico células de memória estática distribuídas por todo FPGA, como mostrado na Figura 2.3a, estas fornecem ao usuário a capacidade de configuração do dispositivo por meio de um fluxo de bits. Em geral, usam-se tais células para definir as linhas de seleção para os multiplexadores que direcionam os sinais de interconexão, bem como para armazenar os dados (valores lógicos 0 ou 1) nas LUT. O maior potencial da tecnologia SRAM é permitir realizar alterações no projeto por meio da reprogramação, por um número indefinido de vezes, reduzindo os custos do projeto. Vale ressaltar que esta tecnologia dissipa maior quantidade de energia comparada as demais tecnologias (IQBAL-CH et al., 2010) e utiliza o padrão CMOS[1], se beneficiando do alto poder de integração em área reduzida.

- *FLASH*: esta faz uso da programação por meio de portas flutuantes, utilizando estas como *switches*, como ilustrado na Figura 2.3b. A tecnologia de programação em *flash* injeta carga energética em uma "porta" que se encontra ("flutua") acima do transistor, acarretando na definição de sua mais importante característica, isto é, a não volatilidade. Com isso, os dados de configuração são armazenados e recarregados automaticamente ao ligar o dispositivo, eliminando a necessidade de recursos extras para realizar tal tarefa. Em se tratando desta tecnologia, pode-se dizer que seu projeto torna-se mais complexo, no sentido de que é necessário controlar a tensão que deve ser repassada a porta flutuante, mantendo-a suficiente baixa para prevenir danos ao dispositivo. Diferentemente da SRAM, sua programação pode ser realizada por um número finito de vezes e requer um processo CMOS não convencional.

- *Anti-fuse*: o funcionamento deste elemento programável é inverso ao de um fusível

[1]CMOS: *Complementery Metal Oxide Semiconductor* (TSAI; WANG, 2004)

convencional. Sua ativação dá-se quando seu estado normal de alta impedância é modificado para baixa impedância devido à aplicação de uma tensão elevada, culminando no seu rompimento e fechamento do circuito (KANG et al., 2007). Basicamente existem duas abordagens desta tecnologia, a abordagem Dielétrica e a *Metal-To-Metal*. Ambas abordagens partem do mesmo princípio, porém, na primeira a resistência gerada é em torno de cinco a seis vezes maior do que na segunda, devido esta última utilizar uma camada com material isolante (como o óxido de silício) entre as duas camadas de metais, conforme Figura 2.3c. As vantagens dessa tecnologia estão relacionadas ao tamanho reduzido da área necessária na pastilha de silício e menor resistência e capacitância, permitindo incluir mais *switches* por dispositivo em relação às demais tecnologias, além da sua alta velocidade e baixo consumo de energia. Em contrapartida, é exigida uma maior programação dos transistores que lhes abastecem com as grandes correntes. Seu caráter de programação permanente garante a redução dos custos do projeto e execução imediata do dispositivo, uma vez que o FPGA recebe o fluxo de bits referente à sua programação no momento que o mesmo é ligado. No entanto, esta tecnologia não permite a reprogramação. Assim como a tecnologia *flash*, seu processo de manufatura também não segue o real padrão CMOS.

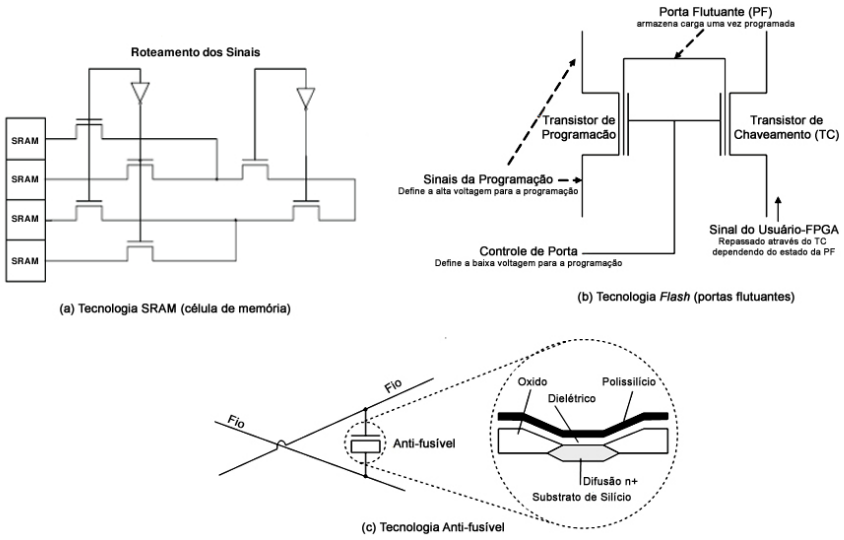


Figura 2.3: Estrutura dos elementos programáveis para FPGA conforme tecnologia. Fonte: Adaptada de (BROWN; ROSE, 1996) (KUON et al., 2008)

Para facilitar o entendimento das tecnologias supracitadas, a Tabela 2.1 relata as principais características associadas a cada uma delas.

Tabela 2.1: *Trade-offs* das principais tecnologias de programação FPGA. Fonte: Adaptada de (KUON et al., 2008)

	SRAM	***FLASH***	***Anti-Fuse***
Volátil?	Sim	Não	Não
Reprogramável?	Sim	Sim	Não
Limite de Programação	Indefinido	Limitado	Único
Área de Silício (transistores necessários)	Grande (6-12)	Moderada (1)	Baixa (0)
Processo de Manufatura CMOS?	Convencional	Específico	Especial
Programável no sistema	Sim	Sim	Não
Programável em campo	100%	100%	>90%

O uso de uma determinada tecnologia para programação de um FPGA é dependente da proposta de solução e necessidades globais atreladas ao projeto que está sendo desenvolvido. Em alguns casos, é necessário disponibilizar alternativas de tais tecnologias para atender a diferentes escopos, como no caso da placa DE2 que fornece a flexibilidade ao usuário de programar o FPGA incorporado tanto por meio da tecnologia SRAM como da *Flash* (ALTERA, 2010a).

2.3 Fluxo de Desenvolvimento de Projeto

O processo de desenvolvimento de aplicações utilizando FPGA, por sua natureza é complexo. Isto remete ao uso de ferramentas CAD (*Computer-Aided Design*) para auxiliar o projetista em todas as fases do projeto, principalmente na criação e configuração do hardware. A utilização de tais ferramentas permite reduzir o esforço necessário para implementar este tipo de aplicação, bem como o tempo e o custo de desenvolvimento (HAUCK; DEHON, 2008).

As ferramentas CAD podem apoiar o desenvolvedor em algumas tarefas como no fornecimento da entrada do projeto (*Design Entry*), otimização lógica, ajuste do dispositivo, simulação e configuração (BROWN; ROSE, 1996). Além destas, tarefas como a criação de megafunções lógicas especificadas pelo usuário e geração de arquivos em HDL podem ser automatizadas.

Em KILTS (2007) e IQBAL et al. (2009), é apresentado o fluxo de projeto de hardware para FPGA, análogo ao descrito em (BARR, 1999). Este será detalhado conforme o

fluxograma visualizado na Figura 2.4.

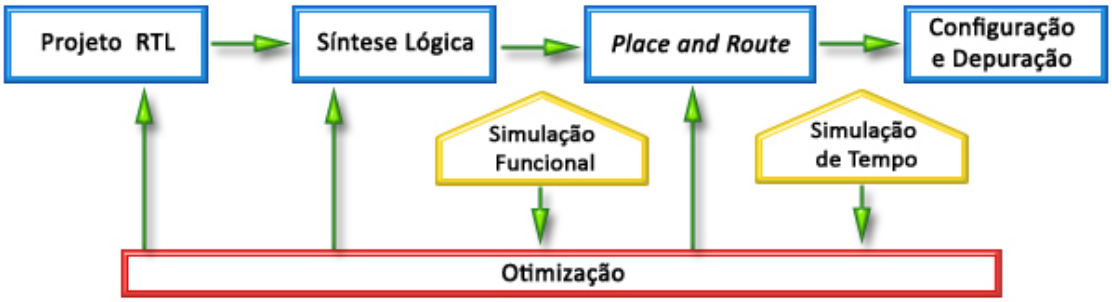


Figura 2.4: Projeto de Hardware para FPGA. Fonte: Adaptada de (KILTS, 2007)

No fluxo do processo ilustrado na Figura 2.4, a etapa de *Projeto RTL* descreve, em alto nível de abstração, os circuitos digitais à nível de registrador (*Register Transfer Level* - RTL). Há alguns anos, esta etapa era denominada de *Entrada do Projeto* e destinava-se a descrever manualmente, pelo projetista do sistema, os componentes e o comportamento do hardware por meio de uma HDL e/ou diagramação do esquemático do circuito lógico. Com o advento da *Síntese Lógica* tornou-se possível converter automaticamente, por meio de ferramentas CAD, a descrição RTL em um *netlist* a nível de portas lógicas codificado em HDL ou diagrama esquemático. Porém, é aconselhado após esta etapa que o projetista realize a *Simulação Funcional* para identificar possíveis erros de lógica na operação do circuito projetado. A etapa seguinte utiliza o *netlist* gerado na etapa de *Síntese Lógica* para criar o *layout* do circuito e ajustar o projeto para que se possa utilizar os elementos lógicos do dispositivo. Isto é feito colocando cada elemento lógico em seu devido local (*place*) e estabelecendo as rotas (*route*) de suas interconexões. Esta etapa, conhecida como *Place and Route*, visa proporcionar melhorias relativas ao desempenho do dispositivo. Na etapa de *Simulação de Tempo*, são analisadas questões de desempenho e restrições de tempo em relação às funcionalidades e configurações especificadas na programação do dispositivo. Dependendo dos resultados, o desenvolvedor pode retornar a etapa de otimização para realizar devidas adequações no projeto. Na etapa de *Otimização* são consideradas as especificações e as restriçoes em relação à fatores, como o tempo, a área e o consumo de energia do dispositivo, de forma que os objetivos do projeto sejam melhor atendidos. A última etapa deste processo é a *Programação e Depuração* do projeto no dispositivo FPGA. Para isto, é gerado o arquivo de configuração que deverá ser carregado e executado no próprio hardware para depuração e verificação do projeto (BROWN; ROSE, 1996) (HAMBLEN; FURMAN, 2001) (KILTS, 2007) e (IQBAL et al., 2009).

Na etapa de *Síntese Lógica*, o *netlist* é a representação intermediária do hardware, ou seja, arquivo de texto que descreve a representação do circuito em termos de portas lógicas e suas interconexões, que independe do dispositivo para o qual é criado. Com a etapa de *Place and Route*, é feito o mapeamento das estruturas lógicas descritas no *netlist* para os elementos lógicos, interconexões programáveis e pinos de E/S do dispositivo. O artefato desta fase é o *bitstream*, isto é, arquivo em formato binário que deve ser carregado no dispositivo FPGA, proporcionando a execução do projeto de hardware definido. No que diz respeito ao *bitstream*, a estrutura final gerada ao término do processo pode ser reutilizada em distintas aplicações, desde que seus requisitos sejam completamente atendidos e o mesmo possa seja compatível com o dispositivo empregado (BARR, 1999) (IQBAL et al., 2009).

Uma observação importante em se tratando de projetos de hardware para FPGA é que apesar do seu processo de desenvolvimento seguir uma sequência de passos básicos, este dispositivo é bastante flexível, no sentido de possibilitar que o desenvolvedor faça as devidas adequações para atender aos objetivos do projeto que se pretende desenvolver. Na literatura este tipo de procedimento é encontrado, por exemplo, nos trabalhos de ALSOLAIM (2002) MURPHY et al. (2003) KANG et al. (2007) BUENO (2007) KUON et al. (2008) DRIMER (2008) GONZALEZ et al. (2008) e JIN (2009).

2.4 *Trade-offs*

Analogamente a qualquer outra tecnologia, os FPGA lidam com a mesma situação de conflito no tocante as vantagens e desvantagens proporcionadas em detrimento de sua adoção como proposta de solução para um determinado problema. Devido à este fator preponderante que implica diretamente nos resultados obtidos ao término de um projeto com caráter computacional envolvendo hardware e software, torna-se indispensável o conhecimento de seus principais *trade-offs*.

Assim, após uma análise minuciosa na literatura (BROWN; ROSE, 1996) (CHOI et al., 2003) (STEPHENSON, 2005) (MESQUITA et al., 2006) (MEHTA, 2006) (HANSEN, 2007) (KANG et al., 2007) (CARVALHO, 2007) (CASTRO, 2007) (CAMPOS, 2008) (LI et al., 2007) (CHOWDHURY, 2007) (MUKILAN, 2008) (KUON et al., 2008) (FERLIN, 2008) (GONZALEZ et al., 2008) (RODRÍGUEZ et al., 2009) (AHMED et al., 2009) (GUHA; AL-DABASS, 2010), identifica-se uma predominância em relação a estes fatores, que serão descritos a seguir:

- Vantagens:

  - *Flexibilidade*: adaptações no projeto podem ser realizadas sem a necessidade de fabricação de um novo dispositivo, já que o próprio desenvolvedor pode reprogramá-lo;
  - *Custo*: gastos adicionais não recorrrentes da engenharia do produto são minimizados;
  - *Desempenho*: em virtude de parte do processamento ser realizado no hardware, a velocidade das operações é aumentada;
  - *Reusabilidade*: o uso de linguagem de descrição de hardware possibilita a portabilidade da aplicação para outros dispositivos, independentemente de fabricante;
  - *Programação*: possibilidade de implementação em linguagem de mais alto nível, tais como C e C++, comparado a Assembly;
  - *Prototipação*: a prototipagem rápida de aplicações, auxilia na correção de erros e na redução dos custos;
  - *Ferramentas CAD*: com auxílio de tais ferramentas na criação e no desenvolvimento dos projetos a complexidade é reduzida, o que contribui, também, para a redução do tempo de desenvolvimento.

- Desvantagens:

  - *Roteamento*: a forma como é realizado o roteamento das interconexões entre os LE, pode influenciar na eficiência do dispositivo;
  - *Área e Atraso*: relacionados à granularidade dos elementos lógicos, quanto menores mais longos são os atrasos, devido a maior propagação do sinal e menor desempenho. No contrário, quanto maiores tais LE, mais espaço no dispositivo é requerido;
  - *Energia*: o consumo de energia é determinado pelo nível de paralelismo implementado em hardware. Quanto menor o paralelismo, maior é o gasto energético para o dispositivo. A proporção do nível de complexidade das funções lógicas também é um fator que interfere no consumo de energia.

2.5 Aplicações

Como dito no início deste Capítulo, uma gama de aplicações estão sendo e podem ser desenvolvidas nos mais diferentes domínios, utilizando como solução os Dispositivos Lógicos Programáveis, em especial, os FPGA. O potencial de tais dispositivos proporciona, não somente ao desenvolvedor, mas também ao usuário final, diversos benefícios, tais como reusabilidade, reconfiguração, programação em campo (local de atuação do dispositivo), redução nos custos e no tempo de desenvolvimento de projetos. Melhor desempenho e maior flexibilidade que uma solução baseada exclusivamente em software ou hardware, são as principais particularidades destas plataformas.

Na Tabela 2.2 são expostos alguns trabalhos relacionados que fazem uso de FPGA para solucionar ou mesmo melhorar inúmeros problemas inerentes à Computação e suas áreas afins.

Tabela 2.2: Trabalhos Relacionados

Domínio	**Objetivo**	**Referência**
Algoritmos	Implementação de algoritmos em grafos em hardware reconfigurável (FPGA) para acelerar a execução.	(AHMED et al., 2009)
Processamento Paralelo	Implementação de um multiprocessador aplicado à robótica sobre uma plataforma FPGA.	(CASTRO, 2007)
Inteligência Computacional	Controlador de Temperatura Difuso (*Fuzzy*) para aplicações industriais pelo método de composição Max-Min.	(ISLAM et al., 2007)
Segurança da Informação	Sistema de decriptação baseado em FPGA para vídeos compostos por sequenciamento de imagens com taxa de 6 fps utilizando o padrão AES (*Advanced Encryption Standard*) com 128 bits.	(BHARGAV et al., 2008)
Bioinformática	Sistema Embarcado para monitor holter de captura, armazenamento e análise de sinais ECG utilizando recursos de hardware e software para obtenção de diagnósticos.	(FARIAS, 2010)
Reconhecimento de Padrões	Arquitetura baseada em FPGA para extração de características de imagem em tempo-real usando o método estatístico *Gray Level Co-occurrence Matrix* (GLCM).	(BARIAMIS et al., 2004)
Processamento de Imagens	Arquitetura multi-pipeline para sistema de reconhecimento de face em FPGA para minimizar o tempo de processamento, mantendo a eficiência do reconhecimento.	(VISAKHASART; CHITSOBHUK, 2009)
Jogos	Projeto de reestruturação do sistema computacional Atari 2600 usando técnicas de projeto de sistemas digitais modernos na plataforma FPGA.	(BEER, 2007)
Sistema em Tempo-Real	Plataforma configurável de hardware e software para processamento de vídeo embarcado em tempo-real.	(BEN ATITALLAH et al., 2008)

2.6 Conclusão

Muitas tarefas executadas em nosso cotidiano são realizadas de forma manual e quando automatizadas são impostas algumas restrições, tais como impossibilidade de reprogramação e caráter reativo dos sistemas. Além disso, existem as atividades de nível crítico ou que devem ser efetuadas em ambientes inacessíveis ou inóspitos, onde torna-se necessário o uso de alguma tecnologia que garanta sua execução e eficiência, evitando perdas materiais, financeiras e, principalmente, humanas.

Neste contexto, os FPGA podem atuar de forma satisfatória, visto que há total possibilidade de incorporar as suas aplicações mecanismos de inferência, agregando a tais dispositivos a capacidade de auto-adaptação às mudanças de contexto ocorridas no ambiente no qual esteja inserido. Desta forma, considerando a contínua evolução tecnológica tanto na indústria eletrônica, quanto nas de hardware e software, o domínio de aplicação dos FPGA é cada vez mais amplo.

Em relação ao mercado global de tais dispositivos, recentes pesquisas têm sido desenvolvidas para análise e previsão de seu crescimento, como em *"Market Share: ASIC and FPGA/PLD Application Market and Vendor Analysis, 2008"*, realizada pela Gartner Dataquest (GARTNER, 2009), e *"Programmable Logic ICs Market Shares and Forecasts, Worldwide, 2010 to 2016"*, pela Wintergreen Research (WINTERGREEN, 2010). No entanto, para acesso a íntegra de tais relatórios, há a necessidade de aquisição financeira junto aos institutos de pesquisas, cujos valores são relativamente baixos para as empresas devido aos benefícios proporcionados, mas exorbitantes para membros da sociedade acadêmica, correspondendo a US$1,295.00 e US$3,500.00 dólares, respectivamente.

De acordo com o resumo dos relatórios supracitados, como também dos relatórios anuais das principais fabricantes destes dispositivos, a *Xilinx, Inc.* [2] (XILINX, 2010b) e a *Altera Corp.* [3] (ALTERA, 2010b) dominam o mercado de Dispositivos Lógicos Programáveis, somando no total 87% deste. Em seu relatório a *Altera Corp.* menciona que 77% das vendas realizadas em 2009 são relacionadas aos FPGA.

[2]Fundada e incorporada na Califórnia (EUA) em 1984, a *Xilinx, Inc.* projeta, desenvolve e comercializa plataformas programáveis, além de oferecer serviços de projeto, treinamento do cliente, engenharia em campo e suporte técnico.

[3]Fundada em 1983, a *Altera Corporation* é uma empresa global de semicondutores, servindo mais de 12.000 clientes com os segmentos de Telecomunicação e *Wireless*, Automação Industrial, Militar e Automotiva, Redes, Computador e Armazenamento, e outros segmentos do mercado.

Capítulo 3

Hardware/Software *Codesign*

A indústria fabricante de tecnologia tem investido fortemente no desenvolvimento de aplicações com alta disponibilidade de recursos, como comunicação sem fio, processamento multimídia, paralelo e concorrente, baixo consumo de energia e, em especial, eficiência no uso desses recursos. Com o advento das Arquiteturas Reconfuguráveis, os Sistemas Embarcados de processamento digital tornam-se cada vez mais complexos, pois novos métodos de implementação de algoritmos para o processamento e o mapeamento de tarefas entre hardware e software são exigidos. Desta forma, abordagens diferenciadas são requeridas para guiar o desenvolvimento destes sistemas.

Este Capítulo foca na apresentação da Metodologia Hardware/Software *Codesign*, trazendo na Seção 3.1 sua contextualização, onde é feito um paralelo entre a indústria de Sistemas Embarcados e a necessidade de se ter um fluxo de projeto condizente com a realidade destes sistemas. Visto tal necessidade, a Seção 3.2 define a abordagem Hardware/Software *Codesign*. Na Seção 3.3 são elencadas e descritas técnicas utilizadas nas mais diversas abordagens de Hardware/Software *Codesign* existentes. Para compreender o fluxo de projeto em questão, a Seção 3.4 disserta sobre suas principais etapas. De forma a atingir um dos objetivos desta pesquisa, a Seção 3.5 relaciona Hardware/Software *Codesign* como a metodologia ideal para o desenvolvimento de sistemas baseados no conceito de Computação Reconfigurável, elencando também seus principais desafios. E, por fim, na Seção 3.6 são debatidas algumas considerações sobre este Capítulo.

3.1 Contextualização

A evolução tecnológica da indústrica eletrônica, principalmente no tocante a miniaturização de seus componentes, tem possibilitado agregar um maior número de elementos em uma menor área geométrica das pastilhas de silício. Aliado a isto, verifica-se um aumento do potencial computacional destes *chips* para suportar em um único dispositivo a integração de complexas plataformas, e, assim, atender às exigências de desempenho no processamento de tarefas de aplicações mais robustas. Na literatura, este tipo de dispositivo é conhecido como *System-on-Chip* (SoC) (JERRAYA; WOLF, 2005).

Em se tratando de Sistemas Embarcados, estes são sistemas micro-processados que combinam hardware e software para executar uma funcionalidade específica (LI; YAO, 2003). Nestes, o software oferece a flexibilidade necessária e ajuda na redução dos custos do sistema, enquanto o hardware aumenta consideralvelmente o seu desempenho (CHATHA; R., 2004). Apesar de SE terem recursos ainda reduzidos – comparados a sistemas computacionais de propósito geral – com hardware projetado sob medida, dimensões reduzidas, independência operacional e controle de execução em tempo-real e reativo, estes são suficientes para suprir suas necessidades (BORGES NETO, 2009) (SILVA, 2009).

Com o aumento da capacidade de integração dos SE a outros sistemas maiores, bem como da quantidade de recursos disponíveis para o qual estes são idealizados, os SE estão se tornando cada vez mais complexos. Devido à sofisticação dos SoC, os SE possibilitaram incluir uma ou mais unidades de processamento para execução paralela de tarefas concorrentes, exigindo de seus projetistas uma abordagem diferenciada de desenvolvimento que permitisse a realização conjunta de ambos os projetos de hardware e software. Tal abordagem ficou conhecida como Hardware/Software *Codesign* (JERRAYA; WOLF, 2005).

Com a introdução de novas plataformas de hardware reconfigurável, em meio a demanda do mercado consumidor de dispositivos computacionais e aparelhos eletrônicos, esta necessidade de uma metodologia que guie o processo de concepção e implementação de aplicações embarcadas torna-se uma obrigação. Isto, advindo das exigências atuais destes sistemas, tais como alto desempenho, flexibilidade para uso multifuncional e eficiência no consumo de energia (PLAKS et al., 2008).

3.2 Definição

Como dito anteriormente, os modernos Sistemas Embarcados necessitam mais ainda da integração entre subsistemas de hardware e software de forma a possibilitar o processamento distribuído e a heterogeneidade das tarefas. A necessidade de alto desempenho associado a estes sistemas é fruto da exigência e demanda do mercado, especialmente devido à sua competitividade. Com isso, as abordagens tradicionais focadas no desenvolvimento isolado de somente hardware ou software já não condizem com a realidade deste mercado, que a cada dia torna-se mais dinâmico, nem com as necessidades e evolução das pesquisas científicas, principalmente quando se trata de Computação Reconfigurável (JERRAYA; WOLF, 2005).

Neste contexto, surge a metodologia de desenvolvimento Hardware/Software *Codesign*, que segundo DE MICHELL e GUPTA (1997) *"representa a união dos objetivos a nível de sistema, explorando a sinergia entre hardware e software por meio do projeto concorrente"*. NIEMANN (1998) afirma que o interesse nesta abordagem tem crescido devido à introdução das ferramentas disponibilizadas pela CAD, expondo que um dos principais objetivos do *Codesign* é minimizar o *time-to-market* ao mesmo tempo em que reduz o esforço dos profissionais de desenvolvimento e os custos do projeto.

A rapidez e a flexibilidade na correção e na otimização de um sistema, como também a capacidade de reutilização e adaptação deste para novas aplicações são outros benefícios proporcionados pelo uso das ferramentas CAD. No entanto, isto somente é possível quando se tem uma equipe de desenvolvimento com conhecimentos técnicos especializados e experiente, pois estes são aspectos que influenciam diretamente na produtividade da equipe.

SCHAUMONT (2010) traz duas definições para a metodologia Hardware/Software *Codesign*, uma mais tradicional e outra com um caráter mais moderno, onde se considera a diversificação dos conceitos e as especificidades dos termos "Hardware" e "Software". Na tradicional, Hardware/Software *Codesign* é definido como "*a concepção colaborativa de componentes de hardware e software em um esforço único de projeto*". Na moderna, "*o particionamento e o projeto de uma aplicação em termos de componentes fixos e flexíveis*". Logo, é possível notar a substituição do termo "Hardware" por "fixo" e "Software" por flexível.

De acordo com WOLF (2003) Hardware/Software *Codesign* permite que sistemas computacionais embarcados possam evoluir mais rapidamente, uma vez que o desenvolvi-

mento cooperativo dos projetos de hardware e software é aperfeiçoado com os avanços em diversas disciplinas, como Arquitetura de Computadores, Sistemas em Tempo-Real e Engenharia de Software. Podendo-se acrescentar ainda a este grupo, a disciplina de Projetos de Sistemas Digitais.

Portanto, segundo JERRAYA e WOLF (2005) Hardware/Software *Codesign* melhora o processo de projetar SoC, aumentando a qualidade e a confiabilidade do hardware e do software, permitindo a verificação inicial, reusabilidade e interoperabilidade. Com isto, usando plataformas reconfiguráveis como componentes de sistemas embarcados torna-se possível oferecer produtos mais diferenciados, fazendo face a uma série de novos desafios técnicos enfrentados pelos projetistas destes sistemas.

3.3 Abordagens

De acordo com NOGUERA e BADIA (2002), CHATHA e R. (2004), QU et al. (2008) e BERTHELOT et al. (2008) existem várias abordagens de Hardware/Software *Codesign*. Dentre estas abordagens, pode-se caracterizá-las, por exemplo, pelo nível de granularidade do particionamento, hardware alvo, suporte a reconfiguração em tempo de execução, políticas de escalonamento, realocação de hardware e software, fluxo de dados e minimização dos atrasos. Tais abordagens utilizam diferentes técnicas de particionamento e escalonamento de tarefas em hardware e software, tais como:

- **Particionamento:**
  - *Granularidade:* a especificação do sistema é particionada a níveis de blocos básicos (granularidade fina) ou a nível de processos ou tarefas (granularidade grossa);
  - *Temporal:* o algoritmo de particionamento constrói uma lista contendo todas as tarefas sistematicamente organizadas em ordem de prioridade. Calcula-se o tempo necessário para execução de cada tarefa, e, em alguns casos, subdivide-as em subtarefas para execução em determinados ciclos de *clock* do processador.

- **Escalonamento:**
  - *Natureza dos Eventos:* estático, quando as tarefas são executadas em uma

ordem pré-fixada, e dinâmico, em que a sequência das tarefas é estabelecida em tempo de execução;

- *Prefetching:* busca prever as próximas tarefas a serem executadas com base na análise do contexto, antecipando a reconfiguração do sistema.

A maioria das abordagens de Hardware/Software *Codesign* fazem uma mistura de técnicas de acordo com as características do projeto do sistema. Nem sempre torna-se necessário o uso de todas as técnicas supracitadas, podendo utilizar-se de algoritmos específicos de particionamento e escalonamento. Muitas vezes, os projetistas fazem adaptações para seus projetos com base em técnicas existentes ou mesmo elaboram técnicas próprias para a solução de seus problemas.

3.4 Fluxo Geral de Desenvolvimento

Segundo JERRAYA e WOLF (2005), o processo Hardware/Software *Codesign* depende do nível de abstração em que o projeto inicia, sendo, portanto, necessário definir inicialmente um fluxo de projeto simplificado.

Com base nas abordagens Hardware/Software *Codesign* apresentadas em (NIEMANN, 1998), (DE-MICHELI et al., 2002), (JERRAYA; WOLF, 2005), (QU et al., 2008) e (HAUCK; DEHON, 2008), foi sistematizado um processo subdividido em cinco etapas, como demonstrado na Figura 3.1, sendo tais etapas descritas a seguir:

- *Especificação:* visa descrever o comportamento do sistema em termos de suas funcionalidades, apresentando seus requisitos e suas restrições por meio da identificação de suas necessidades e objetivos de reconfiguração, como flexibilidade e escalabilidade. Tal especificação, a priori, é parcialmente incompleta e requer a estreita colaboração de todos os participantes do projeto;

- *Particionamento:* caracterizado pela exploração e definição da arquitetura genérica do sistema. Nesta etapa, é feito o mapeamento dos comportamentos do sistema, que devem ser especificados em módulos, onde são agrupadas as funcionalidades que devem ser implementadas em hardware ou em software. Neste instante, as especificações de hardware e software são melhor definidas para possibilitar e garantir a funcionalidade do sistema, quando integrados tais módulos. Uma interface de comunicação e sincronização entre os módulos de hardware e software é necessária,

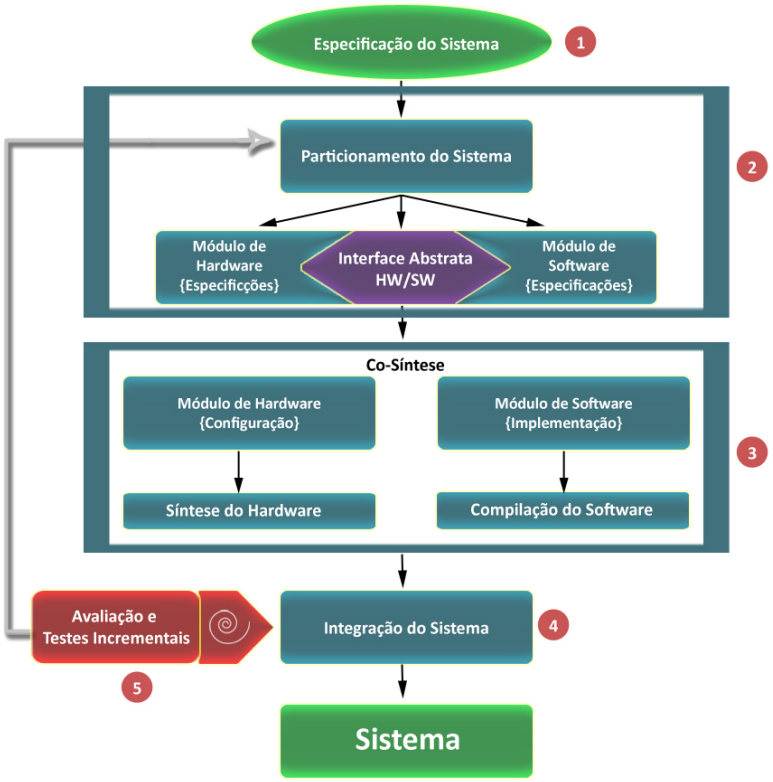


Figura 3.1: Fluxo Geral de Hardware/Software *Codesign*.

seja esta desenvolvida pelo usuário ou abstraída por meio do uso das ferramentas CAD utilizadas. Tal interface inclui mecanismos de comunicação para permitir a troca de dados entre os processadores embarcados. Com isto, a abordagem torna-se ainda mais flexível e operacional, porém, muito mais complexa;

- *Co-Síntese:* esta etapa idealiza ambos os projetos de hardware e software, por meio dos processos de sintetização e compilação, respectivamente. O termo síntese de hardware refere-se ao ato de converter todo o esquema de lógica digital do projeto de hardware em um *netlist* de componentes a nível de registrador, utilizando para isto uma linguagem de descrição de hardware, como, por exemplo, **VHDL** ou **Verilog**. Também é possível fazer uso de uma linguagem a nível de máquina, como **Assembly**. No que diz respeito ao software, linguagens de alto nível como C/C++ podem ser utilizadas na geração de códigos executáveis. Toda a etapa deve ser cuidadosamente acompanhada pelo usuário, uma vez que a forma como as ferramentas CAD realizam tais atividades, pode influenciar no desempenho e no consumo de energia do sistema. Neste caso, o usuário pode realizar algumas otimiza-

ções nos projetos para melhorar estas questões;

- *Integração:* após geração dos módulos de hardware e concepção dos módulos de software, deve-se realizar a integração dos mesmos, abstraindo toda a complexidade inerente ao sistema e tornando-o totalmente funcional;

- *Avaliação e Testes Incrementais:* como forma de verificar e avaliar as funcionalidades implementadas para o sistema, é indispensável a execução desta etapa para a realização de testes, onde são analisadas as entradas e saídas de dados no sistema em tempo de execução. A partir deste momento, é possível retornar a etapa de particionamento seja para corrigir erros detectados ou agregar novas funcionalidades ao sistema.

A abordagem Hardware/Software *Codesign* descrita é um processo genérico que permite a realização de várias adaptações. Tais adaptações não são possíveis ou são difíceis de serem feitas quando se utiliza de processos clássicos em que hardware e software são projetados separadamente. Um exemplo típico é a integração de novos dispositivos de hardware visando atender futuras e constantes demandas do cliente.

3.5 Hardware/Software *Codesign* para Arquiteturas Reconfiguráveis

Mediante o exposto em NOGUERA e BADIA (2002), o nível de complexidade do *codesign* é relativo ao contexto e ao domínio o qual é aplicado. Como a Computação Reconfigurável requer projetos e metodologias apropriadas para lidar com suas particularidades, como capacidade de reconfiguração de seus componentes em tempo de execução (do inglês, *Run-Time Reconfiguration* - RTR) e curto *time-to-market* (WIANGTONG et al., 2005) (BERTHELOT et al., 2008), isto torna o processo de Hardware/Software *Codesign* uma atividade cada vez mais prática. No sentido de adaptar suas abordagens para tratar os problemas advindos da adoção deste grupo de plataformas em diversas soluções computacionais.

> *Uma plataforma é uma arquitetura pré-concebida que os desenvolvedores podem usar para construir sistemas para uma ampla faixa de aplicações, [...] podendo ser um FPGA ou qualquer arquitetura contendo processadores, lógica customizável e interconexões de hardware* (WOLF, 2003).

Em se tratando de Computação Reconfigurável, FPGA é a plataforma que vem sendo bastante utilizada não somente pela comunidade científica, mas também pela indústria de Circuitos Integrados. Sua aceitação no mercado tem crescido rapidamente, em especial, devido esta satisfazer muitos requisitos dos Sistemas Embarcados modernos, tais como alto desempenho, flexibilidade e consumo energético, além de permitir a prototipagem rápida destes sistemas (BERTHELOT et al., 2008).

Neste sentido, FPGA

> *[...] parece ser a plataforma ideal para o qual a cosíntese foi criada, já que esta combina uma matriz de lógica programável com uma ou mais CPUs [unidades de processamento], possibilitando aos projetistas separar tal lógica da implementação dos processadores, meta esta implícita nas técnicas de particionamento* (WOLF, 2003).

3.5.1 Desafios

A exploração da integração cada vez maior de dispositivos computacionais com nosso mundo físico cotidiano, identifica a era da Computação Ubíqua (COULOURIS et al., 2007). Segundo LYYTINEN e YOO (2002) esta une duas grandes áreas: a Computação Móvel e a Computação Pervasiva, por meio de suas respectivas peculiaridades, ou seja, a mobilidade dos serviços computacionais em larga escala e a capacidade de auto-adapatação dos sistemas ao ambiente no qual está inserido.

Para WEISER (1991), considerado um dos precursores da Computação Ubíqua, o termo *"Ubíquo"* fornece à computação a capacidade de estar em todo e qualquer lugar de forma onipresente para o usuário. Dentre seus desafios, MENDES NETO e RIBEIRO NETO (2010) relaciona questões referentes a heterogeneidade dos dispositivos, a escalabilidade do sistema, a forma de como integrar e melhor usufruir de tais tecnologias, como construir dinamicamente modelos computacionais por meio das percepções e mudanças de contexto ocorridas no ambiente e de forma transparente para o usuário, além de considerar também questões de segurança e privacidade.

Em relação a Computação Reconfigurável, os desafios englobam aspectos de reconfiguração de suas plataformas em tempo de execução, sem afetar o desempenho e o consumo de energia do sistema em meio às restrições temporais (PLAKS et al., 2008), bem como a forma de garantir a confiabilidade e a qualidade dos serviços destes sistemas (JERRAYA; WOLF, 2005).

Portanto, ao aliar os desafios técnicos da Computação Ubíqua com os da Computação Reconfigurável, um leque de oportunidades é aberto para modificações no processo de Hardware/Software *Codesign*, dentre estas, JERRAYA e WOLF (2005) cita: a abstração da interface entre hardware e software orientado a requisitos de qualidade de serviços (como comunicação entre subsistemas, largura de banda, *jitter* e comunicação confiável); antecipação da verificação inicial da arquitetura do sistema com o foco na plataforma alvo de hardware; disponibilização de uma arquitetura de interface abstrata altamente reconfigurável e parametrizada para que os desenvolvedores possam otimizá-la de acordo com a aplicação; e, por fim, a elaboração de resumos de tais interfaces para facilitar o diálogo entre os membros da equipe de projeto, independente do setor da empresa ao qual cada membro é vinculado.

3.6 Conclusão

Neste Capítulo buscou-se enfatizar a real importância de uma metodologia especialmente adaptada para o desenvolvimento de aplicações que utilizem a Computação Reconfigurável como um motor para o processamento de tarefas, utilizando como solução o alto desempenho proporcionado pelo hardware e a flexibilidade inerente a um software.

O fato de sua adoção para a concepção simultânea dos projetos de hardware e software para a arquitetura proposta nesta pesquisa é justificado pelas particularidades já mencionadas das plataformas reconfiguráveis. A possibilidade de reuso e a readaptação de projetos anteriormente desenvolvidos para novas aplicações, bem como a prototipagem rápida de sistemas, que agiliza e antecipa a correção de erros, foram outros fatores considerados.

Capítulo 4

Arquitetura Parametrizável para projetos de Sistemas Embarcados

O valor agregado aos produtos e serviços oferecidos à sociedade é consequência direta dos avanços tecnológicos que surgem como um elemento para promover a integração entre áreas que até pouco tempo desenvolviam suas atividades isoladamente. Neste contexto, a tecnologia dá novos rumos a estas áreas que por si só não teriam perspectivas na resolução de determinados problemas, mas que quando somadas por meio da tecnologia, potencializam seus recursos e abrem as portas para novos desafios. Vários são os exemplos e casos de sucessos advindos da relação interdisciplinar promovida por meio da tecnologia, como no caso das áreas militar, medicina, engenharia, matemática, física, agricultura, industrial e computação.

O processo de integrar tecnologias para propor soluções para os mais diversos domínios de aplicação requer o uso de metodologias de desenvolvimento apropriadas e eficientes, especialmente quando se tratam de projetos de Sistemas Embarcados com foco em Arquiteturas Reconfiguráveis. O fato destes sistemas exigirem uma estreita correlação entre hardware e software para satisfazer seus objetivos torna-o ainda mais complexo. Como na maioria destes sistemas há uma interdependência de atividades, faz-se necessário que parte destas sejam executadas em hardware e outras em software, para garantir sua capacidade de alto desempenho. O aspecto flexível de tais sistemas é advindo das próprias características das plataformas reconfiguráveis, aliada a um processo de desenvolvimento consistente.

Mediante o exposto, o objetivo neste Capítulo é usufruir dos recursos disponibilizados pelas plataformas reconfiguráveis, conforme descrito no Capítulo 2, e por meio da

metodologia Hardware/Software *Codesign*, apresentada no Capítulo 3, propor um modelo de arquitetura parametrizável e eficaz para o desenvolvimento de aplicações de Sistemas Embarcados.

Como forma de avaliar o comportamento das aplicações desenvolvidas utilizando a arquitetura proposta, foi desenvolvido no domínio de aplicação de sistemas de segurança patrimonial, um estudo de caso. O mesmo foi disponibilizado como alternativa ao modelo das Centrais de Alarmes atualmente empregadas no setor, sendo projetado por meio da integração de plataformas computacionais reconfiguráveis com dispositivos de segurança eletrônica comumentes utilizados. O processo usado para orientar o desenvolvimento de aplicações com base na arquitetura proposta engloba etapas com atividades concomitantemente realizadas nos projetos de hardware e software.

Assim, neste Capítulo é descrito o Modelo Convencional das Centrais de Alarme na Seção 4.1. A abordagem da arquitetura proposta referente a Central de Alarme Parametrizável é apresentada na Seção 4.2. As Seções 4.3 e 4.4, relatam, respectivamente, sobre a plataforma de hardware reconfigurável utilizada e as ferramentas de apoio ao desenvolvimento dos projetos de hardware e software. Na Seção 4.5 é mostrado o passo-a-passo das etapas da metodologia de Hardware/Software *Codesign* usada para sistematizar o projeto do sistema de controle da arquitetura parametrizável. A Seção 4.6 complementa a implementação da arquitetura proposta para a Central de Alarme Parametrizável, com a aplicação de envio de mensagens SMS (*Short Message Service*), onde se descreve sobre as tecnologias e os equipamentos utilizados. E, por fim, na Seção 4.7, são feitas as considerações finais.

4.1 Concepção

Ao vivenciar a era da ubiquidade, onde os ambientes frequentados estão repletos de dispositivos computacionais espalhados e sensíveis a presença humana, a sociedade se depara com uma situação aparentemente desconfortável se visualizada como uma forma de invasão de privacidade.

Contudo, sob o ponto de vista da segurança pessoal e patrimonial proporcionada por toda a tecnologia por trás destes dispositivos e equipamentos, seus benefícios são bem maiores. Isto porque os índices de criminalidade contra o patrimônio no Brasil, referentes ao período de 2008 à 2009, ainda são muito altos, como demonstrado em relatórios estatísticos disponibilizados pela Secretaria Nacional de Segurança Pública - Ministério

da Justiça/Brasil e apresentados no Anuário do Fórum Brasileiro de Segurança Pública (FBSP, 2010).

Em meio a esta sensação de insegurança por parte da sociedade brasileira, a demanda por sistemas de segurança e vigilância automatizados e totalmente integrados tem aquecido o mercado de automação residencial e de desenvolvimento de equipamentos e soluções tecnológicas para o setor. Como exemplo deste tipo de sistemas, destacam-se o Circuito Fechado de TV (CFTV), por meio de câmeras de vigilância integradas a uma central de monitoramento em tempo-real; Centrais de Alarme, por meio do uso de sensores e atuadores; e os Sistemas de Controle de Acesso Pessoal, baseados nas tecnologias biométricas e de Identificação por Radiofrequência (*Radio Frequency IDentification* - RFID).

Na Figura 4.1 é demonstrada a organização das centrais de alarme comumente empregadas no setor de segurança de patrimônio residencial. Nesta, é possível perceber que o cenário consiste na integração de sensores, atuadores, equipamentos de hardware e um sistema de software.

Figura 4.1: Modelo Convencional de uma Central de Alarme

Em geral, estas centrais utilizam sensores de presença e barreira para controlar o acesso, bem como de fumaça para detectar possíveis focos de incêndio no ambiente monitorado. A comunicação destes dispositivos com a central de monitoramento da empresa provedora do serviço é realizada por meio do controle de sinais elétricos capturados de acordo com a ocorrência dos eventos no ambiente monitorado. Tais dispositivos são interligados a um equipamento de hardware. Neste caso, existe uma placa de circuito impresso, projetada e configurada para gerenciar os dispositivos, obtendo sinais fornecidos por cada sensor como fonte de entrada e enviando, como saída, uma mensagem à central

de monitoramento, indicando qual o imóvel e o tipo de alerta emitido. O tipo de alerta emitido é estabelecido por meio da atribuição e da configuração de setores na placa em questão. Sendo o setor o local onde se conecta o fio referente ao canal de dados de cada sensor. Quanto aos atuadores, estas centrais adotam uma sirene para a emissão de alertas sonoros que despertem a atenção da vizinhança. O próprio usuário operador do sistema é considerado um atuador, visto que o mesmo deverá solicitar uma equipe de apoio conforme o tipo de alerta recebido, para que esta possa se deslocar até o local do acontecido.

No entanto, este modelo apesar de cumprir satisfatoriamente as exigências mínimas de segurança e fornecer um serviço convincente ao cliente, traz consigo várias limitações que influenciam diretamente no desempenho do sistema, como: lógica da solução baseada na interpretação de sinais elétricos, o que acarreta na necessidade de usuários capacitados para operar e gerenciar o sistema, além da contratação de outros serviços de comunicação por parte do cliente, como, por exemplo, telefonia fixa ou internet; restrições no tratamento do sinal dos sensores, já que a central apenas recebe e envia os sinais digitais sem que seja realizada qualquer tipo de análise dos mesmos; e, por fim, a incapacidade de inserir inferência a tais equipamentos, uma vez que estes não disponibilizam qualquer tipo de recurso que possibilite dotá-los de inteligência computacional.

Desta forma, este modelo pode ser melhorado em vários aspectos. Para isto, dispositivos, tecnologias e técnicas de programação mais robustas e sofisticadas na resolução de problemas computacionais podem ser utilizadas. Logo, visando a qualidade, a eficiência e a melhoria dos serviços providos pelas centrais de alarmes residenciais convencionais, é proposta nesta pesquisa uma abordagem tecnológica diferenciada para suprir algumas deficiências do modelo atual, tais como limitação na interpretação dos sinais elétricos advindos dos sensores e incapacidade de reprogramação do sistema.

4.2 Arquitetura Proposta e Estudo de Caso

Objetivando fornecer uma maior flexibilidade e melhor desempenho às Centrais de Alarmes Convencionais, esta pesquisa sistematizou uma solução arquitetural embarcada e de caráter genérico, por meio do uso de plataformas reconfiguráveis integradas à dispositivos de baixo custo e fácil uso, como, por exemplo, sensores digitais. Este projeto busca contribuir não só com o mercado de automação, mas também com a sociedade fornecendo alternativas mais eficientes para a segurança e o controle patrimonial público e pessoal.

O cenário físico para o modelo proposto é apresentado na Figura 4.2.

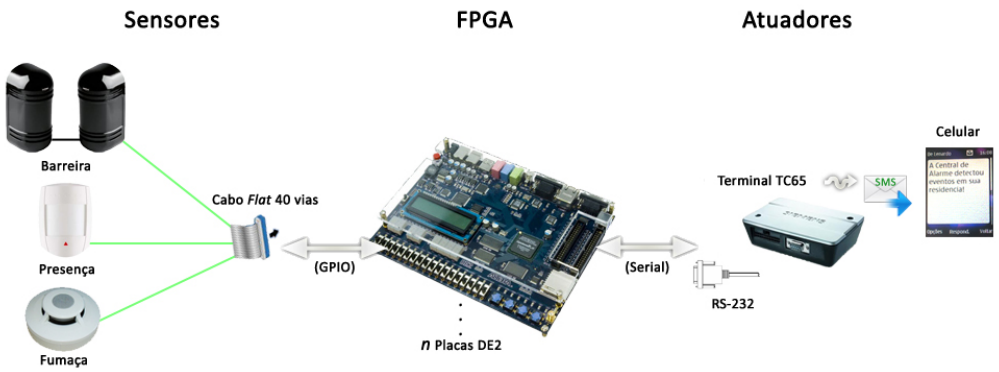


Figura 4.2: Modelo Proposto de uma Central de Alarme Parametrizável

Como ilustrado na Figura 4.2, o modelo da Central de Alarme Parametrizável (CAP), estudo de caso deste trabalho, consiste de uma ou mais placas Altera DE2, um conjunto de sensores e do equipamento terminal TC65 Siemens. A placa Altera DE2 contem como dispositivo reconfigurável o FPGA *Cyclone II* integrado a outros dispositivos para suporte a uma gama de periféricos de entrada e saída. Esta funciona como elemento central da arquitetura parametrizável em substituição à placa de circuito impresso encontrada nas Centrais de Alarme Convencionais. A mesma será apresentada na Seção 4.3. Como fonte de entrada de dados para esta, utilizou-se os mesmos tipos de sensores digitais adotados nas Centrais de Alarmes Convencionais, visto seu baixo custo e usabilidade. Tais sensores fornecem sinais para o FPGA via GPIO (*General Purpose Input/Output*), possibilitando-o realizar uma análise e tratamento destes sinais, para ajudá-lo no processo de tomada de decisão. Como saída, além dos canais acoplados à DE2, tais como LCD de 16x2 (ColunasxLinhas), conjunto de LED, *display* de sete segmentos, dentre outros, foi feita a integração de um terminal GSM/GPRS (*Global System for Mobile Communications*/*General Packet Radio Service*) via interface de comunicação serial, para realizar o envio de mensagens SMS, como resposta aos eventos detectados pelos sensores no ambiente controlado.

Com a idéia de integrar todas estas tecnologias e automatizar grande parte das tarefas necessárias para o monitoramento e a segurança de um ambiente, atribui-se ao modelo proposto características autônomas nas atividades de identificação dos sinais capturados pelos sensores, por meio das interrupções de hardware, e encaminhamento das mensagens SMS, em resposta aos eventos detectados, reduzindo ainda mais a necessidade de intervenção humana no que diz respeito a operação do sistema.

O modelo final da Arquitetura parametrizável elaborada e aplicada a CAP é mostrado

na Figura 4.3.

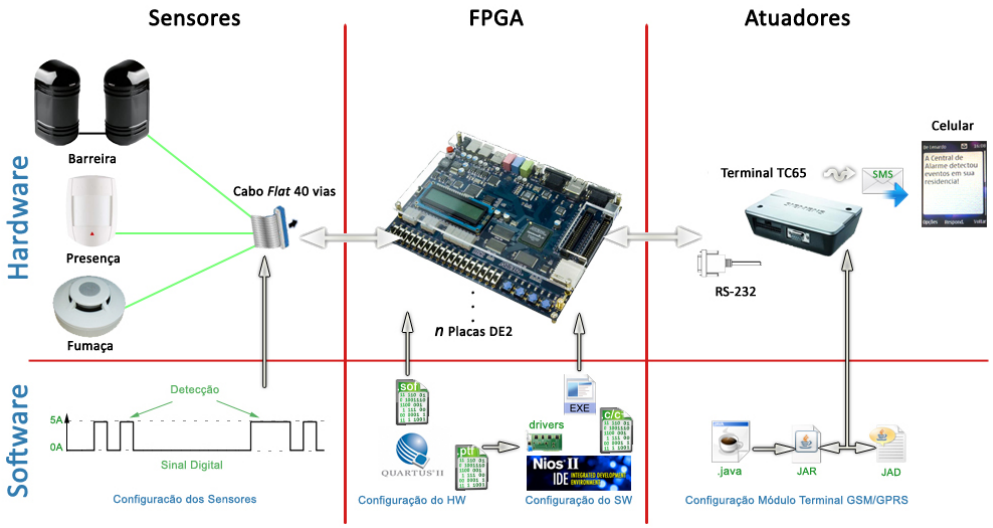


Figura 4.3: Arquitetura Parametrizável aplicada à Central de Alarme Parametrizável

Nesta, duas visões são apresentadas: uma macro visão organizacional da plataforma DE2 integrada aos dispositivos de sensoriamento e atuação, denominada visão de Hardware; e a visão de Software, que demonstra as técnicas de programação e configuração empregadas no desenvolvimento dos módulos de software para a manipulação dos componentes hardware. A visão de Hardware é subdividida em três camadas, a dos Sensores, a do FPGA e a dos Atuadores. Na camada do FPGA, pode-se ter ***n*** placas DE2 associadas, compartilhando recursos e dados, de forma a possibilitar um processamento em paralelo. Na camada dos Sensores, encontram-se sensores de presença, barreira e fumaça, podendo ser incluídos vários outros tipos de sensores digitais. E na camada Atuadores, está o terminal TC65 Siemens responsável por encaminhar as mensagens do tipo SMS. Na visão de Software, é feito o esboço das tecnologias e técnicas utilizadas para realizar o controle e o gerenciamento do hardware correspondente. Para a interpretação dos sinais advindos dos sensores usa-se a análise dos sinais digitais, que indicam quando os mesmos captam ou não os sinais com base nos eventos ocorridos. A solução para o FPGA é desenvolvida com base na definição e na configuração dos componentes de hardware, bem como na implementação da aplicação de software para controle dos dispositivos e canais de comunicação a este integrados. Quanto aos Atuadores, a aplicação para o Terminal TC65 é baseada na plataforma Java para dispositivos móveis. Tal aplicação é composta de dois arquivos, um arquivo executável (JAR), correspondente a própria aplicação, e um arquivo de texto (JAD), que descreve as propriedades da aplicação para facilitar o gerenciamento

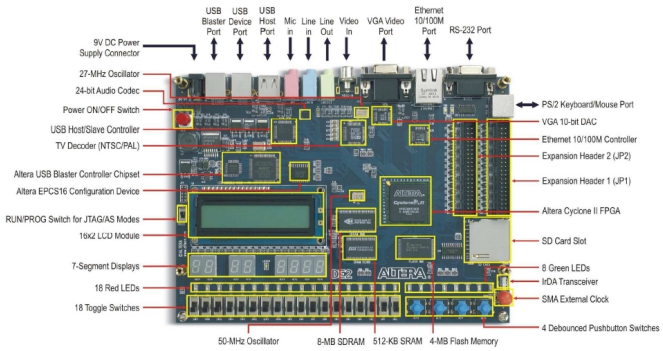
por parte do equipamento terminal.

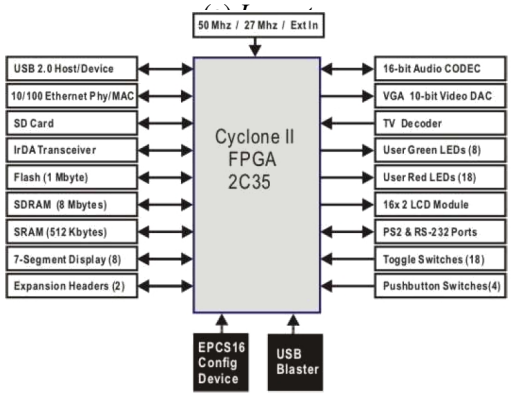
As Seções seguintes apresentam as tecnologias e as metodologias pesquisadas e utilizadas na construção da arquiteura proposta.

4.3 Plataforma Reconfigurável

A plataforma de hardware reconfigurável consiste do FPGA *Cyclone II* e uma variedade de outros componentes de hardware, incluindo interface de comunicação serial (RS-232) e GPIO, controlador *ethernet*, *display* de sete segmentos, módulo LCD (*Liquid Crystal Display*), memórias *flash*, SRAM e SDRAM, bem como um conjunto de LED (*Light-Emitting Diode*), botões e chaves, respectivamente, referenciadas na placa como *keys* e *switches*. Além disso, outros componentes para suporte a dispositivos de aúdio e vídeo, infravermelho (irDA) e cartões de memória estão embutidos no kit de desenvolvimento educacional Altera DE2.

Na Figura 4.4, o *layout* (a) e o esquema lógico (b) da placa DE2 são ilustrados.





(b) Esquema Lógico

Figura 4.4: Plataforma DE2. Fonte: Adaptada de (ALTERA, 2010a)

Os recursos disponíveis na plataforma DE2 permitem que o usuário desenvolva uma gama de aplicações que vão desde projetos mais simples, como circuitos lógicos, até os mais complexos, como aplicações multimídia.

O *Cyclone II* pertence a segunda geração da série de FPGA *Cyclone*, cuja fabricante é a *Altera Corporation*. Sua escolha foi opcional, visto que não faz parte do escopo desta pesquisa realizar um comparativo entre FPGA existentes. No entanto, as duas maiores empresas fabricantes de tais dispositivos são a *Xilinx, Inc.* e a *Altera Corp.*. Estas disponibilizam relatórios técnicos no qual se comparam, em termos de análise de desempenho, os seus respectivos modelos de FPGA com características similares, *Spartan-3* e *Cyclone II*, como pode ser visto em (SHARP, 2005) e (ALTERA, 2004b).

4.3.1 FPGA *Cyclone II*

Fabricados com a tecnologia de 90nm (nanômetros) com base no processo de produção *Low-k Dielectrics* (TSCM, 2004) da TSMC (*Taiwan Semiconductor Manufacturing Company*), o FPGA *Cyclone II* está acoplado a uma pastilha de silício de apenas 300mm de área (ALTERA, 2008).

O processador incorporado a este é o *Nios II*, que permite implementar personalizadas soluções embarcadas, sendo possível adicionar múltiplos processadores para realizar processamento paralelo e melhorar o desempenho do sistema projetado. Isto idealizado por meio de um único chip (SoC).

Abaixo, são elencadas as principais características do FPGA *Cyclone II*. Na Figura 4.5 é demonstrado como está organizada sua arquitetura em relação a seus componentes (ALTERA, 2008):

- Arquitetura de alta densidade, com capacidade de até 68.416 LE;
- Blocos de memória RAM embutida de até 1.1 Mbits, e suporte a memória externa de alta velocidade, tais como DDR e DDR2;
- Suporte de até 150 multiplicadores embarcados;
- Suporte máximo de 4 PLL (*Phase-Locked Loop*) por dispositivo, sendo este utilizado para controle do *clock* a nível de sistema;
- Suporte avançado de E/S, com até 622 pinos.

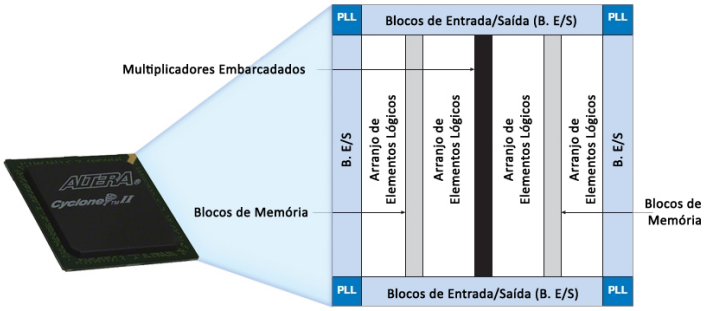


Figura 4.5: Arquitetura do FPGA *Cyclone II*. Fonte: Adaptada de (ALTERA, 2008)

A versão incorporada à plataforma DE2, denominada de EP2C35 (ALTERA, 2010a), é uma versão evoluída e com recursos avançados dentro da classe de dispositivos *Cyclone II* lançados pela Altera (EP2C5, EP2C8, EP2C15, EP2C20, **EP2C35**, EP2C50 e EP2C70) (ALTERA, 2008).

4.4 Ferramentas CAD

Como enfatizado no Capítulo 3, na metodologia Hardware/Software *Codesign* ferramentas CAD podem ser utilizadas para apoiar o desenvolvimento de projetos de Sistemas Embarcados, visto a grandeza de detalhes e cuidados a serem tomados, como também devido à complexidade inerente a estes sistemas.

Cada fabricante de dispositivos de Processamento de Sinal Digital e/ou Dispositivos Lógicos Programáveis disponibiliza um conjunto de ferramentas para facilitar e agilizar o desenvolvimento, bem como depurar seus projetos. Tais ferramentas auxiliam o projetista tanto no projeto de hardware como no de software, seja na criação, customização ou simulação do sistema.

Dentre as ferramentas CAD disponibilizadas pela *Altera Corp.* para suas famílias de dispositivos FPGA, como *Cyclone, Stratix e Arria*, CPLD – *Complex PLD* – como *MAX*, e ASIC, como *HardCopy*, elenca-se a seguir, àquelas que são indispensáveis no projeto da plataforma FPGA apresentada na Seção 4.3:

- *Quartus II Web Edition Software*: como ferramenta de apoio ao projeto de lógica reconfigurável, esta fornece um ambiente de projeto multiplataforma que pode ser utilizado para adaptar às necessidades de um projeto específico, em especial de sistemas programáveis em chip. Este engloba funcionalidades para suporte a todas

as fases de um projeto FPGA, demonstradas na Seção 2.3. Tal ferramenta disponibiliza duas interfaces para executar cada uma destas fases, tanto por meio de uma interface gráfica do usuário (*Graphical User Interface* - GUI) como por meio de um *prompt* de linha de comando, ficando esta escolha de acordo com a preferência do usuário. Seu principal benefício é a possibilidade de readaptação de projetos lógicos programáveis existentes para o desenvolvimento de outras aplicações (ALTERA, 2010e).

- *Nios II EDS (Embedded Design Suite)*: esta é uma coleção de ferramentas de software, utilitários, bibliotecas e *drivers* intencionadas a fazer com que projetos baseados no processador *Nios II* cumpram o *time-to-market* e a qualidade exigida pelo mercado. Dentre as ferramentas e bibliotecas disponibilizadas nesta *suite* de softwares estão incluídas: o Nios II IDE (*Integrated Development Environment*), ambiente de desenvolvimento baseado no *framework* Eclipse IDE; o *GNU Tool Chain*, compilador baseado no padrão GCC (*GNU Compile Collection*) (GNU, 2010); e, a API HAL (*Hardware Abstraction Layer*), uma biblioteca de sistema baseada no padrão ANSI C (ANSI, 1999) que funciona como uma interface de comunicação entre o programa do usuário e o dispositivo de hardware associado, fornecendo serviços de controle e acesso ao hardware (ALTERA, 2006);

- *ModelSim-Altera Software*: ferramenta utilizada para simulação de projetos que sejam totalmente desenvolvidos em HDL (como VHDL ou Verilog), cujo dispositvo alvo seja um FPGA Altera. Através dos arquivos *netlists* gerados pelo *Quartus II* é possível realizar três tipos de simulação: *funcional*, que verifica a sintaxe do código e a funcionalidade do projeto; *pós-síntese*, que analisa se a funcionalidade do projeto foi preservada após a etapa de síntese do hardware; e, *tempo a nível de portas lógicas*, usada para garantir que a funcionalidade do dispositivo atende a todos os requisitos de tempo uma vez realizada a etapa de *Place-and-Route*. Para isto, são necessárias bibliotecas específicas para cada modelo de simulação executado (ALTERA, 2010c). Mediante a não disponibilidade da licença completa para uso de tal ferramenta, esta não pôde ser utilizada.

Assim, durante todo o fluxo de desenvolvimento do projeto da arquitetura proposta, que será detalhado na Seção 4.5, utilizou-se das funcionalidades de tais ferramentas, para reduzir o esforço necessário para projetar e implementar tal solução, abstraindo a complexidade de certas atividades do projetos, como a atribuição dos pinos de entrada e saída dos dispositivos integrados ao FPGA *Cyclone II*.

4.5 Fluxo de Desenvolvimento do Sistema

Como descrito no Capítulo 3, a metodologia mais adequada para o desenvolvimento de Sistemas Embarcados é a Hardware/Software *Codesign*. Conforme ilustrado na Figura 4.6, a abordagem utilizada nesta pesquisa consiste de cinco etapas, que são: Especificação, Particionamento, Co-Síntese, Integração e Avaliação e Testes Incrementais.

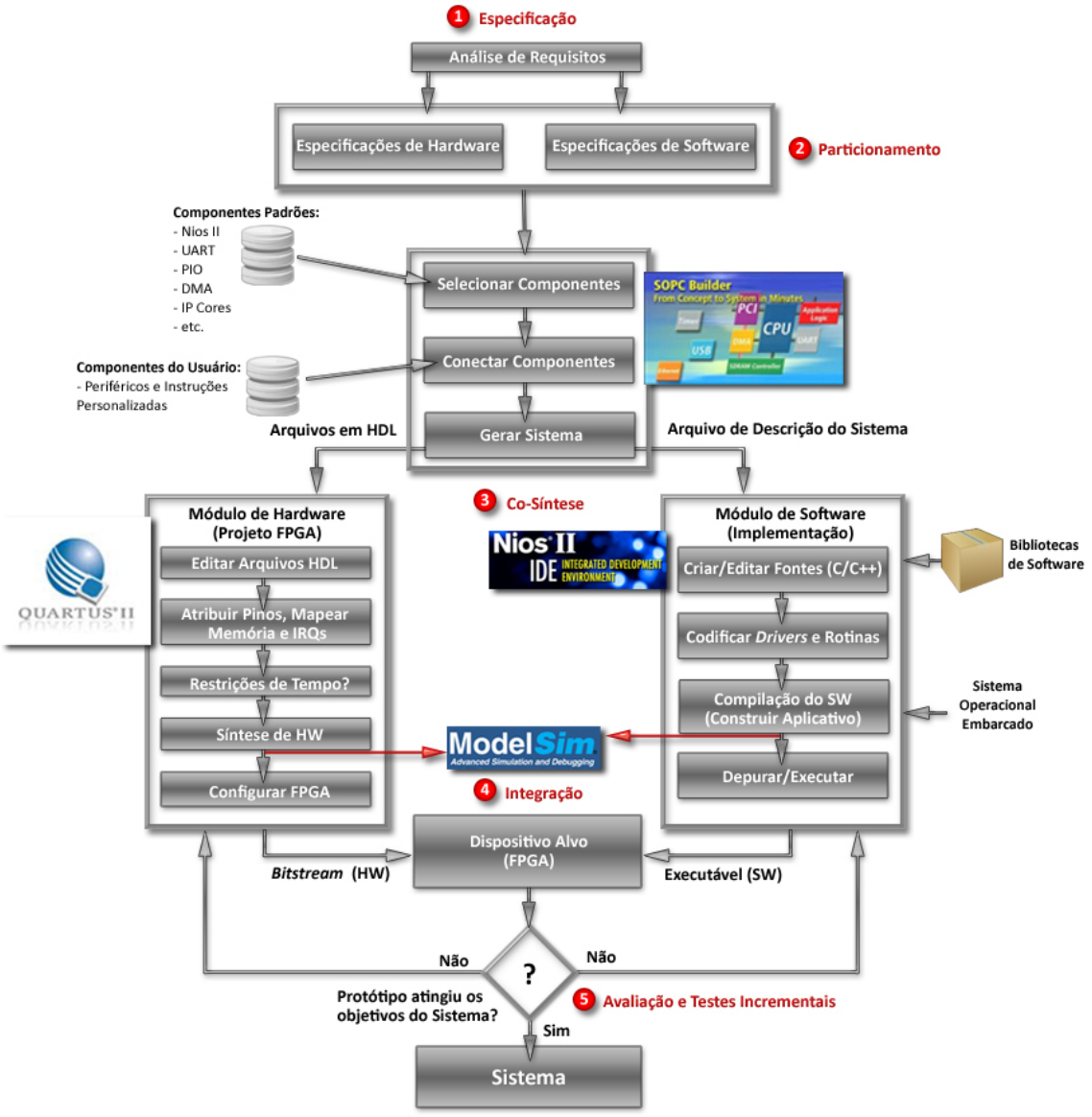


Figura 4.6: Fluxo de projeto *HW/SW Codesign* para a DE2. Fonte: Adaptada de (ALTERA, 2004a)

Cada uma das etapas mencionadas, conforme numeração na figura anterior, faz uso

de uma ou mais ferramentas CAD para auxiliar o projetista do sistema no processo de desenvolvimento de seus projetos, sendo tais etapas descritas a seguir:

4.5.1 Especificação do Sistema

A etapa de Especificação do Sistema define e descreve o que o sistema deve fazer conforme suas funcionalidades (requisitos funcionais), bem como de suas propriedades essenciais e desejáveis (requisitos não funcionais), tais como disponibilidade, desempenho e segurança. Ainda nesta etapa, são estabelecidos os objetivos do sistema.

Em se tratando de SE, os requisitos englobam aspectos de hardware e software, não levando em consideração como elementos isoladamente distintos, mas como um único projeto de sistema.

4.5.2 Particionamento

Tão logo seja realizada a etapa de Especificação do Sistema, dá-se início ao processo de Particionamento do Sistema.

Nesta etapa, explora-se a documentação gerada na etapa anterior para esboçar a arquitetura do sistema, determinando quais as funcionalidades devem ser implementadas em hardware ou em software. Para isto, organiza-se e subdivide-se tal arquitetura em dois módulos respectivamente.

Módulo de Hardware

Este módulo é especificado no tocante a definição dos componentes físicos fundamentais a estrutura de hardware do sistema. Tal processo é realizado com o auxílio da ferramenta *SOPC Builder* que está integrada ao Quartus II (ALTERA, 2010c). Esta disponibiliza um conjunto de componentes padrões e núcleos de propriedade intelectual (*Intellectual Property cores/IP Cores*) que podem ser configurados para definir e gerar um esquema de hardware completo para os SoC. O processador *Nios II*, o controlador DMA (*Direct Memory Access*) e as interfaces de comunicação de E/S são alguns dos componentes disponibilizados. O auxílio de tal ferramenta facilita para usuário a conexão destes componentes e fornece opções de acelerar o desempenho do hardware. Quanto aos *IP cores*, estes são componentes reutilizáveis licenciados por terceiros, que já foram

devidamente testados e otimizados para obter alto desempenho à baixo custo.

Além destes componentes, a ferramenta permite ainda que o hardware contenha elementos customizados pelo próprio usuário, desde que implementados em HDL ou esquematizados em forma de blocos lógicos, utilizando-se para isto o recurso de importação de módulos do *SOPC Builder*. Ao definir todos os elementos necessários do projeto de hardware, é indispensável especificar as faixas de endereços para acesso a todos os seus componentes, assim como também de suas linhas de interrupções (*Interrupt Request Line* - IRQ). As IRQ são usadas para notificar o processador *Nios II* da ocorrência de eventos que devam ser imediatamente tratados por ele. Na Figura 4.7 são apresentados alguns dos componentes utilizados e suas devidas configurações para o projeto de hardware da plataforma reconfigurável adotada no estudo de caso desta pesquisa. Estes componentes estarão disponíveis para acesso e manipulação pelo sistema de software.

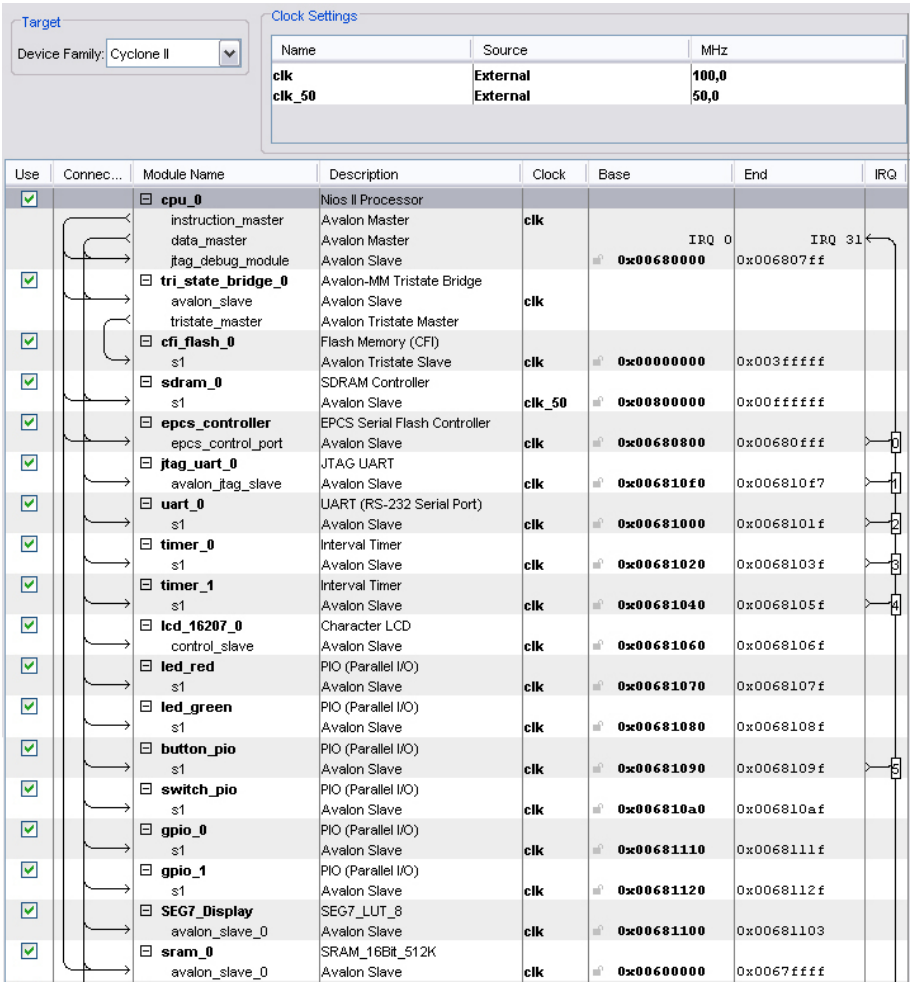


Figura 4.7: Configuração dos componentes de hardware definidos no *SOPC Builder*

Após a conclusão da personalização dos componentes do módulo de hardware requerido pelo sistema, é preciso gerar no *SOPC Builder* os arquivos que são utilizados nas demais etapas do processo de Hardware/Software *Codesign*. Dois deles tratam-se de arquivos que são utilizados no desenvolvimento da aplicação de software pelas ferramentas integradas ao *Nios II IDE*. O primeiro, cuja extensão é do tipo **ptf** (*Peripheral Template File*), descreve o modelo dos componentes periféricos do sistema. O segundo, de extensão **sopcinfo** (*SOPC Information File*), é uma descrição completa do sistema em termos de componentes e conexões. O Arquivo **ptf** é a base para a geração da biblioteca específica que dará acesso aos recursos de hardware disponíveis para o módulo de software.

Outros arquivos em HDL gerados pelo *SOPC Builder* refletem em código a estrutura e a configuração de cada componente de hardware do sistema. Tais componentes serão habilitados para uso na placa DE2, pós-compilação pela ferramenta CAD *Quartus II* (ALTERA, 2009a). Isto possibilita o correto funcionamento do hardware quando desenvolvida a lógica de controle para a aplicação de software.

O *SOPC Builder* disponibiliza automaticamente mais dois tipos de arquivos de extensões ***bsf*** (*Block Symbol File*) e ***bdf*** (*Block Diagram Files*). Ambos contendo a representação dos componentes a nível de hardware. Porém, o primeiro é utilizado pelo projetista do sistema quando deseja alterar a estrutura do hardware por meio do *SOPC Builder*. E o segundo, com um maior nível de abstração, serve como alternativa a configuração do módulo de hardware do sistema mediante o uso do *Quartus II*, assim como os arquivos em HDL.

O resultado parcial desta etapa é a configuração do módulo de hardware por meio da definição dos pinos de entrada e saída para cada bloco de componentes construído, sejam estes *IP cores* ou blocos estruturados pelo próprio projetista do sistema. Na Figura 4.8 é apresentado o esquemático do módulo de hardware criado para o estudo de caso com base na arquitetura proposta.

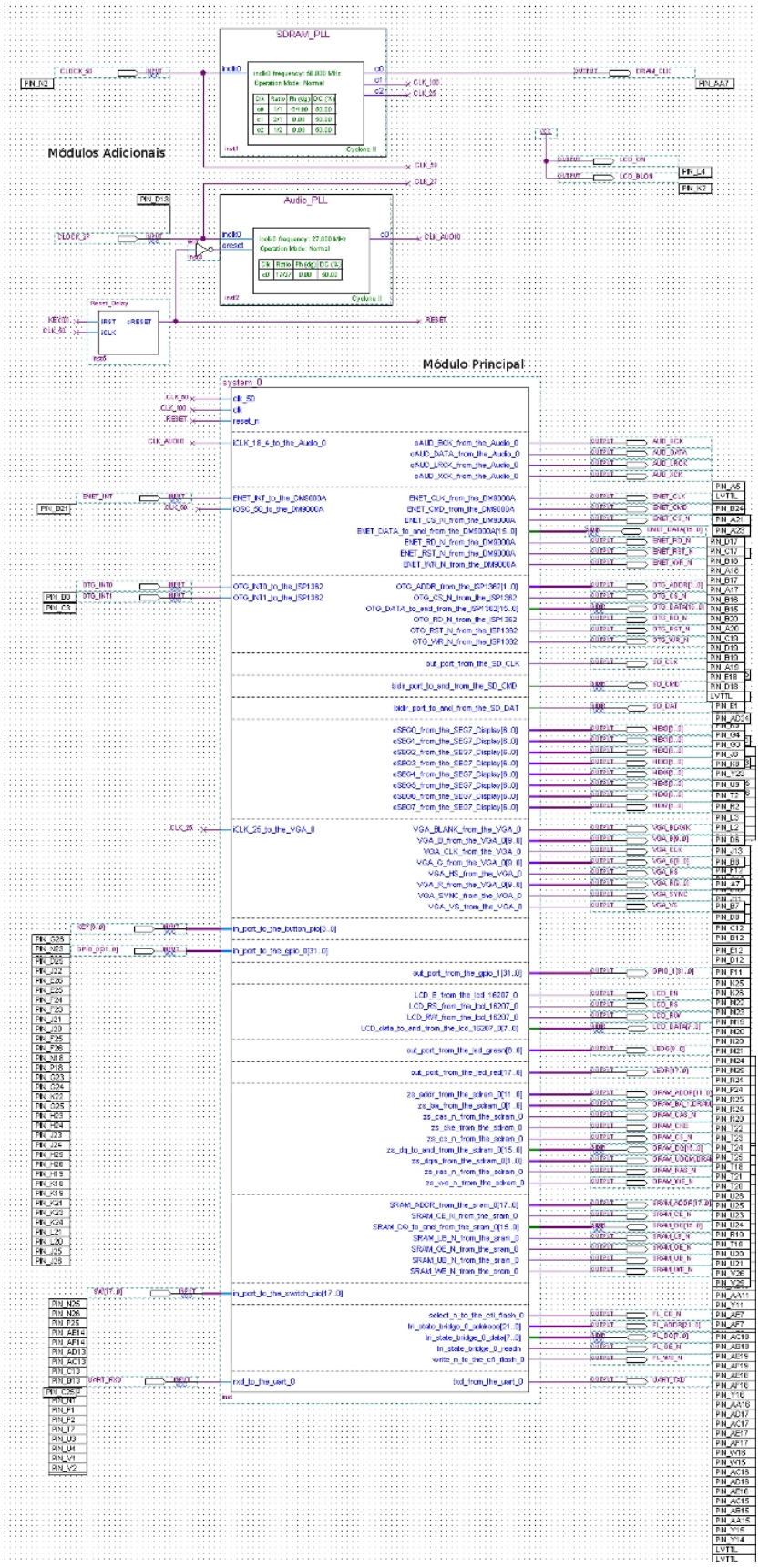


Figura 4.8: Esquemático do módulo de hardware construído no *Quartus II*

Módulo de Software

Quanto ao módulo de software, este é idealizado a partir do módulo de hardware personalizado pelo usuário. Porém, nada impede que durante a configuração de tal módulo sejam codificadas as estruturas de controle e de lógica aritmética do sistema, em linguagem compatível com o projeto de hardware mencionado. Neste caso, a linguagem adotada foi C, utilizando-se o *Nios II IDE* (ALTERA, 2006) como ferramenta de apoio ao desenvolvimento de todas as tarefas relacionadas ao software, desde a criação e edição até a depuração da aplicação.

Em geral, utiliza-se no desenvolvimento do módulo de software bibliotecas de sistemas, *middlewares* e/ou kernel de sistemas operacionais suportados pela arquitetura de hardware em questão. Como exemplo destes, pode-se citar, a API (*Application Programming Interface*) HAL (ALTERA, 2006) e o sistema operacional μLinux (lê-se micro Linux) (LU et al., 2008).

Os projetistas do sistema podem utilizar a API HAl para desenvolver *drivers* específicos para determinados dispositivos que possam ser integrados ao sistema. Com isto, é possível ter acesso direto ao hardware de tais dispositivos. Além disso, rotinas de acesso a memória podem ser programadas para o processador *Nios II*, como também podem ser codificados manipuladores de exceções e interrupções de hardware.

Portanto, por meio da ferramenta CAD *Nios II IDE* que acompanha a *Nios II EDS* é possível realizar o gerenciamento, o desenvolvimento e a implantação do projeto de software para o dispositivo alvo adotado, ou seja, o FPGA *Cyclone II*.

Camada de Interface entre Hardware e Software

Como os projetos de hardware e software elaborados foram executados utilizando o conjunto de ferramentas CAD disponíveis pelo fabricante do FPGA adotado, não houve a necessidade de desenvolvimento de uma interface adicional específica. Isto porque a integração e a comunicação entre os módulos de hardware e software desenvolvidos são totalmente compatíveis com os padrões incorporados a tais ferramentas.

4.5.3 Co-Síntese

A etapa de Co-Síntese dos módulos de hardware e software dá-se a partir do exato momento em que todas as exigências para a configuração do hardware e do software são atendidas e estabelecidas com base em suas especificações.

Síntese do Hardware

Para a síntese do hardware, além de definidos os componentes que estarão a disposição do software, é necessário realizar três procedimentos essenciais.

Os procedimentos são: a definição da localização dos elementos lógicos no FPGA; o mapeamento e o estabelecimento das rotas para as interconexões de tais elementos, processo este conhecido como *Place-and-Route*, e que foi definido na Seção 2.3; e a atribuição dos pinos de entrada e saída do FPGA *Cyclone II*. Na Figura 4.9 é demonstrado o esquema de pinagem do FPGA *Cyclone II* após realizada esta última etapa.

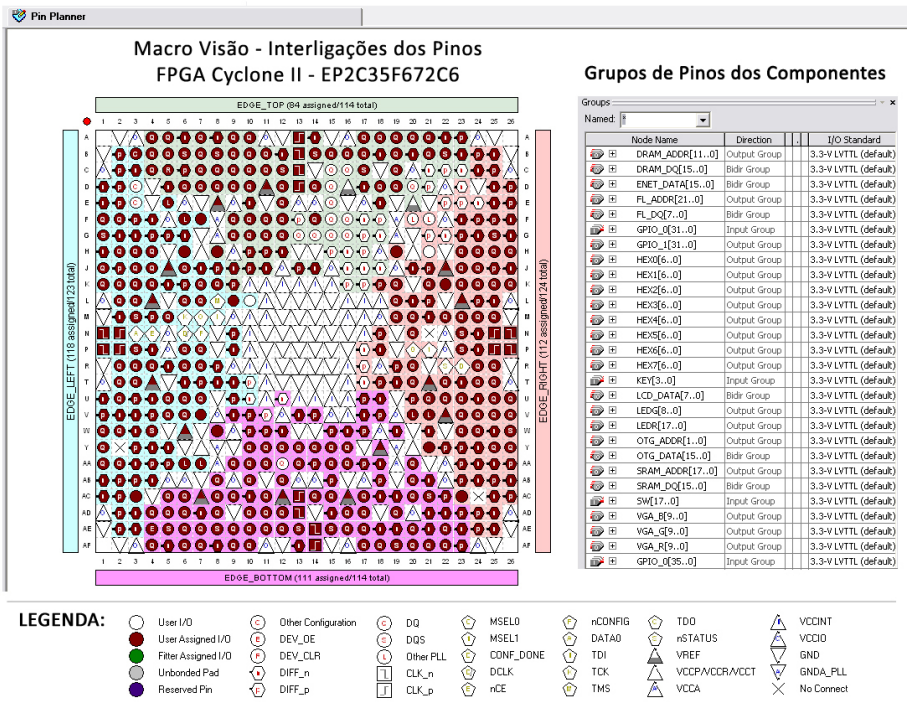


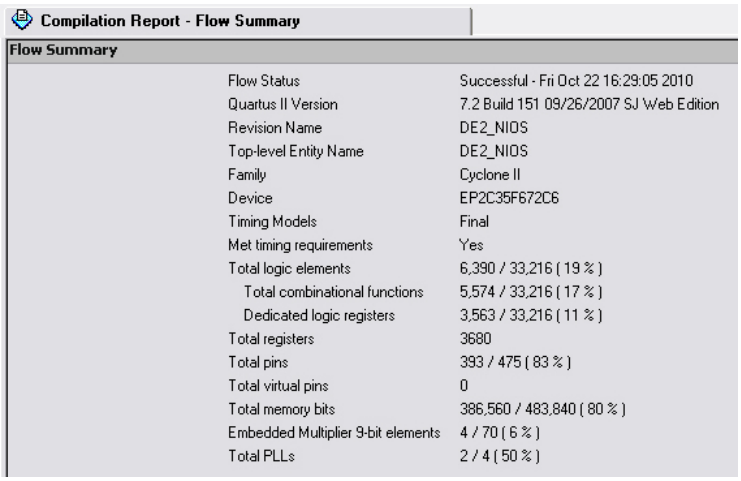
Figura 4.9: Esquema de pinagem do FPGA atribuído com a ferramenta *Pin Planner* integrada ao *Quartus II*

Nesta Figura é especificado como estão agrupados os pinos do FPGA em relação aos componentes de hardware a ele integrados, bem como a direção relativa a cada um destes

grupos, ou seja, se estes são de saída ou entrada de dados. A macro visão apresenta a interligação de cada um destes pinos, acompanhada pela respectiva legenda dos símbolos adotados pela ferramenta CAD para representar os vários tipos de interligação.

O modo como tais procedimentos são realizados pode interferir diretamente na eficiência geral do sistema. Principalmente, em relação a quantidade, o tamanho e a complexidade dos elementos lógicos definidos. Estes aspectos afetam a área interna do FPGA e o atraso (*delay*) na execução das tarefas realizadas pelo processador, atraso este causado pelo mapeamento das interconexões de tais elementos.

Executados tais procedimentos, pode-se agora realizar a síntese do hardware. Para isto, o *Quartus II* gera um *netlist* com base nos arquivos em HDL oriundos do *SOPC Builder*, que deverá ser mapeado em um *bitstream*. O *netlist* é o arquivo que descreve a estrutura lógica dos componentes de hardware e suas conexões. O *bitstream* é a imagem binária de configuração do dispositivo que será embarcada no FPGA para permitir que o processador *Nios II* gerencie suas tarefas. Este também é gerado pelo *Quartus II*, cuja extensão é **sof** (*SRAM Object File*), sendo este arquivo que irá caracterizar a programação do hardware do sistema quando embarcado. Na Figura 4.10 é apresentado o resumo do relatório gerado nesta etapa. Este, especifica dados importantes relativos ao módulo de hardware configurado, dentre estes encontram-se o nome do dispositivo alvo associado ao projeto e o número total de elementos lógicos programáveis, registradores, pinos e PLL utilizados. No relatório completo são fornecidos outros subrelatórios, arquivos e mensagens inerentes a cada etapa do projeto de hardware para análise e otimização pelo projetista do sistema.


Compilation Report - Flow Summary

Flow Summary

Flow Status	Successful - Fri Oct 22 16:29:05 2010
Quartus II Version	7.2 Build 151 09/26/2007 SJ Web Edition
Revision Name	DE2_NIOS
Top-level Entity Name	DE2_NIOS
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	6,390 / 33,216 (19 %)
Total combinational functions	5,574 / 33,216 (17 %)
Dedicated logic registers	3,563 / 33,216 (11 %)
Total registers	3680
Total pins	393 / 475 (83 %)
Total virtual pins	0
Total memory bits	386,560 / 483,840 (80 %)
Embedded Multiplier 9-bit elements	4 / 70 (6 %)
Total PLLs	2 / 4 (50 %)

Figura 4.10: Resumo do relatório de síntese do hardware executado no *Quartus II*

Compilação do Software

A disponibilização da biblioteca do sistema HAL e os arquivos de descrição do módulo de hardware, oriundos do processo de sintetização, possibilitam dá inicío a codificação das *procedures* que permitirão que o software interaja em baixo nível com os componentes de hardware. Isto envolve referenciar endereços de memória, estruturas de dados e um conjunto de diretivas e rotinas para controle de acesso a tais componentes.

Concluída as atividades supracitadas, torna-se possível realizar o processo de compilação do software. Com a ferramenta *Nios II IDE*, vincula-se o projeto implementado em linguagem C, contendo as rotinas de código fonte do aplicativo de software, com a biblioteca personalizada para o hardware do sistema, bem como possíveis *drivers* e *middlewares* de comunicação, se preciso for.

Para a customização da biblioteca do sistema, é necessária a criação de um projeto BSP (*Board Support Packages*). Este é composto por códigos fonte e arquivos de cabeçalhos em C (*header*, com extensão **h**), um conjunto de arquivos de inicialização, além de um arquivo *makefile*, que descreve as regras para a configuração do dispositivo alvo. Isto feito com base nas especificações do usuário, bem como nas funcionalidades do dispositivo alvo gerado pelo *SOPC Builder*. O arquivo referente a biblioteca BSP possui a extensão **a** e serve para gerar a imagem da aplicação quando combinado com o projeto *Nios II C/C++* codificado pelo usuário.

Para a compilação do software, algumas propriedades/configurações também podem ser determinadas, tais como os canais de *debug* do sistema, bem como a definição do dispositivo de *clock* que será utilizado, caso seja disponível mais de um. Após executado o processo de compilação do software, o arquivo executável da aplicação é criado, e assim como o *bistream* gerado pela síntese do hardware, este deverá ser embarcado na placa DE2, para ser gerenciada pelo FPGA.

4.5.4 Integração

Como forma de verificar o funcionamento do sistema como um todo, necessita-se integrar ambos os módulos de hardware e software que foram gerados a partir do processo de Co-Síntese, anteriormente realizado.

Como ilustrado na Figura 4.11, a integração acontece com o processo de descarga dos arquivos gerados nas etapas de sintetização do hardware e compilação do software no

FPGA, arquivos estes denominados de conteúdo de programação. Para isto, é obrigatório especificar qual o canal de comunicação, dentre os listados, deverá ser usado, bem como o aplicativo de hardware e software desejados. O canal mencionado refere-se ao cabo que interliga o computador à placa Altera DE2, denominado de *USB-Blaster*. Junto a este acompanha-se um *driver* a ser instalado no Sistema Operacional que contém todas as ferramentas CAD de apoio ao desenvolvimento do sistema. O canal permite que o computador e a plataforma de hardware reconfigurável possam se comunicar e realizar a (re)configuração do dispositivo alvo.

Figura 4.11: Esquema de *download* dos Projetos de Hardware e Software. Fonte: Adaptada de (ALTERA, 2010d)

Quanto à forma de programação do FPGA na plataforma DE2, esta pode ser realizada por meio do padrão CFI (*Common Flash Interface*) ou Altera EPCS (*Altera Erasable Programmable Configurable Serial*). O CFI é o padrão da indústria que provê uma interface comum e independente de fabricante para dispositivos de memória *flash*. E o Altera EPCS é um dispositivo de memória *flash* Altera que utiliza um esquema de configuração serial ativa para armazenamento dos dados de configuração do FPGA toda vez que este é ligado ou reconfigurado (ALTERA, 2009b) (ALTERA, 2010d).

Na Figura 4.12 é ilustrada a relação entre os módulos de hardware e software após realizadas as etapas de Especificação, Particionamento e Co-Síntese. Uma vez que ambos os projetos de hardware e software foram especificados, o hardware foi definido, configurado e gerado e o software foi codificado pelo usuário, utilizando os *drivers*, bibliotecas e sistema operacional, se necessários, com o apoio das ferramentas CAD, os mesmos podem ser integrados ao dispositivo FPGA.

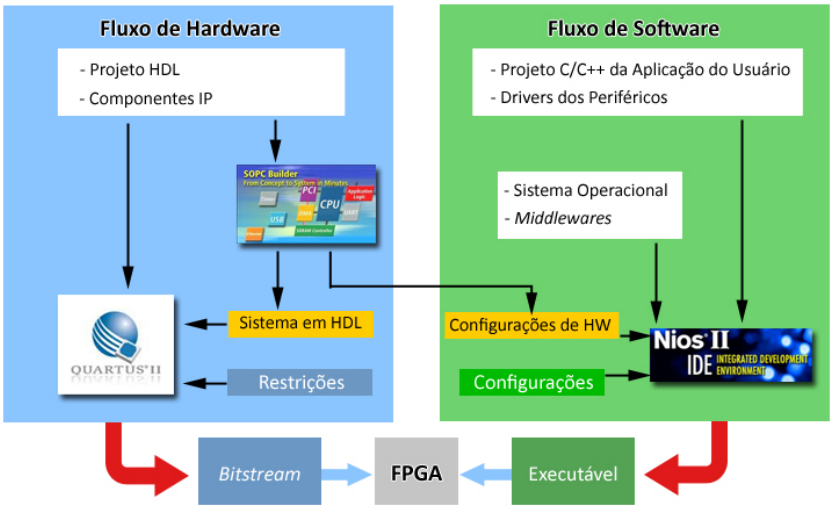


Figura 4.12: Relação entre os Módulos de Hardware e Software

Com o dispositivo alvo configurado, tanto em termos de hardware como de software, o sistema pode passar por avaliação e testes. A diversidade dos testes é determinada não somente pela complexidade do sistema, mas também pelo número e tipos de componentes configurados no hardware, funções lógicas empregadas e número de processadores usados. A etapa de Avaliação e Testes Incrementais é descrita na Seção 4.5.5.

4.5.5 Avaliação e Testes Incrementais

A etapa de Avaliação e Testes Incrementais não obrigatoriamente deve ser realizada quando da conclusão da implementação do sistema. O sistema pode ser subdividido em blocos de funcionalidades que podem ser desenvolvidos em ciclos, de acordo com algum critério como, por exemplo, prioridades do sistema. Assim, à medida que cada bloco é configurado (hardware) e codificado (software) pode-se analisar sua corretude por meio da simulação no dispositivo alvo do protótipo do sistema desenvolvido.

Nesta etapa erros, omissões ou mal funcionamento do sistema seja em relação ao hardware ou ao software podem ser identificados, possibilitando aos projetistas saná-los precocemente, o que irá garantir, ao final de todos os ciclos de implementação dos blocos do sistema, a satisfação total dos objetivos do projeto. Isto é possível com o retorno a etapa de Particionamento do Sistema, seja para solucionar algum problema encontrado durante a simulação ou agregar novas funcionalidades ao sistema, como anteriormente ilustrado na Figura 4.6.

Os testes realizados nesta etapa se deteram a verificação da compatibilidade da plataforma reconfigurável com os dispositivos periféricos externos a ela integrados, tais como os sensores digitais, equipamento terminal de dados e transceptor/leitor de Radiofrequência. Com este último pôde-se detectar a incompatibilidade de comunicação entre tal equipamento e o FPGA adotado, uma vez que este está limitado a enviar e receber apenas um *byte* por vez, sendo que o leitor RFID necessita, em suas operações de leitura e escrita, de um conjunto de *bytes*.

4.6 Aplicação de Controle de Mensagens SMS

A especificação do projeto da aplicação de controle para envio de mensagens SMS foi definida com base na capacidade de transmissão via interface de comunicação serial (RS-232) do FPGA *Cyclone II* embutido na plataforma DE2. Como este trabalha a nível de bits, enviando e recebendo apenas um caracter por vez, a aplicação SMS foi estruturada considerando tal restrição. Tal aplicação foi embarcada no terminal TC65, sendo este o equipamento utilizado para transmissão das mensagens SMS.

As Seções seguintes apresentam os recursos utilizados e descrevem a arquitetura da Aplicação de Controle de Mensagens SMS, visando facilitar a compreensão desta na pesquisa.

4.6.1 Terminal TC65

Para prover a comunicação da Central de Alarme Parametrizável com os dispositivos móveis do usuário final utilizou-se o Terminal TC65.

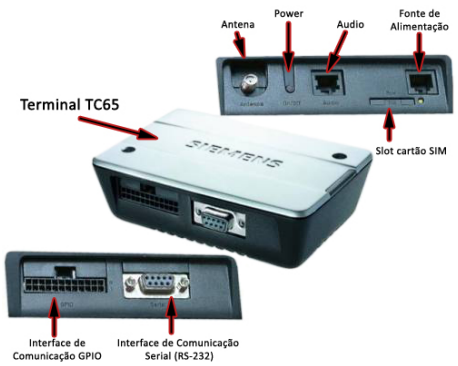


Figura 4.13: Terminal TC65 Siemens

Conforme ilustrado na Figura 4.13, este possui duas interfaces de comunicação, uma baseada no padrão RS-232 e outra GPIO, além de uma saída de áudio, uma antena externa e um *slot* para cartão SIM (*Subscriber Identity Module*). Um cartão SIM permite identificar, gerenciar e armazenar dados referentes a telefones celulares com a tecnologia *GSM quad-band* (850/900/1800/1900 MHz).

Tal terminal foi desenvolvido pela *Siemens AG* para o mercado *Wireless Machine-to-Machine* (M2M) [1] (SIEMENS, 2006).

> *"O M2M interliga as Tecnologias de Informação e Comunicação com os dispositivos de comunicação inteligentes, possibilitando uma interação com os sistemas de organizações ou empresas sem a intervenção humana"* (ORANGE, 2006).

Dentre as principais características deste equipamento pode-se elencar o suporte a plataforma Java, o uso de interfaces industriais padronizadas, fácil instalação e utilização.

O terminal TC65 permite que as aplicações desenvolvidas para este sejam executadas e gerenciadas diretamente por meio de seu microprocessador embutido. O perfil de configuração da plataforma J2ME (*Java 2 Micro Edition*) é o IMP-NG (*IMP - Next Generation*). Este, com o auxílio dos protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*), possibilita que tais aplicações sejam remotamente gerenciadas de forma simples e segura, por meio de mecanismos de criptografia, tais como HTTPS (*HyperText Transfer Protocol Secure*) e PKI (*Public Key Infrastructure*). Tanto a plataforma J2ME como o IMP-NG serão melhor detalhados na Seção 4.6.3.

Em relação ao seu funcionamento, o TC65 é regido por um conjunto de normas e comandos mundialmente padronizados, conhecidos como *Comandos AT*, descritos a seguir. Quanto as suas limitações, estas podem ser encontradas em (SIEMENS, 2005c).

4.6.2 Comandos AT

Oficialmente reconhecida pela União Europeia como a Organização Europeia de Normalização, a ETSI (*European Telecommunications Standards Institute*) [2] elabora normas

[1] O termo Wireless M2M (*Machine-to-Machine*, *Mobile-to-Machine*, *Machine-to-Mobile Communications*) refere-se à transferência e utilização de dados via redes celulares provindos de equipamentos/terminais remotos para o monitoramento, a medição e o controle dos mesmos (BONDE, 2006)

[2] A ETSI é uma organização sem fins lucrativos composta por mais de 700 organizações-membro a partir de 62 países espalhados pelos 5 continentes.

mundialmente aplicáveis às Tecnologias da Informação e Comunicação (TIC), incluindo telefonia fixa, móvel, rádio e convergentes, bem como tecnologias de transmissão e internet (ETSI, 2010).

O grupo SMG (*Special Mobile Group*), incorporado ao ETSI é o responsável pelo Padrão de Telecomunicação Europeu (*European Telecommunication Standard*). Este especificou um conjunto de *Comandos AT* e recomendações para controlar as funcionalidades de equipamentos móveis e serviços de rede GSM a partir de equipamentos e adaptadores terminais dentro do sistema de telecomunicação celular digital. Os comandos AT para os serviços de SMS e GPRS são definidos, respectivamente, nas normas **GSM 07.05** e **GSM 07.60** (ETSI, 1999).

A sigla **AT** é a abreviação de *ATtention*, sendo esta a identificação para o terminal do início de uma linha de comando a ser processada. A sintaxe geral de uma linha de comando AT para GSM é composta de comandos estendidos (ver trecho de código na Figura 4.14).

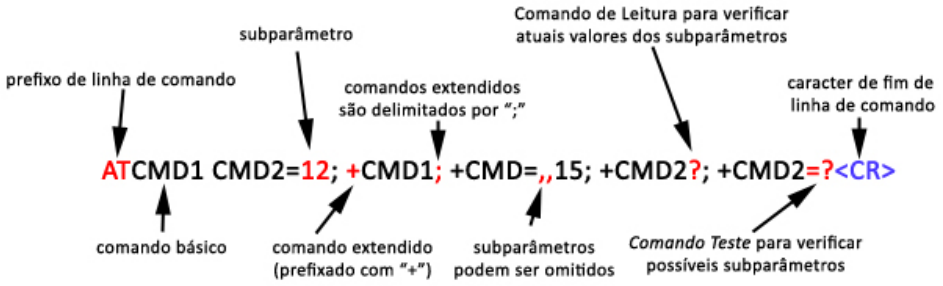


Figura 4.14: Estrutura de uma linha de comando AT. Fonte: Adaptada de (ETSI, 1999)

Como demonstrado na Figura 4.14, o início do comando é prefixado pela sigla "**AT**" seguido por um comando básico, que são comandos definidos pelo setor de normalização de telecomunicações da ITU (*International Telecommunication Union*), setor este conhecido pela sigla ITU-T (ITUT-T, 2003). Cada comando estendido possui um *Comando Teste* (**CMD=?**) que testa a existência do comando e fornece informações acerca de seus possíveis parâmetros. De forma similar, os comandos do tipo estendido (**+CMD;**) podem verificar os valores atuais pré-configurados por meio do *Comando de Leitura* (**CMD?**). Para finalizar a linha de comando AT, utiliza-se o caracter **<CR>** (*Carriage Return*). Se a sintaxe estiver correta, se todos os comandos forem válidos e não houver nenhum problema de comunicação o terminal processa a linha de comando, retornando o seguinte código de confirmação *<CR><LF>***OK***<CR><LF>*, onde **<LF>** (*Line Feed*) in-

dica fim de linha. Caso contrário, o código resultante é *<CR><LF>**ERRO***<CR><LF>*, referindo-se a ocorrência de erros. Neste caso, a linha de comando não foi processada.

Uma resposta à linha de comando enviada ao terminal TC65 e ilustrada na Figura 4.14, é o trecho de código mostrado na Figura 4.15:

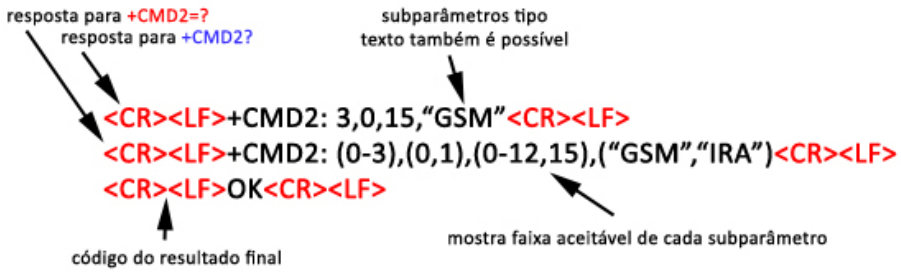


Figura 4.15: Resposta à linha de comando AT. Fonte: Adaptada de (ETSI, 1999)

Percebe-se na Figura 4.15 que além do código do resultado final para a linha de comando anteriormente processada, podem haver respostas intermediárias, conhecidas como URC (*Unsolicited Result Code*) que indicam a ocorrência de eventos não diretamente associadas com o comando enviado ao terminal. Um exemplo típico de URC é a indicação de chamada (*RING*).

Assim, os tipos de comandos AT são (SIEMENS, 2005a):

- *Comando de Teste:* retorna uma lista de parâmetros e uma gama de valores em resposta ao respectivo *Comando de Escrita* (sintaxe: **AT+CXXX=?**);
- *Comando de Leitura:* retorna conjunto de valores atuais do(s) parâmetro(s) (sintaxe: **AT+CXXX?**);
- *Comando de Escrita:* define os valores do(s) parâmetro(s) repassados pelo usuário (sintaxe: **AT+CXXX=<...>**);
- *Comando de Execução:* lê o(s) parâmetro(s) estáticos determinados por processos internos na *engine* GSM (sintaxe: **AT+CXXX**).

Dentre os vários grupos de comandos AT para configuração, controle e execução do terminal TC65 pode-se citar comandos específicos de configuração, controle de estados e interfaces, chamadas telefônicas e mensagens SMS, para a manipulação de aplicações

Java, dentre outros tantos grupos, como, por exemplo, para serviços de rede e internet, bem como relacionados ao cartão SIM (SIEMENS, 2005a).

4.6.3 J2ME

Como enfatizado nos Capítulos anteriores, a evolução na era computacional, influenciada pela crescente demanda do mercado por aplicações cada vez mais robustas, deu origem a um grupo de dispositivos computacionalmente pequenos, que diferentemente dos computadores tradicionais, cujos recursos, como capacidade de armazenamento, memória e poder de processamento, são reduzidos. No entanto, estes são dispositivos menores no tamanho e bastante eficientes no consumo de energia. Alguns exemplos destes dispositivos são telefones celulares, GPS (*Global Positioning System*) e equipamentos *set-top box* para televisão digital.

Juntamente com esta geração de dispositivos portáteis houve a necessidade de novas plataformas computacionais para o desenvolvimento de suas aplicações, tais como **Maemo** (MAEMO,), **SymbianOS** (SYMBIAN, 2010), **J2ME** (ORACLE, 2010a) e, recentemente, **Android** (ANDROID, 2010), sendo a maioria destas plataformas baseadas na linguagem de programação Java.

Em meio a estas plataformas, a J2ME é a plataforma suportada pelo terminal TC65. Esta é uma versão compacta da API Java e JVM (*Java Virtual Machine*) desenvolvida para operar em dispositivos computacionais de pequeno porte e dispositivos *wireless* com recursos limitados, que necessitam incorporar funcionalidade multi-plataforma em seus produtos.

A J2ME herda as características da linguagem Java, como segurança e portabilidade, o que a torna compatível com qualquer dispositivo com suporte a JVM. Contudo, para que seja possível embutir tal plataforma em uma ampla diversidade de dispositivos dos mais variados fabricantes torna-se fundamental entender os conceitos de *Configuração* e *Perfil* (MUCHOW, 2001).

Configuração

A *Configuração* especifica a plataforma Java suportada, por meio do subconjunto de características da linguagem Java e subconjunto de funcionalidades de configuração da JVM. Esta engloba aspectos relacionados, por exemplo, a memória, a tela e a conec-

tividade à rede de um dispositivo. Atualmente, é subdividida em duas categorias, diferenciadas pelo grau de limitação dos recursos disponíveis nos dispositivos embarcados suportados. São elas:

- **CDC** (*Connected Device Configuration*) (SUN, 2005):
  - mínimo de 512KB de memória para a plataforma Java;
  - mínimo de 256KB para alocação de memória em tempo de execução;
  - conectividade a algum tipo de rede (como, GPRS ou *wireless*), conexão persistente e alta largura de banda;
  - suporte a implementação completa da JVM v2.0;
  - interfaces com usuário com diferentes níveis de sofisticação.
- **CLDC** (*Connected, Limited Device Configuration*) (SUN, 2007):
  - mínimo de 192KB de memória para a plataforma Java;
  - processador de 16 ou 32 bits;
  - baixo consumo de energia (bateria);
  - conectividade a algum tipo de rede, conexão intermitente e largura de banda limitada.

Uma observação é que enquanto a CDC executa sobre a máquina virtual java convencional (JVM), a CLDC teve a necessidade de criar sua própria máquina virtual referenciada por KVM (*K Virtual Machine*). O "K" refere-se a capacidade desta ser executada em dispositivos com poucos ***K**ilobytes* de memória.

Perfil

O *Perfil* veio para lidar com o grau de funcionalidades dos dispositivos, no sentido de dá uma maior flexibilidade em relação as mudanças tecnológicas, e permitir especializações de uma *Configuração*. Logo, o *Perfil* reune um conjunto de API que devem ser utilizadas no desenvolvimento de aplicações para um tipo específico de dispositivo, levando em consideração suas limitações de memória e de tela.

A *Oracle Corporation*, detentora dos direitos da *Sun Microsystems Inc.*, disponibiliza alguns perfis definidos pelo JCP (*Java Community Process*) e que podem ser encontrados

em (KEOGH, 2003). Dentre estes perfis, encontra-se o MIDP (*Mobile Information Device Profile*) que é utilizado conjuntamente com a configuração CLDC. Este consiste na especificação de uma arquitetura avançada e API Java associadas, fundamentais para permitir a compatibilidade entre as aplicações de terceiros e os dispositivos de informações móveis (*Mobile Information Devices* - MID).

O perfil MIDP está disponibilizado em quatros versões a saber: MIDP 1.0a (SUN, 2000) e MIDP 2.1 (SUN, 2006), de uso geral; e, duas versões simplificadas, a IMP (*Information Module Profile*) (SIEMENS; NOKIA, 2003) e a IMP-NG (*IMP - Next Generation*) (SIEMENS; NOKIA, 2005), respectivamente baseadas na MIPD 1.0a e na MIPD 2.1. Sendo a IMP-NG adequada aos dispositivos sem características de interface gráfica com o usuário.

Quanto aos MID, dois são os componentes fornecidos pelo fabricante de tais dispositivos, as classes e as aplicações OEM (*Original Equipment Manufacturer*). As classes são usadas pelo MIDP para acesso aos recursos, como, por exemplo, enviar e receber mensagens, bem como acessar dados persistentes específicos do dispositivo. Já as aplicações, são programas fornecidos como um catálogo de endereços, que podem ser acessadas pelo MIDP (KEOGH, 2003).

As aplicações desenvolvidas para serem executadas em dispositivos MID são denominadas de *MIDlet*. Estas são apresentadas na Seção 4.6.4.

4.6.4 *MIDlet*

Uma *MIDlet*, assim como as aplicações Java convencionais, possui classes e métodos associados para manipulação dos dados. Sua declaração é baseada na herança de características da classe *MIDlet* contida no pacote **javax.microedition.***, que fornece um conjunto de métodos, como para inicializá-la, interrompê-la e destrui-la.

As *MIDlet* podem ser agrupadas dentro de um mesmo pacote, com capacidade de compartilhamento de dados e recursos, sendo executadas sob uma mesma JVM. Desta forma, a JVM cria apenas uma instância da classe que é compartilhada pelo grupo de *MIDlet*. Isto pode acarretar erros e inconsistências no tocante aos processos concorrentes. Porém, este risco pode ser reduzido por meio da utilização de primitivas de sincronização que restringem o acesso aos dados voláteis e persistentes (KEOGH, 2003).

Uma *MIDlet* é instalada, executada e removida por meio de um gerenciador de aplica-

tivo (*Application Manager*) em execução no dispositivo. Este deve ser fornecido pelo fabricante do mesmo. Uma vez instalada, o gerenciador de aplicativo disponibiliza-a para execução pelo usuário final. Para isto, é preciso migrar dois tipos de arquivos para o dispositivo móvel alvo de forma a permitir sua execução, são eles os arquivos JAR (*Java Archive*) e JAD (*Java Application Descritor*). O arquivo JAR consiste de um pacote contendo todos os elementos necessários para sua execução. Já o arquivo JAD fornece informações ao gerenciador de aplicativo acerca da MIDlet instalada, ajudando-o na tomada de decisões, como, por exemplo, verificação da compatibilidade ou não desta com o dispositivo em que foi embarcada (MUCHOW, 2001).

Na Figura 4.16 é ilustrado o fluxo de execução de uma *MIDlet*, caracterizado, basicamente, pela implementação de três métodos abstratos a saber: **startApp()**, **pauseApp()** e **destroyApp()** (KEOGH, 2003). Seu ciclo de vida inicia com a *MIDlet* no estado *Pausado*. Ao chamar o método **startApp()** a *MIDlet* altera seu estado para *Ativo*, caracterizando o início de sua execução. A partir deste estado, a *MIDlet* pode ser temporariamente interrompida, por meio do método **pauseApp()**, retornando-a ao estado *Pausado*, ou ter sua execução finalizada por meio do método **destroyApp()**, onde irá para o estado *Destruído*.

Figura 4.16: Ciclo de Vida de uma *MIDlet*.

Como a execução de uma *MIDlet* é baseada em eventos, seus métodos são invocados conforme o evento reportado pelo gerenciador de aplicativo do MID, sendo seu estado atualizado mediante tal execução.

4.6.5 Arquitetura, Configuração e Execução

Como dito anteriormente, o Módulo Terminal TC65 apresentado na Seção 4.6.1 utiliza o modelo de arquitetura J2ME, onde a Configuração é a CLDC 1.1 e o Perfil é o

IMP-NG, sendo esta a especificação adotada. Na Figura 4.17 é apresentada a arquitetura da Aplicação de Controle SMS. Esta será responsável pelo envio automático das mensagens SMS, conforme os parâmetros recebidos e que foram enviados pelo FPGA da placa DE2.

Como o MIDP 2.1, o IMP-NG fornece acesso aos protocolos TCP/IP, possibilitando, utilizar dois tipos de tecnologias para comunicação, ou seja, a CSD (*Circuit Switched Data*) ou GPRS.

Na arquitetura apresentada na Figura 4.17 é possivel perceber que para o desenvolvimento da aplicação utilizou-se de API fornecidas pela fabricante do equipamento terminal (classes OEM, conforme descrito na Subseção 4.6.3). Tais API são a *API I/O Siemens*, para a comunicação com outros dispositivos conectados ao TC65, e a *API de Comandos AT*, que permite controlar as funcionalidades e gerenciar os serviços de conexão e comunicação deste equipamento às redes GSM.

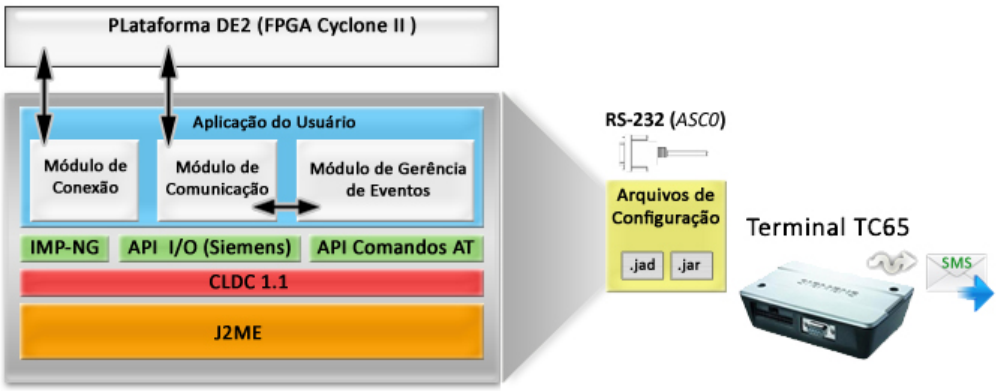


Figura 4.17: Arquitetura da Aplicação de Controle SMS.

A solução para a Aplicação de Controle SMS foi desenvolvida conforme o pseudocódigo do Algoritmo 4.1. Neste, tal aplicação consiste em ficar "escutando" a porta serial do TC65 para obter os parâmetros necessários para o encaminhamento da mensagem SMS, que são o código de área associado ao número do telefone celular de destino, bem como a própria mensagem de texto. Tais parâmetros são enviados pelo FPGA e separados automaticamente pela aplicação embarcada no TC65 por meio de identificadores de fim de comando. Tanto na aplicação embarcada na DE2 quanto nesta foi definido que o formato padrão do parâmetro número de telefone celular é especificado por meio do *COD_AREAFONE$* e da mensagem de texto a ser transmitida o formato *TEXTO#*. Os respectivos identificadores de fim de comando são os caracteres "**$**" e "**#**". Estes possuem

o papel de separar e determinar o final das informações referentes aos parâmetros que compõem a mensagem SMS. Um exemplo de entrada de tais parâmetros seria número = "84xxxxxxxx$" e mensagem = "LES-UERN#".

Algoritmo 4.1: Pseudocódigo da aplicação de controle de mensagens SMS

Require: Pacotes: com.siemens.icm.io.* and javax.microedition.*

Require: Cartão SIM inserido no TC65 com créditos disponíveis

```
1:  Início da Aplicação {Configuração inicial do TC65}
2:  ATCommand {Classe com suporte a execução de Comandos AT}
3:  ATCommandListener {Interface que define funcionalidades para tratar eventos AT}
    {Parâmetros para conexão serial}
4:  conexao ← Nome da Porta Serial e Taxa de Transmissão (Baud rate)
    {Parâmetros para mensagem SMS}
5:  int caracter ← -1;
6:  String mensagem ← null, numero ← null;
7:  stringBuffer ← new StringBuffer();
    {Execução}
8:  while (caracter ≠ 27) do
9:      while (caracter ← lerEntrada() ≠ $) do
10:         stringBuffer.concatena(caracter);
11:     end while
12:     numero ← stringBuffer;
13:     while (caracter ← lerEntrada() ≠ #) do
14:         stringBuffer.concatena(caracter);
15:     end while
16:     mensagem ← stringBuffer;
17:     enviarSMS(mensagem,numero){ ... atc.send("numero+mensagem+EOF"); ... }
18: end while
19: Fim da Aplicação
```

Ainda no pseudocódigo do algoritmo apresentado fica evidente a necessidade de API Java específicas para o desenvolvimento da Aplicação de Controle SMS, bem como de um cartão SIM inserido no Terminal TC65 para envio das mensagens SMS. Inicialmente, no construtor da aplicação são especificados alguns comandos AT para configuração inicial do terminal, como envio de mensagens no modo texto. Além disso, é preciso implementar uma classe com suporte aos comandos AT, os métodos da interface para gerenciar eventos AT e a configuração da comunicação serial para permitir a troca de dados com o FPGA que controla a DE2.

Três módulos foram desenvolvidos para a Aplicação de Controle SMS, que são os módulos de *Conexão*, *Comunicação* e *Gerência de Eventos*:

- *Conexão:* este faz um mapeamento dos métodos de conexão via serial (RS-232) do terminal TC65 com o dispositivo conectado, por meio da *API I/O Siemens*, onde se definem os parâmetros relativos a conexão como nome da porta serial e taxa de transmissão (*baud rate*);

- *Comunicação:* visando o processamento das linhas de comandos AT pelo terminal TC65, foi criado um módulo que especifica suas configurações iniciais, habilitando-o e personalizando-o, conforme as necessidades da aplicação. Este irá permitir a troca de informações com os demais dispositivos, neste caso um aparelho celular e a placa DE2;

- *Gerência de Eventos:* para a execução automática de ações pelo terminal TC65, desenvolveu-se um módulo que possibilitasse observar ("escutar") a ocorrência de eventos, permitindo disparar ações conforme evento reportado. Para esta aplicação, o módulo espera a chegada de um URC referente ao envio de um SMS, que é identificado por meio da recepção dos parâmetros SMS via comunicação serial.

Após o processamento da linha de comando responsável pelo envio da mensagem SMS, a Aplicação de Controle SMS retorna ao estado anterior que remete a observação de novos eventos, ou seja, a espera da definição de novos parâmetros SMS, isto é, número de telefone e texto, referentes a próxima mensagem a ser enviada. Este comportamento cíclico da aplicação é idealizado por meio de um laço (*loop*) executado até que um caracter de fim de aplicação seja dado como entrada pela interface serial. Caracter este definido como o código *ASCII* da tecla *ESC* cujo valor é 27. Outra forma de finalizar tal aplicação é por meio da ocorrência de algum evento externo ao terminal que cause sua desconexão, como, por exemplo, retirada do cartão SIM.

Quanto a forma de implantação (*deployment*) da aplicação *MIDlet* no terminal TC65, esta pode ser realizada por meio de uma ferramenta disponível pelo fabricante do módulo. Com a especificação da porta associada a interface serial ("**ASC0**") do TC65, tal ferramenta dá acesso total a unidade de armazenamento do mesmo. Isto permite fazer uma cópia dos arquivos JAR e JAD da *MIDlet* desejada para tal unidade. Em se tratando da capacidade de armazenamento do TC65, este é limitado até no máximo 2MB (SIEMENS, 2005b).

Veja na Figura 4.18 o fluxo de estados da aplicação quando devidamente configurada e pronta para ser executada pelo terminal TC65.

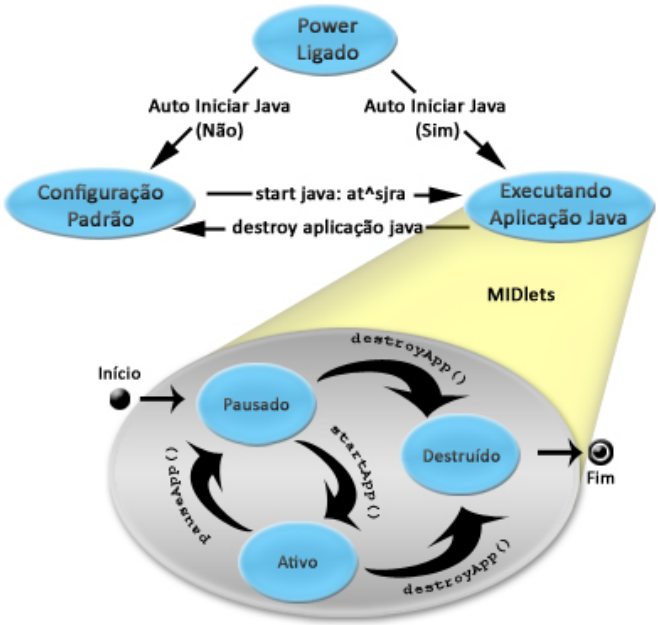


Figura 4.18: Fluxo de estados do Terminal TC65 com aplicação Java embarcada.

O processo de execução da aplicação de controle SMS visualizado na Figura 4.18, dá-se por meio da configuração do terminal TC65 pelo usuário do sistema. Para isso, este utiliza-se de alguns comandos AT específicos para consulta e personalização do terminal (**ATˆSCFG**), como também para manipulação das aplicações Java embarcadas no mesmo (como, por exemplo, **ATˆSJRA**, responsável por dá início a aplicação). Inicialmente, o usuário pode ou não configurar o terminal para auto inicializar a aplicação Java. Quando esta passa a ser executada, inicia-se também o ciclo de vida da *MIDlet*. Desta forma, a finalização da aplicação no terminal é diretamente dependente da destruição da *MIDlet* em execução ou do seu desligamento.

4.7 Conclusão

Neste Capítulo o objetivo foi mostrar o potencial das plataformas reconfiguráveis quando integradas a outras tecnologias e tendo seus projetos guiados por uma metodologia de desenvolvimento consistente, como é o caso da Hardware/Software *Codesign*. Foi possível compreender também como se dá o processo de desenvolvimento para os Sistemas Embarcados, bem como as aplicações desenvolvidas para gerenciamento e controle da placa DE2 e do módulo terminal TC65.

A flexibilidade e o poder computacional fornecido pelos FPGA aliado a alta capacidade de integração de uma arquitetura baseada em componentes de hardware, que, automaticamente, são gerenciados por um sistema embarcado, possibilitou criar uma Central de Alarme Parametrizável com características e recursos suficientemente avançados.

Devido à capacidade de reconfiguração destes dispositivos, o usuário pode adaptar o modelo proposto para adequar à outras aplicações, e, com isto, reduzir o tempo de desenvolvimento de novas soluções embarcadas.

Com a arquitetura parametrizável completamente sistematizada e configurada, dá-se início no Capítulo seguinte a apresentação de uma abordagem de verificação e validação formal para as aplicações desenvolvidas a partir de tal arquitetura.

Capítulo 5

Uma abordagem de verificação e validação formal

A abordagem formal de verificação e validação da arquitetura proposta aplicada ao cenário da Central de Alarme Parametrizável (CAP), descrita detalhadamente no Capítulo 4, consiste dos seguintes passos, como ilustrado na Figura 5.1:

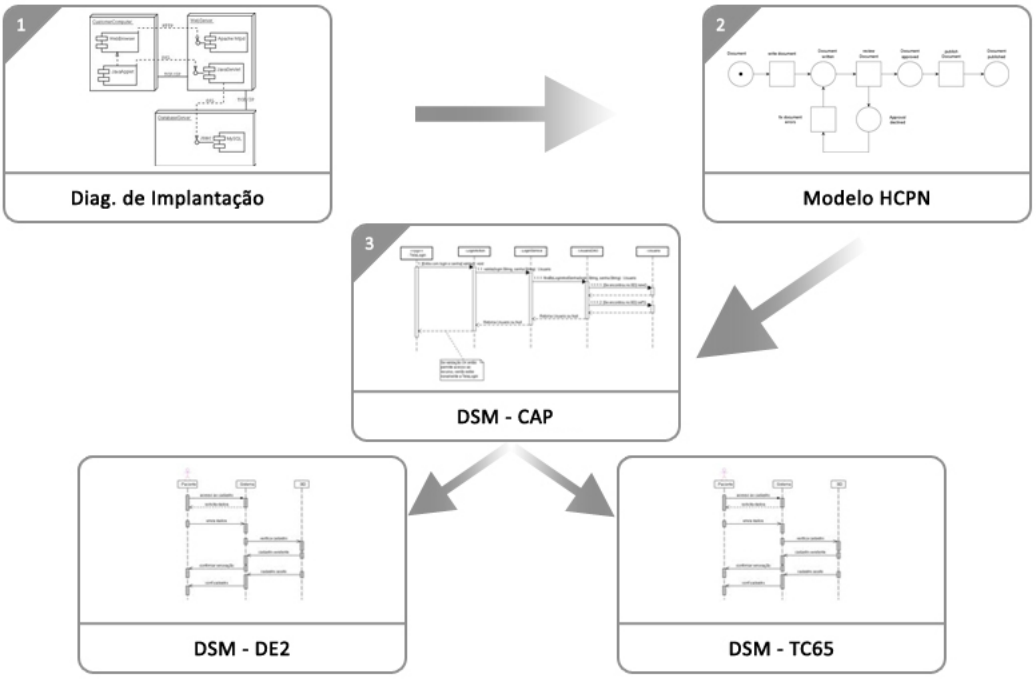


Figura 5.1: Abordagem de verificação e validação da CAP

- Construir o Diagrama de Implantação: usado para modelar os aspectos físicos do sistema. Este representa os dispositivos e a forma com se dá suas relações, bem como os artefatos a serem instalados em tais dispositivos para possibilitar o funcionamento do sistema;

- Construir o modelo HCPN: usado para facilitar a visualização e detectar problemas na especificação do projeto, como, por exemplo, inconsistência de dados e ambiguidades. Com este modelo é possível fazer a análise do comportamento do sistema por meio de simulação, no que diz respeito às suas propriedades dinâmicas;

- Construir os Diagramas de Sequência de Mensagens (DSM): usados para demonstrar as interações entre os objetos de interesse em um cenário que representa um conjunto de operações em um sistema. Para isso, faz-se uso das mensagens trocadas entre tais objetos, considerando a ordem temporal destas mensagens.

5.1 Introdução

A arquitetura parametrizável proposta nesta pesquisa, e apresentada no Capítulo 4, teve seu projeto definido com base em uma abordagem da metodologia Hardware/Software *Codesign*, descrita no Capítulo 3. Esta metodologia apesar de contemplar todas as atividades necessárias para o desenvolvimento de um sistema embarcado complexo, nas etapas de *Integração* e *Avaliação e Testes Incrementais* não são gerados artefatos que facilitem a visualização e a compreensão de tais etapas mediante ao que foi desenvolvido. Isto, devido as mesmas terem suas atividades realizadas pelo próprio projetista do sistema, sendo, portanto, executadas conforme às suas necessidades.

Neste contexto, buscou-se contribuir com tal metodologia integrando as etapas de *Integração* e *Avaliação e Testes Incrementais*, respectivamente, um Diagrama de Implantação e um conjunto de Diagramas de Sequência de Mensagens da UML (*Unified Modeling Language*) (BOOCH et al., 2005).

O Diagrama de Implantação permitirá abstrair para o usuário a forma como os elementos (nós) do sistema projetado estão dispostos e integrados. Além disso, este mostrará os principais componentes e necessidades relacionadas ao funcionamento de tais elementos.

O Diagrama de Sequência de Mensagens facilitará a visualização e a observação de como os objetos integrantes a cada um dos elementos da arquitetura se comunicam, em

termos de dados e operações para manipulação destes dados. Este diagrama analisa o processo de troca de mensagens entre tais objetos, considerando a ordem temporal em que ocorre tais atividades.

Visando a possibilidade de uma análise mais precisa do comportamento dinâmico dos sistemas que fizerem uso da arquitetura proposta, decidiu-se elaborar um modelo formal da mesma. O método formal adotado para a construção de tais modelos foi as Redes de Petri Coloridas Hierárquicas (*Hierarchical Coloured Petri Nets* - HCPN). Este modelo servirá para facilitar a compreensão e verificar a completude e a corretude do sistema, uma vez que modelos executáveis complementam a etapa de especificação. Quando simulações são realizadas erros e falhas no projeto advindos da especificação podem ser detectadas.

Tanto os diagramas UML quanto o modelo formal a serem apresentados neste Capítulo possibilitarão o entendimento prático do funcionamento da arquitetura proposta, quando aplicada a algum domínio de aplicação que tenham como componentes básicos, os dispositivos de hardware adotados para a mesma.

5.2 Diagrama de Implantação

Ao se desenvolver um sistema complexo de software, composto por componentes de software reutilizáveis, é preciso considerar as dimensões lógica e física do sistema. Para a modelagem dos aspectos físicos deste tipo de sistema existem na UML os conceitos de artefatos, nós e conexões. Os artefatos representam pacotes físicos de itens lógicos, tais como classes e interfaces. Um nó é um elemento físico que existe em tempo de execução e representa um recurso computacional, contendo, por exemplo, memória e capacidade de processamento. Os nós são os itens responsáveis pela execução dos artefatos. Por fim, as conexões são, em geral, idealizadas por meio de uma associação, representando uma conexão física entre os nós (BOOCH et al., 2005).

As representações gráficas fornecidas pela UML para um artefato, um nó e uma conexão são, respectivamente, um cubo, uma linha contínua e um retângulo inserido dentro do nó. Todos estes elementos podem usufruir do mecanismo de extensibilidade da UML denominado de esteriótipos. Os esteriótipos são utilizados para adequar a notação gráfica e melhor especificar um elemento modelado, facilitando seu entendimento. Um modelo construído por meio do relacionamento de tais elementos representa alguns dos aspectos estáticos do sistema, definindo o que se denomina de Diagrama de Implantação.

Na Figura 5.2 é apresentado o diagrama de implantação referente ao sistema da Central de Alarme Parametrizável, estudo de caso desta pesquisa. Nesta, ícones são utilizados para identificar os respectivos dispositivos que compõem o sistema.

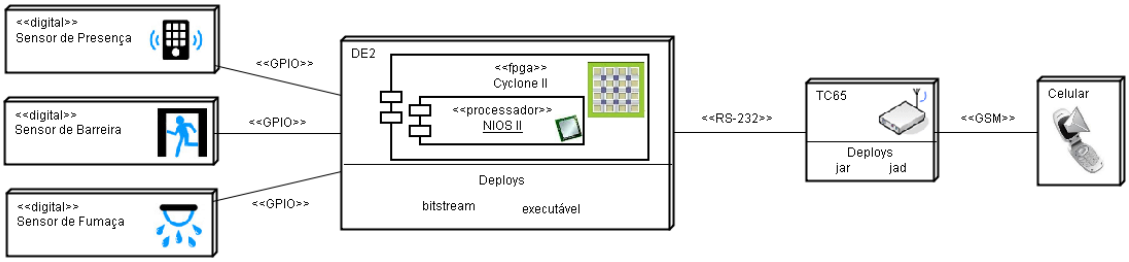


Figura 5.2: Diagrama de Implantação - Aplicação: Central de Alarme Parametrizável

Como se pode observar, os elementos físicos do sistema são do tipo *Sensor*, *DE2* (plataforma reconfigurável), *TC65* (terminal GSM/GPRS) e *Celular*. Para cada nó cujo esteriótipo é do tipo *Sensor* é atribuído um nome e um ícone que especifica os tipos de sensores existentes na arquitetura, que são *Sensor de Presença*, *Sensor de Barreira* e *Sensor de Fumaça*. As conexões que interligam cada sensor à DE2 são esteriotipadas com o nome do canal de comunicação pelo qual tais elementos realizam a troca de dados.

No nó *DE2* é especificado o principal componente de hardware da plataforma reconfigurável, isto é, o *FPGA Cyclone II*, cujo processador é o *Nios II*. Este nó contém ainda os artefatos que necessitam ser embarcados na DE2 para permitir que a aplicação de software, arquivo executável em linguagem C, possa acessar e manipular os componentes de hardware habilitados e disponíveis para o sistema, conforme especificado no arquivo *bitstream*. Assim como o nó *DE2*, o nó *TC65* referente ao módulo terminal GSM, responsável pelo envio de mensagens SMS, necessita de artefatos que são os arquivos JAR e JAD da MIDlet que controla tal equipamento. Estes também devem ser embarcados para permitir a execução da aplicação Java no TC65. A conexão entre os nós *DE2* e *TC65* é esteriotipada mostrando o tipo de conexão entre tais dispositivos, sendo esta do tipo RS-232, que retrata o padrão de comunicação serial. Para completar a estrutura de implantação do sistema, um nó de caráter genérico cujo nome é *Celular* define o tipo de dispositivo que receberá as mensagens SMS advindas do terminal TC65 por meio de uma conexão GSM.

5.3 Modelo Formal

A principal motivação para construir um modelo formal do sistema referente à Central de Alarme Parametrizável é o fato deste tratar-se de um sistema embarcado complexo, onde torna-se necessário analisar suas propriedades dinâmicas. Tais propriedades como dados e restrições de tempo podem ser modeladas, possibilitando a tomada de decisão do projetista do sistema. Pois o modelo permitirá realizar uma análise mais detalhada do fluxo de execução e comportamento do sistema por meio de simulação.

Para tal modelagem utilizou-se dos conceitos e particularidades das Redes de Petri Coloridas Hierárquicas, conhecidas na literatura como HCPN.

5.3.1 HCPN

Uma HCPN é uma extensão das Redes de Petri Colorida (*Coloured Petri Nets* - CPN), sendo, portanto, uma linguagem de modelagem adequada para a verificação da corretude de sistemas. Uma CPN combina as capacidades de redes de Petri com os recursos de uma linguagem de programação de alto nível, fornecendo a notação gráfica e as primitivas base para a modelagem de concorrência, comunicação e sincronização (JENSEN; KRISTENSEN, 2009).

Um modelo HCPN pode ir além das características de uma CPN, representando aspectos de concorrência, paralelismo e não-determinismo com diferentes níveis de abstração do sistema (RIBEIRO NETO et al., 2007) .

Um sistema que englobe estas características é fundamental que funcione corretamente desde sua concepção, uma vez que sua execução pode proceder de diversas formas e dependendo do domínio de aplicação, falhas podem acarretar sérios prejuízos, sendo necessário, portanto, testá-lo e depurá-lo com maior precisão (JENSEN; KRISTENSEN, 2009).

Conforme ilustrado na Figura 5.3, os níveis de abstração permitidos para uma HCPN é alcançado por meio de dois mecanismos: transições de substituição e lugares de fusão (RIBEIRO NETO, 2006) (RIBEIRO NETO et al., 2007).

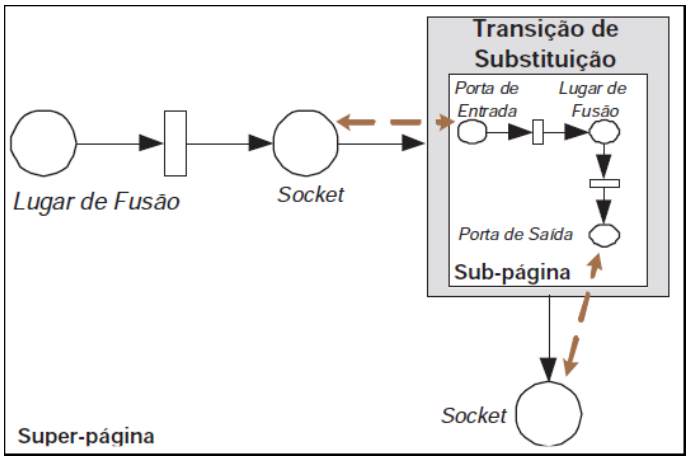


Figura 5.3: Níveis de abstração de uma HCPN. Fonte: (RIBEIRO NETO, 2006)

Uma transição de substituição é uma transição que descreve mais detalhadamente as atividades modeladas por elas. Esta é representada por uma sub-página CPN que deve estar contida em uma super-página CPN, isto é, em um nível mais abstrato do modelo. A relação entre uma sub-página e uma super-página é idealizada por meio de *sockets* e portas. Os *sockets* são todos os lugares de entrada e saída da transição na super-página. As portas são os lugares na sub-página associados aos *sockets*, podendo ser de entrada, saída ou entrada-saída.

Quanto aos lugares de fusão, estes são estruturas que permitem especificar um conjunto de lugares como funcionalmente um único lugar. Isto significa que quando uma ficha é removida ou adicionada em um destes lugares, uma ficha idêntica é removida ou adicionada dos demais lugares pertencentes ao conjunto. Assim, todos os lugares pertencentes ao conjunto de fusão possuem a mesma marcação, ou seja, o mesmo número e tipo de ficha em um determinado instante da execução. A cada ficha associa-se um valor denominado *cor da ficha* que pode representar tipos arbitrários de dados complexos como, por exemplo: inteiros, reais, registros, etc. Logo, a marcação de todos os lugares em uma rede de petri representa o estado do sistema naquele momento.

5.3.2 *CPN Tools*

CPN Tools (JENSEN; KRISTENSEN, 2009) é a ferramenta computacional utilizada na edição, simulação, análise de espaço de estado e análise de desempenho de modelos CPN. Esta dá suporte às redes de Petri do tipo HCPN, oferecendo também a capacidade

de agregar aos modelos, características de tempo para simular o comportamento dos Sistemas em Tempo-Real.

As principais características do projeto *CPN Tools* são: utilização de notação gráfica para representar o modelo CPN; suporte a checagem de tipos e sintaxe dos objetos modelados; realização de simulação interativa e automática com possibilidade de gerar relatórios por meio desta; suporte a geração e a análise do espaço de estados do modelo, permitindo evidenciar erros no sistema; análise de desempenho dos modelos baseada em simulação; e, capacidade de estender suas funcionalidades por meio de um conjunto de bibliotecas adicionais disponíveis.

Por ser uma ferramenta distribuída gratuitamente, esta passou a ser uma das mais importantes ferramentas utilizadas para a modelagem de redes de Petri. Além de permitir trabalhar e manipular com tipos de dados complexos (conjunto de cores) com o uso da linguagem *Standard ML* (SML) (RIBEIRO NETO et al., 2007), suas versões são suportadas pelas plataformas operacionais Windows e Linux (JENSEN; KRISTENSEN, 2009).

5.3.3 Modelo HCPN

Declarações

Como forma de facilitar a compreensão do modelo HCPN, as cores, as variáveis e as funções especificadas são apresentadas e comentadas, conforme as necessidades da arquitetura proposta e estudo de caso desenvolvido.

Cores

```
(*** Cor de Inicialização ***)

   color UNIT = unit timed;

(*** Cores dos Tipos Primitivos ***)

   color STRING = string;
   color INT = int;
   color BOOL = bool;
   color STATUS = int;
```

```
(*** Cor de Fim de Comando ***)

   color EOF = string;

(*** Especifica os diferentes tipos de sensores:  Presença (P),
Barreira (B) ou Fumaça (F) ***)

   color TipoSensor = with P | B | F;

(*** Especifica os possíveis valores de estado dos sensores ***)

   color TipoSinal = int with 0 ..  1;

(*** Especifica os diferentes tipos de URC dos eventos AT ***)

   color TipoURC = with SMS_ | OUTROS_;

(*** Define os parâmetros dos pacotes de mensagens ***)

   color Num_Fone = string with "0" ..  "9";
   color Msg_Texto = string;

(*** Registro que define a estrutura de um pacote de mensagem ***)

   color Pacote = record Num_Fone:STRING * Msg_Texto:STRING;

(*** Associa o URC do evento AT conforme pacote recebido ***)

   color URC_ = product Pacote * TipoURC timed;

(*** Registro que define a estrutura de sensor ***)

   color Sensor = record tipoSensor:TipoSensor * status:STATUS;

(*** Define uma lista de sensores ***)

   color Sensores = list Sensor;

(*** Define a DE2 com suporte a região de tempo ***)

   color DE2 = with de2 timed;

(*** Especifica a estrutura para processamento da lista de eventos dos
sensores ***)

   color DE2xSensor = product DE2 * Sensor timed;

(*** Registro que define a estrutura de uma mensagem SMS ***)

   color SMS = record Num_Fone:STRING * Msg_Texto:STRING;
```

```
(*** Especifica a estrutura para envio das mensagens SMS ***)

   color SMSxEOF = product SMS * EOF timed;
```

Variáveis

```
(*** Constantes de fim de comando ***)

   val EON = "$"; //End-Of-Number
   val EOM = "#"; //End-Of-Message
   val EoF = "26"; //End-of-File

(*** Constantes das mensagens ***)

   val numero_fone = "84xxxxxxx";
   val mensagem_texto1 = "Presença Detectada!";
   val mensagem_texto2 = "Fumaça Detectada!!!";

(*** Parâmetros dos pacotes de mensagens ***)

   var Num_SMS:Num_Fone;
   var Msg_SMS:Msg_Texto;

(*** Define o sensor que detecta o evento ***)

   var sensor :  Sensor;

(*** Define a lista de sensores que detectaram eventos ***)

   var sensores :  Sensores;

(*** Define o pacote de mensagem ***)

   var pct :  Pacote;

(*** Define a mensagem SMS ***)

   var sms :  SMS;

(*** Define o URC do pacote ***)

   var urc :  TipoURC;
```

Funções

```
(*** Calcula o tempo médio para gerar um evento ***)

fun TempoEsporadico(mean:int) =
   let
      val realMean = Real.fromInt mean
      val rv = exponential((1.0/realMean))
   in
      floor(rv+0.5)
   end;
```

(*** Gera eventos para um determinado sensor escolhido aleatoriamente. Quando o campo *status* assume o valor 1 indica que o sensor escolhido detectou um evento, caso assuma o valor 0 nenhum evento foi detectado ***)

```
fun novoEvento() :  Sensor = {
   tipoSensor = TipoSensor.ran(), status = TipoSinal.ran()

};

(*** Atribui os parâmetros do pacote de mensagem ***)

fun criarPacote(sensor:Sensor) :  Pacote = {
   Num_Fone = numero_foneˆEON,
   Msg_Texto = if (#tipoSensor sensor <> F) then
      mensagem_texto1ˆEOM
   else
      mensagem_texto2ˆEOM

};
```

(*** Simula a consulta do tipo de URC conforme evento AT recebido pelo TC65. Se o parâmetro *#Num_Fone* do pacote recebido for diferente de uma cadeia de caracteres vazia então o URC é relativo a um SMS, senão a outro tipo de URC, como, por exemplo, uma chamada telefônica (*RING*) ***)

```
fun obterURC(pct:Pacote) :  TipoURC = {
   case (String.compare(#Num_Fone pct,"")) of
      LESS => SMS_ | GREATER => SMS_ | EQUAL => OUTROS_;

};
```

```
(*** Atribui os parâmetros da mensagem SMS ***)

fun criarSMS(pct:Pacote) :  SMS = {
   Num_Fone = substring(#Num_Fone pct,0,10),
   Msg_Texto = substring(#Msg_Texto pct,0,19)

};
```

Modelagem

Assim como no Diagrama de Implantação UML apresentado na Seção 5.2, a modelagem HCPN dos dispositivos pertencentes à arquitetura proposta foi adaptada para o cenário da Central de Alarme Parametrizável. Esta foi sistematizada conforme as funcionalidades inerentes a tais dispositivos, como ilustrado na Figura 5.4.

O modelo HCPN para o cenário da CAP consiste de quatro partes. A parte *Sensores* que possui uma transição de substituição denominada de *Sensoriar*. A parte *Plataforma DE2 (FPGA)* composta pela transição de substituição *Processar*. A parte TC65 contém a transição de substituição *Atuar*. E, finalmente, a parte *Celular* com a transição *Receber SMS*. A interface entre *Sensores* e *Plataforma DE2 (FPGA)* é realizada pelo lugar *Detectando*, enquanto que entre a *Plataforma DE2 (FPGA)* e o *TC65* pelo lugar *Processando*. Entre as partes *TC65* e *Celular* a interface é fornecida pelo lugar *Enviando SMS*.

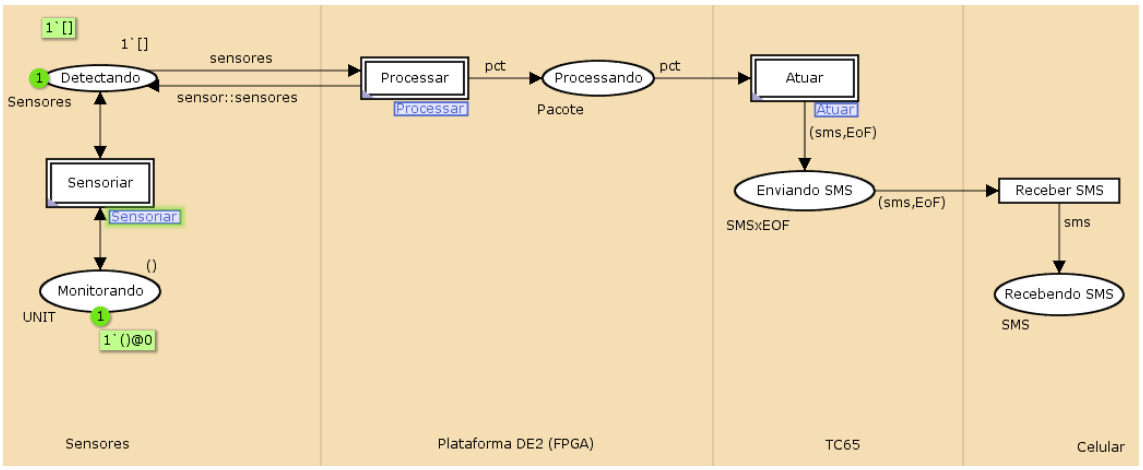


Figura 5.4: Modelo HCPN da CAP (super-página)

O lugar *Monitorando* localizado em *Sensores* funciona como o ponto de partida para a inicialização e o controle de tempo de execução do sistema. Para isto, adicionou-se

a inscrição do arco de saída um rótulo de tempo representado por *()@+TempoEsporadico(100)*, como ilustrado na Figura 5.5.

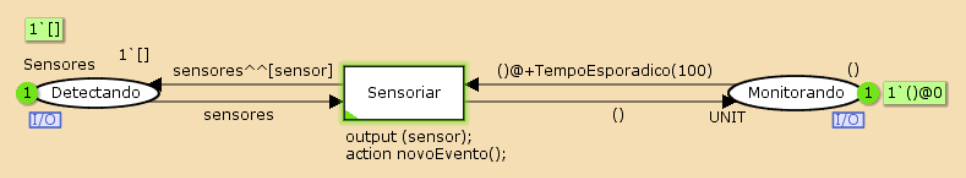


Figura 5.5: Sub-página da transição de substituição Sensoriar

A simbologia @+ especifica uma região de tempo, que serve para definir o tempo correpondente a realização de uma determinada operação. *TempoEsporadico(100)* é uma função ML declarada, codificada utilizando a sintaxe da linguagem de programação funcional *Standard ML* (SML), que retorna um valor referente ao tempo médio para habilitar a transição de substituição *Sensoriar* durante todo o decorrer da simulação. Isto indica o tempo em que o próximo evento deve ser gerado. O valor *100* é usado como parâmetro para a função exponencial randômica que retorna um valor médio entre 100 unidades de tempo. O rótulo de tempo é usado para garantir que o primeiro sensor nem sempre chegará no instante 0 para as diferentes simulações. O tipo de marcação suportado pelo lugar *Monitorando* é o *UNIT* temporizado (*unit timed*), que inicialmente possui marcação vazia *()*.

A transição *Sensoriar* localizada na sub-página da transição de substituição *Sensoriar* ao ser habilitada executa a função ML *novoEvento()*. O papel desta função é gerar um evento para um dos tipos de sensores definidos e escolhido do conjunto de cores *TipoSensor* juntamente com um estado associado do conjunto de cores *TipoSinal*. A função *novoEvento()* possui como saída um elemento do tipo *Sensor* especificado como um registro contendo os campos *tipoSensor* e *status*. Uma possível saída para a variável *sensor* é *tipoSensor = B* e *status = 1*. Isto corresponde a um evento gerado para o sensor de barreira com detecção de sinal, devido ao valor 1 da variável *status*. Após retornada a variável *sensor* pela função *novoEvento()*, esta é adicionada a variável *sensores* que é definida como uma lista de elementos do tipo *Sensor*. Apesar de ser uma lista, seu funcionamento é similar ao de uma fila do tipo FIFO (*First-In-First-Out*), sendo esta construída para processar todos os eventos advindos dos sensores. A inscrição do arco de saída da transição *Sensoriar* para o lugar de entrada-saída *Detectando* é *sensores^^[sensor]*. A marcação do lugar *Detectando* é do tipo *Sensores*, que de início contém uma lista vazia *[]*.

Na Figura 5.6 são ilustradas as atividades modeladas referentes a sequência de operações executadas pela aplicação embarcada na *Plataforma DE2 (FPGA)*.

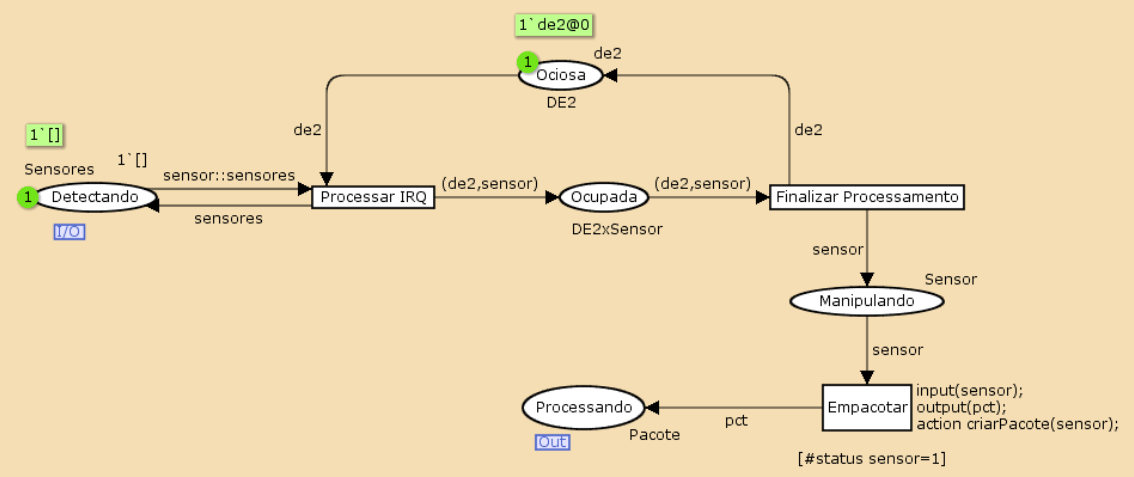


Figura 5.6: Sub-página da transição de substituição Processar

Tais operações estão relacionadas ao processamento dos eventos capturados pelos sensores, que foram encaminhados pelo lugar *Detectando* e que estão descritas na transição de substituição *Processar*. Ao receber um elemento da fila *sensor::sensores* e verificada a existência de uma ficha no lugar *Ocioso*, que representada o estado disponível da *DE2*, dá-se início ao processamento de tais eventos por meio da transição *Processar IRQ*. Esta transição refere-se na prática uma interrupção de hardware advinda do sinal emitido pelo sensor. Quando disparada esta transição, o sensor o qual está tendo seu evento processado pela DE2 é retirado da fila. A marcação do lugar *Detectando* é atualizada para que, ao término de tal processamento, um outro sensor possa ter seu evento também processado. Uma ficha no lugar *Ocupada* indica que a *DE2* está processando um evento gerado por um determinado sensor. A marcação inicial do lugar *Ocupada* é vazia. Ao finalizar o processamento do evento pela DE2, a transição *Finalizar Processamento* envia uma ficha do tipo *de2* para o lugar *Ociosa* e outra do tipo *sensor* para o lugar *Manipulando*. Esta será utilizada para criar o pacote de mensagem que conterá os parâmetros SMS que serão repassados para o terminal *TC65*, que se responsabilizará pelo envio da mensagem SMS para o *Celular*. Na transição *Empacotar* uma condição de guarda é adicionada para que apenas os sensores com o campo *status* igual a um tenham os pacotes de mensagens personalizados, descartando àqueles cujo *status* é igual a zero. Assim, em tal transição existe uma função ML *criarPacote(sensor)*, que recebe como entrada um elemento do tipo *Sensor* e gera como saída um elemento do tipo *Pacote*. Um elemento

do tipo *Pacote* é definido como um registro com os campo *Num_Fone* e *Msg_Texto*, ou seja, os parâmetros para envio da mensagem SMS. Tal pacote é enviado para o lugar *Processando*, que é um lugar de saída na transição de substituição *Processar* e de entrada na transição *Gerenciar Evento AT* da sub-página da transição de substituição *Atuar*.

As atividades modeladas na sub-página da transição de substituição *Atuar* são apresentadas na Figura 5.7.

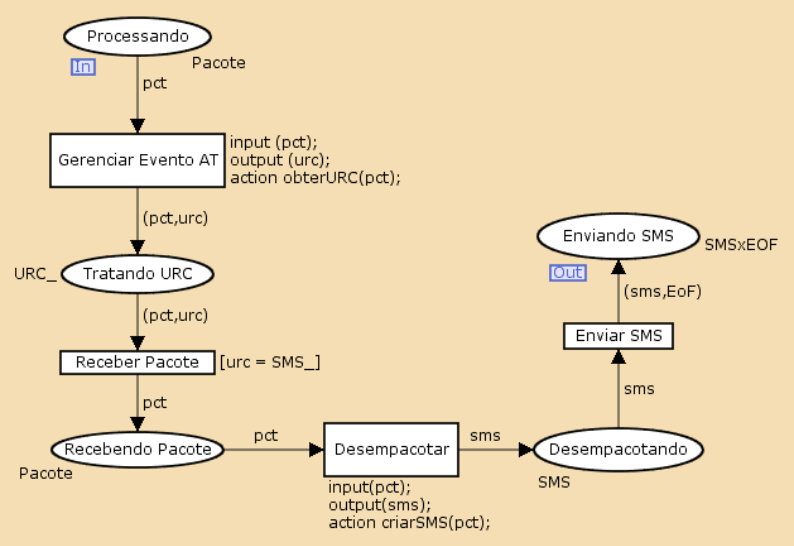


Figura 5.7: Sub-página da transição de substituição Atuar

Tais atividades fazem parte das ações que devem ser realizadas pelo terminal TC65 responsável pelo envio das mensagens SMS. Percebe-se na figura mencionada que ao receber um pacote de mensagens da DE2, habilita-se a transição *Gerenciar Evento AT*. Nesta, uma função ML *obterURC(pct)* recebe como entrada uma variável do tipo *Pacote*, retornando como saída o *TipoURC*, que corresponde ao evento AT recebido pelo TC65. Esta situação foi apenas modelada para descrever o comportamento real que é realizado por tal equipamento. Assim, ao verificar que o campo *Num_Fone* do pacote recebido é diferente de vazio *“”*, o valor retornado para a variável *urc* é *SMS_*, caso contrário esta assume o valor *OUTROS_*. Feito isto, o lugar *Tratando URC* aguarda o pacote juntamente com o tipo especificado de *URC*, repassando tais informações para a transição *Receber Pacote*. Esta transição possui uma condição de guarda que especifica que somente os pacotes cuja variável *urc* contiver o valor igual a *SMS_* devem ter seu conteúdo extraído, caso contrário, estes serão descartados. O lugar *Recebendo Pacote* repassa para a transição *Desempacotar* apenas os pacotes válidos. Nesta transição, a função ML

criarSMS(pct) ao receber o pacote encaminhado pela DE2, extrai os parâmetros SMS com base no conteúdo do pacote. Para isto, utilizou-se identificadores de fim de comando concatenados aos parâmetros enviados, possibilitando separar tais parâmetros e criar a mensagem SMS. A variável *sms* é do tipo *SMS*, que consiste de um registro contendo a mesma estrutura do tipo *Pacote*, porém sem os identificadores de fim de comando. O lugar *Desempacotando* recebe a ficha do tipo *sms* e encaminha-a juntamente com o identificador de fim de arquivo *EoF*, cujo valor foi anteriormente definido, para o lugar de saída *Enviando SMS*. Tal atividade é realizada por meio da transição *Enviar SMS*. Com isto, é especificada a estrutura geral necessária para que o terminal TC65 envie a mensagem SMS para o dispositivo de destino, isto é, o celular.

Na parte *Celular*, de acordo com o modelo HCPN anteriormenre ilustrado na Figura 5.4, a transição *Receber SMS* recebe as informações transferidas pelo lugar *Enviando SMS*, e uma ficha do tipo SMS é enviada para o lugar *Recebendo SMS*. Na prática, esta transição seria encarregada de executar a linha de comando AT responsável por enviar a mensagem SMS. Isto finaliza um ciclo completo de execução do sistema da Central de Alarme Reconfigurável, culminando na interação de todos os dispositivos da arquitetura parametrizável proposta neste trabalho.

5.4 Diagramas de Sequência de Mensagens

Os Diagramas de Sequência de Mensagens (DSM) fazem parte de um conjunto de diagramas UML, conhecidos como Diagramas de Interação, que são utilizados na modelagem de aspectos dinâmicos de um sistema.

Um Diagrama de Interação mostra as interações entre um conjunto de objetos por meio de seus relacionamentos. Esta relação é caracterizada pela troca de mensagens entre tais objetos. Um DSM é uma especialização dos diagramas de interação com ênfase na ordem temporal das mensagens, ilustrando o comportamento de um cenário em um dado momento (BOOCH et al., 2005).

Na modelagem de sistemas complexos, como no caso dos sistemas embarcados, a atividade de observar as inúmeras situações e estados do sistema em tempo de execução torna-se ainda mais difícil, visto a possibilidade da existência de vários fluxos de controle. Portanto, os DSM servem para modelar os vários cenários de um sistema, com base na análise do comportamento dos objetos de interesse, considerando o tempo de ocorrência das mensagens trocadas entre eles.

Graficamente, a UML representa os objetos de interação por meio de retângulos distribuídos horizontalmente no diagrama, colocando-se da esquerda para direita àqueles que executam a ordem temporal da interação. Tais mensagens são organizadas verticalmente, da parte superior para parte inferior do diagrama, evidenciando visualmente o fluxo de controle no decorrer do tempo. Cada objeto possui uma linha de vida associada, que representa o período de tempo de sua existência, podendo estes serem criados ou destruídos durante a interação (BOOCH et al., 2005).

Os diagramas de sequência de mensagens UML apresentados a seguir, descrevem o fluxo de controle de mensagens trocadas entre os componentes da arquitetura proposta. O cenário principal, que será apresentado na Seção 5.4.1, demonstra a interação entre tais componentes no contexto geral da Central de Alarme Reconfigurável, de uma forma mais abstrata. Para o cenário da aplicação que controla os componentes de hardware da DE2, apresentado na Seção 5.4.2, os objetos são referentes às funcionalidades implementadas para seu controle interno e execução de entrada e saída dados. E, por último, o cenário da aplicação de controle do terminal TC65 tem as interações de seus objetos enfatizadas no DSM apresentado na Seção 5.4.3, demonstrando como esta realiza a gerência de eventos AT e encaminha as mensagens SMS para o Celular de destino.

5.4.1 Diagrama de Sequência de Mensagens - CAP

Na Figura 5.8 é apresentado o DSM da Central de Alarme Parametrizável. Conforme ilustrado, o sistema possui quatros tipos de objetos participantes que são os *Sensores*, a placa *DE2*, o terminal *TC65* e o aparelho *Celular*. A priori, os sensores estão integrados à placa DE2 que está executando uma aplicação embarcada em linguagem C, cujo objetivo é realizar o controle dos componentes de hardware por meio de seu FPGA.

O monitoramento do ambiente é de responsabilidade dos sensores de presença, barreira e fumaça. Quando estes detectam seus respectivos eventos, enviam um sinal digital para a placa DE2 por meio da interface de comunicação GPIO. Como a placa DE2 permanece realizando o gerenciamento das interrupções de hardware, esta consegue capturar os sinais emitidos pelos sensores. Ao detectar o sinal, a aplicação da DE2 cria um pacote contendo uma cadeia de caracteres com os parâmetros *número do telefone celular* e *mensagem de texto*. A cada um destes parâmetros é associado um identificador de fim de comando, sendo definido para o primeiro parâmetro o caractere $ e para o segundo o caractere #. Tal pacote é encaminhado via interface de comunicação serial para o equipamento terminal TC65, sendo logo após destruído. No TC65 também encontra-se

embarcada uma aplicação em linguagem Java que aguarda os devidos parâmetros para envio da mensagem SMS. Esta permanece "escutando" a porta serial e ao obter o pacote encaminhado pela DE2, é necessário que tal aplicação desmembre-o para extrair os parâmetros *número do celular* e *mensagem de texto*.

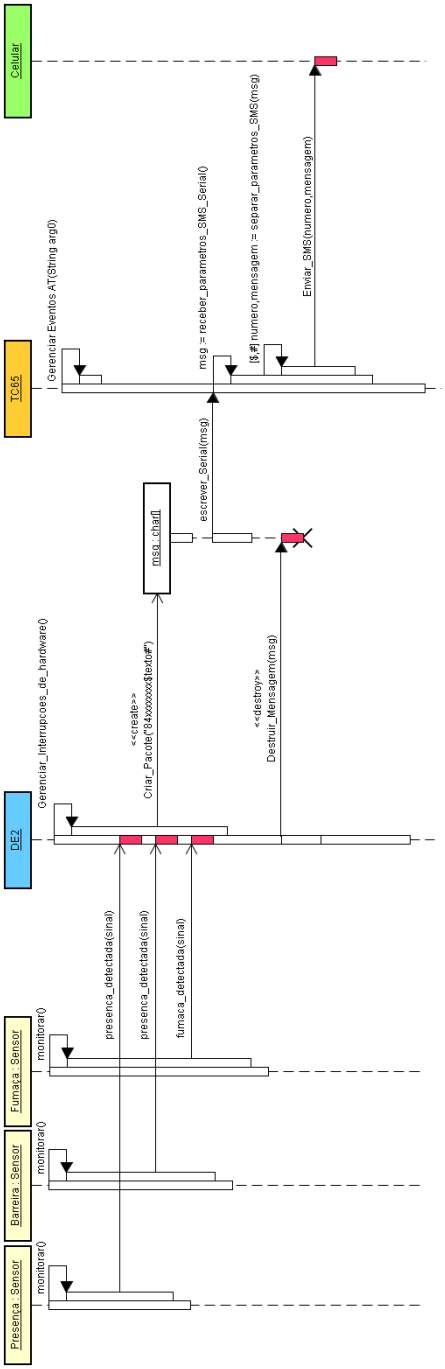


Figura 5.8: Diagrama de Sequência de Mensagens - Central de Alarme Parametrizável

A aplicação possui ainda uma funcionalidade que agrega ao terminal TC65 a capacidade de identificar o tipo de evento AT que o mesmo recebe, e, portanto, orientá-lo em relação às proximas atividades a serem realizadas. Para finalizar, o TC65 encaminha a mensagem SMS para o celular de destino, de acordo com os parâmetros recebidos.

5.4.2 Diagrama de Sequência de Mensagens - DE2

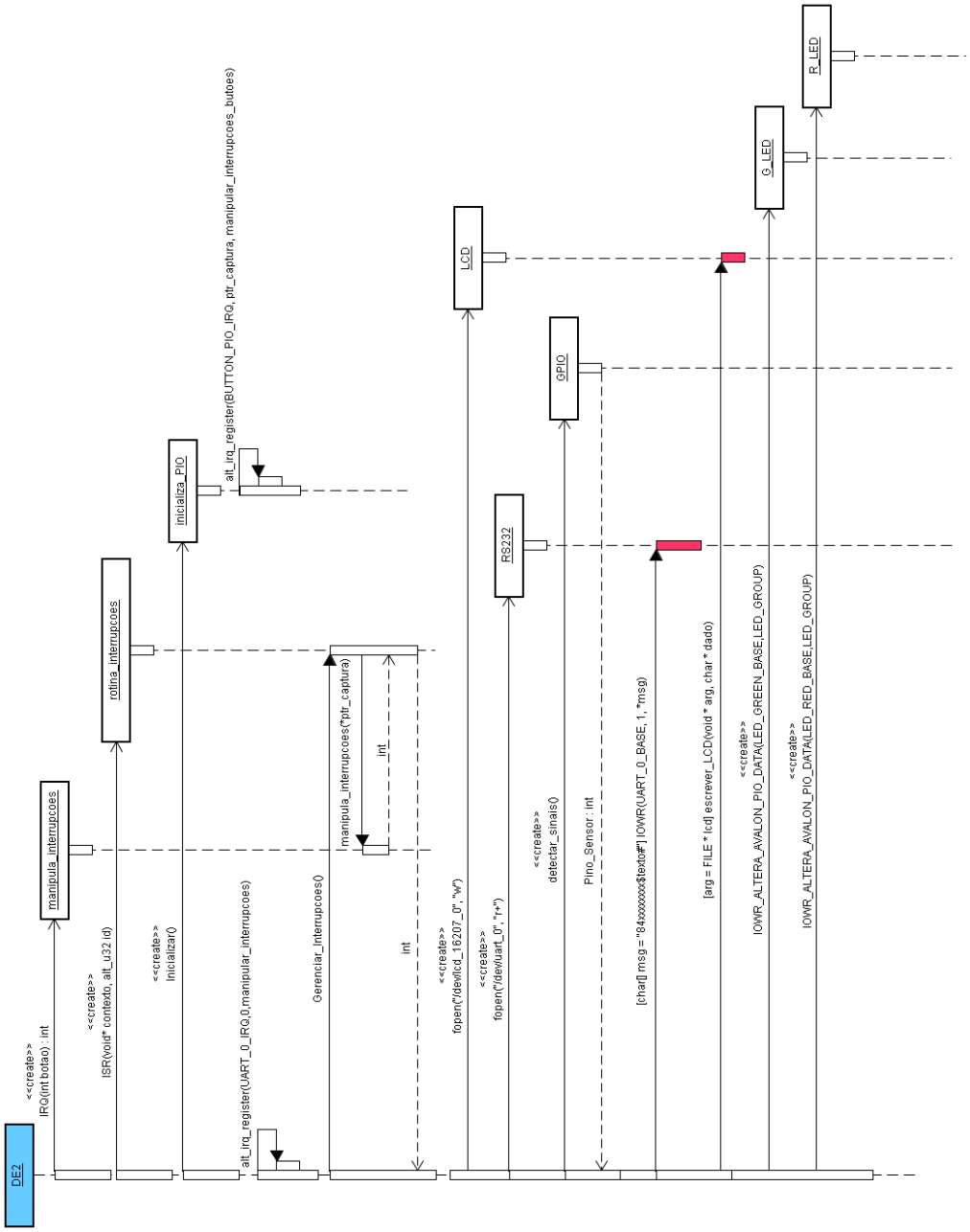


Figura 5.9: Diagrama de Sequência de Mensagens - DE2

Conforme ilustrado na Figura 5.9, este representa o cenário da aplicação embarcada na plataforma reconfigurável DE2. Inicialmente, para permitir o gerenciamento das interrupções de hardware é preciso definir três procedimentos: a função que recebe os identificadores das linhas de interrupção (IRQ); a função que escreve a rotina de serviço de interrupção (*Interrupt Service Routine* - ISR), responsável por invocar o componente de hardware em resposta aos IRQ e informá-lo como agir; e, por fim, a função de inicialização, que habilita tais componentes para possibilitar que a aplicação obtenha seus IRQ. Após definidos tais procedimentos é necessário fazer o registro das ISR para os componentes utilizados. A partir deste instante a plataforma DE2 está apta a manipular e realizar o gerenciamento das interrupções de hardware.

Ao inicializar a aplicação, alguns componentes, tais como display LCD e canal serial (UART), têm suas interfaces de comunicação abertas para acesso de leitura e escrita pela aplicação. Os componentes de hardware, cujas atividades foram enfatizadas neste DSM são: *LCD*, *RS232*, *GPIO*, *G_LED* e *R_LED*. O objeto *DE2* modelado neste diagrama representa o FPGA e seu respectivo processador, cuja responsabilidade é controlar os demais componentes de hardware.

A função definida como *detectar_sinais()* é responsável por obter os sinais dos sensores que estão integrados ao canal de comunicação representado pelo objeto GPIO. Um sensor ao capturar um evento, seja ele de presença, barreira ou fumaça, emite um sinal digital por meio do pino ao qual este se integra à GPIO, possibilitando que a aplicação identifique a origem deste sinal. Ao detectar a origem da interrupção, um pacote de mensagem é personalizado pela aplicação contendo como conteúdo uma cadeia de caracteres com informações referentes ao número de celular e mensagem de texto, sendo estes separados por caracteres identificadores de fim de comando. Esta mensagem, é então enviada via interface de comunicação serial representado pelo objeto *RS232*. Após realizar o envio do pacote, informações são inseridas no *display* LCD, onde é exibido o conteúdo da mensagem contida no pacote.

Os objetos *G_LED* e *R_LED* representam, respectivamente, o conjunto de *led* verdes e vermelhos. Estes *led* acendem conforme acionamento de botões e chaves da plataforma DE2, bem como também são utilizados para exibir o estado de algumas operações, como a finalização da aplicação.

5.4.3 Diagrama de Sequência de Mensagens - TC65

O DSM que descreve o cenário da aplicação embarcada no terminal TC65 é apresentado na Figura 5.10.

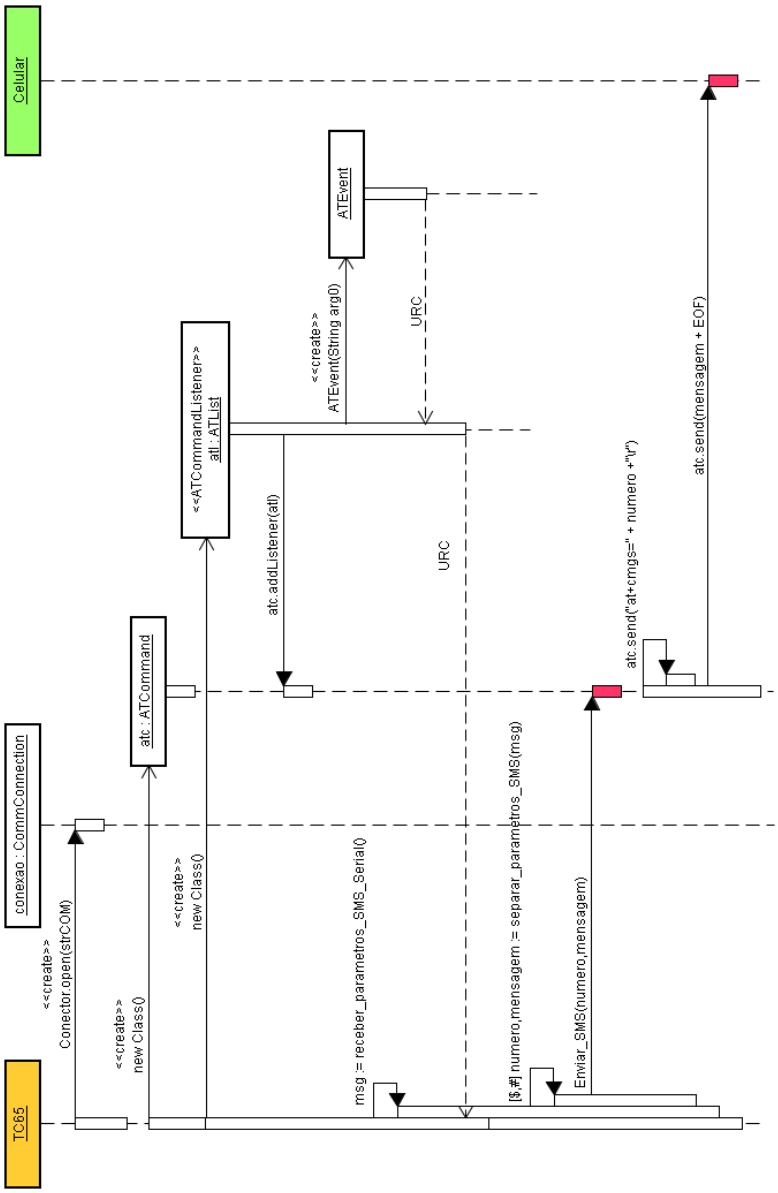


Figura 5.10: Diagrama de Sequência de Mensagens - TC65

Para esta aplicação é indispensável implementar uma instância da classe **ATCommand**, bem como os métodos da interface **ATCommandListener** por meio de uma outra classe Java. Tal classe e interface fazem parte da API de programação contida no pacote **com.siemens.icm.io**, específica para os equipamentos da *Benq Siemens*. Uma das prin-

cipais operações da classe **ATCommand** nesta aplicação, é realizar a configuração do equipamento terminal e encaminhar as mensagens SMS. Já a interface **ATCommandListener** define métodos que possibilitam o terminal TC65 receber eventos AT não solicitados (*Unsolicited Result Code* - URC), como uma chamada de voz ou SMS.

Ao receber o pacote com a mensagem advinda da DE2 via interface de comunicação serial, tal aplicação deve separar os parâmetros para envio do SMS, que são o *número* e a *mensagem* de texto. Isto é possível, devido aos identificadores de fim de comando que encontram-se concatenados ao conteúdo de tal pacote. Finalizada a separação de tais parâmetros, estes são agrupados com o caractere de fim de arquivo (*End-of-File* - EOF) para o encaminhamento da mensagem SMS.

O formato definido nas configurações do TC65 é do tipo *modo texto*. O método da classe **ATCommand** com a finalidade de enviar a mensagem SMS é o método *send()*. Primeiro, este recebe como parâmetros o comando AT juntamente com o número do celular de destino concatenado com *"\r"*, como no exemplo mostrado neste DSM, ou seja, *"atc.send(at+cmgs =" + numero + "\r")*. Após o terminal TC65 processar esta linha de comando AT, a mensagem propriamente dita é encaminhada também por meio do método *send()*. Porém, nesta linha de comando AT, os parâmetros são a mensagem de texto concatenada com o identificador de fim de arquivo, conforme o seguinte exemplo: *"atc.send(mensagem + EOF)*. Na aplicação, EOF é uma variável do tipo *char* que assume o valor decimal igual a 26, cuja tabela ASCII especifica-o como um caractere de controle denominado de *Substitute*, que tem seu funcionamento similar ao presionamento das teclas *Crtl+Z* (W3C, 2010).

5.5 Conclusão

Neste Capítulo o objetivo foi apresentar mais duas contribuições, além da arquitetura proposta nesta pesquisa. A primeira, de agregar à metodologia de Hardware/Software *Codesign* os artefatos Diagrama de Implantação e Diagrama de Sequência de Mensagens UML às suas respectivas etapas de *Integração* e *Avaliação e Testes Incrementais*. E a segunda, a elaboração de um modelo formal em HCPN construído a partir da arquitetura de hardware reconfigurável desenvolvida com base em uma metodologia consistente, que representa fidedignamente tal arquitetura e permite analisar seu comportamento por meio de simulações do modelo executável.

Tais contribuições irão esclarecer mais ainda a compreensão do funcionamento da

arquiterura proposta, por parte de outros pesquisadores, quando aplicadas à domínios adequados, como no caso da Central de Alarme Parametrizável, estudo de caso desta dissertação.

Capítulo 6

Conclusões e Trabalhos Futuros

A idealização deste trabalho é oriundo de um projeto de pesquisa, cujo título é "Ambientes Inteligentes utilizando conceitos da Computação Ubíqua". Tal projeto é financiado pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e possui o período de vigência de 2009 à 2011. O foco deste projeto é o desenvolvimento de soluções computacionais utilizando conceitos referentes ao paradigma da Computação Ubíqua. O objetivo do desenvolvimento de tais soluções é construir sistemas que venham agregar qualidade aos serviços fornecidos, beneficiando e melhorando o modo como o ser humano realiza determinadas tarefas do cotidiano.

Esta dissertação buscou apoiar o projeto supracitado no que diz respeito à estrutura computacional necessária para o desenvolvimento dessas soluções. Vários são os projetos de aplicações embarcadas que podem ser desenvolvidos por meio da utilização da arquitetura parametrizável proposta neste trabalho. Dentre eles, sistemas de controle de acesso, monitoramento de processos industriais, sistemas críticos, acompanhamento dos sinais vitais de um paciente, exploração de ambientes inóspitos, como também sistemas de vigilância patrimonial, como o apresentado no estudo de caso deste trabalho.

Para alcançar este objetivo, primeiramente, foi necessário entender o funcionamento e elencar as principais características relacionadas ao paradigma da Computação Reconfigurável. Ao observar que os FPGA são dispositivos com capacidade de reprogramação por parte dos usuários desenvolvedores de aplicações, verificou-se a importância de uma metodologia adequada para o desenvolvimento de projetos com base nestas plataformas. Visto que as etapas da metodologia Hardware/Software *Codesign* contemplam as principais atividades inerentes aos projetos de sistemas embarcados complexos, esta foi estudada. Objetivando comprovar o potencial e a eficácia advinda da utilização das plata-

formas reconfiguráveis guiada por uma metodologia de desenvolvimento consistente, foi desenvolvido o projeto do sistema de uma Central de Alarme Reconfigurável como estudo de caso.

Durante esta pesquisa algumas dificuldades foram evidenciadas, dentre estas pode-se mencionar que o fato de desenvolver projetos para sistemas embarcados requer cuidados que em sistemas de software convencionais não precisam ser tomados, como, por exemplo, entender a nível de hardware como se dá o processo de comunicação entre os dispositivos. Além disso, observou-se que, para este tipo de sistema, é essencial que os projetos de hardware e software sejam desenvolvidos concomitantemente, de forma sistemática e harmônica, pois o funcionamento correto do mesmo depende de ambos. Por fim, uma outra dificuldade vivenciada foi que sempre deve-se realizar testes isolados em cada dispositivo ou equipamento de hardware utilizado, uma vez que estes podem conter algumas falhas de produção em sua estrutura, que acabam comprometendo ou contribuindo diretamente para o não funcionamento correto de um sistema, causando atrasos e aumentando os custos de projetos.

6.1 Contribuições

A principal contribuição desta dissertação é a disponibilização de uma arquitetura parametrizável que pode ser adaptada para o desenvolvimento de aplicações em vários domínios, como, por exemplo, algoritmos, processamento multimídia, reconhecimento de padrões e sistemas em tempo-real. Tal adaptação é realizada por meio da redefinição da estrutura de hardware em relação aos seus componentes, bem como com a integração de outros dispositivos periféricos externos compatíveis e que possam ser integrados à plataforma reconfigurável adotada nesta pesquisa.

As demais contribuições alcançadas com este trabalho são:

- Adaptação da Metodologia Hardware/Software *Codesign* por meio da incorporação de artefatos importantes na compreensão e na visualização dos aspectos estáticos e dinâmicos dos sistemas embarcados;
- Disponibilização de um modelo formal que reflete a estrutura computacional atrelada a arquitetura parametrizável proposta nesta pesquisa. Este modelo trata-se de um modelo executável que irá permitir, por meio de simulações, a análise do comportamento dos sistemas desenvolvidos. Realizada tal verificação, é possivel

detectar erros advindos da etapa de especificação da metodologia adotada, sendo, portanto, as ambiguidades dos projetos, em geral, elimidadas, em virtude da precisão do modelo.

6.2 Trabalhos Futuros

Como pôde ser visto nesta dissertação, as plataformas reconfiguráveis possuem potencial o bastante para serem aplicadas nos mais diversos domínios, sem comprometer o rendimento do sistema. Além disso, estas garantem a flexibilidade necessária para alterações em seus projetos. No entanto, para possibilitar a criação dos chamados Ambientes Inteligentes, objetivo do projeto de pesquisa associado, torna-se necessário dotá-las de mecanismos e técnicas de Inteligência Artificial, tais como Redes Neurais, Algoritmos Genéticos ou Agentes Inteligentes, questões estas não discutidas nesta pesquisa.

Esta obrigatoriedade é fruto dos conceitos que originam o paradigma da Computação Ubíqua, cujas soluções envolvem aspectos de mobilidade, autonomia, sensibilidade ao contexto e capacidade de auto-adaptação dos sistemas. Tudo isto, de forma transparente e imperceptível para os usuários de tais ambientes, e, sem a necessidade de intervenção humana (MENDES NETO; RIBEIRO NETO, 2010).

Portanto, nesta dissertação foram identificadas algumas questões que necessitam ainda serem discutidas, bem como novos desafios a serem explorados para possibilitar a criação dos, assim chamados, Ambientes Inteligentes. A seguir são listados, alguns direcionamentos para trabalhos futuros decorrentes desta pesquisa:

- Geração de outros artefatos a partir do modelo formal construído em HCPN, tais como diagrama de sequência de mensagens e de *logs* automáticos que demonstrem todo o comportamento do sistema a partir da simulação;
- Como melhor fazer uso da arquitetura parametrizável proposta nesta pesquisa no desenvolvimento de suas aplicações;
- Como permitir a integração e a comunicação eficiente da plataforma reconfigurável em meio a heterogeneidade de dispositivos e equipamentos que podem ser a ela integrados;
- Além, de como embutir capacidade de inferência nas plataformas reconfiguráveis, uma vez que estas, assim como todo sistema embarcado, possuem certas limitações

em relação as capacidades de armazenamento e processamento, sem que atrasos advindos da execução destes algoritmos possam comprometer o rendimento e atender as restrições de tempo e os objetivos de suas aplicações.

Referências Bibliográficas

AHMED, I.; ALAM, S.; RAHMAN, M. A. U.; ISLAM, N. Implementation of graph algorithms in reconfigurable hardware (fpgas) to speeding up the execution. *Fourth International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, p. 880–885, nov. 2009.

ALSOLAIM, A. M. *Dynamically reconfigurable architecture for third generation mobile systems*. Tese (Doutorado) — Ohio University, ago. 2002.

ALTERA. *Nios Hardware Development Tutorial*. jan. 2004a. Disponível em: <http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf>. Acesso em: Ago. 2010. Document Version: 1.2.

ALTERA. *Cyclone vs. Spartan-3 Performance Analysis*. [S.l.], abr. 2004b. White Paper, Document Version: 1.0.

ALTERA. *Nios II Software Developer's Handbook*. nov. 2006. Disponível em: <http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf>. Acesso em: Set. 2010. Document Version: 6.1.0.

ALTERA. *Cyclone II Device Handbook*. fev. 2008. Disponível em: <http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf>. Acesso em: Ago. 2010. Document Version: 3.2.

ALTERA. *Nios II System Architect Design Tutorial*. jun. 2009a. Disponível em: <http://www.altera.com/literature/tt/tt_nios2_system_architect.pdf>. Acesso em: Ago. 2010. Document Version: 1.0.

ALTERA. *Configuration Handbook*. dez. 2009b. Disponível em: <http://www.altera.com/literature/hb/cfg/config_handbook.pdf>. Acesso em: Set. 2010.

ALTERA. *DE2 Development and Education Board - User Manual*. 2010a. Disponível em: <ftp://ftp.altera.com/up/pub/Webdocs/DE2_introduction.pdf>. Acesso em: Jun. 2010. Document Version: 1.41.

ALTERA. *Annual Report 2009*. [S.l.], maio 2010b.

ALTERA. *Quartus II Handbook Version 10.0.1*. jul. 2010c. Disponível em: <http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf>. Acesso em: Ago. 2010. Document Version: 10.0.1.

ALTERA. *Nios II Flash Programmer User Guide*. fev. 2010d. Disponível em: <http://www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf>. Acesso em: Out. 2010. Document Version: 2.1.

ALTERA. *Introduction to the Quartus II Software*. 2010e. Disponível em: <http://www.altera.com/literature/manual/intro_to_quartus2.pdf>. Acesso em: Set. 2010. Document Version: 10.0.

ANDERSON, H.; KHOO, S. Improving responsiveness of hard real-time embedded systems. In: *Third IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*. [S.l.: s.n.], 2009. p. 13. ISBN 978-0-7695-3757-3.

ANDROID. *Android Developer*. 2010. Disponível em: <http://developer.android.com/index.html>. Acesso em: Out. 2010.

ANSI. *ISO/IEC 9899:1999 Programming languages – C*. 1999. Disponível em: <http://webstore.ansi.org/RecordDetail.aspx?sku=ISO%2FIEC+9899%3A1999>. Acesso em: Out. 2010.

AOUDNI, Y.; GOGNIAT, G.; ABID, M.; PHILIPPE, J.-L. Custom instruction integration method within reconfigurable soc and fpga devices. *International Conference on Microelectronics*, p. 131–134, dez. 2006.

BARIAMIS, D. G.; IAKOVIDIS, D. K.; MAROULIS, D. E.; KARKANIS, S. A. An fpga-based architecture for real time image feature extraction. In: *International Conference on Pattern Recognition (ICPR)*. [S.l.: s.n.], 2004. v. 1, p. 801–804.

BARR, M. Programmable logic: What's it to ya? *Embedded Systems Programming*, p. 75–84, jun. 1999.

BECKER, T. et al. Towards benchmarking energy efficiency of reconfigurable architectures. *18th International Conference on Field Programmable Logic and Applications (FPL'08)*, p. 691–694, set. 2008.

BEER, D. *ATARI on an FPGA*. [S.l.], maio 2007.

BEN ATITALLAH, A.; KADIONIK, P.; MASMOUDI, N.; LEVI, H. Fpga implementation of a hw/sw platform for multimedia embedded systems. *Journal Design Automation for Embedded Systems*, v. 12, n. 4, p. 293–311, dez. 2008.

BERTHELOT, F.; NOUVEL, F.; HOUZET, D. A flexible system level design methodology targeting run-time reconfigurable fpgas. *EURASIP Journal on Embedded Systems*, 2008.

BHARGAV, S.; CHEN, L.; MAJUMDAR, A.; RAMUDIT, S. *FPGA-based 128-bit AES decryption*. [S.l.], maio 2008.

BONDE, I. *Tutoriais do Mercado Brasileiro de Wireless M2M I*. [S.l.], jul. 2006.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário*. [S.l.]: Elsevier, 2005. ISBN 85-352-1784-3.

BORGES NETO, J. B. *PROST: Um Protocolo de Roteamento Sensível ao Tempo para Redes de Sensores Sem Fio*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará (UFC), ago. 2009.

BROWN, S.; ROSE, J. Architecture of fpgas and cplds: A tutorial. *IEEE Design and Test of Computers*, v. 13, n. 2, p. 42–57, 1996.

BUENO, M. A. F. *Análise e implementação de suporte a SMP (multiprocessamento simétrico) para o sistema operacional eCos com aplicação em robótica móvel*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP), abr. 2007.

CAMPOS, A. A. N. *Algoritmo de Criptografia AES em Hardware, Utilizando Dispositivo de Lógica Programável (FPGA) e Linguagem de Descrição de Hardware (VHDL)*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Itajubá (UNIFEI), jun. 2008.

CARVALHO, M. G. *Implementação Hardware/Software da estimação de movimento segundo o padrão H.264*. Dissertação (Mestrado) — Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte (UFRN), set. 2007.

CASTRO, E. O. *Multiprocessador em Eletrônica Reconfigurável para Aplicações Robóticas*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica da Universidade Estadual de Campinas (UNICAMP), dez. 2007.

CHATHA, K. S.; R., V. Hardware-software codesign for dynamically reconfigurable architectures. In: *Field Programmable Logic and Applications*. [S.l.]: Springer Berlin / Heidelberg, 2004, (Lecture Notes in Computer Science, v. 1673). p. 175–185. 10.1007/978-3-540-48302-1_18.

CHEN, D.; CONG, J.; ERCEGOVAC, M.; HUANG, Z. Performance-driven mapping for cpld architectures. *ACM/SIGDA 9th Int. Symp. Field-Program Gate Arrays*, v. 22, p. 39–47, 2003.

CHOI, S.; SCROFANO, R.; PRASANNA, V. K.; JANG, J.-W. Energy efficient signal processing using fpgas. *ACM Field Programmable Gate Array*, fev. 2003.

CHOWDHURY, S. R. *CHOWDHURY, S. R., FPGA-Based Clinical Diagnostic System using Pipelined Architectures in the Nios II Processor*. [S.l.], 2007.

COMPTON, K. *Architecture Generation of Customized Reconfigurable Hardware*. Tese (Doutorado) — Nortwestern University, ECE Department, 2003.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos: conceitos e projeto*. 4. ed. Porto Alegre, RS: Bookman, 2007.

DE-MICHELI, G.; ERNST, R.; WOLF, W. H. *Readings in hardware/software co-design*. [S.l.]: Morgan Kaufmann, 2002. ISBN 1-55860-702-1.

DE MICHELL, G.; GUPTA, R. K. Hardware/software co-design. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1997. p. 349–365. ISSN 0018-9219.

DE PAULO, V.; ABABEI, C. 3d network-on-chip architectures using homogeneous meshes and heterogeneous floorplans. *International Journal of Reconfigurable Computing*, Hindawi Publishing Corporation, v. 2010, 2010. Article ID 603059.

DRIMER, S. *Volatile FPGA design security – A survey*. abr. 2008. Version 0.96. Disponível em: <http://www.cl.cam.ac.uk/šd410/papers/fpga_security.pdf>. Acesso em: Set. 2010.

ETSI. *GLOBAL SYSTEM FOR MOBILE COMMUNICATIONS: Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment (ME)*. 8. ed. [S.l.], dez. 1999. ETS 300 916 (GSM 07.07 version 5.9.1 Release 1996).

ETSI. *ETSI: European Telecommunications Standards Institute*. 2010. Disponível em: <http://www.etsi.org>. Acesso em: Out. 2010.

FARIAS, T. M. T. *Sistema Embarcado para um Monitor Holter que Utiliza o Modelo PPM na Compressão de Sinais ECG*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba (UFPB), mar. 2010.

FBSP. *IV Anuário do Fórum Brasileiro de Segurança Pública*. 2010. Disponível em: <http://www2.forumseguranca.org.br/node/24104>. Acesso em: Jan. 2011.

FERLIN, E. P. *Arquitetura Paralela Reconfigurável Baseada em Fluxo de Dados Implementada em FPGA*. Tese (Doutorado) — Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, nov. 2008.

GARTNER. *Market Share: ASIC and FPGA/PLD Application Market and Vendor Analysis, 2008*. [S.l.], maio 2009.

GNU. *GCC, the GNU Compiler Collection*. 2010. Disponível em: <http://gcc.gnu.org/>. Acesso em: Out. 2010.

GONZALEZ, I. et al. Classification of application development for fpga-based systems. In: *National Aerospace & Electronics Conference (NAECON)*. [S.l.: s.n.], 2008. p. 203–208.

GUHA, R.; AL-DABASS, D. Performance prediction for parallel computations of streaming applications on fpga platform. In: *12th International Conference on Computer Modeling and Simulation (UKSIM2010)*. [S.l.: s.n.], 2010. p. 579–585.

HAMBLEN, J. O.; FURMAN, M. D. *Rapid prototyping of digital systems: a tutorial approach*. 2. ed. [S.l.]: Springer, 2001.

HANSEN, M. *Implementing Real-Time Video Deblocking in FPGA Hardware*. Dissertação (Mestrado) — University of Waterloo, Department of Electrical and Computer Engineering, maio 2007.

HARTENSTEIN, R. A decade of reconfigurable computing: a visionary retrospective. In: *Conference on Design, Automation and Test in Europe*. [S.l.]: IEEE Press, 2001. p. 642–649.

HAUCK, S.; DEHON, A. *Reconfigurable computing: the theory and practice of FPGA-based computation*. [S.l.]: Morgan Kaufmann, 2008. ISBN 978-0-12-370522-8.

IQBAL-CH, M. M.; JAVED, M. Y.; IQBAL, M. A. Reconfigurable computing systems related hardware and software perspectives. *International Journal on Computer Science and Engineering*, v. 2, n. 5, p. 1580–1587, 2010.

IQBAL, M. A.; KHAN, S. A.; AWAN, U. S. Reconfigurable computing systems related hardware and software perspectives. *International Journal of Intelligent Information Technology Application*, v. 2, n. 5, p. 209–217, 2009.

ISLAM, S. et al. Fpga realization of fuzzy temperature controller for industrial application. *WSEAS Transactions on Systems and Control*, v. 2, n. 7, p. 484–490, dez. 2007.

ITUT-T. Itu-t recommendation v.250: Serial asynchronous automatic dialling and control. In: *Series V: Data Communication Over the Telephone Network*. [S.l.]: ITU-T, 2003.

JENSEN, K.; KRISTENSEN, L. M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. [S.l.]: Springer, 2009. ISBN 978-3-642-00283-0.

JERRAYA, A. A.; WOLF, W. Hardware/software interface codesign for embedded systems. *Computer*, v. 38, n. 2, p. 63–69, fev. 2005.

JIN, Z. *A remote controlled embedded system implemented in FPGA*. Dissertação (Mestrado) — Chalmers University of Technology, Department of Computer Science and Engineering, nov. 2009.

KANG, C.-W.; IRANLI, A.; PEDRAM, M. A synthesis approach for coarse-grained, antifuse-based fpgas. *IEEE Trans. on CAD of Integrated Circuits and Systems*, v. 26, n. 9, p. 1564–1575, set. 2007.

KEOGH, J. *J2ME: The Complete Reference*. [S.l.]: McGraw-Hill/Osborne, 2003. ISBN ISBN 0-07-222710-9.

KHARAT, M. U.; RAHANGADALE, V. S. Analysis of resource utilization in fpga implementation of an embedded system using soft core processor. *The Pacific Journal of Science and Technology*, v. 10, n. 1, p. 272–279, maio 2009.

KILTS, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. [S.l.]: John Wiley & Sons, 2007. ISBN 978-0-470-05437-6.

KUON, I.; TESSIER, R.; ROSE, J. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, v. 2, n. 2, p. 135–253, 2008.

LI, L.; ZHANG, Y.; GE, C. *Fingerprint Identification System Based on the Nios II Processor*. [S.l.], 2007.

LI, Q.; YAO, C. *Real-Time Concepts for Embedded Systems*. San Francisco: CMP Books, 2003. ISBN 1578201241.

LU, Y.; WU, H.; WU, Z. Robust speech recognition using improved vector taylor series algorithm for embedded systems. *IEEE Transactions on Consumer Electronics*, v. 56, n. 2, p. 764, maio 2010. ISSN 0098-3063.

LU, Z.; ZHANG, X.; SUN, C. An embedded system with uclinux based on fpga. *Pacific-Asia Workshop on Computational Intelligence and Industrial Application, IEEE*, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, p. 691–694, 2008.

LYYTINEN, K.; YOO, Y. Issues and challenges in ubiquituos computing. *Communications of the ACM*, v. 45, n. 12, dez. 2002.

MAEMO. *Maemo Development*. Disponível em: <http://maemo.org/development/>. Acesso em: Out. 2010.

MEHTA, N. *Programmable Logic Design: Quick Start Handbook*. [S.l.]: Xilinx Inc., 2006. Disponível em: <http://www.xilinx.com/publications/products/cpld/logic_handbook.pdf>. Acesso em: Set. 2010.

MENDES NETO, F. M.; RIBEIRO NETO, P. F. (Ed.). *Designing Solutions-Based Ubiquitous and Pervasive Computing: New Issues and Trends*. [S.l.]: IGI Global, 2010. Hardcover. ISBN 978-1-61520-843-2.

MESQUITA, L.; BOTURA JUNIOR, G.; ROCHA, P. M. S. Implementação de controladores lógicos difusos usando fpga ou asic. *Revista Ciências Exatas*, v. 12, n. 2, p. 9–17, 2006.

MUCHOW, J. W. *Core J2ME Technology & MIDP*. [S.l.]: Prentice Hall PTR, 2001. ISBN 0-13-066911-3.

MUKILAN, P. *An Fpga Implementation of Alamouti?s Transmit Diversity Technique*. Dissertação (Mestrado) — Universiti Teknologi Malaysia, Faculty of Electrical Engineering, maio 2008.

MURPHY, P.; FRANTZ, P.; DICK, C. An fpga implementation of alamouti's transmit diversity technique. *Wireless Networking Symposium (WNCG)*, out. 2003.

NIEMANN, R. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems*. Boston: Kluwer Academic Publishers, 1998. ISBN 0-7923-8299-4.

NOGUERA, J.; BADIA, R. M. Hw/sw codesign techniques for dynamically reconfigurable architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 10, n. 4, p. 399–415, ago. 2002. ISSN 1063-8210.

OHSAKI, K.; ASAMOTO, N.; TAKAYA, Y. Sippos (single poly pure cmos) eeprom embedded fpga by news ring interconnection and highway path. *IEEE 1994 Custom Integrated Circuits Conference*, p. 189–192, maio 1994.

ORACLE. *Java Plataform ME*. 2010. Disponível em: <http://developer.apple.com/devcenter/ios/index.action>. Acesso em: Out. 2010.

ORACLE. *x86 Assembly Language Reference Manual*. [S.l.], mar. 2010. PartNo: 817–5477–11.

ORANGE. *Machine To Machine - stakes and prospects*. jun. 2006. Disponível em: <http://www.steria.com/documents/File/WP_Machine-to-Machine% _M2M.pdf>. Acesso em: Out. 2010. White Paper.

PHILLIPS, J. et al. Methodology to derive context adaptable architectures for fpgas. *IET Computers & Digital Techniques*, v. 3, n. 1, p. 124–141, 2009. ISSN 1751-8601.

PLAKS, T. P.; SANTAMBROGIO, M. D.; SCIUTO, D. Reconfigurable computing and hardware/software codesign. *EURASIP Journal on Embedded Systems*, 2008.

QU, Y.; TIENSYRA, K.; SOININEN, J.-P.; NURMI, J. Design flow instantiation for run-time reconfigurable systems: A case study. *EURASIP Journal on Embedded Systems*, 2008.

REDDY, E. S. S. et al. Detecting seu-caused routing errors in sram-based fpgas. *18th International Conference on VLSI Design*, p. 736–741, jan. 2005.

RIBEIRO NETO, P. F. *Mecanismos de Qualidade de Serviços para o Gerenciamento de Dados e Transações em Tempo-Real*. Tese (Doutorado) — Programa de Pós-Gradução em Engenharia Elétrica da Universidade Federal de Campina Grande (UFCG), mar. 2006.

RIBEIRO NETO, P. F.; PERKUSICH, M. L. B.; DE ALMEIDA, H. O.; PERKUSICH, A. *A Formal Verification and Validation Approach for Real-Time Databases*. [S.l.]: Idea Group Publishing, 2007. Chapter V.

RODRÍGUEZ, S. C.; ALVAREZ, R. P. M.; CASTAÑO, F. J. G.; CASTIÑEIRA, F. G. Measuring fpga soft processor performance in streaming applications. In: *Conference on Design of Circuits and Integrated Systems*. [S.l.: s.n.], 2009.

SCHAUMONT, P. R. *A Practical Introduction to Hardware/Software Codesign*. 1. ed. [S.l.]: Springer, 2010. Hardcover. ISBN 978-1-4419-5999-7.

SHARP, S. *Spartan-3 vs. Cyclone II Performance Analysis: Cyclone II Performance Drops In Recent Releases*. [S.l.], maio 2005. White Paper n. 226, Document Version: 1.0.

SHIH, D.; CHIANG, H.; LIN, B.; SB., L. An embedded mobile ecg reasoning system for elderly patients. *IEEE Transactions on Information Technology in Biomedicine*, v. 14, n. 3, p. 854–865, maio 2010. ISSN 1089-7771.

SIEMENS. *TC65 AT Command Set*. [S.l.], maio 2005a. Document Version: 00.521.

SIEMENS. *TC65 JAVA User's Guide*. [S.l.], mar. 2005b. Document Version: 01.

SIEMENS. *TC65 Module - TC65 Terminal: Release Notes*. [S.l.], out. 2005c. Document Version: 01.041.

SIEMENS. *TC65 Terminal - Plugado no mercado Wireless de Machine-to-Machine: Datasheet*. 2006. Disponível em: <http://www.siemens.com.br/templates/produto.aspx?channel=7372&channel%_pri_nivel=7372&produto=18170>. Acesso em: Out. 2010.

SIEMENS; NOKIA. *JSR-195: Information Module Profile Specification*. [S.l.], jul. 2003. JCP Specification, version 1.0 (Final Release).

SIEMENS; NOKIA. *JSR-228: Information Module Profile Next Generation Specification*. [S.l.], fev. 2005. JCP Specification, version 2.0 (Proposed Final Draft).

SILVA, L. C. *Java Binding FirmSYS – Uma API Java para o RFID Starter kit – Estudo de Caso: SiControlAT*. Dissertação (Mestrado) — Universidade do Estado do Rio Grande do Norte (UERN), fev. 2009.

SPEERS, T. et al. 0.25μ flash memory based fpga for space applications. *Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, set. 1999.

STEPHENSON, J. *Design Guidelines for Optimal Results in FPGAs*. mar. 2005. Disponível em: <http://www.altera.com/literature/cp/fpgas-optimal-results-396.pdf>. Acesso em: Ago. 2010. CF-031405-1.0.

STERPONE, L.; BATTEZZATI, N.; FERLET-CAVROIS, V. Analysis of set propagation in flash-based fpgas by means of electrical pulse injection. *IEEE Transactions on Nuclear Science*, v. 57, n. 4, p. 1820–1826, ago. 2010.

SUN. *JSR-37: Mobile Information Device Profile Specification*. [S.l.], dez. 2000. Document Version: 1.0a.

SUN. *JSR-036: Connected Device Configuration Specification*. [S.l.], dez. 2005. Document Version: 1.0b (Maintenance Release).

SUN. *JSR-118: Mobile Information Device Profile Specification*. [S.l.], maio 2006. Document Version: 2.1.

SUN. *JSR-139: Connected Limited Device Configuration Specification*. [S.l.], nov. 2007. Document Version: 1.1.1 (Maintenance Release).

SVENSSON, H. *Reconfigurable Architectures for Embedded Systems*. Tese (Doutorado) — Lund University, out. 2008.

SYMBIAN. *The Symbian Plataform*. 2010. Disponível em: <http://www.symbian.org/symbian-platform>. Acesso em: Out. 2010.

TSAI, M.-D.; WANG, H. A 0.3-25-ghz ultra-wideband mixer using commercial 0.18-μm cmos technology. *IEEE Microw. Wireless Compon. Lett.*, v. 14, n. 11, p. 522–524, nov. 2004.

TSCM. *Low-K Dielectrics: Key to Success at 90 Nanometers and Beyond*. abr. 2004. Disponível em: <http://www.tsmc.com/download/english/a05_literature/enliterature/html-newsletter/April04/AdvancedTechnology/index.html>. Acesso em: Set. 2010.

VIOLANTE, M. et al. Analyzing seu effects in sram-based fpgas. *IEEE On-Line Testing Symposium (IOLTS)*, p. 119–123, jul. 2003.

VISAKHASART, S.; CHITSOBHUK, O. Multi-pipeline architecture for face recognition on fpga. *International Conference on Digital Image Processing (ICDIP)*, p. 152–156, mar. 2009.

W3C. *HTML ASCII Reference*. 2010. Disponível em: <http://www.w3schools.com/tags/ref_ascii.asp>. Acesso em: Jan. 2011. W3schools.com.

WANG, J. J. et al. Clock buffer circuit soft errors in antifuse-based fpgas. *IEEE Trans. Nucl. Sci.*, v. 47, p. 2675–2681, dez. 2000.

WEISER, M. The computer for the 21st century. *Scientific American*, set. 1991.

WIANGTONG, T.; CHEUNG, P. Y. K.; LUK, W. Hardware/software codesign: A systematic approach targeting data-intensive applications. *Signal Processing Magazine, IEEE*, v. 22, n. 3, p. 14–22, maio 2005. ISSN 1053-5888.

WINTERGREEN. *Programmable Logic ICs Market Shares and Forecasts, Worldwide, 2010 to 2016*. [S.l.], set. 2010.

WOLF, W. A decade of hardware/software codesign. *Computer*, IEEE Computer Society, v. 36, n. 4, p. 38–43, abr. 2003. ISSN 0018-9162.

XILINX. *Xilinx Corporate Overview*. [S.l.], 2010b.

ZLOKOLICA, V. et al. Video processing real-time algorithm design verification on embedded system for hdtv. In: *IEEE Eastern European Conference on the Engineering of Computer Based Systems*. [S.l.: s.n.], 2009. p. 150. ISBN 978-1-4244-4677-3.