Implementação do barramento on-chip AMBA baseada em computação reconfigurável

Daniel Cruz de Queiroz

Implementação do barramento on-chip AMBA baseada em computação reconfigurável

Daniel Cruz de Queiroz

Orientador: *Prof. Dr. Eduardo Marques*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Janeiro/2005

Dedicatória

Dedico este trabalho a minha família, amigos, professores e todas as outras pessoas que, de uma forma ou de outra, me ajudaram a crescer.

Agradecimentos

Aos meus pais Cláudio e Ivone, ao meu irmão Paulo e demais parentes, por todo o apoio em todos os sentidos;

Ao meu orientador Eduardo Marques, que está sempre de bom humor e não mede esforços em ajudar no que for preciso;

Aos colegas do LCR, em especial ao José Arnaldo, por compartilharem comigo momentos de dedicação e descontração;

Aos demais amigos e colegas, sejam do curso de mestrado ou não, com os quais tive a oportunidade de aprender muito;

E, por fim, ao CNPq pelo apoio financeiro.

"Somos o que fazemos, mas somos, principalmente,
o que fazemos para mudar o que somos"
(Eduardo Galeano)

Resumo

QUEIROZ, D. C. **Implementação do barramento on-chip AMBA baseada em Computação Reconfigurável**. 2005. 113 p. Dissertação de Mestrado - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

A computação reconfigurável está se fortalecendo cada vez mais devido ao grande avanço dos dispositivos reprogramáveis e ferramentas de projeto de hardware utilizadas atualmente. Isso possibilita que o desenvolvimento de hardware torne-se bem menos trabalhoso e complicado, facilitando assim a vida do desenvolvedor. A tecnologia utilizada atualmente em projetos de computação reconfigurável é denominada FPGA (*Field Programmable Gate Array*), que une algumas características tanto de software (flexibilidade), como de hardware (desempenho). Isso fornece um ambiente bastante propício para desenvolvimento de aplicações que precisam de um bom desempenho, sem que estas devam possuir uma configuração definitiva. O objetivo deste trabalho foi implementar um barramento eficiente para possibilitar a comunicação entre diferentes *CORES* de um robô reconfigurável, que podem estar dispersos em diferentes dispositivos FPGAs. Tal barramento seguirá o padrão AMBA (*Advanced Microcontroller Bus Architecture*), pertencente à ARM. Todo o desenvolvimento do *core* completo do AMBA foi realizado utilizando-se a linguagem VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) e ferramentas EDAs (*Electronic Design Automation*) apropriadas. É importante notar que, embora o barramento tenha sido projetado para ser utilizado em um robô, o mesmo pode ser usado em qualquer sistema on-chip.

Palavras-chave: AMBA, barramento on-chip, computação reconfigurável, FPGA.

Abstract

QUEIROZ, D. C. **Implementation of on-chip AMBA bus based on Reconfigurable Computing**. 2005. 113 p. Masters Thesis - Institute of Mathematical Sciences and Computation, University of São Paulo, São Carlos.

The reconfigurable computing is each time more fortified, what leads to a great advance of reprogrammable devices and hardware design tools. This has become hardware development less laborious and complicated, thus, facilitating the life of the designer. The technology currently used in projects of reconfigurable computing is called FPGA (Field Programmable Gate Array), which combines some characteristics of software (flexibility) and hardware (performance). This technology provides a propitious environment to the development of applications that need a good performance. Those that don't need a definitive configuration. The purpose of this work was to implement an efficient bus to make possible the communication among different modules of a reconfigurable robot. This bus is based on a bus standard called AMBA (Advanced Microcontroller Bus Architecture), which belongs to ARM. All the development of full AMBA core was carried through using VHDL (Very High Speed Integrated Circuit the Hardware Description Language) language and appropriated EDA (Electronic Design Automation) tools. It is important to notice that, even so the bus have been projected to be used in a robot, it could be used in any system on-chip.

Keywords: AMBA, on-chip bus, reconfigurable computing, FPGA.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ABEL	*Advanced Boolean Expression Language*
ADC	*Analog to Digital Converter*
AHB	*Advanced High-performance Bus*
AHDL	*Altera Hardware Description Language*
AMBA	*Advanced Microcontroller Bus Architecture*
APB	*Advanced Peripheral Bus*
ARMOSH	Aprendizado em Robôs Móveis via Software e Hardware
ASB	*Advanced System Bus*
ASIC	*Application Specific Integrated Circuit*
AXI	*Advanced eXtensible Interface*
CI	Circuito Integrado
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
DCI	*Digitally Controlled Impedance*
DCM	*Digital Clock Manager*
DCR	*Device Control Register*
DDR	*Double Data Rate*
DDR2	*Double Data Rate 2*
DMA	*Direct Memory Access*

DSP	*Digital Signal Processing*
EDA	*Electronic Design Automation*
ESA	*European Space Agency*
E/S	Entrada/Saída
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
FIA	Federação Internacional de Automobilismo
FPGA	*Field Programmable Gate Array*
GPIO	*General Purpose I/O*
HDL	*Hardware Description Language*
HP	*Hewlett-Packard*
I2C	*Inter-Integrated Circuit*
IBM	*International Business Machines*
ICMC	Instituto de Ciências Matemáticas e de Computação
IDE	*Integrated Development Environment*
IP	*Intellectual Property*
ISE	*Integrated Software Environment*
I/O	*Input/Output*
JTAG	*Joint Test Action Group*
LABIC	Laboratório de Inteligência Computacional
LCD	*Liquid Crystal Display*
LCR	Laboratório de Computação Reconfigurável
LUT	*Look-Up Table*
MAC	*Media Access Control*
MMU	*Memory Management Unit*
NASA	*National Aeronautics and Space Administration*
NoC	*Network-on-Chip*
OCP	*Open Core Protocol*
OCP-IP	*Open Core Protocol International Partnership*
OPB	*On-chip Peripheral Bus*

PAL	*Programmable Array Logic*
PCI	*Peripheral Component Interconnect*
PDA	*Personal Digital Assistance*
PLA	*Programmable Logic Array*
PLB	*Processor Local Bus*
PLD	*Programmable Logic Device*
PLL	*Phase-Locked Loop*
PWM	*Pulse Width Modulation*
QDRII	*Quad Data Rate 2*
RISC	*Reduced Instruction Set Computer*
RLDRAM II	*Reduced Latency Dynamic Random Access Memory II*
RTC	*Real Time Clock*
SDR	*Single Data Rate*
SDRAM	*Synchronous Dynamic Random Access Memory*
SoC	*System-on-Chip*
SoPC	*System-on-a-Programmable-Chip*
SPARC	*Scalable Processor Architecture*
SPI	*Serial Peripheral Interface*
SPLD	*Simple Programmable Logic Device*
SRAM	*Static Random Access Memory*
UART	*Universal Asynchronous Receiver/Transmitter*
USP	Universidade de São Paulo
VC	*Virtual Component*
VCM	*Vehicle Control and Monitoring*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
XPP	*Extreme Processing Platform*

Lista de Códigos

Capítulo 1

Introdução

Nos últimos anos várias pesquisas vêm sendo desenvolvidas sobre robôs, especialmente pesquisas relacionadas com robôs móveis e reconfiguráveis. A computação reconfigurável é uma tecnologia bastante interessante, do ponto de vista técnico e comercial, e tem se mostrado muito útil para a construção de robôs, ou sistemas embutidos, cuja arquitetura pode ser modificada por software. Tal característica permite que o robô se modifique, em tempo de execução, para melhor se adequar a uma determinada situação. Neste caso é o hardware que se adapta à aplicação e não o contrário (Gonçalves, 2000).

Para a construção de sistemas reconfiguráveis, como um robô, pode-se utilizar FPGAs (*Field Programmable Gate Arrays*). Tal tecnologia de dispositivos evoluiu bastante nos últimos anos, alcançando uma grande densidade, alto índice de desempenho e menor custo de fabricação. Isso despertou o interesse pela realização do projeto ARMOSH (Romero, 2000), que foi financiado pela FAPESP (processo nº: 2000/02959-3) e realizado no Instituto de Ciências Matemática e de Computação (ICMC) da Universidade de São Paulo (USP). Os dois laboratórios responsáveis pelo projeto foram o LCR (Laboratório de Computação Reconfigurável) e o LABIC (Laboratório de Inteligência Computacional). O projeto ARMOSH inspirou o desenvolvimento de um novo ambiente para projeto e

implementação de um sistema de controle evolucionário embutido para robôs móveis e reconfiguráveis dinamicamente baseado em FPGA (ver figura 1.1). Este trabalho está envolvido no projeto de construção de um robô móvel reconfigurável que será construído a partir de uma parceria entre o LCR e o LABIC, ambos do ICMC-USP.

Para que a realização do projeto mencionado obtenha sucesso é necessário que exista uma estrutura de comunicação entre os diversos módulos (dispositivos/*cores*[1]) do robô. Isso pode ser feito com a utilização de um barramento. Essa comunicação deve ser baseada em algum padrão para contornar o problema da integração de dispositivos contruídos por diferentes desenvolvedores/empresas.

Atualmente, existem algumas arquiteturas de barramentos (on-chip) publicamente disponíveis, tais como WISHBONE (SILICORE, 2001), AMBA (*Advanced Microcontroller Bus Architecture*) (ARM, 1999), Avalon (Altera, 2003a) e CoreConnect (IBM, 1999) dos respectivos fabricantes: Silicore, ARM, Altera e IBM. Optou-se, neste trabalho, pela escolha do barramento AMBA, pois se trata de um padrão confiável e que pode ser licenciado sem a cobrança de taxas, além de ser amplamente utilizado por fabricantes de SoC (System-on-Chip), como: Altera (Altera, 2004a), PACT (PACT, 2004), MIPS (MIPS, 2004), etc. O protocolo AMBA[2] também é bastante usado, para implementação de seus SoC (System-on-Chip), por diversos grupos de pesquisa da área acadêmica. Alguns dos trabalhos podem ser vistos em (Becker e Vorbach, 2003), (Kalte et al., 2002), (Lee e Bergmann, 2003), (Hellmich et al., 2000), (Tsai et al., 2003) e (Moch et al., 2003).

1.1 Objetivos

O objetivo principal deste trabalho é fazer com que os diversos módulos do robô movél reconfigurável sejam interconectados utilizando o barramento AMBA, especificado pela ARM. Mais especificamente, fazer com que as aplicações em FPGA desenvolvidas no LCR/ICMC-USP e dispositivos de terceiros possam ser interligados e se comunicarem de uma maneira padronizada, confiável e com uma velocidade satisfatória. Para isso, faz-se

[1] *Core*: Bloco que possui alguma lógica funcional.

[2] Protocolo AMBA: Trata-se do conjunto de regras do barramento AMBA. Quando usa-se a expressão "barramento AMBA", na verdade está se referindo ao seu conjunto de regras.

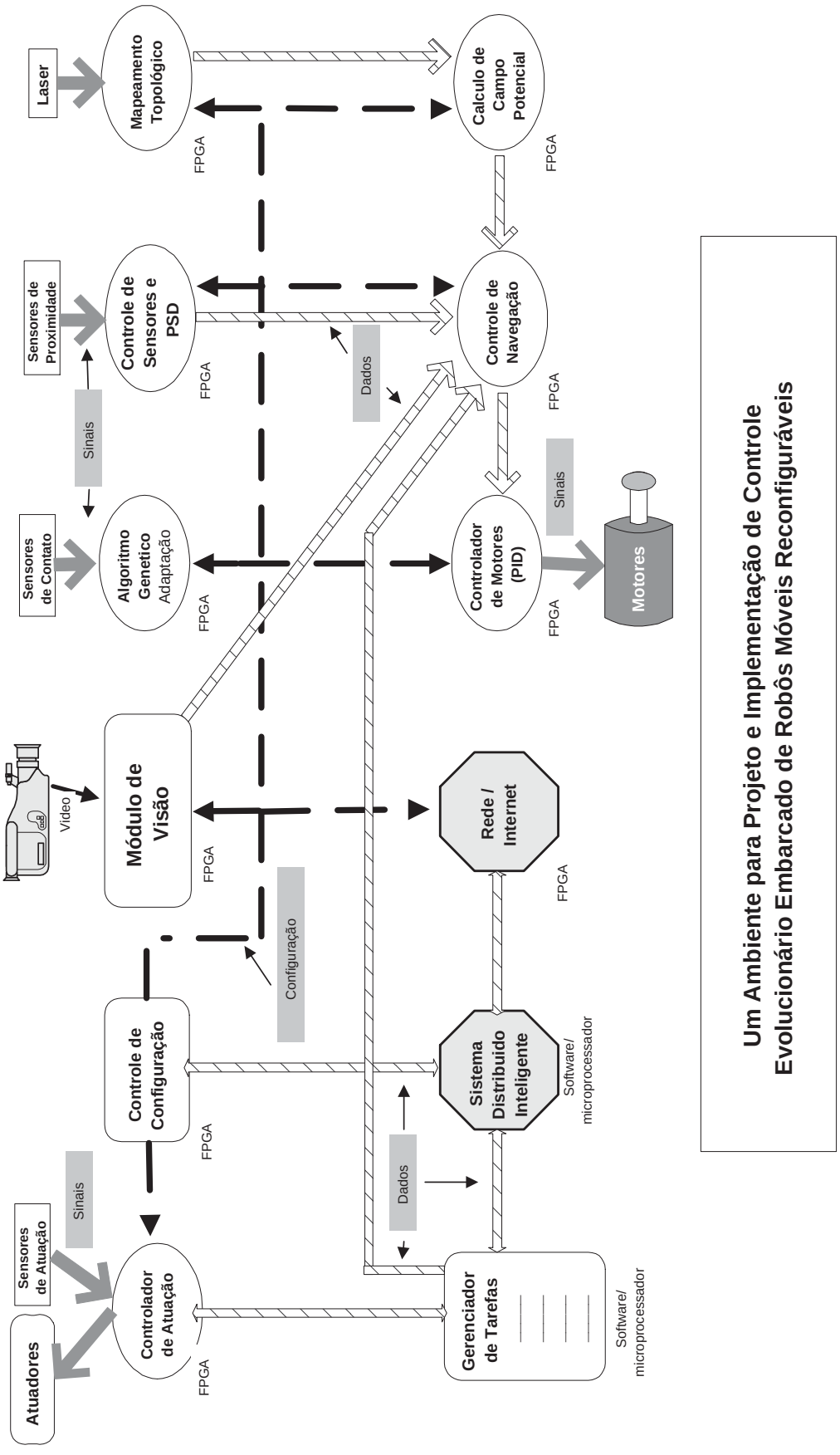

Figura 1.1: Ambiente do Robô Móvel Reconfigurável.

necessário que cada módulo se torne compatível com o protocolo AMBA. Desta maneira, foi preciso implementar o barramento, usando uma linguagem de descrição de *hardware*, de modo que todos os módulos do robô (implementados em FPGA) possam se comunicar através desse protocolo que foi desenvolvido para comunicação on-chip.

Cabe ressaltar que o barramento AMBA já foi implementado no processador LEON2 (Gaisler, 2003) e encontra-se à disposição de todos. Tal implementação é livre e gratuita. Dessa forma, a implementação proposta neste trabalho foi realizada com base nela. Porém, várias mudanças foram feitas com o intuito de cobrir de forma mais abrangente a especificação e de tornar seu uso mais amigável e genérico para a solução do problema apresentado na figura 1.1.

1.2 Organização do trabalho

Este trabalho encontra-se dividido em sete capítulos e dois apêndices, como mostrado abaixo:

Capítulo 1: Fornece uma breve introdução sobre este trabalho e o objetivo do projeto de mestrado.

Capítulo 2: Faz uma pequena introdução sobre sistemas embutidos e apresenta a abordagem de desenvolvimento de SoCs baseados na reutilização de componentes e utilização de padrões.

Capítulo 3: Apresenta alguns tópicos sobre lógica programável e reprogramável, faz um pequena descrição sobre FPGA, mostra o que há de mais novo e moderno na tecnologia FPGA, e por último mostra algumas aplicações em que a computação reconfigurável já atua e onde ela pode atuar.

Capítulo 4: Este capítulo apresenta algumas características e funcionalidades gerais de um barramento e mostra alguns exemplos de barramentos on-chip.

Capítulo 5: Entra em detalhes sobre o barramento AMBA, que foi o barramento on-chip escolhido para ser implementado neste projeto.

Capítulo 6: Apresenta detalhes da implementação, testes e resultados. Além de mostrar detalhes sobre a configuração e utilização do barramento.

Capítulo 7: São apresentadas as conclusões do trabalho e algumas sugestões para possíveis trabalhos futuros.

Apêndice A: Mostra alguns detalhes da placa de desenvolvimento Nios - Edição Stratix.

Apêndice B: Apresenta os resultados de algumas simulações de transferências no barramento AMBA implementado.

Capítulo 2

Sistemas Embutidos

Este capítulo apresenta um breve introdução sobre sistemas embutidos, assim como alguns tópicos relacionados ao desenvolvimento de tais sistemas. Será abordado também o desenvolvimento de sistemas baseados em IP-*cores* e a importância da padronização do desenvolvimento de componentes reutilizáveis.

2.1 Introdução

Algumas décadas atrás, os computadores ocupavam uma grande área física e eram usados (individualmente) por um grande número de usuários. Com a evolução da computação e eletrônica, os computadores se tornaram bem mais rápidos, menores e mais acessíveis aos usuários. Isso possibilitou a diminuição do número de pessoas que utilizam um mesmo computador. Os *mainframes* dominaram as décadas de 50 e 60, depois vieram os minicomputadores na década 70. Nos anos 80 e 90 os microcomputadores invadiram os escritórios e domicílios de milhões de pessoas. Hoje em dia os sistemas embutidos (ou embarcados) já estão dominando o mercado. Isso pode ser visto em uma previsão feita pela empresa HP mostrada na figura 2.1. É importante notar também que com a evolução

dos sistemas embutidos, uma simples pessoa pode fazer uso de até 100 CPUs.

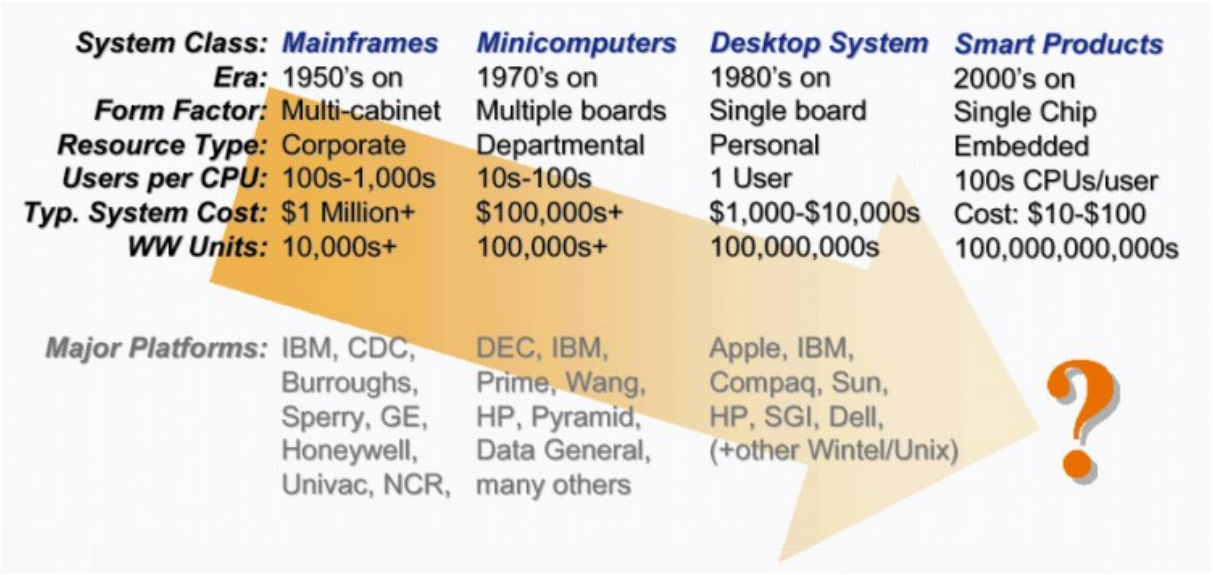

Figura 2.1: Evolução dos sistemas computacionais.

Sistema embutido é um sistema computacional (complexo ou simples), construído dentro de um produto, pedaço de equipamento ou outro sistema computacional (iDLAB, 2004). É um sistema programado para realizar alguma tarefa computacional, que pode ser muito simples ou bastante complexa. Diferentemente dos sistemas computacionais mais genéricos, os sistemas embutidos são implementados para realizar uma tarefa bem definida dentro de um outro sistema maior.

Uma das grandes vantagem de tais sistemas é que, por serem compactos (em grande parte dos casos), podem ser construídos totalmente em um único chip. Ou seja, todo o conjunto de dispositivos necessários ao sistema pode ser colocado em um mesmo CI (Circuito Integrado). A figura 2.2 mostra a diferença entre sistemas embutidos e tradicionais (formados por vários chips em uma mesma placa). Cabe ressaltar, que existem casos em que os sistemas embutidos utilizam vários chips, já que podem estar embutidos em sistemas computacionais muito grandes (carros, robôs, satélites, etc).

É interessante notar que ao se construir um sistema embutido em um único chip, há uma série de vantagens em relação aos sistemas tradicionais (onde os vários dispositivos são interligados com a utilização de uma ou mais placas de circuito impresso). Tais vantagens são: diminuição nos custos (ver figura 2.3), eliminação da latência na comunicação entre

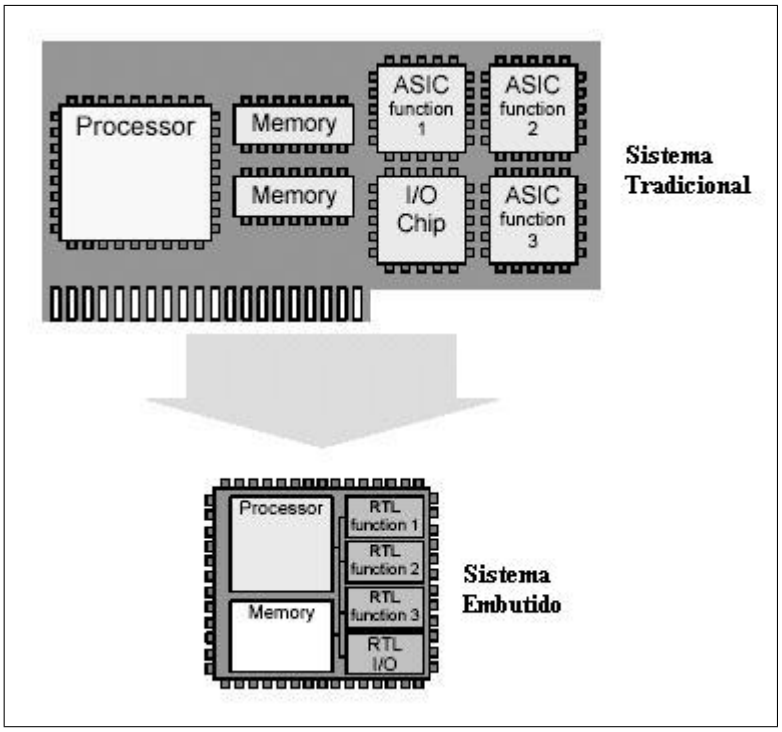

Figura 2.2: Sistema Tradicional x Embutido (Leibson, 2002)

componentes, maior confiabilidade, aumento no desempenho e um menor consumo de energia.

Os sistemas embutidos já estão presentes em um grande número de atividades humanas e fazem parte do dia-a-dia de milhões de pessoas, que muitas vezes nem notam que tais sistemas existem. Alguns exemplos são: máquinas fotográficas digitais, telefones celulares, PDAs, fornos de microondas, máquinas de lavar, video-games, etc. Dessa maneira, a previsão da HP está se tornando uma realidade, onde uma pessoa pode possuir vários processadores.

2.2 Desenvolvimento de Sistemas Embutidos

O desenvolvimento de um sistema embutido deve levar em conta algumas questões como: portabilidade, baixo consumo de energia (sem perda de desempenho), baixa quantidade de memória, segurança, confiabilidade e curto tempo de projeto (Wolf, 2001). Como tais sistemas podem possuir uma grande quantidade de dispositivos em um único chip, torna-se necessário que exista alguma estrutura de comunicação que pode variar de

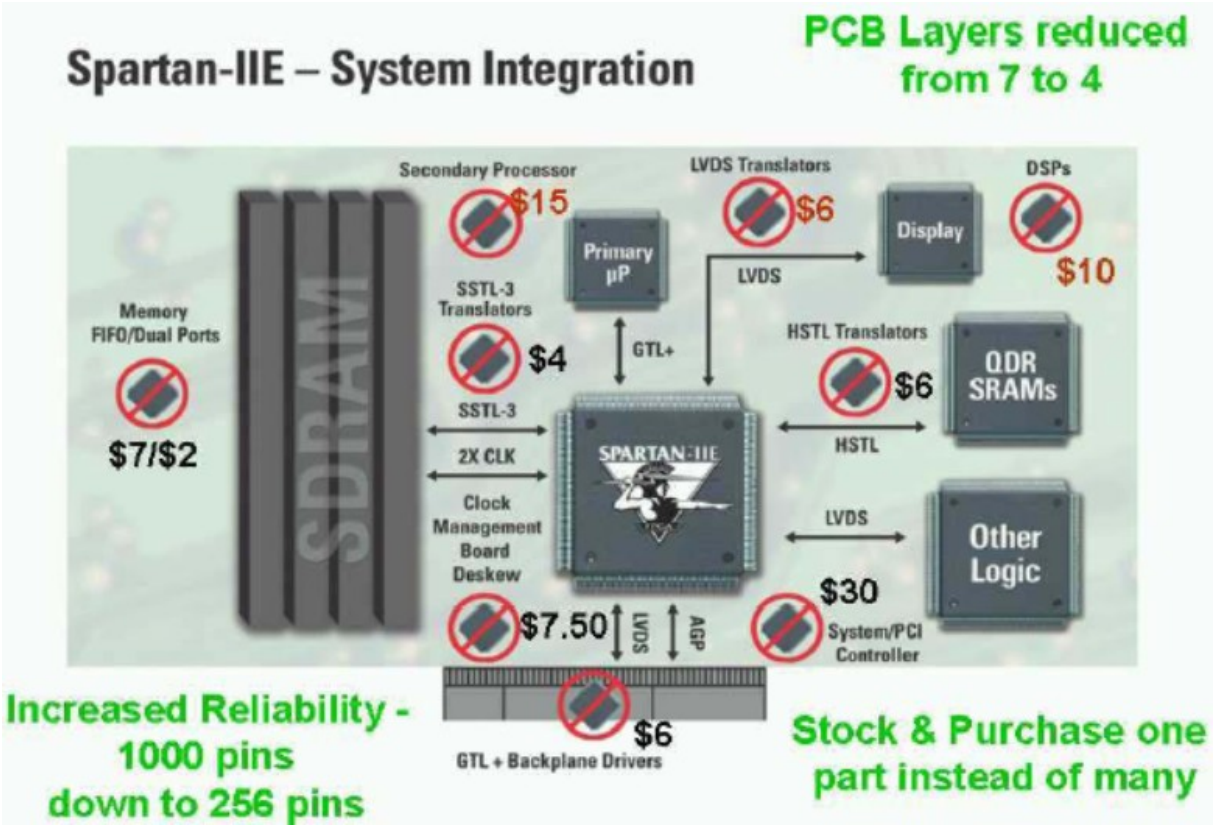

Figura 2.3: Diminuição de custos usando um único chip (Xilinx, 2003)

um barramento a uma rede complexa (NoC - *Network-on-Chip*) (Carro e Wagner, 2003).

O mercado mundial também influencia no desenvolvimento de sistemas embutidos, já que as empresas são pressionadas a desenvolver novos sistemas em períodos de tempo cada vez mais curtos (poucos meses). Atualmente, o ciclo de vida desses produtos também são bem pequenos, o que leva a necessidade de um rápido retorno financeiro. Isso implica que atrasos nos lançamentos dos produtos comprometem seriamente as margens de lucro esperados pelas empresas (ver figura 2.4). Para sistemas embutidos complexos, empresas devem possuir uma equipe multidisciplinar, já que precisam trabalhar com diversas áreas do conhecimento (hardware digital, hardware analógico, software, testes). Esse fator gera uma elevação nos custos do projeto.

Para a implementação dos sistemas, podem ser utilizadas várias tecnologias. Entre elas estão: ASIC (*Application Specific Integrated Circuit*) e FPGA (*Field Programmable Gate Array*). Para aplicações onde existam sérias restrições a requisitos como área, desempenho e potência, deve-se utilizar a tecnologia ASIC. Porém, isso elevará os custos do projeto, já

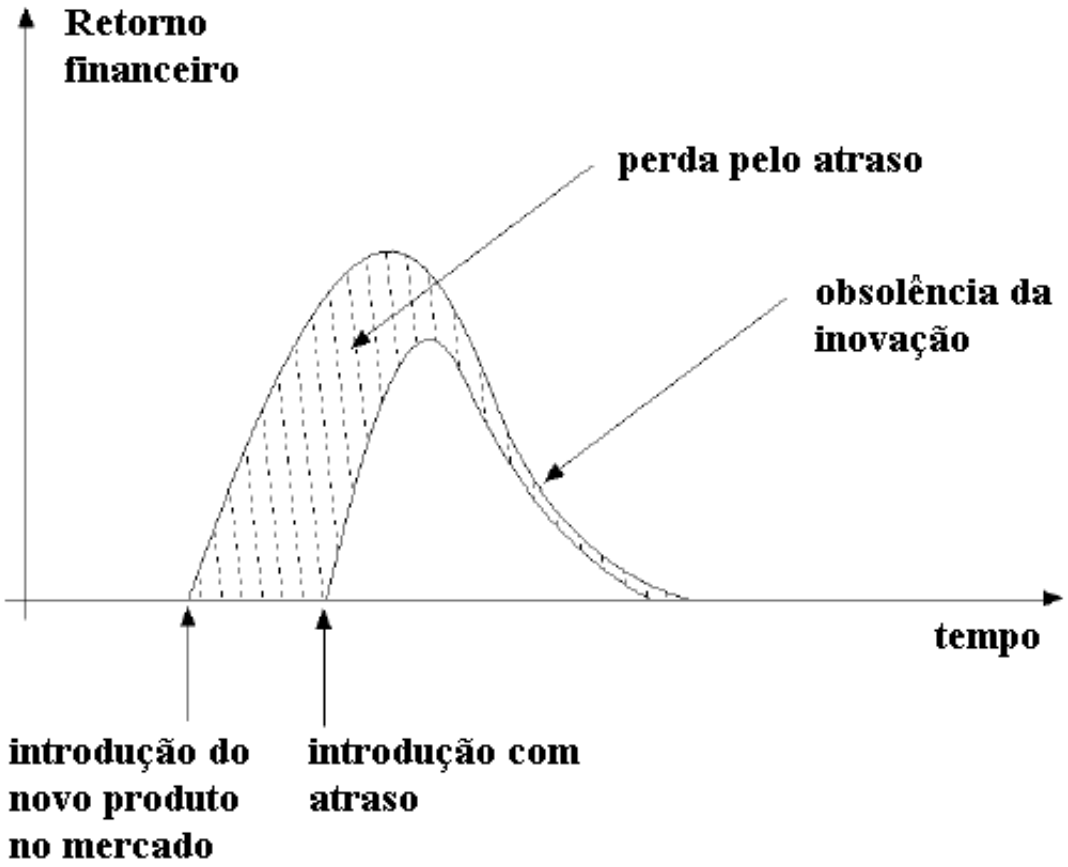

Figura 2.4: Análise do retorno financeiro (Carro e Wagner, 2003).

que tal tecnologia é bem cara e só torna-se viável para produções em larga escala. Já os FPGAs têm um custo bem menor quando se trata de uma produção em pequena escala. Porém, possui um desempenho inferior ao ASIC. A figura 2.5 mostra uma comparação entre sistemas em FPGA e ASIC, no que se diz respeito a desempenho e flexibilidade. A figura 2.6 mostra as características que os projetistas devem levar em consideração na implementação de um sistema. É importante notar que todos os requisitos devem estar encaixados para se conseguir um resultado satisfatório. Partindo-se de um alto nível (a especificação do sistema) deve-se então procurar pela melhor solução através do correto particionamento entre hardware e software (ver figura 2.7). Isso pode ser feito através de estimativas de diferentes particionamentos. A partir daí é possível achar o ponto ideal ou o mais próximo para a correta implementação do sistema embutido. Um outro fator de extrema importância são os testes. Que devem seguir algum tipo de metodologia. Tais testes podem ser realizados com diversas ferramentas EDAs (*Electronic*

Design Automation) disponíveis no mercado. A ferramenta utilizada neste trabalho foi o Quartus II da Altera.

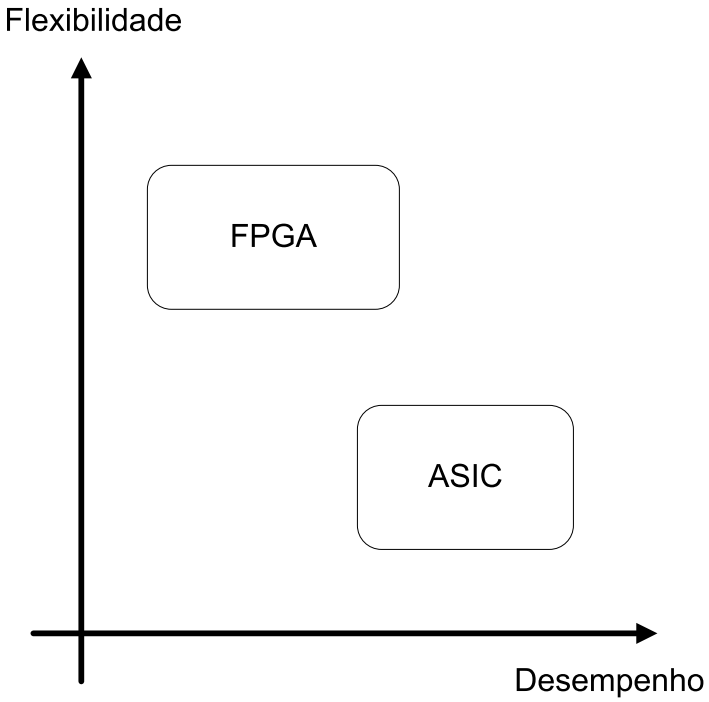

Figura 2.5: Comparação entre desempenho e flexibilidade

2.3 IP-cores

Um *core*, ou componente virtual (VC - *Virtual Component*), nada mais é do que um bloco que realiza alguma tarefa baseada em operações lógicas. Exemplos de *cores* são: processadores, controladores de barramento, DSPs, controladores DMA, etc. Segundo (Palma et al., 2001), os *cores* podem ser classificados em três maneiras distintas: *soft*, *firm* e *hard*. A tabela 2.1 mostra a classificação de acordo com cinco critérios (rigidez, modelagem, flexibilidade, previsibilidade e proteção da propriedade intelectual). Algumas empresas se especializaram em construir *cores* que pudessem ser usados no desenvolvimento de SoCs. Surgiram assim os IP *cores* (IP - *Intellectual Property*), onde as empresas mantém seus direitos de propriedade intelectual, comercializando somente a licença de

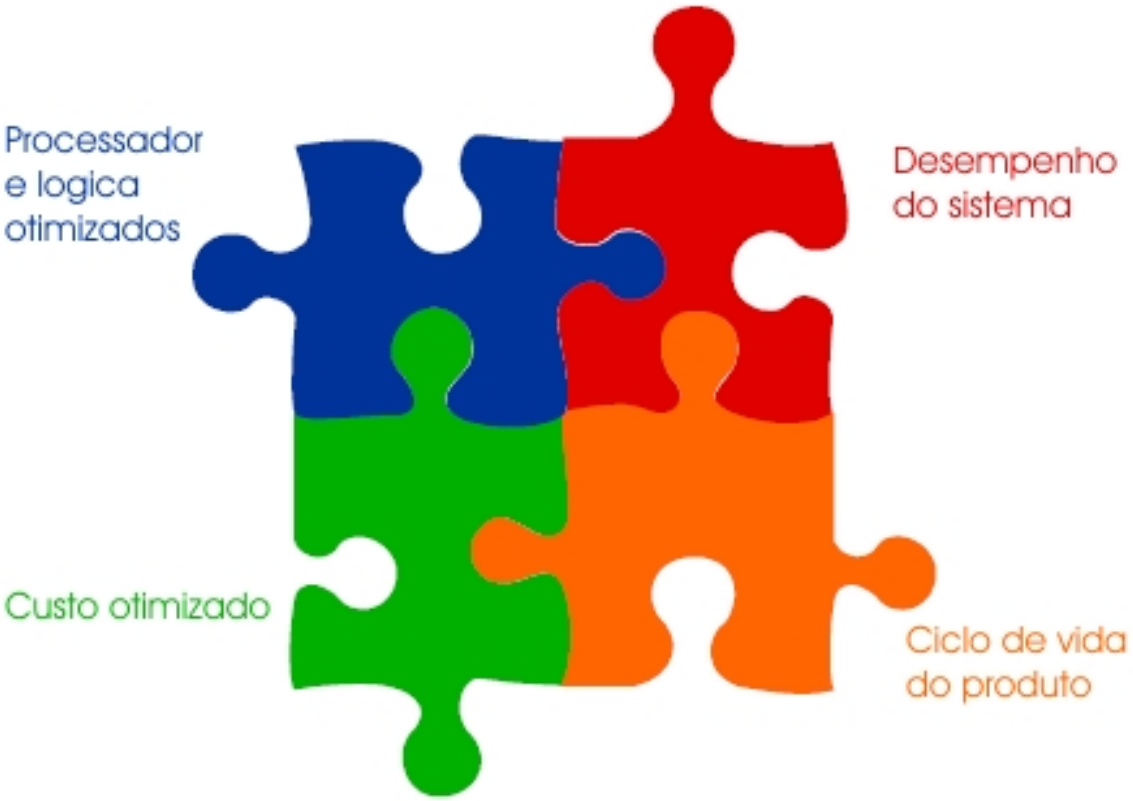

Figura 2.6: Requisitos de um sistema embutido

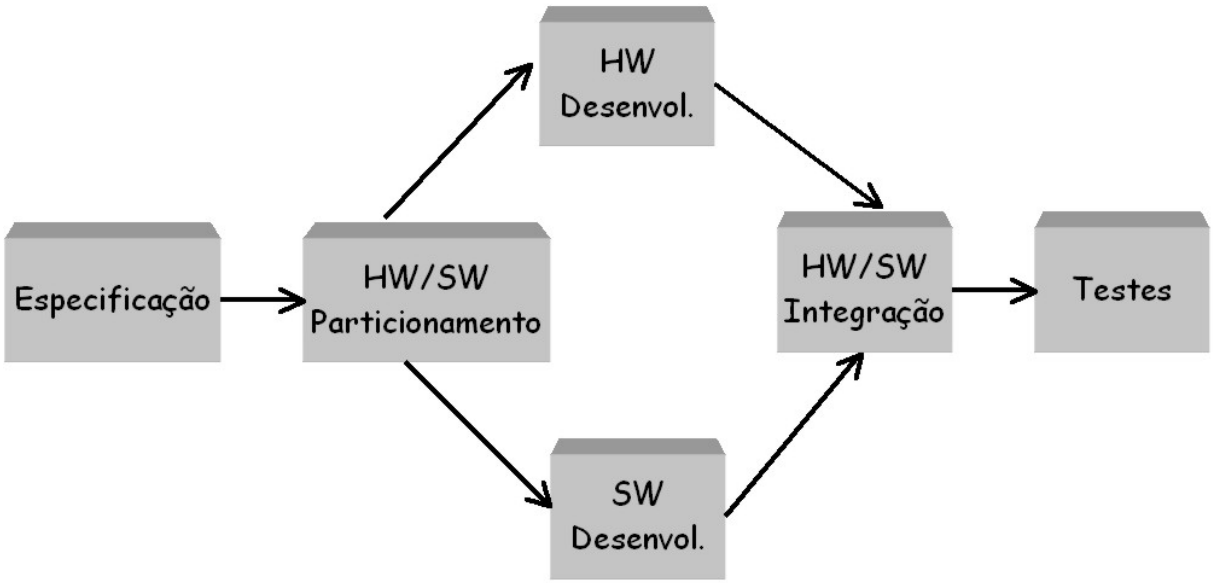

Figura 2.7: Particionamento *Hardware/Software* (Berger, 2002)

uso de seus módulos. É importante ressaltar que também existem organizações que distribuem *cores* gratuitamente, sem nenhuma restrição de uso. A tabela 2.2 mostra algumas empresas e organizações que distribuem *cores*. A maioria dos módulos são desenvolvidos utilizando-se linguagens de descrição de hardware (HDL), tais como: VHDL, Verilog e SystemC.

	Hard Core	***Firm Core***	***Soft Core***
Rigidez	A organização é predefinida.	Combinação de código fonte de net list dependente de tecnologia.	Apresenta um código fonte comportamental independente da tecnologia.
Modelagem	Modelado como um elemento de biblioteca.	Combinação de blocos sintetizáveis fixos. Permite compartilhar recursos com outros cores.	Sintetizável com diversas tecnologias.
Flexibilidade	Não pode ser modificado pelo projetista.	A personalização de funções específicas é dependente da tecnologia.	O projeto pode ser modificado e independente da tecnologia.
Previsibilidade	Temporização é garantida.	Caminhos críticos com temporizações fixas.	A temporização não é garantida, podendo não atender os requisitos do projeto.
Proteção da propriedade intelectual	Alta. A descrição normalmente corresponde a um *layout*.	Média.	Muito pequena. Código fonte geralmente aberto.

Tabela 2.1: Classificação dos *cores* (Palma et al., 2001).

2.3.1 Reuso de IP cores

A reutilização de componentes já vêm sendo utilizada a bastante tempo nos sistemas tradicionais, onde os fabricantes de computadores e periféricos já reaproveitam vários componentes desenvolvidos por terceiros. E agora está sendo bastante difundida para sistemas embutidos. O projeto e desenvolvimento de SoCs têm se tornado cada vez mais complexos, devido a grande quantidade de recursos disponíveis nos chips mais modernos. Para facilitar o desenvolvimento de tais sistemas é necessário utilizar uma abordagem de reutilização de *cores*, desenvolvidos por terceiros. Isso já tem se tornado bastante comum em grandes empresas que necessitam lançar seus produtos o mais rápido possível no mercado (Wilson, 2003), pois isso possibilita uma menor perda de tempo nos projetos.

Como visto anteriormente, existem empresas e organizações que disponibilizam vários *cores* previamente testados e validados. Isso significa que um projetista de sistema poderá

Fornecedor	Endereço eletrônico	IP
Free-IP	http://www.free-ip.com	Aberto
OpenCollector	http://www.opencollector.org	Aberto
In-Silicon	http://www.in-silicon.com	Teste/Pago
Vautomation	http://www.vautomation.com	Pago
CMOSexod	http://www.cmosexod.com	Aberto
FMF	http://www.vhdl.org/fmf	Aberto
Design-Reuse	http://www.design-reuse.com	Teste/Pago
OpenCores	http://www.opencores.org	Aberto
Arasan Chip systems	http://www.arasan.com	Pago
Amphion Semiconductor	http://www.amphion.com	Pago
Eureka Technology	http://www.eurekatech.com	Pago
Altera	http://www.altera.com/products/ip/ipm-index.html	Teste/Pago
Innovative Semiconductors	http://www.isi96.com	Pago
ARM	http://www.arm.com	Pago
Brazil IP	http://www.brazil-ip.org	Aberto
Xilinx	http://www.xilinx.com/ipcenter	Teste/Pago

Tabela 2.2: Distribuidores de *cores* (baseado em (Palma et al., 2001)).

utilizar tais recursos em seu projeto, sem precisar entender o funcionamento interno do *core*. É claro que ao comprar algum IP-*core*, espera-se que o vendedor disponibilize a documentação do mesmo. Pode-se notar então, que a complexidade no desenvolvimento de SoCs é bastante amortizada com a reutilização de componentes. Isso quer dizer que grande parte dos sistemas desenvolvidos atualmente só tem o trabalho de escolher quais *cores* desejam usar (Carro e Wagner, 2003) e integrá-los corretamente.

Muitas ferramentas estão disponíveis no mercado com o intuito de facilitar a construção de SoCs. Como exemplo pode-se citar o SoPC Builder (Altera, 2004d), da Altera. Tal ferramenta permite a inclusão de novos *cores* no sistema com apenas alguns comandos. A figura 2.8 mostra uma tela do ambiente de desenvolvimento desta ferramenta, onde no lado esquerdo são apresentados os *cores* que podem ser reutilizados e do lado direito tem-se todos os componentes que formam o sistema.

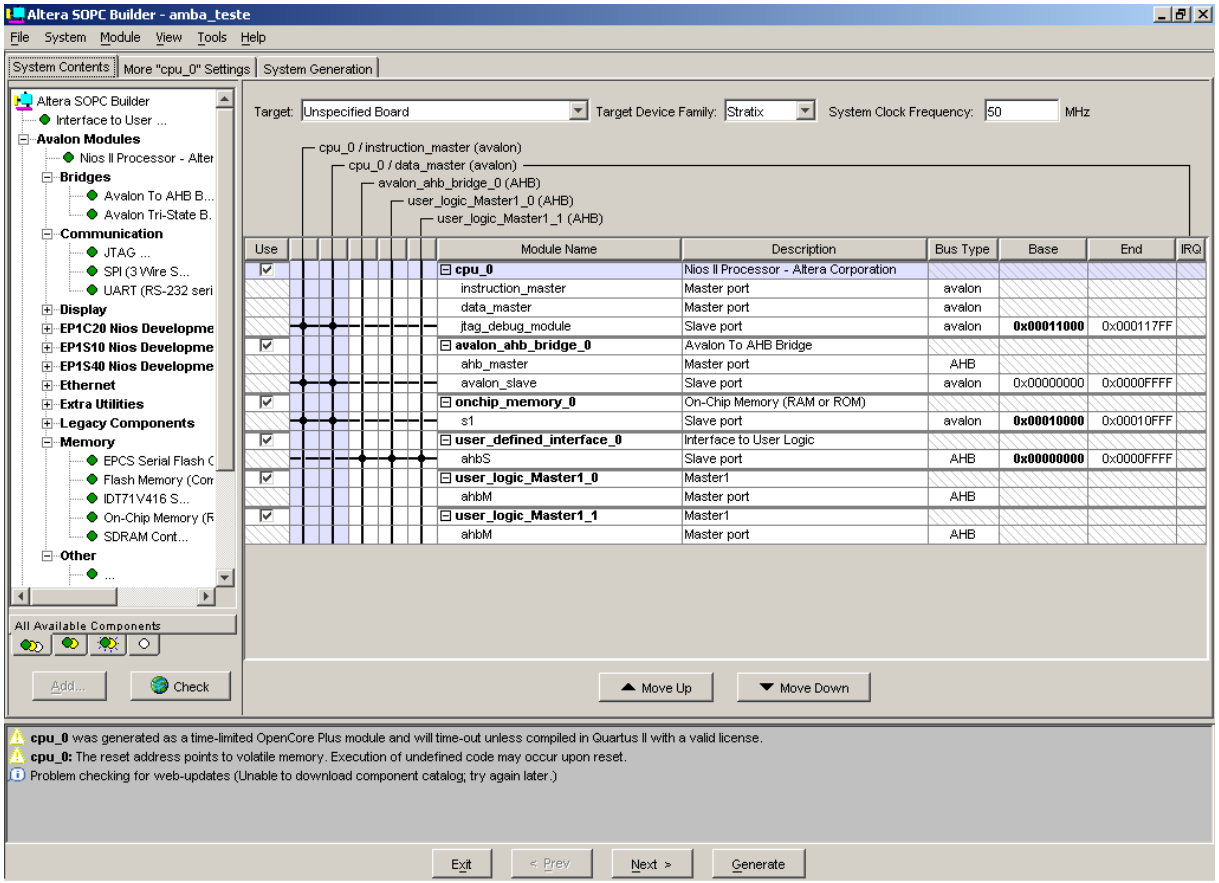

Figura 2.8: Ambiente de desenvolvimento do SoPC Builder

2.3.2 Padronização da Comunicação

De nada adianta várias empresas produzirem seus *cores* se elas não seguirem um determinado padrão. Pois, desta forma, ficaria muito difícil (ou até mesmo impossível) realizar a comunicação entre os mesmos. Com o intuito de resolver tal problema, vários esforços vêm sendo feitos para a padronização das interfaces de comunicação dos *cores*.

Projetos de SoC podem usar diversas formas de comunicação. Uma das mais comuns é a abordagem de projeto baseada em barramentos, onde os componentes do sistemas encontram-se interligados através de um ou mais barramentos, que por sua vez estão ligados entre si através de pontes (*bridges*) (Carro e Wagner, 2003). Com a especificação do barramento, é possível que empresas criem componentes já usando a interface do barramento. Isso possibilita que tais componentes possam ser ligados diretamente ao barramento escolhido.

Criar componentes baseados em algum padrão já é de extrema importância hoje em

dia, já que é desnecessário e, muitas vezes inadequado, perder tempo criando-se protocolos de comunicação customizados. Exemplos de padrões de comunicação são os baseados em determinado barramento on-chip (AMBA, CoreConect, Avalon, WISHBONE, etc) ou os que são independentes de barramento (OCP - *Open Core Protocol*). Este último está sob controle da OCP-IP (*Open Core Protocol International Partnership*).

Padrões baseados em barramentos são bastante utilizados por empresas ou organizações desenvolvedoras de IP-*cores*, como: ARM, OpenCores, Altera, etc. Porém, atualmente, também tem-se produzido vários componentes utilizando o padrão OCP, que é independente de barramento. Uma organização que utiliza o OCP é a BrazilIP (http://www.brazil-ip.org), que é um consórcio constituído por 8 universidades brasileiras (UNICAMP, USP, UFPE, UFCG, UFRGS, UFMG, UnB e PUC-RS) para a formação de pessoal qualificado em projeto de hardware e desenvolvimento de IP-*cores*.

2.3.2.1 Padrões baseados em barramentos

Os padrões baseados em barramentos já são utilizados há algum tempo pelos mais diversos vendedores de IP-*cores*. Uma vantagem desta abordagem é que os *cores* desenvolvidos com interfaces para determinado barramento, podem ser adicionados diretamente à sistemas que usam o barramento apropriado. Um exemplo pode ser visto na figura 2.9, onde todos os componentes possuem uma interface para o barramento AMBA. Já uma desvantagem é que quando deseja-se incluir tais componentes em algum outro barramento, deve-se alterar a interface do mesmo, ou utilizar-se de algum adaptador. A figura 2.10 mostra *cores* desenvolvidos com a interface AMBA utilizando adaptadores para o barramento WISHBONE. Nesta caso não foi necessário modificar a interface dos componentes. Porém, isso causa um atraso no sistema, já que os mesmos não estão ligados diretamente ao barramento.

2.3.2.2 Padrões independentes de barramentos

OCP-IP é uma associação dedicada a padronizar a comunicação de IP-*cores* e, desta maneira, facilitar o desenvolvimento de SoCs (OCP-IP, 2004b). Diferentemente da abor-

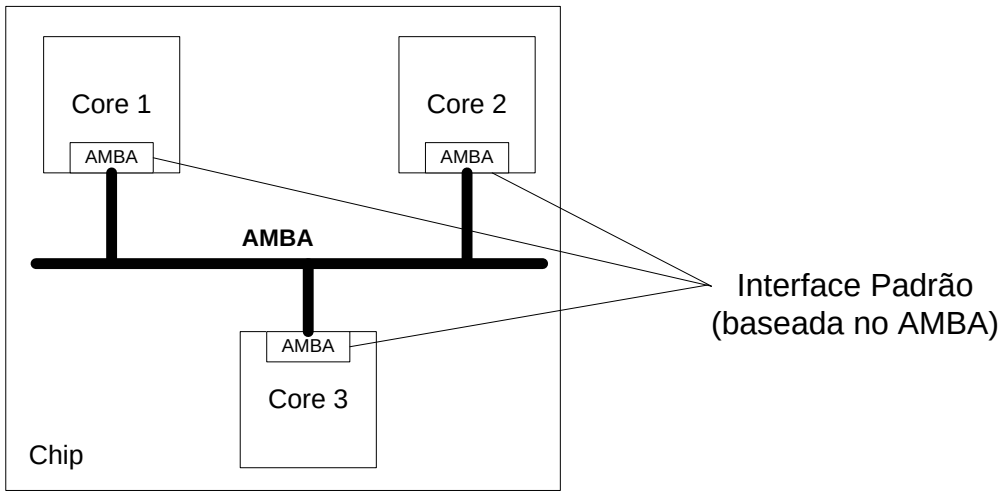

Figura 2.9: Sistema on-chip utilizando *cores* com interfaces AMBA

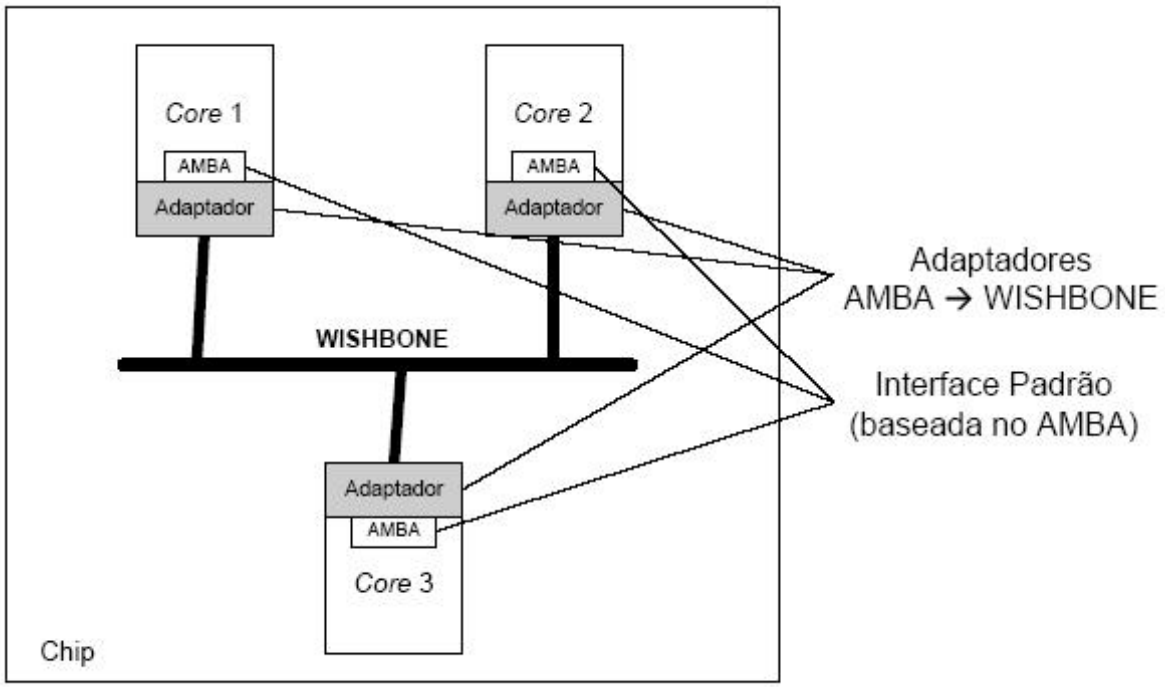

Figura 2.10: Sistema on-chip utilizando adaptadores de interface para o barramento WISHBONE

dagem baseada em barramentos, o padrão OCP não possui nenhuma relação direta com a maneira de comunicação a ser realizada (barramento, NoC ou outro qualquer) entre os componentes de um SoC. Tal padrão especifica apenas o protocolo de comunicação. Qual método de comunicação que será usado, para ligar os diversos *cores*, fica a critério do usuário, onde o mesmo pode usar qualquer tipo de barramento on-chip ou até mesmo NoC (Zhou, 2004). Uma das vantagens de ser independente de barramento é que o desenvolvedor do IP-*core* não precisa ter nenhum conhecimento do ambiente em que o componente será utilizado, já que pode ser ligado a qualquer meio de comunicação. Essa ligação é feita através de adaptadores que traduzem os sinais provenientes do protocolo OCP para o protocolo específico a ser utilizado. Desta maneira, para que um componente desenvolvido utilizando OCP possa ser inserido em um SoC baseado no barramento AMBA por exemplo, é necessário que exista um adaptador OCP/AMBA. A figura 2.11 mostra dois componentes (um mestre e um escravo) OCP, na parte superior, ligados ao barramento AMBA através de adaptadores.

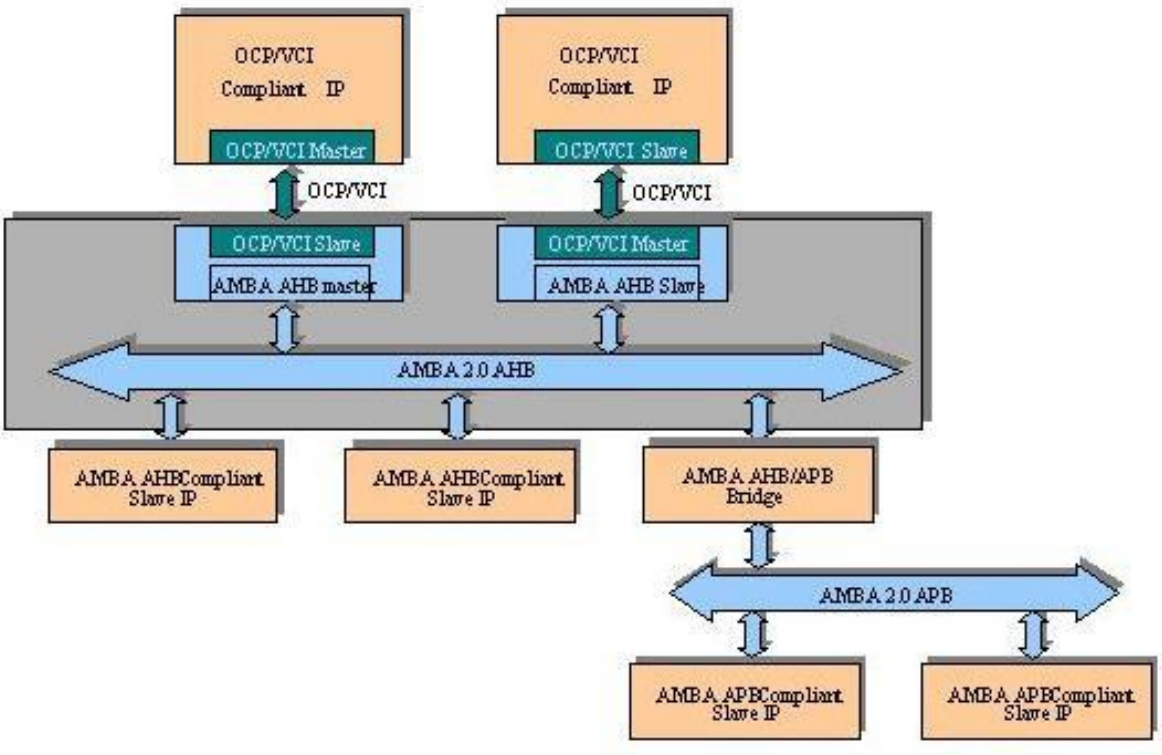

Figura 2.11: Componentes OCP ligados ao barramento AMBA através de adaptadores (Prosilog, 2004)

Uma desvantagem de usar o padrão OCP é que sempre que os componentes forem adicionados a diferentes tipos de barramentos ou NoC, eles terão que utilizar os adaptadores apropriados. Ou seja, não podem ser ligados de maneira direta. Isso causa uma perda de desempenho, já que existe uma determinada lógica em tais adaptadores. É claro que a grande vantagem de utilizar OCP é que o desenvolvedor poderá produzir seus *cores* sabendo que os mesmos poderão ser facilmente ligados em qualquer que seja o meio de comunicação de qualquer sistema on-chip. E para isso, não é necessário realizar nenhuma mudança em seus componentes, bastando apenas trocar seus adaptadores. A figura 2.12 mostra a diferença de adaptadores que traduzem os sinais de um barramento 'A' para um barramento 'B', por exemplo, e adaptadores OCP (que traduzem sinais OCP para algum outro barramento). Os adaptadores OCP são mais padronizados que os outros, já que uma das suas extremidades sempre será ligada a algum *core* OCP. Já os outros adaptadores devem possuir, em suas extremidades, as interfaces do barramento de origem e destino. Então, caso seja necessário a inclusão de um componente AMBA no barramento WISHBONE, o adaptador deve possuir as interfaces AMBA e WISHBONE, caso o mesmo componente precise ser incluído no barramento Avalon, deve-se possuir as interfaces AMBA e Avalon. No caso dos adaptadores OCP uma das interface sempre será a mesma (interface OCP).

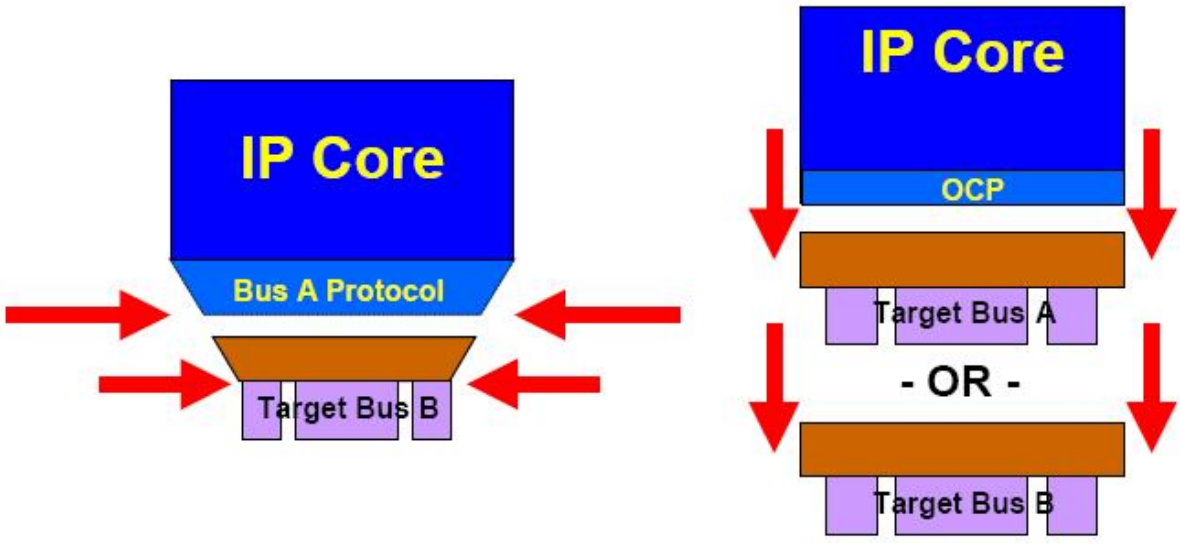

Figura 2.12: Diferença entre adaptadores de um barramento para outro e do padrão OCP para algum barramento (OCP-IP, 2004a)

Capítulo

3

Computação Reconfigurável

Neste capítulo são apresentados alguns tópicos sobre lógica (re)programável, como a evolução dos primeiros dispositivos lógicos programáveis (PLDs) até a criação dos CPLDs e FPGAs. São abordadas também características de algumas famílias de FPGAs mais modernas, como Stratix e Stratix II da Altera e Virtex-II Pro e Virtex-4 da Xilinx. Por último, são apresentadas algumas áreas e aplicações que utilizam e poderiam utilizar a tecnologia de computação reconfigurável.

3.1 Dispositivos Lógicos Programáveis

Existe uma grande variedade de dispositivos (circuitos integrados) que podem ter suas funções lógicas programadas após sua fabricação. Tais dispositivos são chamados de PLDs (*Programmable Logic Devices*) e muitos deles permitem também que as funções sejam reprogramadas de maneira que eventuais erros sejam corrigidos ou que a funcionalidade do CI (circuito integrado) seja alterada (Wakerly, 2001).

Alguns destes dispositivos são apresentados a seguir, seguindo a ordem cronológica em que os mesmos surgiram.

PLA: *Programmable Logic Arrays* foram os primeiros dispositivos programáveis e são compostos por dois níveis de portas lógicas 'E' e 'OU'. Tal dispositivo pode ser programado para realizar qualquer soma de produtos. Um exemplo de um PLA com 4 entradas e 3 saídas pode ser visto na figura 3.1, cujas equações resultantes são:

O1 = I1·I2+I1'·I2'·I3'·I4'

O2 = I1·I3'+I1'·I3·I4+I2

O3 = I1·I2+I1·I3'+I1'·I2'·I4'

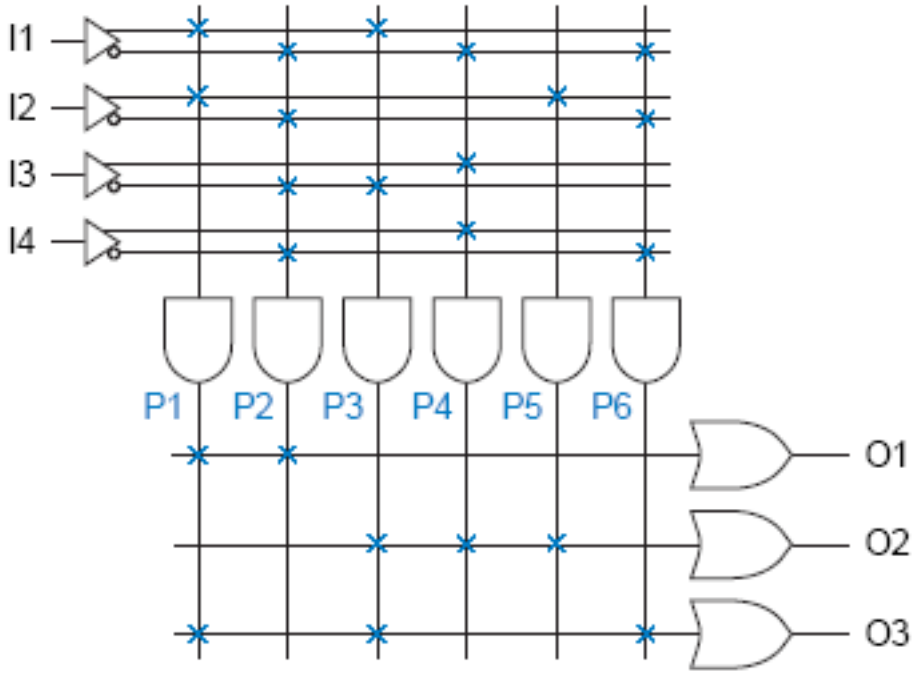

Figura 3.1: Esquema de um PLA 4x3 (Wakerly, 2001).

PAL: *Programmable Array Logic* trata-se de um caso especial do PLA, onde possui um número fixo de portas 'OU'. Isso ocasionou uma redução nos custos de produção dos PLDs.

CPLD: *Complex Programmable Logic Devices* são dispositivos que possuem uma maior capacidade em relação aos SPLDs (*Simple Programmable Logic Devices*) vistos anteriormente, como: PLAs e PALs. O grande aumento de capacidade dos CIs permitiu a produção de PLDs maiores, porém a estrutura de níveis de portas lógicas 'E' e 'OU'

não era adequada para dispositivos de alta capacidade (Wakerly, 2001). A partir daí foram criados os CPLDs, que nada mais são do que uma coleção de SPLDs. Todo o mecanismo de interconexão entre esses diversos SPLDs também é programável. Uma ilustração desta abordagem pode ser vista na figura 3.2.

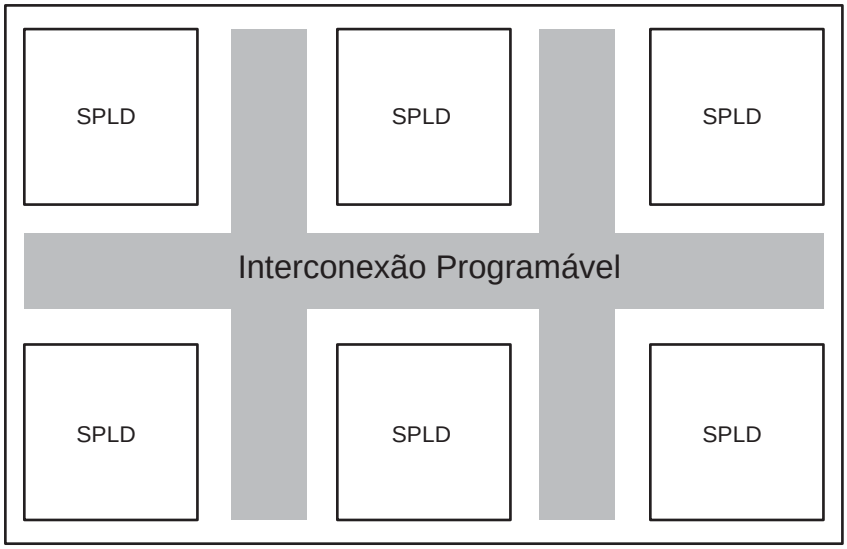

Figura 3.2: Esquema geral de um CPLD.

3.2 FPGAs

No mesmo período em que os CPLDs começaram a ser inventados, surgiu uma nova abordagem de construção de CIs da alta capacidade. Foram criados então, pela empresa Xilinx em 1984, os FPGAs (*Field Programmable Gate Arrays*) que são arranjos de blocos lógicos contidos em um único chip. A arquitetura de um FPGA depende de três fatores fundamentais, que são (Aragão e Marques, 1997):

- Tecnologia de programação, como: SRAM (*Static Random Access Memory*), *Anti-fuse* e *Gate* Flutuante.
- Arquitetura dos blocos lógicos, como: Multiplexadores e *Look-up-tables* (LUTs), entre outros.
- Arquitetura de roteamento.

A figura 3.3 ilustra a estrutura geral de um FPGA.

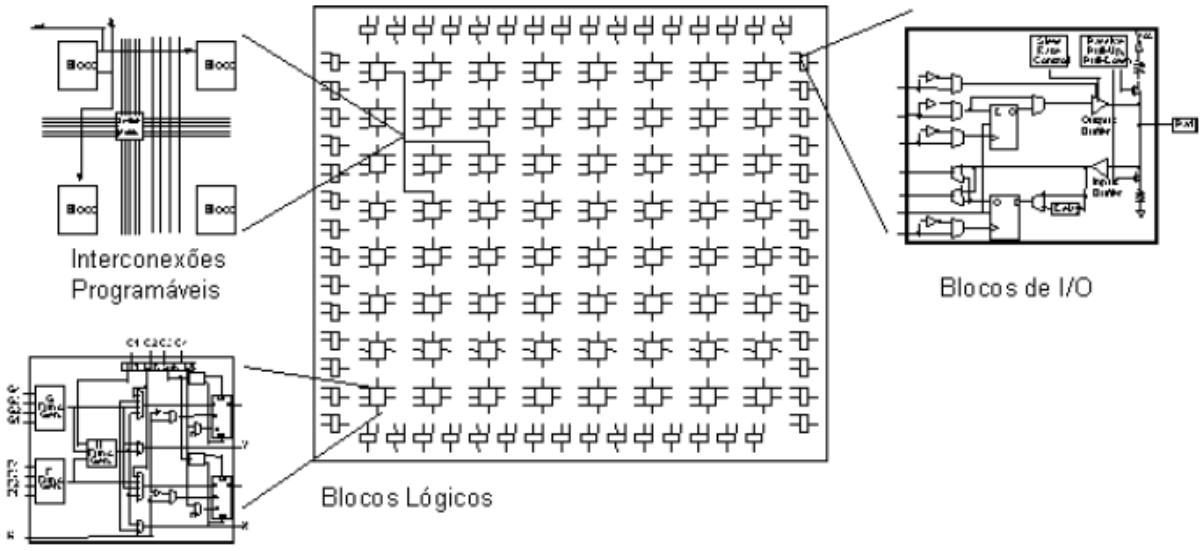

Figura 3.3: Estrutura geral de um FPGA (Xilinx, 1999).

3.2.1 FPGAs atuais

Existem diversos fabricantes de FPGAs, tais como: Altera, Xilinx, Actel, QuickLogic, Atmel, etc. Os FPGAs mais modernos estão bem mais densos que os antigos, possuindo em alguns casos 10 milhões de portas lógicas e várias outras características que tornam possível o aumento no desempenho de SoCs (*Systems On-Chip*). Serão apresentadas a seguir, quatro das mais recentes famílias de FPGAs da Altera e Xilinx. São elas: Stratix, Stratix II, Virtex-II Pro e Virtex-4.

3.2.1.1 Stratix - Altera

Stratix é uma das famílias mais recentes de FPGAs da Altera e possui sete membros: EP1S10, EP1S20, EP1S25, EP1S30, EP1S40, EP1S60 e EP1S80. Os primeiros FPGAs desta família começaram a ser produzidos em maio de 2002 (Altera, 2003c). Os dispositivos desta família possuem de 10.570 a 79.040 elementos lógicos[1] e várias outras características que podem ser visualizadas na tabela 3.1 (os dados da tabela são referentes a quantidade de elementos apresentados na primeira coluna). A arquitetura Stratix também foi projetada para aumentar os benefícios de desempenho do processador embu-

[1]Elementos Lógicos: Blocos lógicos básicos que podem ser (re)programados.

tido Nios[2]. As características da arquitetura Stratix juntamente com o processador Nios tornam possível o melhoramento do desempenho de sistemas embutidos, que pode chegar à uma freqüência de processamento de 150MHz. Dispositivos da família Stratix também possuem algumas características avançadas que aumentam o desempenho e a largura de banda em SOPCs (*System-On-a-Programmable-Chips*), além de diminuir o tempo em que o produto poderá ser lançado no mercado (*time-to-market*). Algumas das características são mostradas a seguir (Altera, 2003c).

	Dispositivo						
Característica	**EP1S10**	**EP1S20**	**EP1S25**	**EP1S30**	**EP1S40**	**EP1S60**	**EP1S80**
Elem. Lógicos	10570	18460	25660	32470	41250	57120	79040
Mem. 512 bits	94	194	224	295	384	574	767
Mem. 4 Kbits	60	82	138	171	183	292	364
Mem. 512 Kbits	1	2	2	4	4	6	9
bits RAM total	920448	1669248	1944576	3317184	3423744	5215104	7427520
Blocos DSP	6	10	10	12	14	18	22
Multip. Embut.	48	80	80	96	112	144	176
Máx. pin. E/S	426	586	706	726	822	1.022	1.203

Tabela 3.1: Algumas características dos dispositivos da família Stratix (Altera, 2003c).

Arquitetura de alto desempenho :

- Aumento de desempenho de 50%, em média, em relação aos outros dispositivos anteriores da Altera, quando se utiliza o software Quartus II[3];

Memória TriMatrix (Três tamanhos diferentes de blocos de memória) :

- Até 7 Mbits de memória embutida;
- Blocos de memória com três tamanhos (512Kbits, 4 Kbits e 512 bits) mais os bits de paridade;
- Até 8 Terabits/s de largura de banda total da memória;
- Bits de paridade para checagem de erros;
- Registrador de deslocamento embutido.

[2]Nios: Processador embutido da Altera.
[3]Quartus II: Ferramenta de desenvolvimento de hardware da Altera.

Blocos DSP :

- Até 22 blocos DSP de alto desempenho por dispositivo;
- Circuitos dedicados de pipeline e multiplicadores.

3.2.1.2 Stratix II - Altera

Stratix II é a família mais recente de FPGAs da Altera, e possui atualmente 6 membros: EP2S15, EP2S30, EP2S60, EP2S90, EP2S130 e EP2S180. Tais dispositivos possuem entre 15.600 a 179.400 elementos lógicos, além de várias outras características que podem ser vistas na tabela 3.2 (os dados da tabela são referentes a quantidade de elementos apresentados na primeira coluna). Eles oferecem também uma completa solução de gerenciamento de *clock*, com freqüência interna de até 500 MHz (Altera, 2004e).

	Dispositivo					
Característica	**EP2S15**	**EP2S30**	**EP2S60**	**EP2S90**	**EP2S130**	**EP2S180**
Elem. Lógicos	15.600	33.880	60.440	90.960	132.540	179.400
Mem. 512 bits	104	202	329	488	699	930
Mem. 4 Kbits	78	144	255	408	609	768
Mem. 512 Kbits	0	1	2	4	6	9
bits RAM total	419.328	1.369.728	2.544.192	4.520.488	6.747.840	9.383.040
Blocos DSP	12	16	36	48	63	96
Máx. pin. E/S	365	499	717	901	1.109	1.173

Tabela 3.2: Algumas características dos dispositivos da família Stratix II (Altera, 2004e).

Os dispositivos da família Stratix II, também possuem, entre outras, as seguintes características:

- Até 9.383.040 bits disponíveis de memória RAM, sem reduzir os recursos lógicos;
- Memória TriMatrix;
- Blocos DSP de alta velocidade que permitem a implementação de multiplicadores de até 370 Mhz;
- Suporte para memórias externas de alta velocidade, incluindo DDR e DDR2 SDRAM, RLDRAM II, QDR II SRAM e SDR SDRAM;
- Suporte para projetos seguros, usando criptografia;

- Suporte para atualizações remotas de configuração;
- Suporte para diferentes padrões de E/S.

3.2.1.3 Virtex-II Pro - Xilinx

Virtex-II Pro é uma das recentes famílias de FPGAs da Xilinx. Baseada na arquitetura do Virtex-II, a família Virtex-II Pro expande as capacidades das aplicações que utilizam a tecnologia FPGA através de suas novas características, como por exemplo a possibilidade de utilização de *cores* do processador embutido IBM PowerPC 405 (PPC405) (Xilinx, 2002b).

Além disso a família Virtex-II Pro possui as seguintes características:

- Tecnologia de controle digital de impedância (DCI - *Digitally Controlled Impedance*) para gerenciamento da integridade de sinais;
- Gerenciadores de clock digital (DCM - *Digital Clock Managers*) para criação de domínios avançados de clock;
- Multiplicadores XtremeDSP avançados para aplicações DSP de alto desempenho.

A família Virtex-II Pro possui 10 membros, variando sua capacidade de 3K a 125K células lógicas. Possuem também uma grande quantidade de memória RAM, assim como gerenciadores de clock (DCM). Algumas características dos 10 membros da família Virtex-II Pro podem ser vistas na tabela 3.3.

A Xilinx também possui uma ferramenta que provê suporte ao desenvolvimento de SoC (*System On-Chip*) nos FPGAs da família Virtex-II Pro. A partir da versão 4.2i da ISE/Foundation já é fornecido um suporte total aos dispositivos da Virtex-II Pro.

3.2.1.4 Virtex-4 - Xilinx

Virtex-4 é a mais nova família de FPGAs da Xilinx. Ela é subdividida em 3 famílias distintas: LX, FX e SX. A LX é ideal para soluções para aplicações lógicas que requerem um alto desempenho, a FX oferece uma solução mais completa que a LX para aplicações

Dispositivo	Células Lógicas	SelectRAM (Kbits)	Blocos DCM	Mult. Xtreme
2VP2	3168	216	4	12
2VP4	6768	504	4	28
2VP7	11088	792	4	44
2VP20	20880	1584	8	88
2VP30	30816	2448	8	136
2VP40	43632	3456	8	192
2VP50	53136	4176	8	232
2VP70	74448	5904	8	328
2VP100	99216	7992	12	444
2VP125	125136	10008	12	556

Tabela 3.3: Algumas características dos dispositivos da família Virtex-II Pro (Xilinx, 2002b).

em plataformas embutidas, já a SX é a mais indicada para aplicações de DSP. Existe também nessa família alguns IPs (*Intelectual Property*) que já vêm no chip, tais como: Processador PowerPC, *Ethernet* MAC (*Media Access Control*), *transceivers* seriais, blocos para monitoramento de voltagem e temparatura, DSP dedicados, circuito para gerenciamento de *clock*.

A família é formada por 17 dispositivos (sendo 8 da subfamília LX, 6 da FX e 3 da SX), e possui uma quantidade de células lógicas que variam de 12.312 a 200.448 e pode trabalhar a uma frequência de 500MHz. Algumas das características dos dispositivos desta família podem ser vistas na tabela 3.4.

3.3 Computação Reconfigurável e seu uso prático

Sistemas de computação reconfigurável são sistemas que utilizam FPGAs e podem modificar suas configurações a qualquer momento, inclusive em tempo real, para melhor se adaptar à tarefa/problema que foi proposto a solucionar. No caso do hardware reconfigurável, sua configuração pode ser modificada para criar a melhor arquitetura possível para que determinada aplicação possa tirar proveito máximo da mesma. Neste caso, há uma adaptação do hardware ao software.

A computação reconfigurável combina a flexibilidade de dispositivos programáveis com o desempenho do hardware específicos (como ASICs - *Application Specific Integrated Circuits*). Além disso existem várias ferramentas EDAs (*Electronic Design Automation*)

Dispositivo	Células Lógicas	Ethernet MAC	PowerPC	Xtreme DSP
XC4VLX15	13824	N/A	N/A	32
XC4VLX25	24192	N/A	N/A	48
XC4VLX40	41472	N/A	N/A	64
XC4VLX60	59904	N/A	N/A	64
XC4VLX80	80640	N/A	N/A	80
XC4VLX100	110592	N/A	N/A	96
XC4VLX160	152064	N/A	N/A	96
XC4VLX200	200448	N/A	N/A	96
XC4VSX25	23040	N/A	N/A	128
XC4VSX35	34560	N/A	N/A	192
XC4VSX55	55296	N/A	N/A	512
XC4VFX12	12312	2	1	32
XC4VFX20	19224	2	1	32
XC4VFX40	41904	4	2	48
XC4VFX60	56880	4	2	128
XC4VFX100	94896	4	2	160
XC4VFX140	142128	4	2	192

Tabela 3.4: Algumas características dos dispositivos da família Virtex-4 (Xilinx, 2004b)

para o projeto, testes, simulação, síntese e implementação de hardwares reconfiguráveis. Com tais ferramentas é possível projetar o hardware utilizando linguagens de descrição de hardware (HDL - *Hardware Description Language*), como por exemplo: VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), Verilog, AHDL e ABEL. Assim, como softwares podem ser facilmente atualizados pela internet, o hardware também poderá ser atualizado de maneira similar (Ribeiro e Marques, 2004). Isso já é possível com a utilização de dispositivos lógicos reprogramáveis como os FPGAs.

A computação reconfigurável pode atuar em uma grande variedade de aplicações, de diversas áreas (comercial, militar, acadêmica, aeroespacial, etc), que necessitem de um alto desempenho. Alguns dos campos que podem utilizar computação reconfigurável são:

Robótica: Onde as características do robô podem ser alteras em tempo real. Com isso o robô poderá melhor se adaptar à determinado ambiente e reagir da forma que for mais adequada.

Telefones Celulares: As funcionalidades do telefone podem ser alteradas ou apenas atualizadas e expandidas.

Aeronáutica: Os pilotos-automáticos ou outros tipos de equipamentos eletrônicos dos aviões poderiam ser reconfigurados de acordo com as necessidades dos vôos. Correções em equipamentos poderiam ser feitas remotamente.

Automobilismo: Determinado controlador de algum dispositivo de um carro poderia ser modificado para atender as necessidades do piloto/motorista. No caso da fórmula 1, a equipe poderia mexer no carro, até mesmo de forma remota, para conseguir um maior desempenho nas pistas.

Urnas para Votação Eletrônica: As configurações de uma eleição (nomes dos candidatos, números, fotos, etc) poderiam ser facilmente modificadas por hardware. Como tudo seria implementado via hardware, o eleitor gastaria menos tempo para votar, já que seu desempenho comparado com o software é bem superior.

Prototipação de Hardware: A construção de protótipos de hardware usando computação reconfigurável é um processo rápido e barato.

Computação de Alto Desempenho: Sistemas paralelos, caso necessário, podem ser implementados através de FPGAs, já que um FPGA pode conter mais de um processador. Ou diferentes FPGAs podem trabalhar em paralelo, trocando informações entre si. Processamento de imagens e aplicações de tempo real são outros exemplos em que se pode utilizar a abordagem de computação reconfigurável (Aragão et al., 2000).

A seguir serão descritos alguns projetos, de algumas instituições/empresas, que utilizam computação reconfigurável.

Robô Móvel Reconfigurável: Projeto, ao qual este trabalho está inserido, que visa construir um robô móvel reconfigurável. O robô será composto por vários módulos e os mesmos serão totalmente reconfiguráveis. FPGA é a principal tecnologia utilizada e os módulos do robô estão sendo desenvolvidos por diversos alunos de pós-graduação no LCR (Laboratório de Computação Reconfigurável) e LABIC (Laboratório de Inteligência Computacional), que estão localizados no campus de São Carlos da Universidade de São Paulo - USP.

Satélites da ESA: A Agência Espacial Européia (ESA - *European Space Agency*) utiliza, a mais de uma década, a tecnologia FPGA em alguns de seus satélites. Maiores informações podem ser obtidas no endereço eletrônico http://www.estec.esa.nl/wsmwww/asic/fpga.html.

VCM BMW WilliamsF1: A equipe de fórmula 1 BMW WilliamsF1 utiliza FPGA no VCM (*Vehicle Control and Monitoring*) de seus carros. Isso permite que os engenheiros da BMW WilliamsF1 possam, facilmente, adicionar ou remover funcionalidades aos VCMs dos carros, para que os mesmos sejam aproveitados da melhor forma possível e estejam de acordo com o regulamento da FIA (Federação Internacional de Automobilismo). Maiores informações em http://www.xilinx.com/publications/xcellonline/xcell_47/xc_formula47.htm.

Guitarra Digital: A empresa GIBSON é pioneira na produção de guitarras digitais, que possuem FPGAs para transformação dos sinais sonoros analógicos em digitais. É possível também transportar os sinais através de uma rede ethernet. Maiores detalhes em (Xilinx, 2002a).

Robôs Aeroespaciais: A NASA (*National Aeronautics and Space Administration*) utilizou FPGAs em seus robôs enviados ao planeta Marte em 2004. Isso possibilitou a reconfiguração dos mesmos a milhões de quilômetros de distância. Mais informações em (Xilinx, 2004a).

Como pôde ser visto anteriormente, existe uma grande quantidade de áreas em que a computação reconfigurável pode ser utilizada. Porém, para que seu uso seja mais significativo é necessário que exista mão-de-obra (engenheiros de hardware) qualificada, o que no Brasil ainda é um assunto problemático.

Capítulo 4

Barramentos de Comunicação

Serão abordadas, neste capítulo, algumas funcionalidades gerais de um barramento, assim como funcionalidades de barramentos on-chip. Serão apresentados, também, alguns dos barramentos on-chip mais conhecidos no mercado e algumas diferenças entre barramentos on-chip e tradicionais.

4.1 Conceitos gerais sobre barramentos

O barramento é um caminho de comunicação entre diversos dispositivos (Stallings, 2000). É responsável por prover subsídios para que os mesmos possam trocar informações entre si. Uma característica do barramento é que se trata de um meio de transmissão compartilhado, onde são usados diversos fios elétricos (*wires*) para a ligação entre os dispositivos (Patterson e Hennessy, 1998). O barramento utilizado para interligar os principais componentes de um computador (memória, processador e módulos de E/S) é chamado de **barramento de sistema**.

Como diversos dispositivos podem ser ligados ao barramento (sendo ele um meio compartilhado), um sinal (dígito binário) enviado por um dos dispositivos pode ser recebido

por todos os outros. É possível também que mais de um dispositivo tente enviar um sinal ao mesmo tempo, gerando assim uma informação inconsistente pois os sinais podem se sobrepor ou se intercalar. Para que este problema seja resolvido é preciso que haja apenas um dispositivo por vez transmitindo sinais ao barramento.

Os sinais são enviados através dos fios elétricos. Podem ser enviados de maneira seqüencial ou paralela.

Seqüencial: Uma seqüencia de *bits* é enviada através de um único fio elétrico (linha).

Paralela: Vários *bits* são transmitidos ao mesmo tempo através de diversos fios elétricos.

O barramento tipicamente possui linhas de controle, de endereços e de dados. Além desses três tipos, devem existir também linhas de distribuição de energia. A figura 4.1 mostra o três tipos funcionais de linhas.

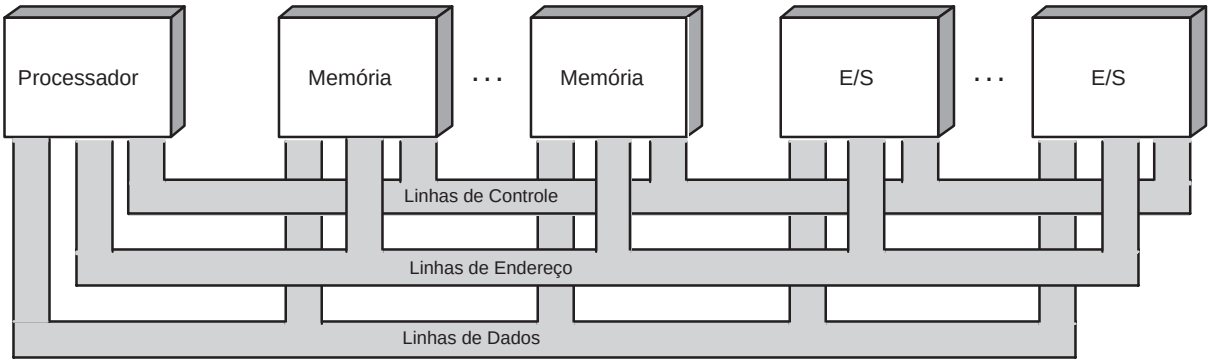

Figura 4.1: Três tipos de linhas (Stallings, 2000).

Linhas de Controle: Usada para enviar sinais de controle (operações a serem realizadas) sobre a utilização e o acesso às linhas de dados e endereço. Tal controle deve ser realizado, pois tais linhas são compartilhadas. O sinais de controle são utilizados também para transmitir informações de temporização (*clock*). O comando *reset* (que inicializa todos os componentes do sistema) também utiliza as linhas de controle.

Linhas de Dados: São utilizadas para o transporte dos dados entre diversos componentes de um sistema. Tais linhas são conhecidas como **barramento de dados** e o

número de fios elétricos, que forma o barramento de dados, especifica a largura do mesmo. Tal largura é um ponto chave para um bom desempenho do sistema.

Linhas de Endereço: Utilizadas para especificar a fonte ou destino dos dados trafegados pelas linhas de dados (barramento de dados). O número de linhas utilizadas para compor o **barramento de endereço** determina a capacidade máxima de memória que pode ser utilizada no sistema.

O funcionamento básico de um barramento pode ser visto das seguintes formas:

- Componente deseja enviar dados para outro.
  1. Conseguir o controle do barramento.
  2. Transferir dados através do barramento de dados.
- Componente deseja requisitar dados de um outro.
  1. Conseguir controle do barramento.
  2. Fazer a requisição, usando para isso as linhas de controle e endereço. Depois deve esperar que o outro dispositivo envie os dados requisitados.

4.1.1 Hierarquia de múltiplos barramentos

Com o objetivo de evitar a perda de desempenho quando existe uma grande quantidade de dispositivos conectados ao barramento, muitos sistemas computacionais utilizam o esquema de múltiplos barramentos organizados de maneira hierárquica. Para que os dispositivos mais lentos não acabem prejudicando o desempenho global do sistema, eles geralmente são conectados no último nível de hierarquia dos barramentos. Dessa forma, os dispositivos mais rápidos são ligados aos primeiros níveis da hierarquia. Um exemplo de hierarquia de múltiplos barramentos pode se visualizado na figura 4.2. Tal figura mostra um barramento local ligando o processador com o controlador da memória cache, que por sua vez está ligada ao barramento de sistema (em que encontra-se a memória principal) e também atua como ponte para o barramento de alta velocidade. Tal barramento

conecta vários dispositivos que necessitam de uma grande velocidade, além de ser ligado ao barramento de expansão através de uma ponte (*bridge* - *Interface* para barramento de expansão). O barramento de expansão, neste exemplo, conecta os dispositivos mais lentos do sistema.

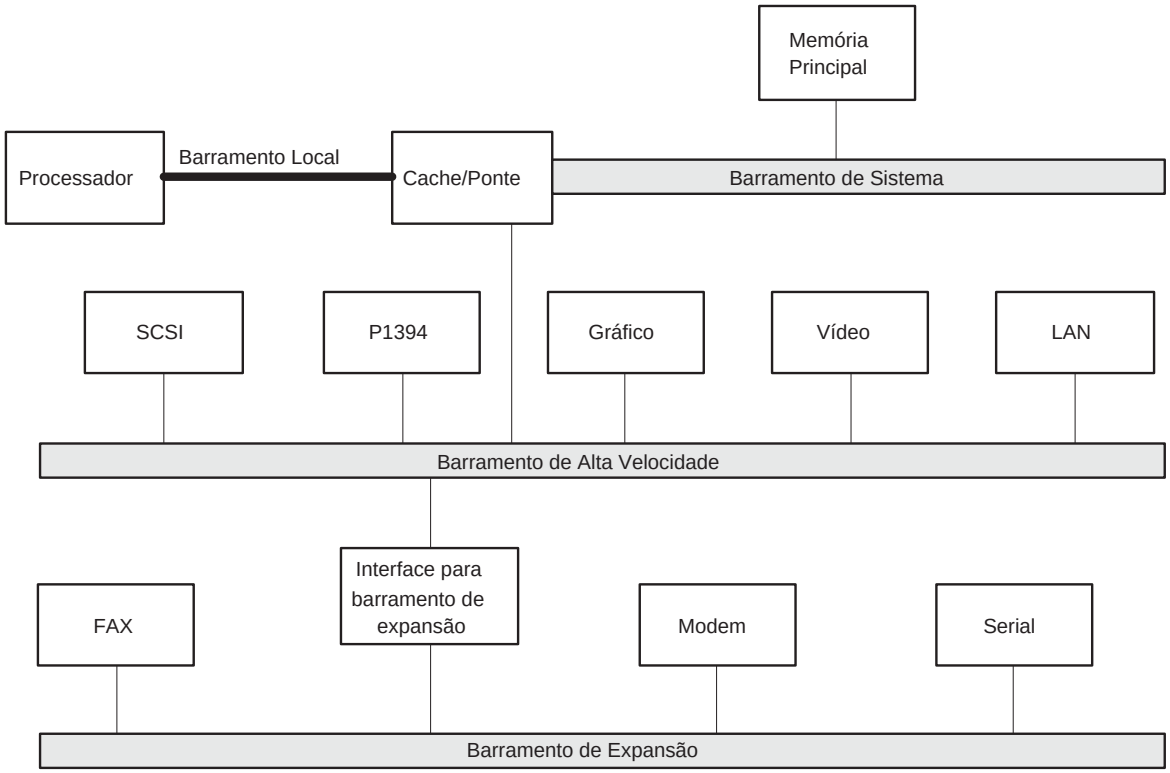

Figura 4.2: Hierarquia de múltiplos barramentos (Stallings, 2000).

4.1.2 Projeto de barramentos

Existem diferentes maneiras de se implementar um barramento. Para a obtenção de um melhor desempenho em determinado sistema, várias características devem ser observadas durante o projeto do barramento. Algumas das características serão abordadas nas subseções seguintes.

4.1.2.1 Largura do barramento

A largura do barramento tem impacto direto no desempenho do sistema. Quanto maior a largura, maior o número de bits transmitidos ao mesmo tempo (em paralelo). No caso do **barramento de dados**, quanto maior sua largura, maior a quantidade de

dados que serão transmitidos. No caso do **barramento de endereço**, quanto maior sua largura, maior será a capacidade de memória do sistema.

4.1.2.2 Tipos de barramento

As linhas de determinado barramento podem ser classificadas de acordo com dois tipos: dedicado e multiplexado.

Dedicado: Linhas de barramento dedicadas são associadas a um subconjunto de dispositivos de um sistema ou podem ser responsáveis por determinadas funções fixas. No caso das funções fixas existirão linhas diferentes para o envio/recebimento de dados e endereços. O barramento também pode ser utilizado para conectar um subconjunto de dispositivos. Neste caso um sistema completo pode ser formado por vários barramentos (no esquema de hierarquia). Isso trás uma grande vantagem, já que a utilização do barramento de um determinado subconjunto de dispositivos não necessariamente interfere na utilização do barramento de outro subconjunto.

Multiplexado: As mesmas linhas podem ser utilizadas para vários propósitos distintos, como o envio/recebimento de dados e endereço. Uma vantagem desta abordagem é que com a utilização de poucas linhas, o espaço físico necessário para as linhas do barramento é bem menor. Isso causa uma diminuição nos custos de construção de tais linhas. Porém, esta abordagem torna-se mais lenta que a anterior, já que será necessário um número maior de acessos às linhas.

4.1.2.3 Temporização

Existem duas maneiras distintas de coordenar a ocorrência de eventos em um barramento. A coordenação pode ser feita de maneira síncrona ou assíncrona.

Síncrona: Eventos baseados em um relógio (*clock*). Neste esquema deve existir no barramento, uma linha específica para o *clock*. Todos os dispositivos podem ler essa linha e todos os eventos devem iniciar no começo de um ciclo de *clock*[1].

[1]Ciclo de *Clock*: Transmissão de um '1' e um '0'. Também chamado como ciclo de barramento.

Assíncrona: Eventos são baseados em outros eventos que ocorreram anteriormente. Ou seja, só vai ocorrer alguma variação de sinal, caso tenha ocorrido uma outra variação, anteriormente, em algum outro sinal que influencia no primeiro sinal citado. Neste caso não há um relógio para controle dos eventos.

A abordagem síncrona é bem menos flexível que a assíncrona, porém possui a vantagem de ser mais fácil de se implementar e testar. Como nesta abordagem todos os eventos são baseados no **ciclo do barramento**, o sistema não poderá tirar proveito de dispositivos que, por ventura, possuam um ciclo de *clock* mais veloz. Já a abordagem assíncrona permite que todos os tipos de dispositivos (velozes ou lentos) compartilhem a utilização do barramento.

4.1.2.4 Arbitragem

Deve existir uma maneira de controlar o acesso ao barramento, pois caso não exista, mais de um dispositivo poderia acessar o barramento ao mesmo tempo. Isso geraria uma confusão no sistema. Este problema pode ser solucionado com a utilização de um ou mais mestres, escravos e mecanismos de arbitragem.

Mestre (*Master*): Controla o acesso ao barramento, controlando e iniciando todas as requisições ao barramento. Um exemplo de um dispositivo mestre é o processador, responsável por várias requisições à diferentes dispositivos.

Escravo (*Slave*): Dispositivo responsável por responder as requisições do mestre. A memória é um típico dispositivo escravo, já que sua tarefa é responder à requisições de outros dispositivos.

Vale ressaltar que um sistema pode possuir diversos dispositivos mestres e escravos ligados ao mesmo barramento. E que um mesmo dispositivo pode ser considerado mestre ou escravo, dependendo da sua atuação em determinada transação. Em sistemas mais simples existe somente um mestre. Ou seja, todo o controle sobre o acesso ao barramento é controlado por este único mestre. Já em sistemas mais complexos, quando há mais de um mestre, é possível que dois ou mais mestres desejem iniciar uma transação ao mesmo

tempo. Isso pode gerar inconsistência já que os mestres poderiam sobrepor informações de outros mestres. Para solucionar este problema é preciso a utilização de um **protocolo de arbitragem** antes do uso do barramento. Este protocolo define qual mestre poderá desempenhar seu papel na próxima transação (Maccabe, 1993). Neste caso, somente um mestre poderá atuar por vez. Existem várias políticas que podem ser utilizadas no esquema de arbitragem. Política de prioridade, ordem de chegada e *round robin* são alguns exemplos. Esquemas de arbitragem podem ser centralizados ou distribuídos (Stallings, 2000), e podem ser divididos em quatro classes (Patterson e Hennessy, 1998):

***Daisy Chain*:** Este esquema usa a política de prioridade, onde a prioridade dos dispositivos é determinada pela posição em que ele se encontra no barramento. Os dispositivos de maior prioridade sempre receberão os sinais, enviados pelo barramento, antes dos dispositivos de baixa prioridade. Neste caso, caso mais de um dispositivo deseje usar o barramento ao mesmo tempo, o sinal de permissão de acesso ao mesmo (sinal *grant*), enviado pelo árbitro, sempre passará pelos dispositivos de maior prioridade primeiro. Com isso, eles podem interceptar o sinal, impedindo assim que os dispositivos de menor prioridade consigam acessar o barramento. Uma desvantagem desta abordagem é que os dispositivos com menor prioridade podem nunca conseguir acesso ao barramento. Tal abordagem pode ser visualizada na figura 4.3.

Centralizada, Paralela: Nesta abordagem existem várias linhas de requisição (daí o paralelismo) e um árbitro central. O árbitro recebe várias requisições e determina qual dispositivo terá acesso ao barramento. O dispositivo selecionado recebe uma notificação e então pode atuar como mestre do sistema. Uma desvantagem dessa abordagem é que o árbitro central pode tornar-se o gargalo do barramento. O barramento PCI (*Peripheral Component Interconnect*) utiliza esta abordagem (Shanley e Anderson, 1999).

Distribuído por auto-seleção: Neste esquema não há um árbitro centralizado, porém há a necessidade de utilizar-se várias linhas de requisição. Todo dispositivo que

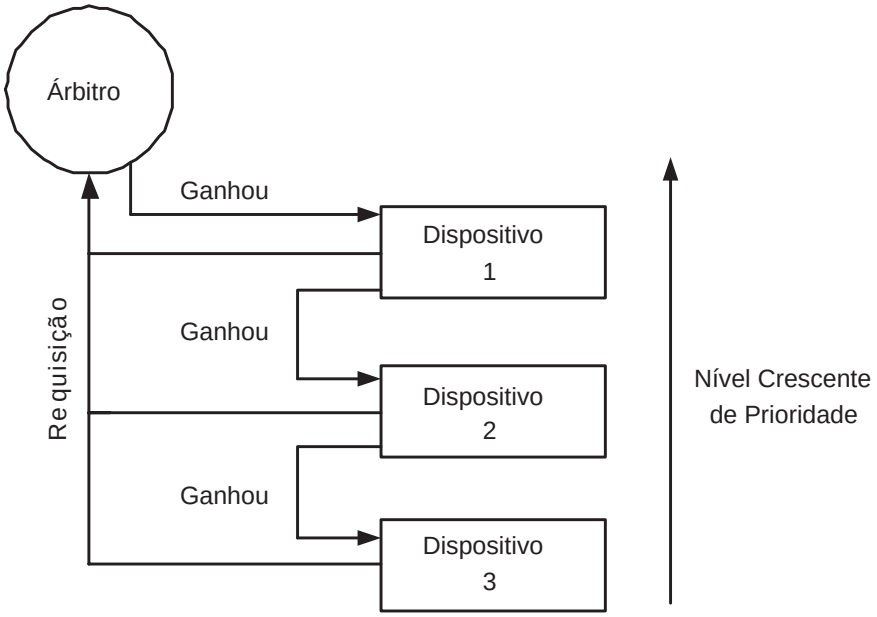

Figura 4.3: Arbitragem *Daisy Chain*.

desejar acessar o barramento põe seu código no mesmo. Com isso os dispositivos, através de uma observação do barramento, podem então determinar qual possui uma maior prioridade. Neste esquema cada dispositivo consegue determinar se o mesmo tem uma maior prioridade ou não.

Distribuído por detecção de colisão: Quando mais de um dispositivo tenta acessar o barramento há uma colisão. Tal esquema detecta a colisão e apenas um dos dispositivos que causou a colisão pode acessar o barramento. O barramento da rede *ethernet* trabalha deste modo (Spurgeon, 2000).

A escolha de uma dessas classes de esquema de arbitragem é determinada por vários fatores. Como, por exemplo, a capacidade de expansão do barramento e quão rápido deve ser a arbitragem. Um exemplo completo de um projeto de barramento proprietário pode ser encontrado em (Marques, 1988).

4.2 Barramentos On-Chip

A principal funcionalidade de um barramento on-chip é prover um meio de comunicação entre diversos *cores* contidos em um chip. Para isso é necessário que tais *cores* sigam

um mesmo padrão (protocolo). O barramento on-chip, então, serve como uma maneira de padronizar a comunicação entre tais *cores*. Sua funcionalidade principal é mostrada na figura 4.4. Onde vários módulos (contidos em um único chip) são ligados entre si através do barramento AMBA.

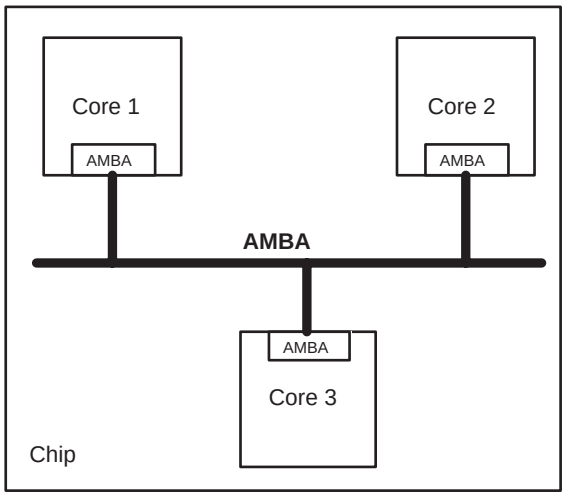

Figura 4.4: Barramento on-chip interconectando diversos *cores* em um mesmo chip.

Embora o barramento on-chip sirva, primariamente, para ligar diversos dispositivos que estejam em um mesmo chip, o protocolo do barramento também pode ser usado para ligar dispositivos que não se encontram necessariamente dentro do mesmo chip. Pode ligar, por exemplo, vários FPGAs de um sistema reconfigurável Multi-FPGA, como o apresentado na figura 1.1.

Existem vários barramentos on-chip, como pode ser observado na tabela 4.1. Destes barramentos, se destacaram para o projeto, pelo fato de serem gratuitos, o AMBA (*Advanced Microcontroller Bus Architecture*), CoreConnect e WISHBONE. Um outro barramento interessante é o barramento Avalon. Porém, sua licença restringe seu uso apenas nos dispositivos da Altera. Algumas considerações sobre os quatro barramentos citados anteriormente podem ser vistas a seguir.

AMBA: Barramento, da ARM, que pode ser dividido em três tipos: AHB (*Advanced High-performance Bus*), ASB (*Advanced System Bus*) e APB (*Advanced Peripheral Bus*). Ele foi escolhido para ser utilizado no projeto por possuir várias características interessantes e estar sendo usado por inúmeras empresas de hardware, tais como:

BARRAMENTO	FABRICANTE
AMBA	ARM
Atlantic	Altera
Avalon	Altera
CoreConnect	IBM
FISPbus	Mentor Graphics
FPI Bus	Infineon Technologies
SiliconBackplane	Sonics Inc
VSIA On-Chip Bus	Virtual Socket Interface Alliance
WISHBONE	OpenCores/Silicore

Tabela 4.1: Alguns barramentos on-chip (BOC).

Altera (Altera, 2004a), PACT (PACT, 2004), MIPS (MIPS, 2004), etc. Também vem sendo bastante utilizado no meio acadêmico. Mais especificamente na área de system-on-chip, existem três boas referências: (Becker e Vorbach, 2003), (Lee e Bergmann, 2003) e (Tsai et al., 2003). Além disso, vale destacar os trabalhos de (Kalte et al., 2002), (Hellmich et al., 2000) e (Moch et al., 2003). Mais detalhes sobre este barramento serão vistos no capítulo 5.

CoreConnect: Barramento desenvolvido pela IBM que, assim como o AMBA, está dividido em três tipos: PLB (*Processor Local Bus*), OPB (*On-Chip Peripheral Bus*) e DCR Bus (*Device Control Register*). A figura 4.5 mostra como a arquitetura CoreConnect pode ser usada para ligar diversos dispositivos em um SoC baseado em PowerPC 440. O barramento PLB é o barramento de alta velocidade, enquanto o OPB e DCR trabalham com dispositivos mais lentos (IBM, 1999).

WISHBONE: Barramento on-chip da Silicore que possui, entre outras, as seguintes características (SILICORE, 2001):

- Interface simples, compacta e lógica, que necessita de poucas portas lógicas;
- Conjunto completo de protocolos populares de transferência de dados através de barramento;
- Suporta diferentes métodos de interconexão, como: ponto-a-ponto, barramento compartilhado e *crossbar switch*;
- Suporte à múltiplos mestres;

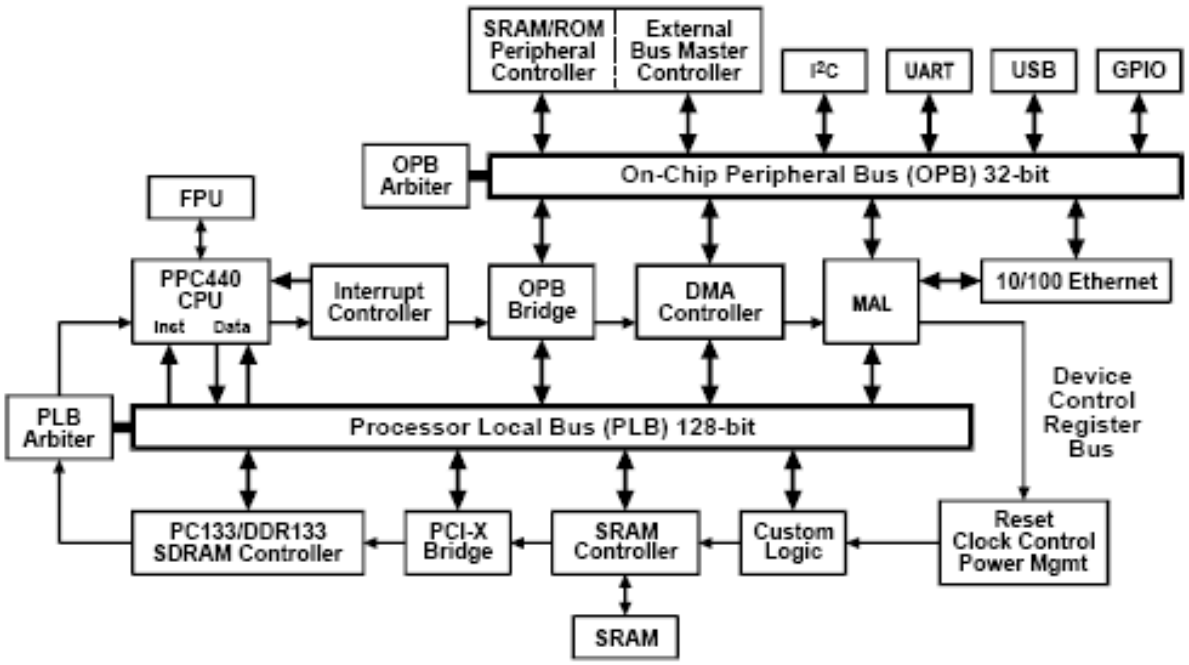

Figura 4.5: SoC baseado em CoreConnect (IBM, 1999).

- Metodologia de arbitragem definida pelo usuário (ex: por prioridade, *round-robin*, etc);
- Independente da tecnologia da hardware (FPGA, ASIC, etc).

Avalon: Mais um tipo de barramento on-chip, desenvolvido pela Altera, cuja única restrição de licença é a sua utilização em dispositivos da Altera. Algumas de suas características são citadas a seguir (Altera, 2003a):

- Simplicidade;
- Realização de operações síncronas;
- Suporte a múltiplos mestres;
- Largura de dados de 8, 16 e 32 bits;
- 32 bits para endereçamento;
- Linhas separadas de endereço, controle e dados.

A arbitragem da comunicação entre mestres e escravos é feita no lado do escravo (*slave-side*). Tal arbitragem determina qual mestre poderá se comunicar com o

escravo, caso mais de um mestre esteja querendo se comunicar com um mesmo dispositivo escravo. Essa abordagem de arbitragem oferece dois benefícios:

- Cada mestre realizará a comunicação com os dispositivos escravos como se fosse o único mestre;
- Múltiplos mestres podem realizar transações simultaneamente, caso não acessem o mesmo dispositivo escravo no mesmo ciclo de barramento.

4.3 Barramentos Tradicionais x Barramentos On-Chip

Barramentos tradicionais e on-chip possuem o mesmo objetivo, que é fazer com que diversos componentes possam se comunicar através de um meio compartilhado. Porém, o barramento on-chip é bem mais flexível que o tradicional.

Os barramentos on-chip podem usar diferentes tamanhos para endereço e dados. Isso é possível, pois as especificações geralmente fornecem mais de uma opção para tais tamanhos. Dessa forma cada implementação pode ser diferenciada da outra neste quesito. Já o barramento tradicional deve respeitar o tamanho padrão, pois isso limitará o número de linhas a serem construídas na placa onde o mesmo será implementado. Uma outra diferença é encontrada na decodificação de endereços, já que da maneira tradicional, usualmente ela é feita usando-se o endereço completo. Já no barramento on-chip a decodificação pode ser feita usando apenas parte do endereço, tornando mais rápida a decodificação dos mesmos. A velocidade dos barramentos on-chip também pode variar de acordo com sua implementação (pois não é definido dados sobre tempo em suas especificações), já nos tradicionais ela deve seguir rigorosamente a especificação.

Capítulo 5

O Barramento AMBA

Este capítulo entrará em detalhes sobre o barramento AMBA, o qual foi escolhido para ser implementado neste trabalho. Serão mostrados, também, alguns exemplos de aplicações (da área acadêmica e comercial) que utilizam o AMBA.

5.1 Introdução ao barramento AMBA

O AMBA (*Advanced Microcontroller Bus Architecture*) é um barramento on-chip de padrão aberto, desenvolvido pela ARM, que tem como principal funcionalidade realizar a interconexão de módulos/*cores* de um SoC (*System-on-Chip*) (ARM, 2003b).

Outras características do AMBA é que ele encoraja o desenvolvimento modular de sistemas e minimiza a utilização de silício necessária para suportar um SoC eficiente, além de ser independente de tecnologia (ARM, 1999). Isso garante uma grande flexibilidade ao projeto.

Um típico microcontrolador baseado em AMBA possui um barramento de sistema que funciona como um *backbone* que interliga os dispositivos que necessitam de alto desempenho. Alocado ao barramento principal (barramento de sistema) pode existir uma ponte

que faz a ligação com um outro barramento menos veloz, no qual podem estar ligados diversos periféricos que não necessitam de alto desempenho.

Existem quatro tipos diferentes de barramento AMBA. Três que podem ser usados como **barramentos de sistema** (AXI, AHB e ASB) e um que é utilizado como **barramento de periféricos** (APB). Este último, é ligado ao barramento principal (de sistema) através de uma ponte (*bridge*). A tabela 5.1 mostra quais são os tipos de barramentos mencionados acima.

BARRAMENTO AMBA	
Barramento de Sistema	Barramento de Periféricos
AMBA AXI AMBA AHB AMBA ASB	AMBA APB

Tabela 5.1: Tipos de barramento AMBA.

A figura 5.1 mostra um típico sistema AMBA que realiza a interconexão de vários dispositivos usando diferentes tipos de barramentos AMBA em conjunto.

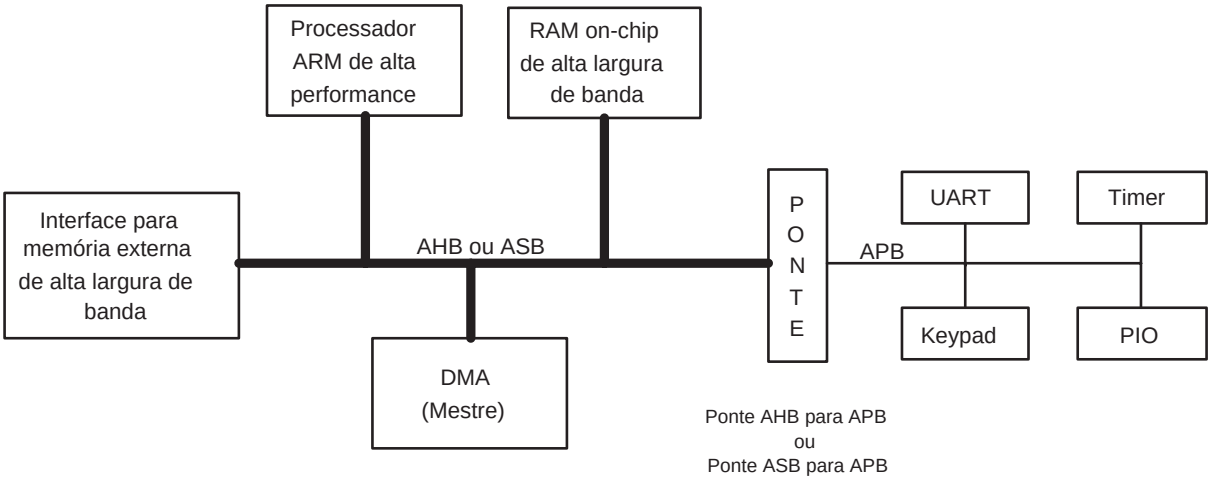

Figura 5.1: Um típico sistema AMBA (ARM, 1999).

5.2 AMBA AHB

O AHB (*Advanced High-performance Bus*) trata-se de uma geração recente de barramento AMBA que foi projetado para ligar dispositivos de alto desempenho e que utilizam

uma alta freqüência de *clock*. Atua como um *backbone* e suporta a conexão de processadores, memórias on-chip, interfaces de memória externa *off-chip*, etc.

AMBA AHB suporta múltiplos mestres, além de fornecer suporte a operações que necessitem de uma alta largura de banda. Ele possui os recursos necessários para sistemas de alto desempenho e de alta freqüência de *clock*, incluindo:

- Transferências intermitentes (*Burst Transfers*).
- Transações com divisão (*Split Transactions*).
- Operação com única borda de *clock*.
- Operações de transferência com a utilização de *pipeline*.
- Implementação *non-tristate*.
- Configuração com alta largura para o barramento de dados (8/16/32 até 1024 bits).

Tal barramento também pode ser ligado a outro barramento mais lento. Isso é feito, de forma bastante eficiente, através da utilização de uma ponte (*bridge*). Qualquer dispositivo do sistema pode ser incluído, como escravo, no AHB. Porém, dispositivos de baixa largura de banda devem ser ligados ao barramento APB.

5.2.1 AMBA AHB Muilt-Layer

AMBA AHB *Multi-Layer* é baseado no protocolo AHB e trata-se de um esquema de interconexão (através de uma matriz de interconexão) que possibilita o acesso paralelo entre múltiplos mestres e escravos (ARM, 2001b). Uma vantagem do AHB *Multi-Layer* é que ele é totalmente compatível com o AHB, pois nenhuma modificação precisa ser feita nos mestres e escravos de um sistema AHB para que ele funcione normalmente no AHB *Multi-Layer*. O esquema básico de interconexão entre mestres e escravos, de várias camadas, pode ser visto na figura 5.2.

Outro benefício propiciado pelo AHB *Multi-Layer* é o aumento da utilização da largura de banda do sistema, já que múltiplos mestres podem acessar diversos escravos ao mesmo

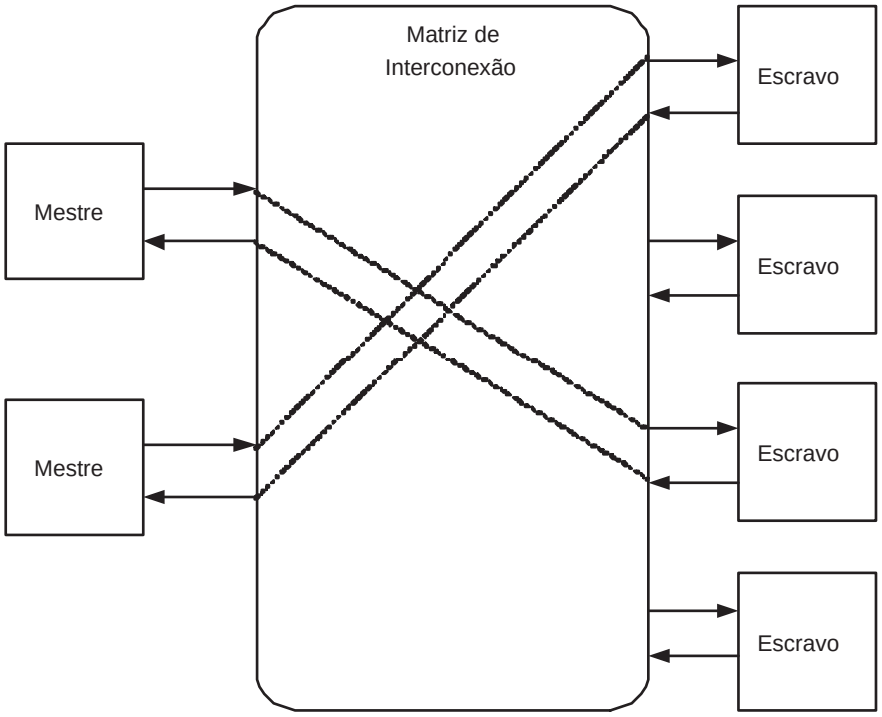

Figura 5.2: Esquema básico de interconexão *Multi-Layer* (ARM, 2001b).

tempo. A figura 5.3 mostra um exemplo mais detalhado onde dois mestre (em duas camadas) podem acessar dois escravos. Deve existir uma política de sincronização caso os dois mestres tentem acessar um mesmo escravo ao mesmo tempo. Tal política pode ser a *round-robin* ou baseada na prioridade das camadas, por exemplo. Neste exemplo, o decodificador é responsável pela localização dos dispositivos escravos e o multiplexador por conectar o múltiplos mestres aos dispositivos. Se cada camada do sistema *Multi-Layer* contiver somente um mestre, sua implementação torna-se muito simples, já que pode ser utilizado o protocolo AHB-*Lite* em cada camada.

5.2.2 AMBA AHB-Lite

AMBA AHB-*Lite* nada mais é do que um subconjunto da especificação do AMBA AHB Completo (*Full*) e é utilizado em sistemas que necessitem de somente um mestre. Isto é, pode ser usado quando existe um sistema com apenas um mestre ou em um sistema AHB *Multi-Layer*, onde cada camada possua apenas um mestre (ARM, 2001a). Outra característica do AHB-*Lite* é a ausência de um árbitro, pois existirá apenas um único mestre. A figura 5.4 mostra um diagrama de blocos de um sistema AHB-*Lite* com único

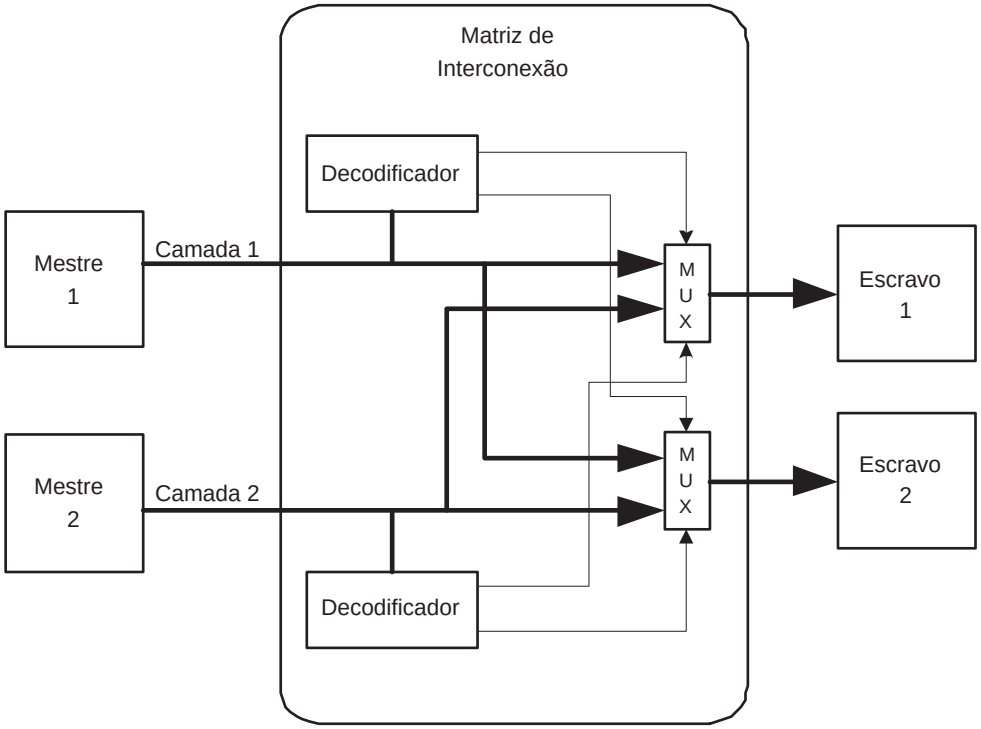

Figura 5.3: Sistema *Multi Layer* com dois mestres e dois escravos (ARM, 2001b).

mestre.

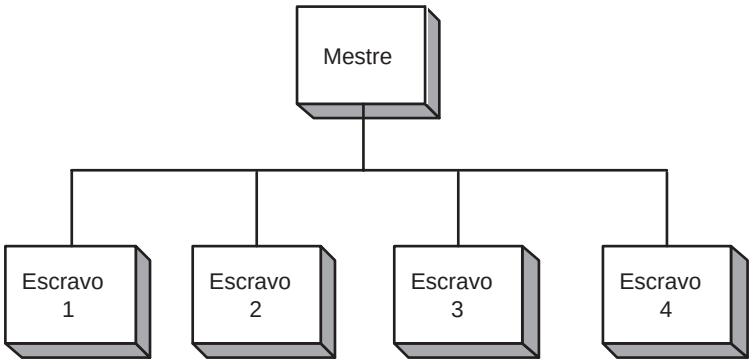

Figura 5.4: Sistema AHB-Lite com único mestre

AHB-*Lite* remove o protocolo necessário para suportar múltiplos mestres em um sistema AHB. Desta forma, sua implementação torna-se mais simples que o AHB *Full*. Cabe ressaltar, que qualquer mestre desenvolvido para sistemas AHB *Full* funcionará normalmente em sistema AHB-*Lite*. Também é possível migrar um mestre AHB-*Lite* para um AHB *Full*. Para isso, basta adicionar as funcionalidades restantes (aquelas que só são implementadas no AHB *Full*).

Os dispositivos escravos em sistemas AHB-*Lite* também são compatíveis com o AHB

Full. Porém, para que os escravos do AHB *Full* sejam compatíveis com o AHB-*Lite* é preciso fazer alguns reajustes (tirando algumas funcionalidades). Uma comparação de compatibilidade pode ser visualizada na tabela 5.2.

Componente	**Sistema AHB *Full***	**Sistema AHB-*Lite***
Mestre AHB *Full*	✓	✓
Mestre AHB-*Lite*	Necessita alteração	✓
Escravo AHB (sem todas as funcionalidades)	✓	✓
Escravo AHB (com todas as funcionalidades)	✓	Necessita alteração

Tabela 5.2: Compatibilidade entre AHB *Full* x AHB-*Lite* (ARM, 2001a).

5.3 AMBA ASB

O ASB (*Advanced System Bus*) foi a primeira geração de barramento de sistema AMBA (surgiu em 1995) e especifica algumas características necessárias à sistemas de alto desempenho, incluindo (ARM, 1999):

- Transferências intermitentes (*Burst Transfers*).
- Operações de transferência com a utilização de *pipeline*.
- Múltiplos mestres.

Como se pode notar, o ASB assemelha-se muito ao AHB, porém o AHB possui mais características úteis para sistemas de alto desempenho, motivo pelo qual foi escolhido para ser o barramento de sistema neste projeto.

5.4 AMBA APB

O APB (*Advanced Peripheral Bus*) é parte da hierarquia de barramentos AMBA e é otimizado para trabalhar com um mínimo consumo de energia, além de possuir uma reduzida complexidade de interface. Ele aparece como um barramento local secundário que está encapsulado como um simples dispositivo escravo AHB. APB fornece um prolongamento de baixa energia ao barramento de sistema. A ponte APB (APB *Bridge*) aparece

como um módulo escravo, ligado ao AHB, representando o barramento periférico local. Cabe ressaltar, que a ponte AHB/APB é o único mestre do barramento APB. AMBA APB deve ser utilizado para conectar qualquer periférico que não necessite de uma alta largura de banda e não precise de um alto desempenho. Os novos melhoramentos obtidos, em relação às suas versões anteriores, fazem com que os periféricos do APB possam ser mais facilmente integrados em qualquer projeto, com as seguintes vantagens:

- Melhoramento do desempenho em operações com freqüências maiores.
- A análise estática cronometrada é simplificada pelo uso de uma única borda de *clock*.
- Considerações especiais não são necessárias para inserção automática de teste.
- Muitas bibliotecas ASIC (*Application Specific Integrated Circuit*) têm uma melhor seleção de registros de subida de borda.
- Fácil integração com simuladores baseados em ciclos.

Estas características fazem a interface com o barramento AHB muito mais simples. Uma implementação AMBA APB, tipicamente, contém uma única ponte APB que é responsável por converter a transferências provenientes do AHB (ou ASB) em um formato confiável para os escravos localizados no APB.

No final de 2003 a ARM lançou a terceira versão do barramento APB, ao qual foi chamada de APB 3 (ARM, 2004). A principal diferença em relação a versão anterior foi a inclusão de dois sinais extras para tratamento de erro e inclusão de *wait states*. Isso permite uma melhor interface com o barramento AXI.

5.5 AMBA AXI

AXI (*Advanced eXtensible Interface*) é o nova geração de protocolo da ARM. Foi desenvolvido para sistemas que necessitam de alto desempenho e uma alta freqüência. Alguns das caracteríticas do AMBA AXI são (ARM, 2003a):

- Apropriado para projetos que necessitam de uma alta largura de banda e baixa latência;
- Torna possível operações de alta freqüência sem a utilização de pontes complexas;
- Provê flexibilidade na implementação de arquiteturas de interconexão;
- Compatível com o AHB e APB;
- Suporte a transferências fora de ordem.

Nota: O barramento AXI não será visto em detalhes, pois o trabalho está focado no AHB e APB, que já são bastante usados e testados.

5.6 Componentes AMBA AHB

Alguns componentes são necessários para que se possua um sistema AMBA, são eles: mestres, escravos, árbitros, decoder e multiplexadores. Os últimos são responsáveis por manter a correta comunicação entre mestres e escravos envolvidos em uma transferência de dados, através do compartilhamento do barramento. Os outros componentes serão vistos em mais detalhes a seguir.

O conjunto de todos os sinais referentes ao AHB pode ser visto na tabela 5.3. Para maiores detalhes sobre os sinais, consultar a especificação (ARM, 1999).

5.6.1 Mestres AHB

Os mestres são os responsáveis por iniciar toda e qualquer transferência no barramento. Eles inicialmente devem fazer uma requisição ao árbitro indicando que querem começar uma transferência, só após a confirmação do árbitro o mestre pode então realizá-la. Sua interface é mostrada na figura 5.5.

5.6.2 Escravos AHB

Os escravos não conseguem iniciar uma transferência de dados, eles são totalmente dependentes das ações dos mestres. Os mestres podem tanto requisitar dados para os

Nome	Fonte	Descrição
HCLK	Fonte do Clock	Clock (borda de subida)
HRESETn	Controlador de Reset	Reset (borda de decida)
HADDR[31..0]	Mestre	Endereço de 32 bits
HTRANS[1..0]	Mestre	Tipo de transferência: NONSEQ, SEQ, IDLE, BUSY
HWRITE	Mestre	Indica se a transferência é de leitura ou escrita
HSIZE[2..0]	Mestre	Tamanho da transferência
HBURST[2..0]	Mestre	Indica se a transferência faz parte de uma rajada
HPROT[3..0]	Mestre	Sinal de Proteção
HWDATA[31..0]	Mestre	Dados de escrita (dados do mestre para os escravos)
HSELx	Decoder	Sinal de seleção para o escravo correto. O 'x' indica o escravo.
HRDATA[31..0]	Escravo	Dados de leitura (dados dos escravos para o mestre)
HREADY	Escravo	Indica a finalização de uma transferência
HRESP[1..0]	Escravo	Tipo de resposta: OKAY, ERROR, RETRY, SPLIT
HBUSREQx	Mestre	Requisição para usar o barramento. O 'x' representa o mestre.
HLOCKx	Mestre	Indica que uma transferência não pode ser interrompida por outro mestre.
HGRANTx	Árbitro	Sinal que indica ao mestre que o mesmo pode usar o barramento.
HMASTER[3..0]	Árbitro	Sinal enviado ao escravos informando o mestre que está realizando a transferência.
HMASTLOCK	Árbitro	Indica ao escravo que o mestre atual está realizando uma transferência travada (locked)
HSPLITx	Escravo (com suporte SPLIT)	Sinal que indica que o mestre que recebeu um SPLIT pode voltar a usar o barramento.

Tabela 5.3: Sinais do AHB (ARM, 1999).

escravos como também podem enviá-los (essas operações são chamadas de leitura e escrita respectivamente). Sua interface é mostrada na figura 5.6.

5.6.3 Árbitro AHB

O árbitro é responsável por selecionar qual dos mestres pode utilizar o barramento, caso mais de um estejam fazendo requisições simultaneamente. Isso impede que dois mestres mudem os sinais do barramento ao mesmo tempo, evitando assim inconsistência no sistema. Sua interface é mostrada na figura 5.7.

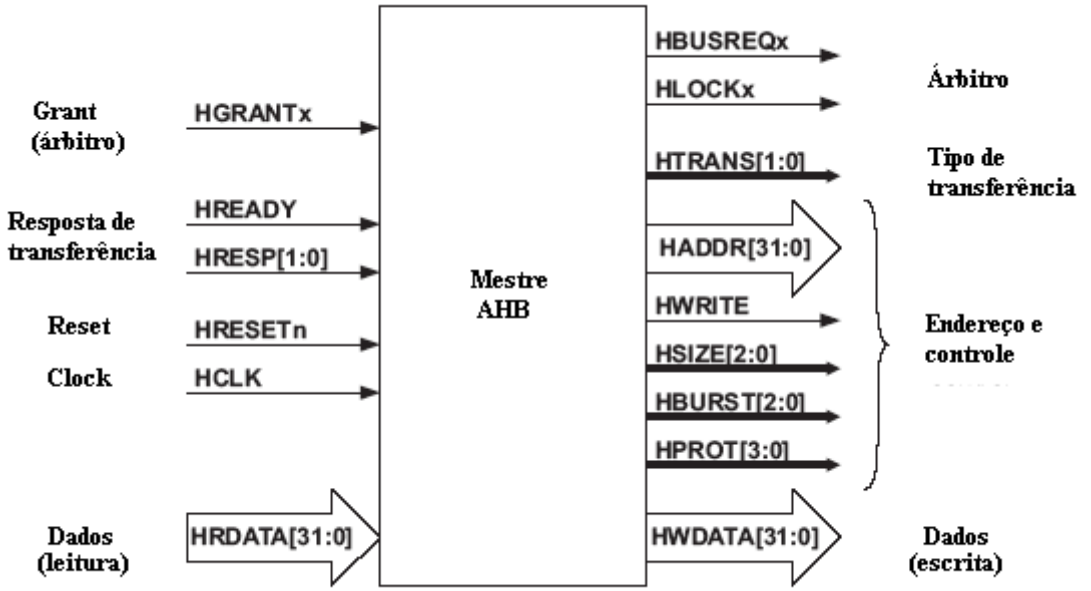

Figura 5.5: Interface do Mestre AHB (ARM, 1999)

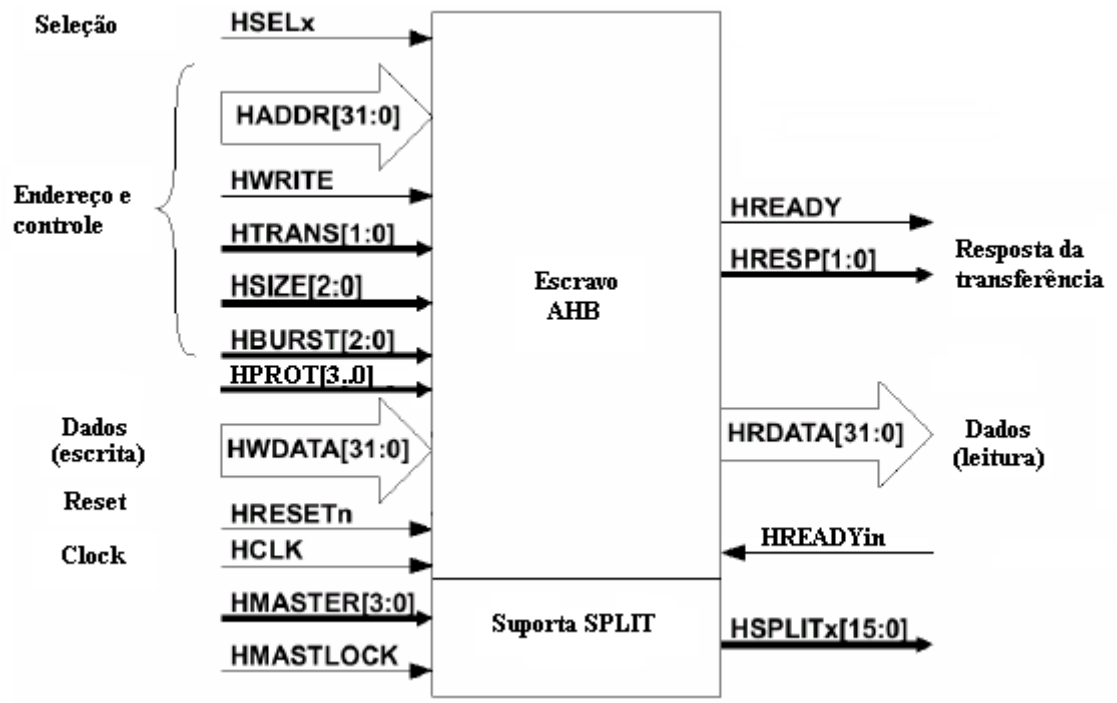

Figura 5.6: Interface do Escravo AHB (ARM, 1999)

5.6.4 Decoder AHB

O papel do *decoder* é selecionar o escravo correto de acordo com o endereço passado pelo mestre. Ele deve ser bem simples já que quanto menos tempo demorar para de-

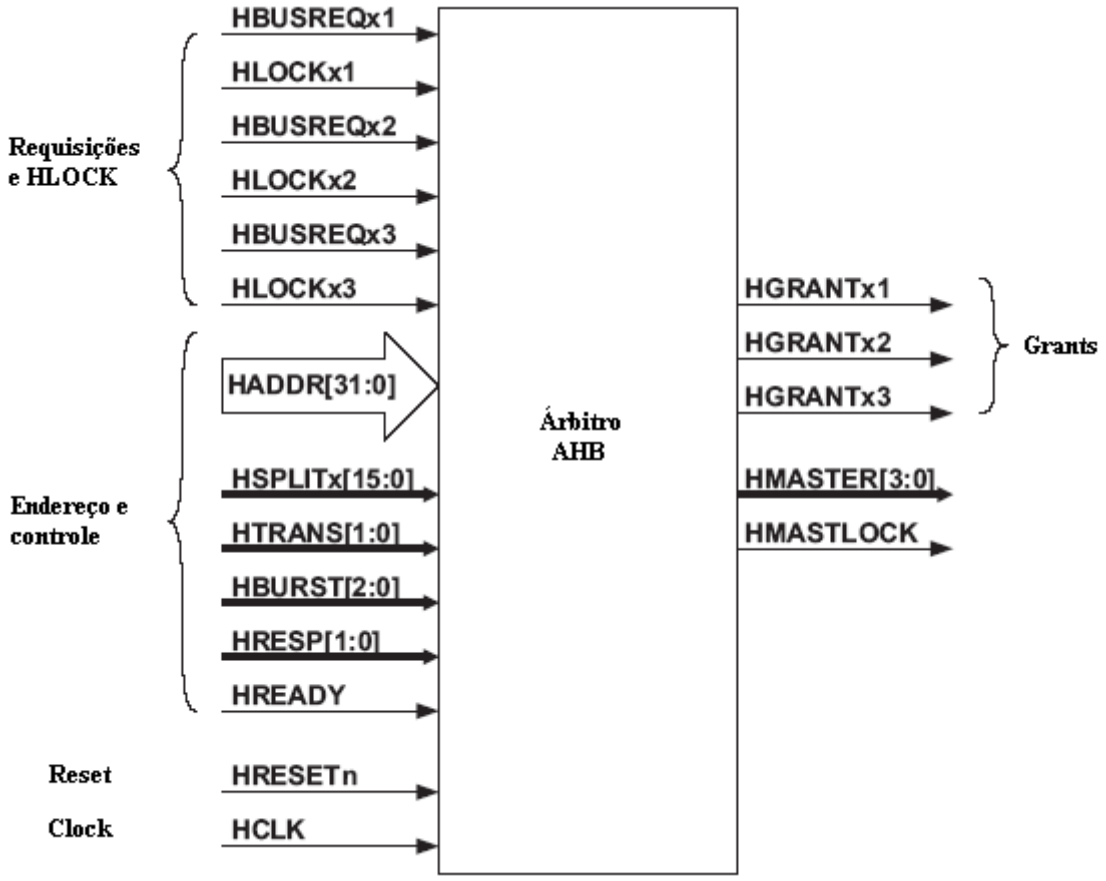

Figura 5.7: Interface do Árbitro AHB (ARM, 1999)

codificar o endereço, mais rápido será o funcionamento do sistema como um todo. Sua interface é mostrada na figura 5.8.

5.7 Componentes AMBA APB

O barramento APB é constituído de apenas um único mestre (ponte AHB/APB) e pode possuir vários escravos. Existe também um multiplexador que retorna os dados dos escravos para o mestre.

Todos os sinais utilizados no APB podem ser vistos na tabela 5.4. Para maiores detalhes sobre os sinais, consultar a especificação (ARM, 1999).

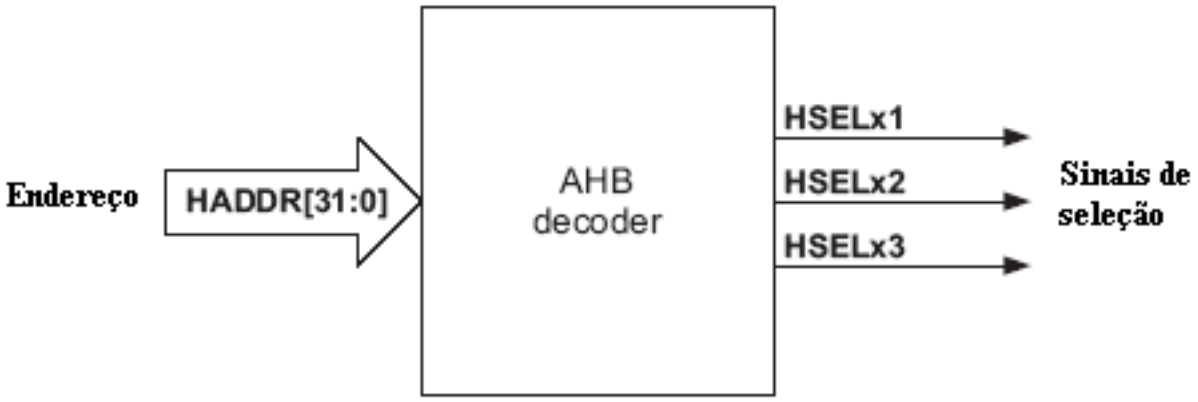

Figura 5.8: Interface do *Decoder* AHB (ARM, 1999)

Nome	Fonte	Descrição
PLCK	Fonte do Clock	Clock (borda de subida)
PRESETn	Controlador de Reset	Reset (Borda de decida)
PADDR[31..0]	Mestre	Endereço de 32 bits
PSELx	Mestre	Sinal de seleção para escravo. O 'x' indica o escravo.
PENABLE	Mestre	Sinal que habilita a operação de leitura ou escrita
PWRITE	Mestre	Indica operação de leitura ou escrita
PRDATA	Escravo	Dados de leitura (dos escravos para o mestre)
PWDATA	Mestre	Dados de escrita (do mestre para os escravos)
PREADY	Escravo (somente APB 3)	Indica o final de uma transferência
PSLVERR	Escravo (somente APB 3)	Indica que ocorreu algum erro

Tabela 5.4: Sinais do APB (ARM, 1999).

5.7.1 Mestre APB (Ponte AHB/APB)

O mestre APB é responsável por traduzir os sinais provenientes do barramento AHB em sinais do APB. Como só existem um mestre no barramento, não é necessário nenhum esquema de arbitragem. Sua interface é mostrada na figura 5.9.

5.7.2 Escravos APB

Os escravos APB, assim como os escravos AHB, não conseguem iniciar nenhum tipo de transferência. Eles somente realizam escritas dos dados provenientes do mestre ou lêem dados e retornam para o mestre (operações de escrita e leitura respectivamente). Sua interface é mostrada na figura 5.10.

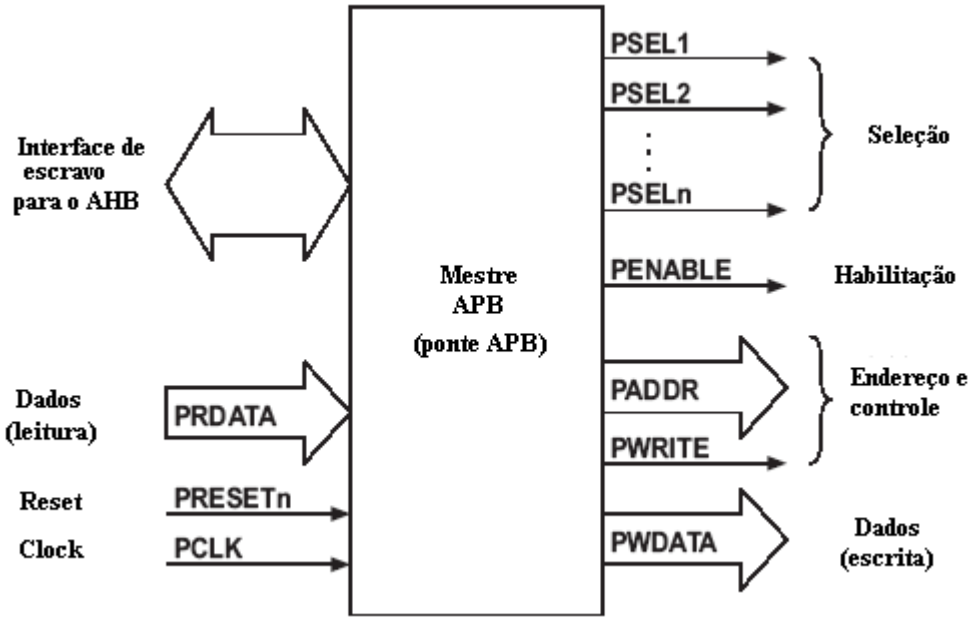

Figura 5.9: Interface do Mestre APB (ponte AHB/APB) (ARM, 1999)

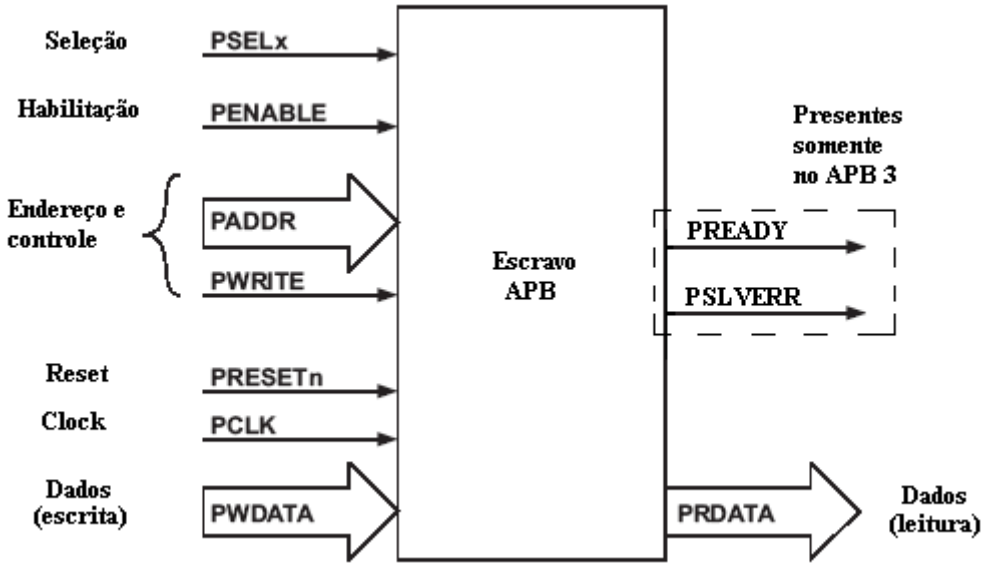

Figura 5.10: Interface do Escravo APB (ARM, 1999)

5.8 Exemplos de aplicações que utilizam o barramento AMBA

Como já citado anteriormente, existem várias empresas e grupos da área acadêmica trabalhando com o barramento AMBA em seus projetos de *hardware*. Serão apresentados, a seguir, alguns exemplos de aplicações que utilizam o protocolo AMBA.

5.8.1 SMeXPP - Smart Media Processor

SMeXPP é um SoC integrado, feito pela PACT, que visa permitir a utilização de multimídia em dispositivos *hand-held* de baixo custo, utilizando o mínimo de bateria possível. Tal sistema possui diversas aplicações que podem ser vistas na lista a seguir:

- Codificador de vídeo para vídeo-fones;
- *Codec* MPEG4;
- Codificador/Decodificador de imagem;
- Filtro de Imagem e melhoria do *display*;
- Aceleração na renderização de jogos 2D e 3D.

A arquitetura do dispositivo possui um microcontrolador ARM7EJ-S com periféricos e o *core* de um processador XPP reconfigurável com *interfaces* de alta velocidade. O SMeXPP pode atuar tanto como processador ou coprocessador. Tal sistema possui 4 camadas do barramento AMBA AHB para conectar os dispositivos que precisam de um maior desempenho e também implementa o barramento APB para os periféricos de baixa velocidade. A figura 5.11 ilustra a arquitetura do sistema.

Algumas das características da arquitetura são apresentadas na lista a seguir:

- Microprocessador 32 bits ARM7EJ-S;
- Acelerador de mídia 16-bit XPP;
- Três 64k x 32 SRAM;

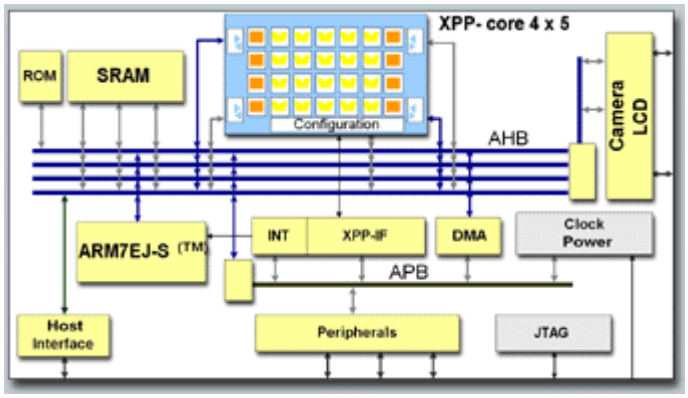

Figura 5.11: Sistema SMeXPP usando o barramento AMBA (PACT, 2003)

- Controladores DMA;
- *Interface* para câmera e *displays* coloridos LCD;
- Periféricos como: I2C, GPIO, *interfaces* seriais e *timers*;
- Domínios separados de *clocks* programáveis para ARM7 e XPP-*core*;
- *Interface* para troca de dados, de maneira paralela, em alta velocidade;
- *Interface* JTAG;
- Controlador de força (para gerenciamento da energia).

O SMeXPP é uma solução de baixo custo, que utiliza pouca energia e que é bastante flexível, pois se trata de um sistema completamente programável.

5.8.2 Processador Samsung S3C2410

S3C2410 é um microprocessador RISC 16/32 bits, desenvolvido pela Samsung, baseado no *core* do processador ARM920T, da ARM. Ele adota o barramento AMBA para realizar a comunicação interna entre alguns de seus componentes. Tem como objetivo prover uma solução de baixo custo, que utilize pouca energia e possua um alto desempenho. Foi desenvolvido para utilização em dispositivos *hand-held* e aplicações móveis em geral.

Algumas das características e componentes do S3C2410 podem ser observados na lista a seguir:

- *Caches* de dados e instruções separadas (16KB cada);
- MMU para tratar o gerenciamento de memória virtual;
- Controlador LCD;
- 3 canais UART com *handshake*;
- 4 canais DMA;
- 4 canais *timers* com PWM;
- Portas de E/S;
- RTC;
- 8 canais ADC de 10 bits e uma *interface touch-screen*;
- 2 canais SPI e PLL para geração de *clock*.

Mais detalhes sobre o S3C2410 podem ser vistos na figura 5.12.

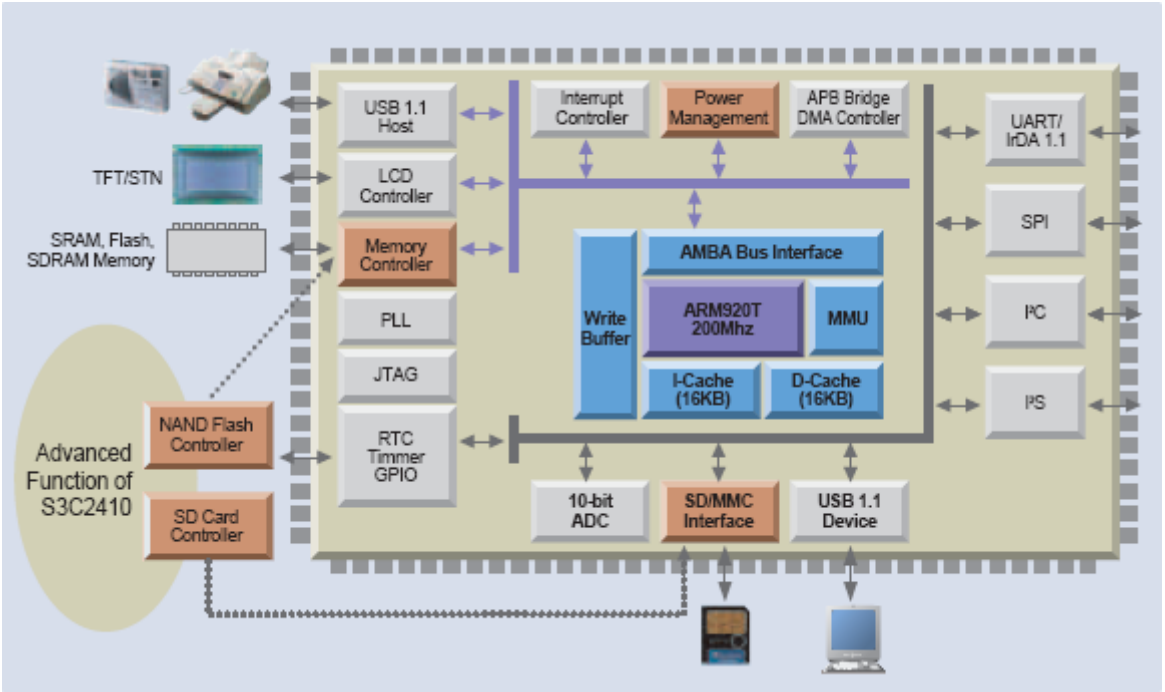

Figura 5.12: Diagrama de blocos do processador S3C2410 (Samsung, 2003)

5.8.3 HiBRID-SOC: Arquitetura Multi-core para aplicações de Imagem e Vídeo

HiBRID-SOC trata-se de uma arquitetura multi-*core*, desenvolvida na Universidade de Hannover - na Alemanha, que visa o tratamento de vídeos e imagens. Tal arquitetura combina flexibilidade (devido a seus *cores* programáveis) com um alto poder de processamento de dados multimídia.

A arquitetura multi-*core* HiBRID-SOC, mostrada na figura 5.13, é formada basicamente pelos seguintes itens:

- Três *cores* programáveis (HiPAR-DSP, *Macroblock Processor* e *Stream Processor*);
- Vários módulos de memória on-chip;
- Barramento AMBA AHB 64-Bits;
- Diversas *Interfaces* para comunicação externa.

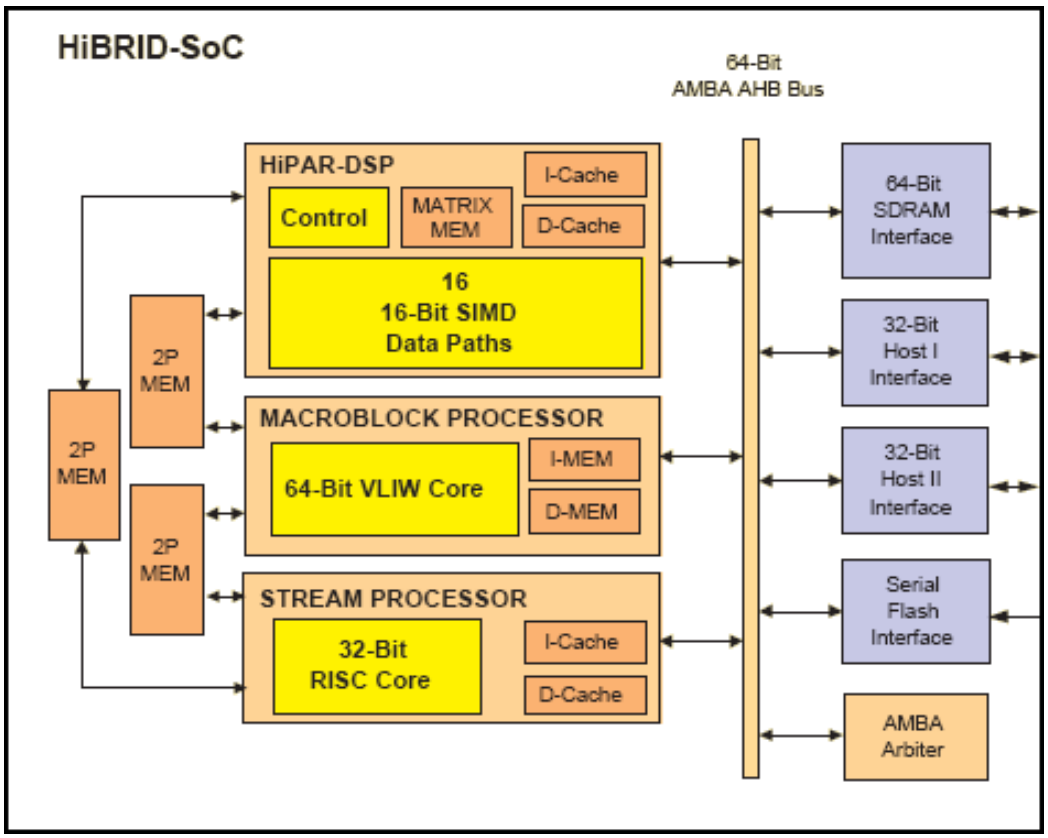

Figura 5.13: Arquitetura Multi-*core* HiBRID-SOC (Moch et al., 2003)

O AMBA AHB (64-Bits) é utilizado como **barramento de sistema** para esta arquitetura e conecta todos os *cores* (mestres) com as *interfaces* externas (escravos). A velocidade do barramento é a mesma atribuídas aos *cores* e *interfaces* (145 MHz).

Tal arquitetura provê uma eficiente solução SoC para vários tipos de aplicações de processamento de imagens e vídeos, pois como seus *cores* programáveis podem se adaptar a diferentes classes de algoritmos, diferentes aplicações podem ser mapeadas dentro desta arquitetura.

Capítulo 6

Implementação e Resultados

Este capítulo mostra os principais detalhes da implementação, dos testes e dos sistemas de validação que foram criados. Apresenta toda a organização e uma pequena descrição dos arquivos do projeto e também as instruções para utilização e configuração do barramento. Algumas informações sobre velocidade e área ocupada no chip, que foi utilizado, também são vistas neste capítulo.

6.1 Considerações Iniciais

O trabalho foi iniciado com um estudo detalhado da especificação do barramento AMBA (ARM, 1999). Foram observadas todas as interfaces mestres e escravas dos barramentos AHB e APB, assim como todos os detalhes sobre o comportamento dos mesmos. O ASB não foi estudado pois não foi utilizado neste projeto. Houve também um estudo sobre o LEON2 (Gaisler, 2003) que possui aproximadamente 67000 linhas de código-fonte e, como citado anteriormente, serviu como base para a implementação feita neste trabalho. LEON2 trata-se de um processador RISC (*Reduced Instruction Set Computing*) 32 bits de código aberto, compatível com a arquitetura SPARC V8, que foi desenvolvido

pela *Gaisler Research*. A arquitetura do processador pode ser vista na figura 6.1.

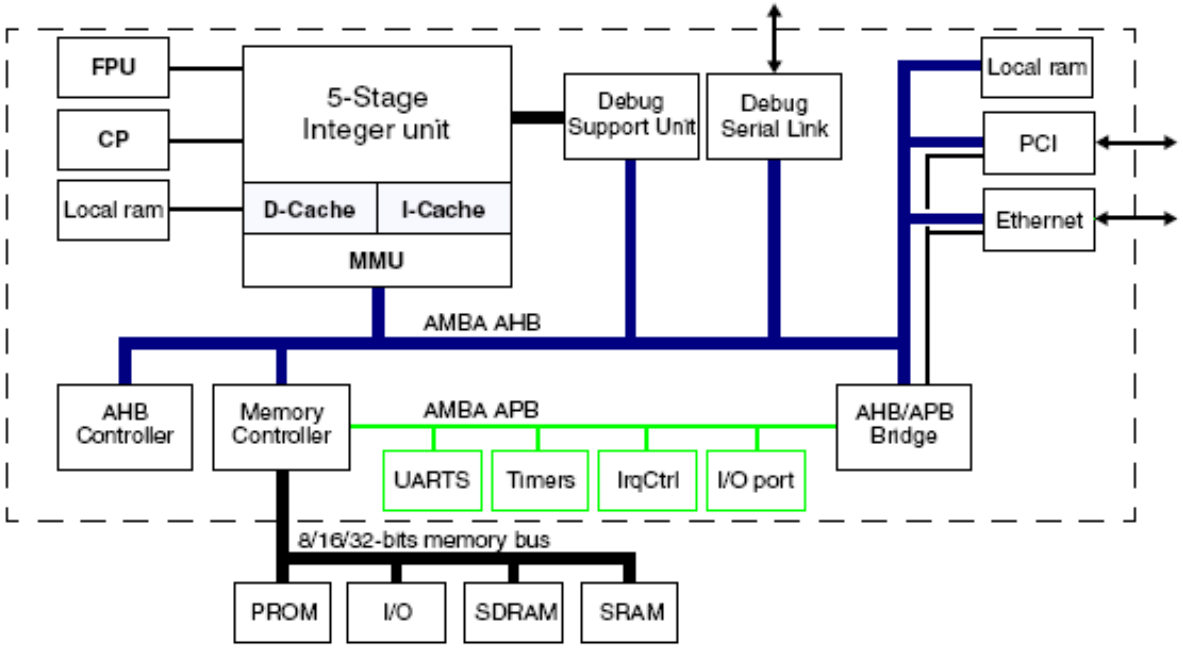

Figura 6.1: Diagrama de blocos do processador LEON2 (Gaisler, 2003).

Durante a implementação do projeto, foi lançado o LEON3 (Research, 2004). Porém, o mesmo não foi utilizado como base para o trabalho, pois o mesmo já encontrava-se bem adiantado. Além disso, os dois processadores implementaram a mesma versão do barramento AMBA.

O LCR (Laboratório de Computação Reconfigurável) possui convênio universitário com a empresa Altera. Portanto, todo o desenvolvimento e demais etapas do projeto (testes, simulação, depuração, etc) foram realizadas utilizando as ferramentas EDAs da Altera, como: Quartus II (Altera, 2004c), SoPC Builder (Altera, 2004d) e Nios II IDE (*Integrated Development Environment*) (Altera, 2004b). Alguns testes foram realizados utilizando o kit de desenvolvimento Nios - Edição Stratix (ver apêndice A). A linguagem de descrição de hardware (HDL) utilizada foi a VHDL. E todo o ambiente de desenvolvimento rodava na plataforma Windows 2000 Server, da Microsoft.

6.2 Organização e descrição dos módulos

Primeiramente foram identificados os módulos referentes ao barramento pertencentes ao processador LEON2. Como o mesmo possui cerca de 67000 linhas de código-fonte, foi necessário realizar um estudo básico sobre como estava organizada a arquitetura do processador. Depois de identificados os módulos que tratavam do barramento AMBA, foi feita uma organização e centralização de tudo que dizia respeito ao tratamento do protocolo AMBA. Então, foi criada uma estrutura de arquivos (em VHDL), cuja organização e descrição estão mostradas na tabela 6.1. Esses arquivos implementam todo o barramento AHB, APB e APB3.

Arquivo VHDL (.vdh)	**Descrição**
AHB_Types_Const	Declaração de tipos e constantes utilizadas no barramento AHB.
APB_Types_Const	Declaração de tipos e constantes utilizadas no barramento APB.
APB3_Types_Const	Declaração de tipos e constantes utilizadas no barramento APB3.
AHB_Components	Declaração de todos os componentes que serão ligados ao barramento AHB.
APB_Components	Declaração de todos os componentes que serão ligados ao barramento APB.
APB3_Components	Declaração de todos os componentes que serão ligados ao barramento APB3.
APB_Master	Ponte AHB/APB. É o único mestre do barramento APB.
APB3_Master	Ponte AHB/APB3. É o único mestre do barramento APB3.
AHB_Arbiter	Árbitro do sistema AHB. Responsável também por realizar a decodificação dos endereços e gerar os sinais de *select* para os escravos corretos. Possui também os multiplexadores responsáveis por multiplexar os sinais dos mestres para o escravo e dos escravos para o mestre. É o principal módulo do AMBA.
APB_System	Entidade onde é feita a ligação de todos os componentes utilizados no barramento APB.
APB3_System	Entidade onde é feita a ligação de todos os componentes utilizados no barramento APB3.
AMBA_System	Entidade onde é feita a ligação de todos os componentes utilizados no barramento AMBA (incluindo todos os componentes do APB e APB3).

Tabela 6.1: Organização e descrição dos arquivos VHDL referentes ao barramento AMBA

Foi implementado no trabalho todo o barramento AHB, APB e a versão mais nova do APB (versão 3), não contida no LEON2. Também, neste mestrado, foram feitas várias alterações no código do processador LEON2 com o intuito de tornar a implementação mais completa, configurável e otimizada. Para isso, foram definidos mais alguns tipos

de dados, constantes, além da utilização de registradores extras para aumentar o desempenho do barramento. Foram incluídas também algumas novas funcionalidades como o tratamento de *split* em transferências travadas (*Locked Transfer*) e a possibilidade de escolher a política de arbitragem a ser utilizada pelo árbitro (foram implementadas duas políticas distintas, a *round-robin* e por prioridade - maior índice). Modificações na ponte AHB/APB também foram feitas com a finalidade de diminuir a quantidade de *wait states* utilizadas na implementação do LEON2.

Os componentes básicos utilizados nos módulos da tabela 6.1 podem ser vistos na figura 6.2 (não mostra detalhes dos sinais AHB e APB). O módulo do árbitro (AHB_Arbiter.vhd), além de realizar a arbitragem, implementa também os multiplexadores, *decoder* e escravo padrão (tal escravo é acionado sempre que um mestre tentar acessar um endereço que não existe). Toda a ação do barramento AHB é centralizada no árbitro, onde são ligados todos os mestres e escravos do sistema. Já as pontes AHB/APB (APB_Master.vhd) e AHB/APB3 (APB3_Master.vhd) são os únicos mestres do APB e APB3, respectivamente. Eles também são responsáveis por multiplexar os dados de seus escravos para o mestre. A funcionalidade de tais pontes é traduzir os sinais AHB em sinais APB.

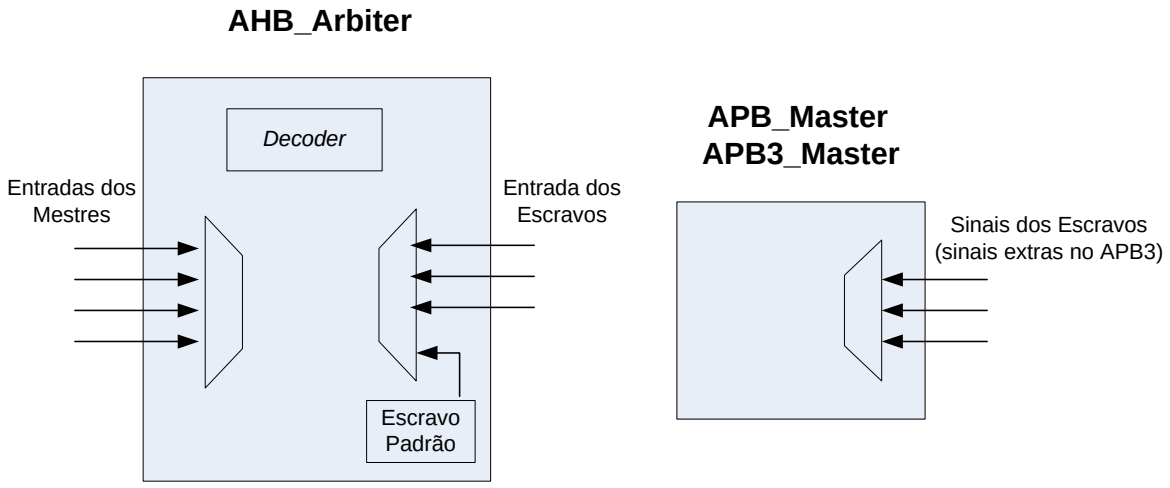

Figura 6.2: Componentes básicos do AHB_Arbiter, APB_Master e APB3_Master

Todos os mestres e escravos AHB devem ser ligados diretamente ao árbitro (AHB_Arbiter), inclusive as pontes APB e APB3. Já os escravos APB e APB3 devem ser ligados direto nas pontes. Um esquema da ligação de mestres (no máximo 16) e escravos ao árbitro e

pontes pode ser visto na figura 6.3. A quantidade de mestres é limitada pelo barramento. Ao total pode-se ter até 16 mestres AHB. Isso é devido ao sinal HMASTER só possuir 4 bits para representar os 2^4 mestres. Já a quantidade de escravos AHB e APB não possui nenhuma restrição de limite, só no que se diz respeito a faixa de endereçamento (de 32 bits). Na figura 6.3 a quantidade de escravos AHB vai até N-1 pelo fato do escravo padrão ser implementado dentro do AHB_Arbiter.

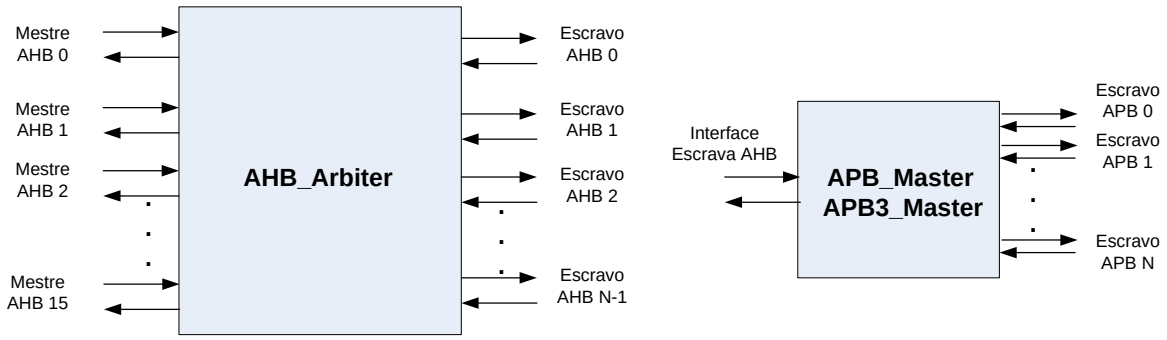

Figura 6.3: Ligação de mestres e escravos ao barramento

Nos módulos que apresentam os tipos e constantes usadas no barramento, foram definidos registros (*records*) que englobam todos os sinais de entrada e saídas de mestres e escravos AHB, APB e APB3. Detalhes sobre tais registros podem ser vistos na tabela 6.2. Como sistemas reais utilizam vários mestres e escravos, foram criados também vetores, cujos elementos são dos tipos citados anteriormente. Cada mestre e escravo estará contido no vetor apropriado.

Registro	**Descrição**
AHB_Mst_In_Type	Sinais de entrada do mestre AHB
AHB_Mst_Out_Type	Sinais de saída do mestre AHB
AHB_Slv_In_Type	Sinais de entrada do escravo AHB
AHB_Slv_Out_Type	Sinais de saída do escravo AHB
APB_Slv_In_Type	Sinais de entrada do escravo APB
APB_Slv_Out_Type	Sinais de saída do escravo APB
APB3_Slv_In_Type	Sinais de entrada do escravo APB3
APB3_Slv_Out_Type	Sinais de saída do escravo APB3

Tabela 6.2: Registros dos sinais de entrada e saída dos mestres e escravos do barramento

O código 6.1 mostra um exemplo de registros (*records*) de entrada e saída de mestres AHB. Registros são utilizados para facilitar a conexão de mestres e escravos ao barra-

mento, já que poupa trabalho de ter que ligar todos os sinais. Dessa forma, basta apenas ligar os registros de entrada e saída dos mestres. Um exemplo que mostra a utilidade do uso desses registros pode ser visualizado na figura 6.4.

```
-- Entradas do mestre AHB (HCLK e HRESETn estão separados)
type AHB_Mst_In_Type is
  record
    HGRANT:     Std_ULogic;  -- indica se o árbitro selecionou o escravo
    HREADY:     Std_ULogic;  -- indica a possibilidade de realizar uma transferência
    HRESP:      Std_Logic_Vector(1       downto 0); -- tipo de resposta
    HRDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- dado de leitura
  end record;

-- Saídas do mestre AHB
type AHB_Mst_Out_Type is
  record
    HBUSREQ:    Std_ULogic;                         -- requisição do barramento
    HLOCK:      Std_ULogic;                         -- transferência travada
    HTRANS:     Std_Logic_Vector(1       downto 0); -- tipo de transferência
    HADDR:      Std_Logic_Vector(HAMAX-1 downto 0); -- endereço
    HWRITE:     Std_ULogic;                         -- leitura/escrita
    HSIZE:      Std_Logic_Vector(2       downto 0); -- tamanho da transferência
    HBURST:     Std_Logic_Vector(2       downto 0); -- tipo de burst
    HPROT:      Std_Logic_Vector(3       downto 0); -- proteção
    HWDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- dado de escrita
  end record;
```

Código-fonte 6.1: Registros com sinais de entrada e saída de mestres AHB

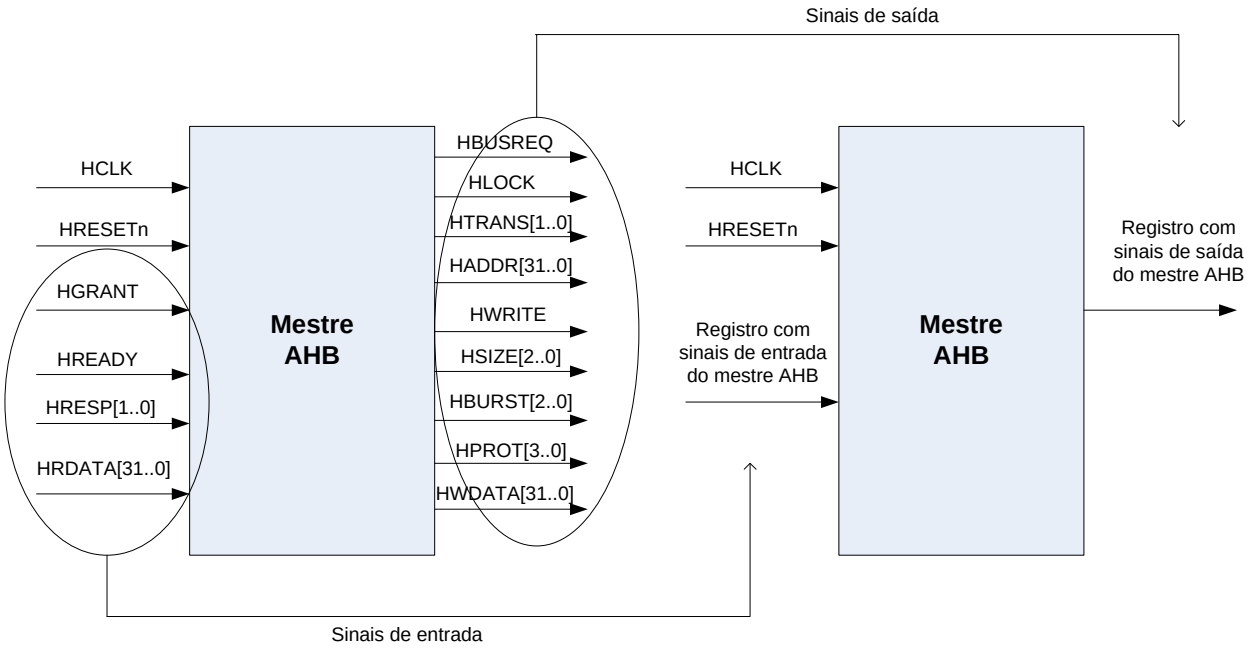

Figura 6.4: Utilização dos registros

Embora a utilização dos registros facilite a conexão dos mestres e escravos, a mesma possui uma desvantagem, pois não é possível representar estrutura do tipo *record* no modo esquemático da ferramenta de desenvolvimento Quartus II. Isso impossibilita a inclusão de mestres e escravos ao barramento utilizando uma interface gráfica. Dessa forma, todos os componentes a serem incluídos no barramento devem ser feitos através do código-fonte em VHDL.

6.3 Configuração do barramento AMBA

Várias mudanças podem ser feitas no barramento com a finalidade de configurar o mesmo para ser utilizado de acordo com as necessidades do usuário. Algumas modificações que podem ser realizadas dizem respeito aos itens citados a seguir:

- Largura do barramento de dados
- Número total de mestres e escravos (AHB e APB)
- Tipo de arbitragem a ser usada
- Escolha do barramento de periféricos (APB, APB3 ou ambos)

6.3.1 Largura do barramento de dados

A largura do barramento de dados do AHB pode variar de 32 a 1024 bits, já a do APB/APB3 varia de 8 a 32 bits. Para alterar a configuração do AHB, basta modificar o valor da constante `HDMAX`, contida no arquivo `AHB_Types_Const.vhd`. As configurações do APB e AP3 podem ser feitas alterando-se as constantes `PDMAX` e `P3DMAX`, contidas nos arquivos `APB_Types_Const.vhd` e `APB3_Types_Const.vhd`, respectivamente.

6.3.2 Número total de mestres e escravos

O número total de mestres AHB pode ser configurado modificando-se a constante `AHB_MST_MAX`, contida no arquivo `AHB_Types_Const.vhd`. Já o número de mestres APB e

APB3 não podem ser alterados, pois ambos os barramentos só devem possuir um mestre (a ponte).

Para modificar o número de escravos do AHB deve-se alterar a constante `AHB_SLV_MAX`, contida do arquivo `AHB_Types_Const.vhd`. Já para alterar o número de escravos do APB e APB3 é necessário modificar as constantes `APB_SLV_MAX` e `APB3_SLV_MAX`, contidas nos arquivos `APB_Types_Const.vhd` e `APB3_Types_Const.vhd`, respectivamente. É importante ressaltar que o barramento AHB sempre possuirá, no mínimo, um escravo, já que o escravo padrão é implementado no módulo do árbitro (`AHB_Arbiter.vhd`).

6.3.3 Tipo de arbitragem

A implementação permite a inclusão de vários tipos de arbitragem, ficando a critério do usuário, escolher a política mais adequada. Pode-se alterar o tipo de arbitragem modificando-se a constante `AHB_ARB_NUM`, contida no arquivo `AHB_Types_Const.vhd`. Porém, o código da nova política de arbitragem deve ser incluído no arquivo `AHB_Arbiter.vhd`. Um exemplo de utilização de duas políticas de arbitragem pode ser visto no código 6.2. A inclusão de uma terceira política é mostrada no código 6.3.

```
if arbitration = 0 then -- arbitragem por prioridade
  ... código da arbitragem por prioridade ...
elsif arbitration = 1 then -- arbitragem round-robin
  ... código da arbitragem round-robin ...
end if;
```

Código-fonte 6.2: Duas políticas de arbitragem

```
if arbitration = 0 then -- arbitragem por prioridade
  ... código da arbitragem por prioridade ...
elsif arbitration = 1 then -- arbitragem round-robin
  ... código da arbitragem round-robin ...
  elsif arbitration = 2 then -- nova política de arbitrage
    ... código da nova política de arbitragem ...
end if;
```

Código-fonte 6.3: Inclusão de uma nova política de arbitragem

É claro que para se incluir uma nova política de arbitragem é necessário realizar uma análise da implementação do módulo do árbitro (AHB_Arbiter). Já que é preciso utilizar os recursos disponíveis no módulo (como registradores, por exemplo).

6.3.4 Escolha do tipo de barramento de periféricos

Caso seja necessário o uso de um barramento de periféricos, pode-se optar pela inclusão do APB ou APB3. Também é possível utilizar os dois ao mesmo tempo. A inclusão dos barramentos de periféricos (`APB_System` e `APB3_System`) é feita de maneira similar a inclusão de escravos ao barramento AHB. Isso ocorre pelo fato dos barramentos de periféricos serem vistos, pelo sistema AHB, como simples escravos. Detalhes sobre a inclusão de escravos AHB serão vistos na próxima seção.

6.4 Inclusão/Remoção de Mestres e Escravos

Para adicionar qualquer componente ao sistema, deve-se, inicialmente, criar um projeto e adicionar os arquivos descritos na tabela 6.1. Depois, os componentes devem ser adicionados ao projeto e dependendo de quais barramentos serão usados, deve-se incluir a declaração de tais componentes nos arquivos `AHB_Components.vhd`, `APB_Components.vhd` ou `APB3_Components.vhd`. Logo após, os componentes devem ser ligados aos barramentos por meio dos arquivos `AMBA_System.vhd`, `APB_System.vhd` e `APB3_System.vhd`. Toda a ligação é feita de maneira estrutural (através do uso de componentes), usando-se o comando `PORT MAP`. A figura 6.5 mostra uma idéia de onde os componentes devem ser inseridos.

As instruções anteriores servem para a inclusão de qualquer componente ao sistema. Porém, há algumas particularidades na inclusão de mestres e escravos dos barramentos de sistema e periféricos. A seguir serão mostradas as diferenças de tais barramentos e também algumas máquinas de estados finitos para o auxílio no desenvolvimento de mestres e escravos de acordo com o protocolo AMBA. Tais máquinas foram construídas usando o software da ALDEC chamado Active-HDL (Aldec, 2005). Essa ferramenta permite a criação de máquinas de estados e a transformação delas em código VHDL, gerando assim *templates* para mestres e escravos, baseados nos estados da máquina. Isso auxilia bastante projetistas leigos em AMBA.

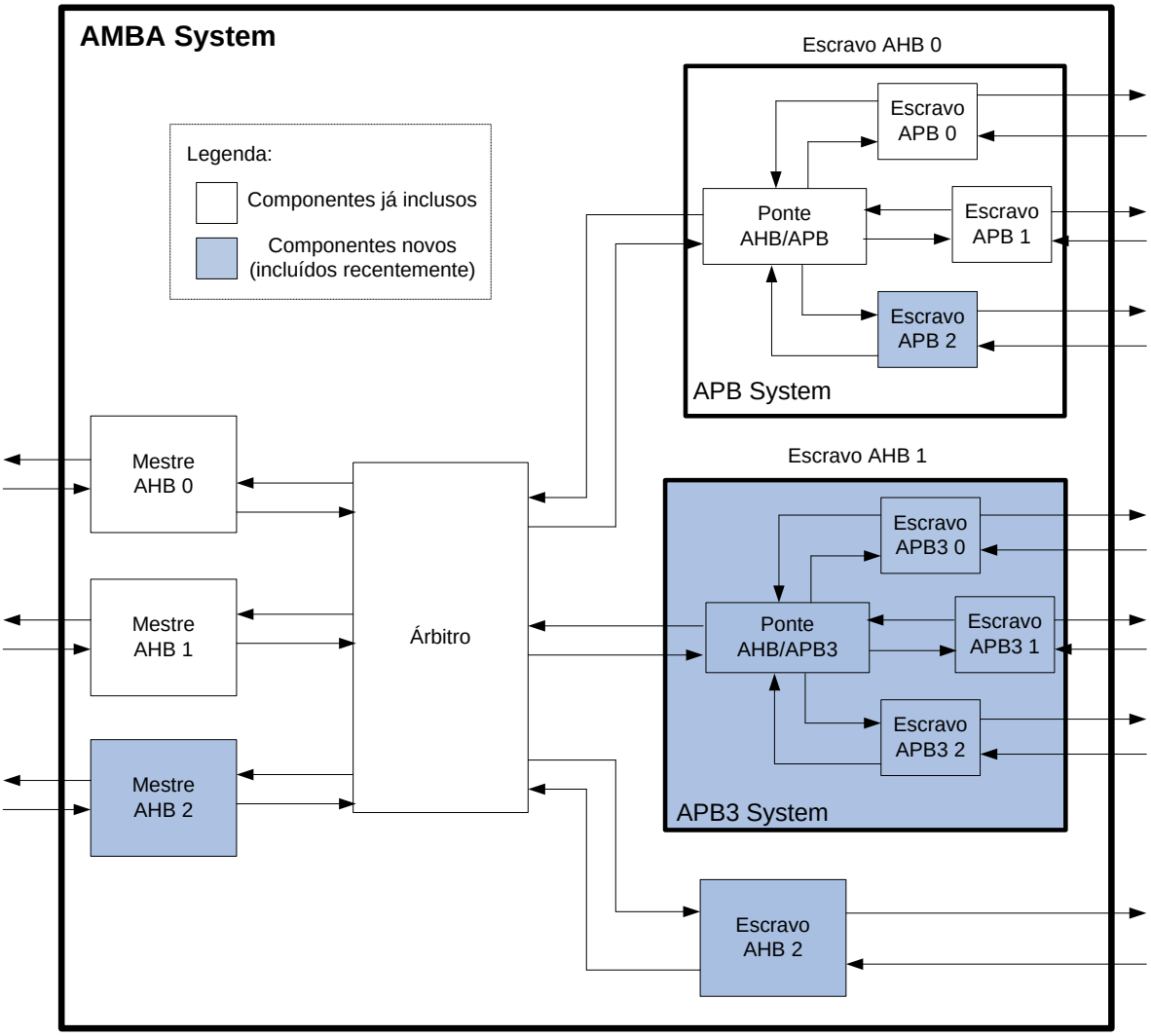

Figura 6.5: Inclusão de novos componentes ao barramento

6.4.1 Incluindo/Removendo mestres AHB

Para incluir um mestre AHB é necessário incrementar o valor da constante `AHB_MST_MAX` contida do arquivo `AHB_Types_Const.vhd` e também fazer a ligação do novo componente diretamente no árbitro do barramento. Isso é feito através do arquivo `AMBA_System.vhd`. Para excluir um mestre AHB, basta fazer o processo reverso da inclusão.

Uma máquina de estados de um mestre AHB é mostrada na figura 6.6, que possui os seguintes estados:

IDLE: Mestre encontra-se ocioso.

REQ_LOCK: Mestre fez uma requisição para transferência travada (*locked transfer*).

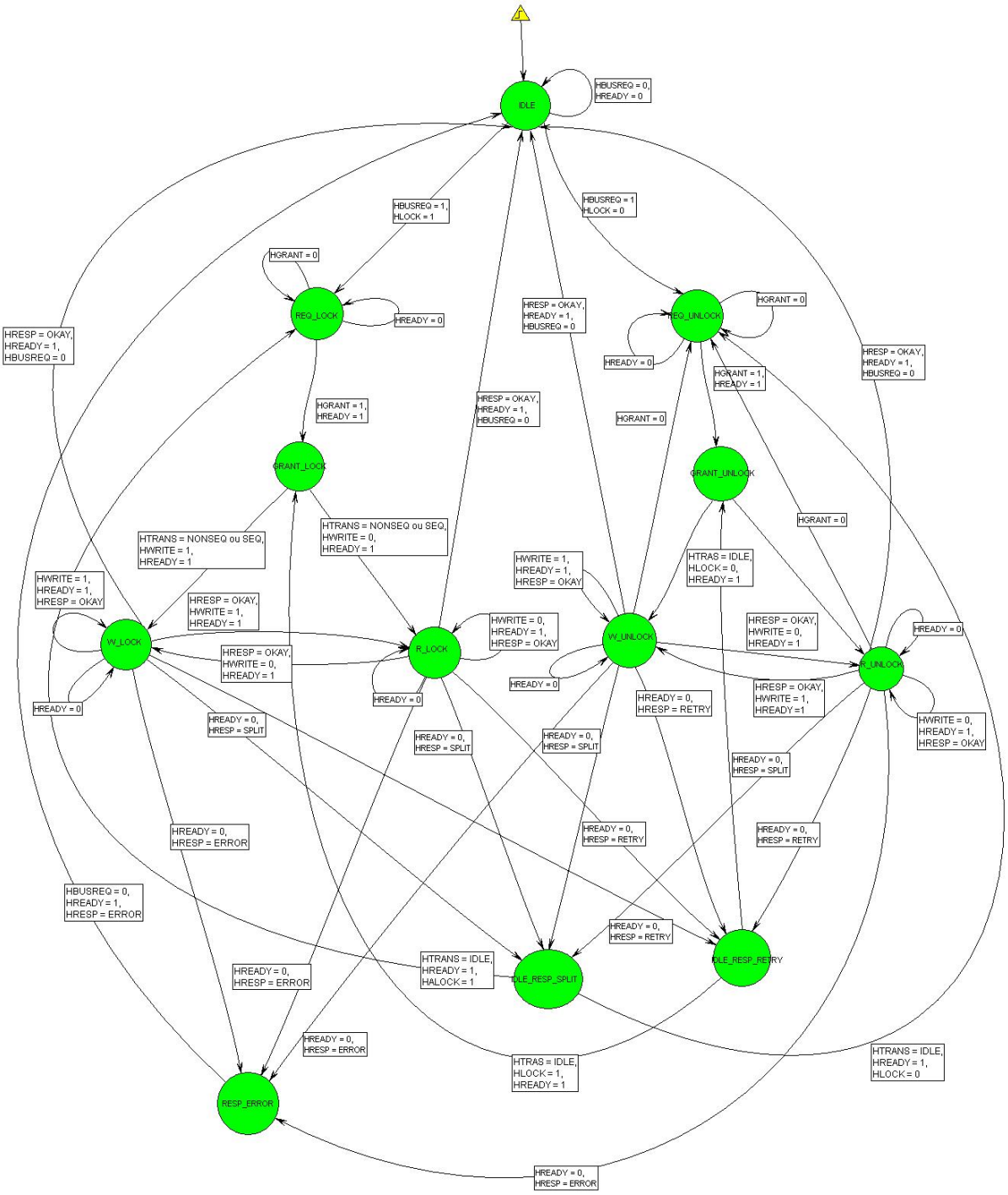

Figura 6.6: Máquina de estados finitos do mestre AHB.

REQ_UNLOCK: Mestre fez uma requisição que não trava o barramento.

GRANT_LOCK: Mestre possui um barramento e pode realizar *locked transfers*.

GRANT_UNLOCK: Mestre possui o barramento, porém não podem realizar transferências travadas.

W_LOCK: Mestre está realizando operações de escrita em uma *locked transfer*.

R_LOCK: Mestre está realizando operações de leitura em uma *locked transfer*.

W_UNLOCK: Mestre está realizando operações de escrita sem travar o barramento.

R_UNLOCK: Mestre está realizando operações de leitura sem travar o barramento.

RESP_ERROR: Mestre recebe uma mensagem de erro e aborta transferências.

IDLE_RESP_SPLIT: Mestre recebeu a resposta *split* do primeiro ciclo.

IDLE_RESP_RETRY: Mestre recebeu a resposta *retry* do primeiro ciclo.

6.4.2 Incluindo/Removendo escravos AHB

Adicionar um escravo AHB é um pouco mais trabalhoso que os mestres, pois deve-se alterar também o esquema de decodificação do endereço. Primeiramente, deve-se incrementar o valor da constante `AHB_SLV_MAX` contida do arquivo `AHB_Types_Const.vhd` e fazer a ligação do novo componente diretamente no árbitro do barramento. Depois, é preciso alterar o processo de decodificação do endereço contido no arquivo `AHB_Arbiter.vhd`. Um exemplo de como pode ser feita essa alteração é mostrado no código 6.4. Para excluir o escravo AHB, basta fazer o processos reverso da inclusão.

```
case haddr(HAMAX-1 downto 12) is -- primeiros 5 dígitos em hexadecimal
  when x"00000" | x"00001" | x"00002" | x"00003" |
       x"00004" | x"00005" | x"00006" | x"00007" |
       x"00008" | x"00009" | x"0000A" | x"0000B" |
       x"0000C" | x"0000D" | x"0000E" | x"0000F"
                           => nslave := 0; -- escravo 0
  when x"00010" | x"00011" => nslave := 1; -- escravo 1
  when x"00012" | x"00013" => nslave := 2; -- escravo 2
  when x"00014" | x"00015" => nslave := 3; -- escravo 3 (ESCRAVO ADICIONADO)
  when others => nslave := slaves;     -- escravo padrão
end case;
```

Código-fonte 6.4: Alteração no esquema de decodificação de endereços

Uma máquina de estados de um escravo AHB é mostrada na figura 6.7, que possui os seguintes estados:

IDLE: Escravo está ocioso.

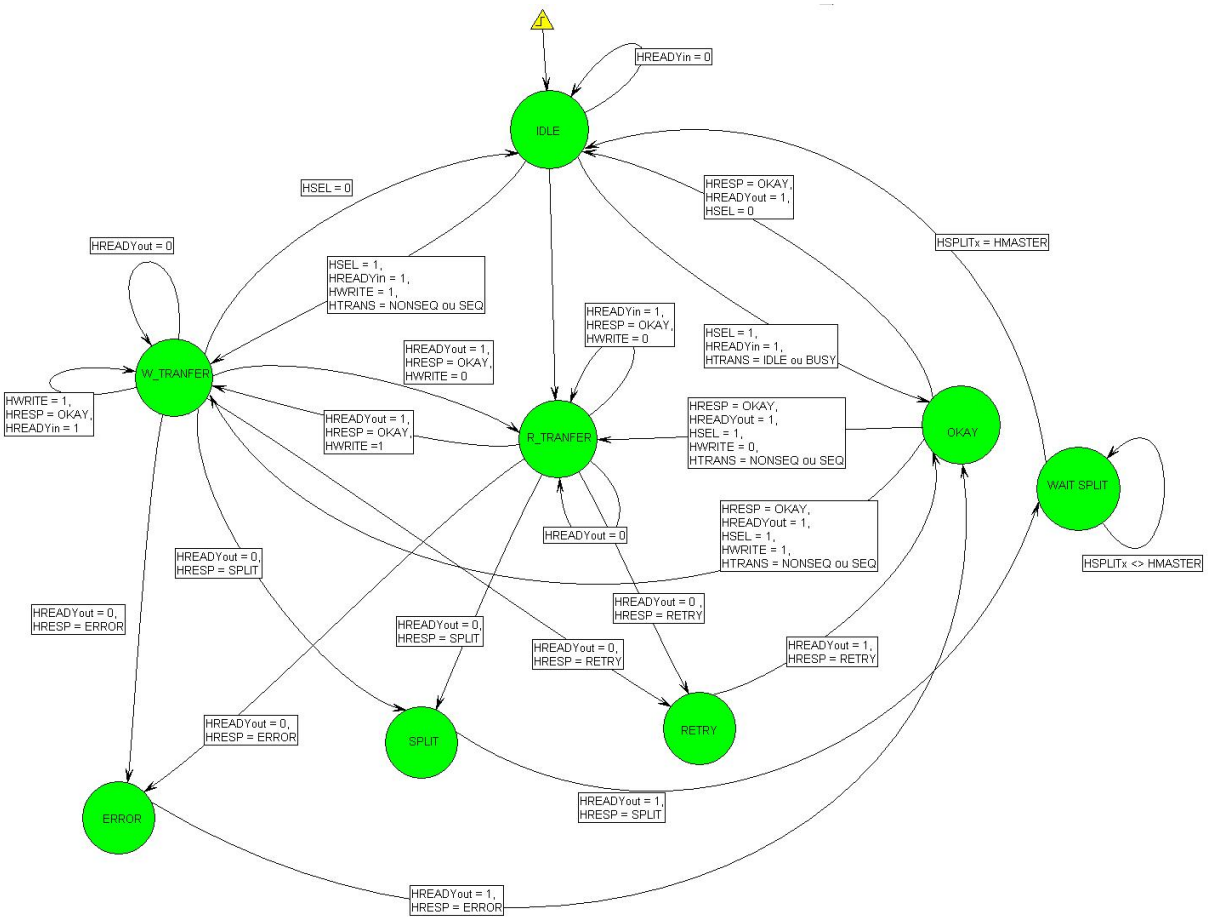

Figura 6.7: Máquina de estados finitos do escravo AHB.

W_TRANSF: Escravo encontra-se realizando operação de escrita.

R_TRANSF: Escravo encontra-se realizando operação de leitura.

OKAY: Escravo fornece uma resposta OKAY caso receba uma transferência do tipo IDLE ou BUSY. Ou caso entre no segundo ciclo de respostas: *split*, *retry* ou *error*.

ERROR: Escravo enviou o primeiro ciclo da resposta de erro.

SPLIT: Escravo enviou o primeiro ciclo da resposta de *split*.

RETRY: Escravo enviou o primeiro ciclo da resposta de *retry*.

WAIT_SPLIT: Estado em que o escravo permanece até que consiga responder a um mestre que recebeu a resposta *split*.

6.4.3 Incluindo/Removendo escravos APB/APB3

Para incluir um escravo no APB é preciso incrementar o valor da constante `APB_SLV_MAX`, contida no arquivo `APB_Types_Const.vhd`. Depois, é necessário ligar o escravo na ponte APB (`APB_Master.vhd`), através do arquivo `APB_System.vhd`. O esquema de decodificação do endereço (contido na ponte) também deve ser alterado. A inclusão de escravos no APB3 é similar ao APB, bastando apenas trocar o prefixo "`APB_`" dos arquivos por "`APB3_`". A remoção dos escravos APB/APB3 pode ser feita fazendo o processo reverso da inclusão. Cabe ressaltar, que mestres não podem ser adicionados ou removidos dos barramentos de periféricos, pois estes devem conter, obrigatoriamente, um único mestre (que é a ponte).

As figuras 6.8 e 6.9 mostram as máquinas de estados de escravos APB e APB 3, respectivamente.

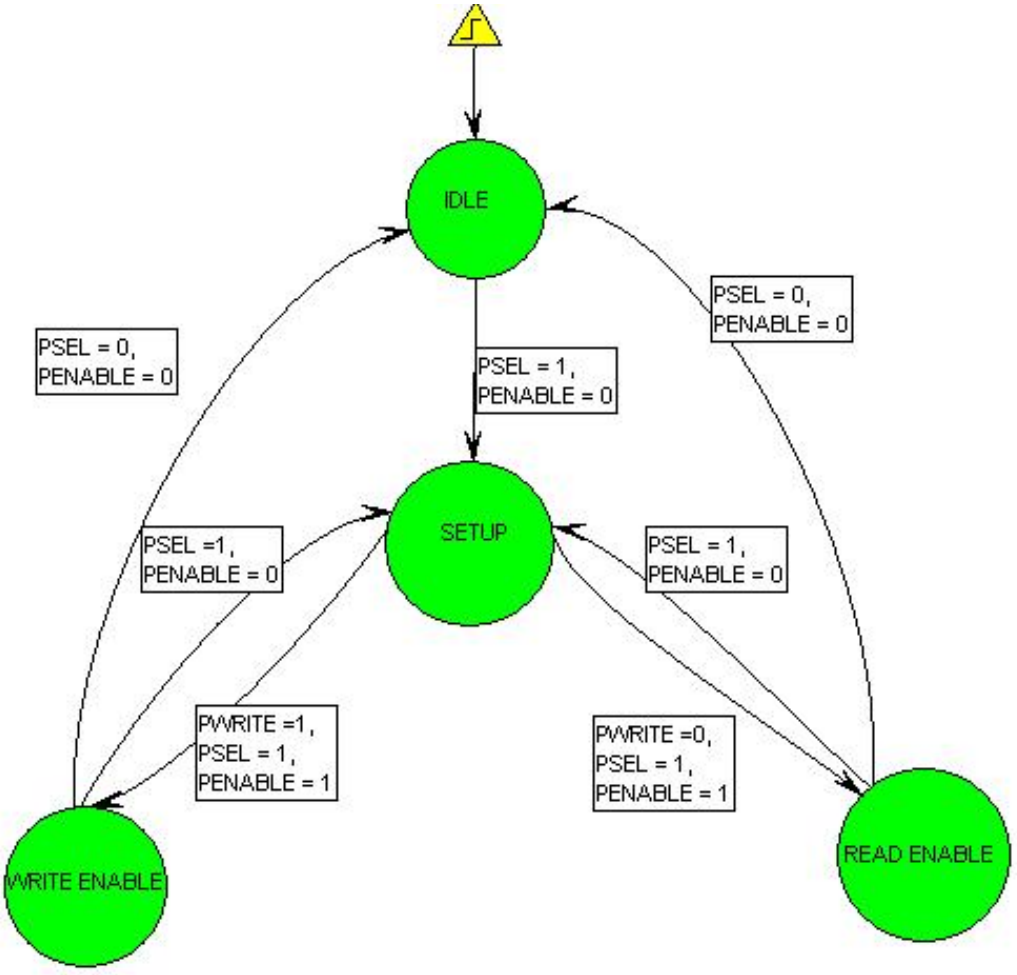

Figura 6.8: Máquina de estados finitos do escravo APB.

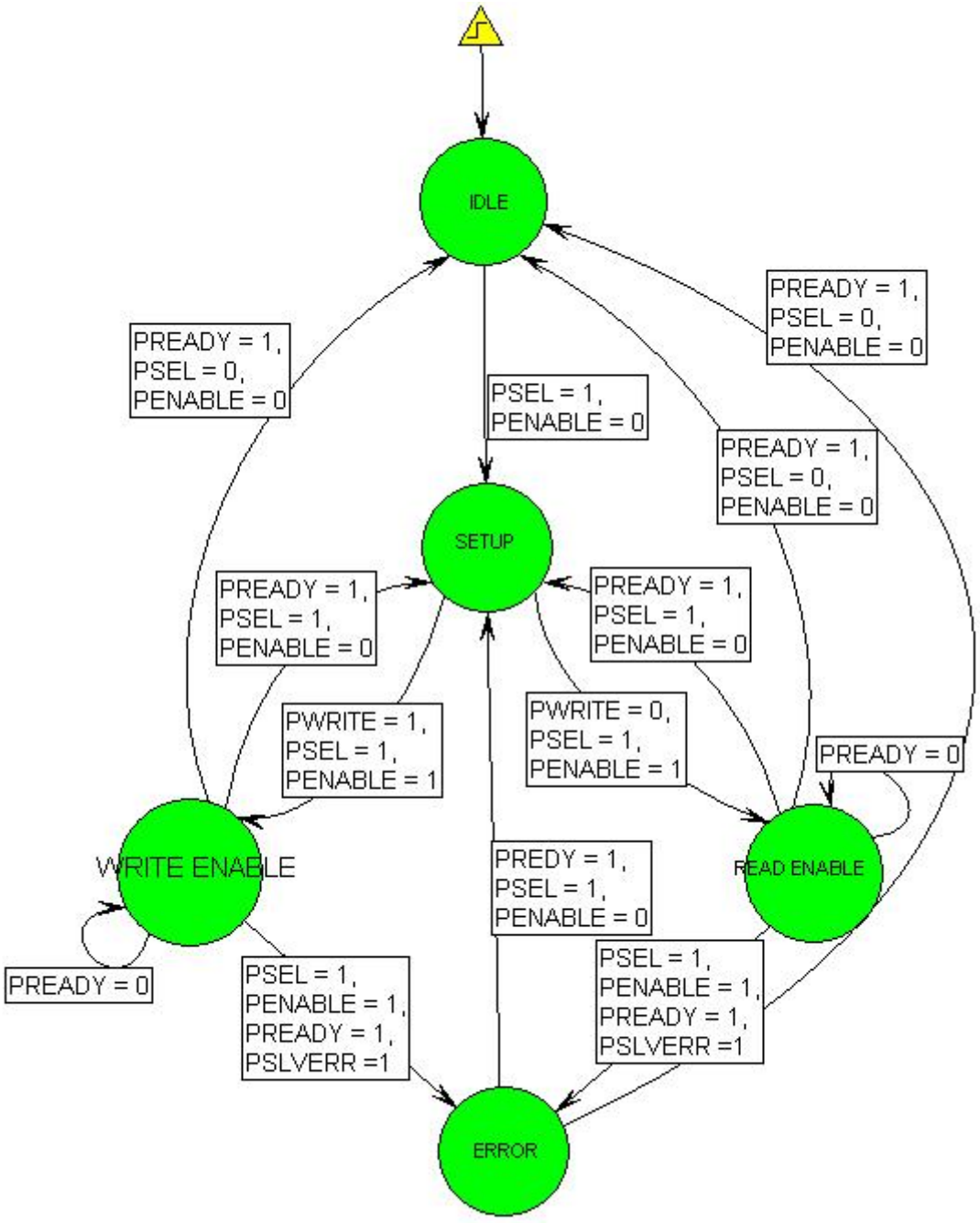

Figura 6.9: Máquina de estados finitos do escravo APB 3.

Os estados da máquina do escravo APB são citados a seguir:

IDLE: Escravo ocioso.

SETUP: Escravo encontra-se no primeiro ciclo da transferência.

WRITE ENABLE: Escravo realiza operação de escrita no segundo ciclo da transferência.

READ ENABLE: Escravo realiza operação de leitura no segundo ciclo da transferência.

Os estados do APB 3 são similares ao APB, exceto o estado **ERROR** que determina que o escravo enviou uma resposta de erro.

6.5 Validação

Para a validação do barramento implementado, foi necessária a criação de um pequeno sistema, formado por três mestres e quatro escravos AHB (dos quais dois são memórias e os outros dois são pontes APB e APB3). Foram incluídos também 3 escravos (memórias) em cada barramento de periféricos. A figura 6.10 ilustra tal sistema. O mestre padrão recebe o *grant* caso nenhum outro mestre esteja requisitando o barramento. Já o escravo padrão é acionado quando um mestre tenta acessar um endereço que não exista, colocando uma mensagem de erro de 2 ciclos.

As faixas de endereços atribuídos a todos os escravos do barramento podem ser vistas na figura 6.11.

O sistema de validação é formado por todos os arquivos contidos na tabela 6.1 e mais os componentes: `APB_Slave0.vhd`, `APB3_Slave0.vhd`, `AHB_Slave1.vhd`, `AHB_Slave2.vhd`, `AHB_Master0_Default.vhd` e `AHB_Master1.vhd`. Na figura 6.10, os mestres 1 e 2 são instâncias do `AHB_Master1`, já os escravos APB são instâncias de `APB_Slave0` e os escravos APB3 instâncias de `APB3_Slave0`. Uma pequena descrição de cada componente pode ser vista no cabeçalho de seu código-fonte, disponível em `http://www.icmc.usp.br/~lcr/amba`. A figura 6.12 mostra a hierarquia do sistema.

Depois que o sistema inteiro foi construído, foram realizadas várias simulações com os diferentes componentes, assim como simulações com o sistema completo (ver apêndice B). Todas as simulações atenderam aos requisitos impostos na especificação do barramento (ARM, 1999). Vários arquivos com *wave forms* foram gerados em tais simulações. Após a realização das simulações, foram desenvolvidos mais dois sistemas que possibilitaram a realização de testes na placa de desenvolvimento Nios - Edição Stratix (Altera, 2003b). No primeiro sistema, foram incluídos ao sistema da figura 6.10 um processador Nios II

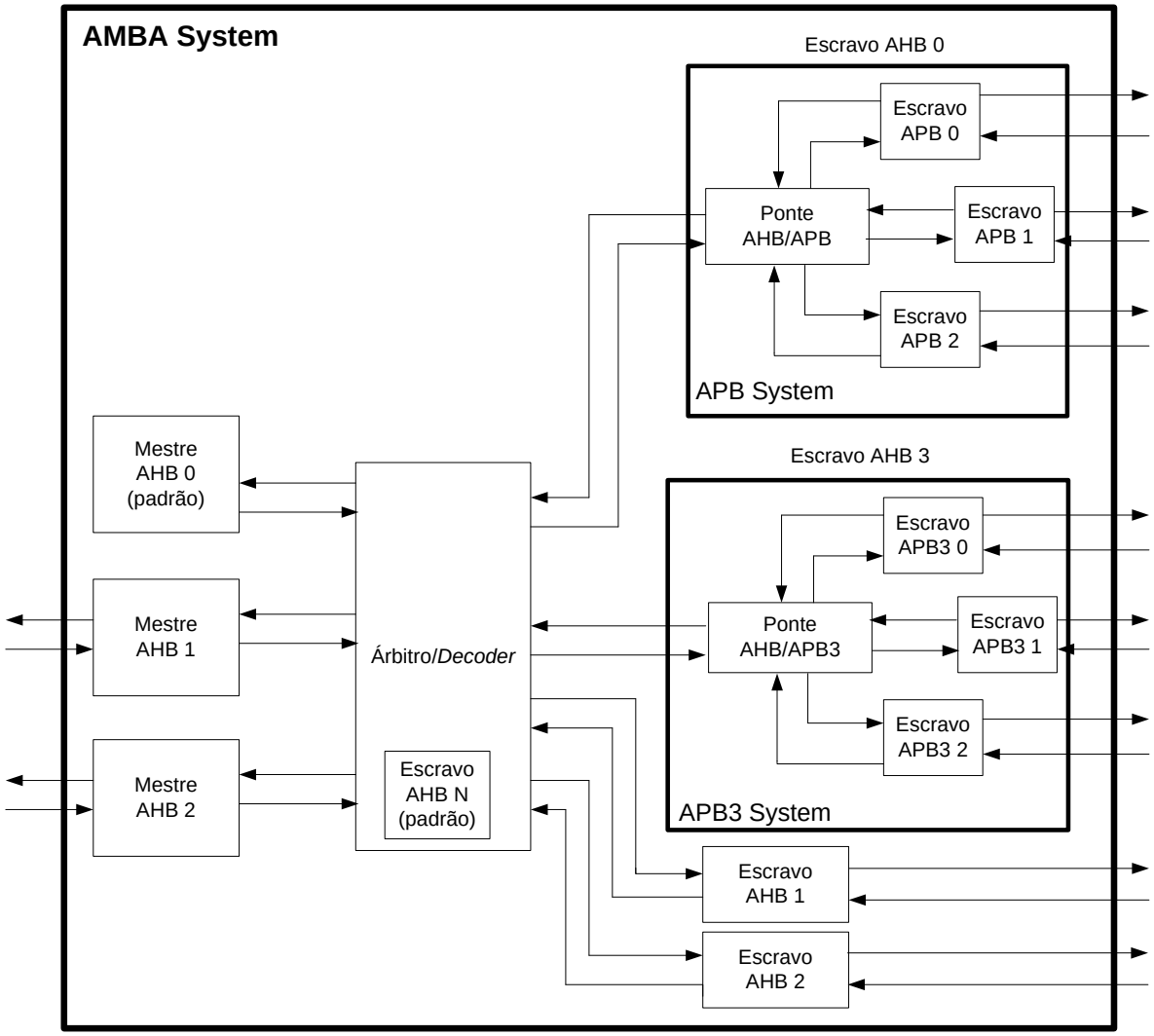

Figura 6.10: Sistema de validação

e mais um escravo AHB. A figura 6.13 ilustra o primeiro sistema, e seu funcionamento acontece da seguinte maneira:

1. Mestre 1 inicia requisição de dados para o escravo 4.
2. Escravo envia a resposta para mestre 1.
3. Mestre 1 envia as respostas recebidas para o processador Nios II.
4. Nios II exibe na tela do computador os dados de resposta e o endereço ao qual o dado foi lido.

É importante ressaltar que a comunicação entre o mestre 1 e escravo 4 ocorre totalmente utilizando-se o protocolo AMBA AHB. Já a comunicação do mestre 1 com o

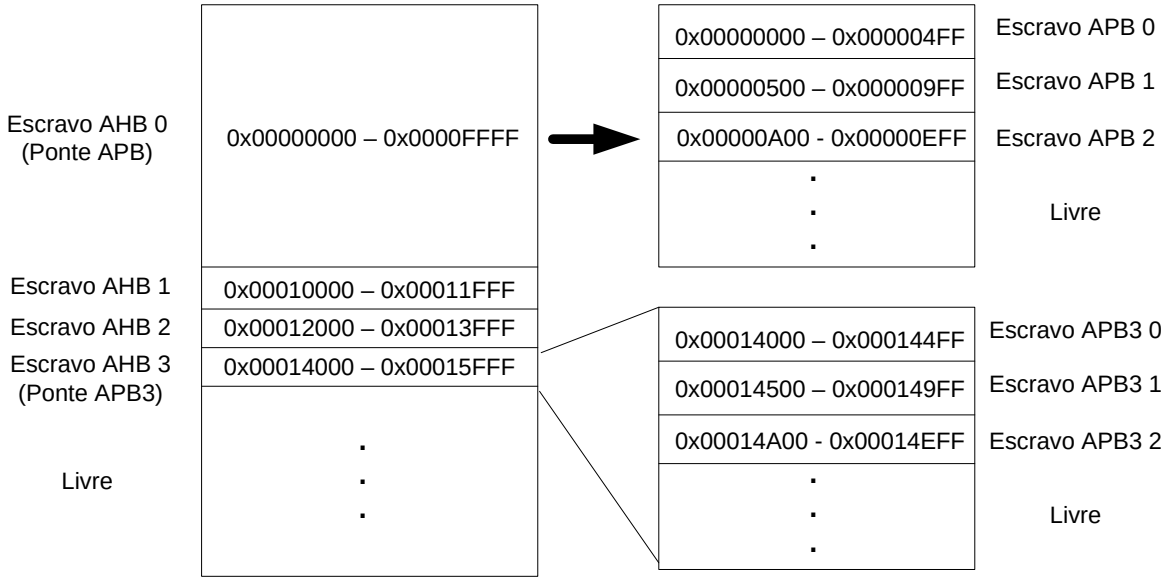

Figura 6.11: Endereçamento do Sistema de Validação

Nios II não segue o padrão AMBA. Foram utilizadas portas paralelas de E/S para essa comunicação. Esse sistema só foi implementado para que fosse possível realizar testes no FPGA da placa e que as respostas fossem visualizadas na tela do computador, pois todo o funcionamento do barramento já tinha sido validado através de simulações. A freqüência máxima atingida foi 50 MHz, devido à limitações da placa de desenvolvimento, apesar do AMBA poder operar até 117 MHz com os FPGAs da família Stratix.

Já o segundo sistema criado (ver figura 6.14) utiliza uma ponte Avalon/AMBA para ligar o Nios II ao barramento AMBA. O Nios II só possui uma interface Avalon, e como trata-se de um processador com código fechado, não é possível criar uma interface AMBA para o mesmo. Neste caso ele só pode ser ligado a outros barramentos (que não Avalon) através do uso de pontes ou adaptadores. Então, neste caso, o Nios II e a ponte serão vistos, pelo barramento, como se fosse um mestre AHB. Esse sistema é similar ao primeiro, porém, desta vez quem gera todas as requisições é o próprios Nios II.

Em ambos os sistemas foi-se desenvolvido um programa em linguagem C para realizar simples operações de leitura. Os programas foram desenvolvidos e compilados usando a ferramenta Nios II IDE (Altera, 2004b), da Altera.

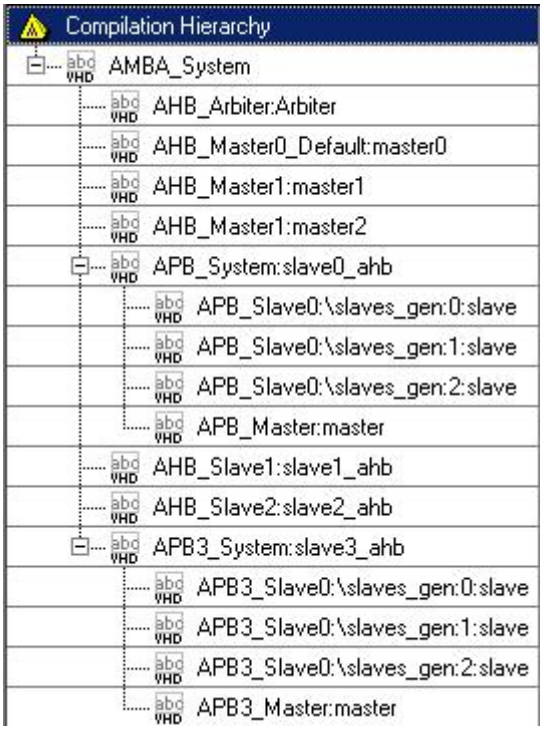

Figura 6.12: Hierarquia do Sistema de Validação

6.6 Resultados

Ao final da implementação e testes, foram colhidas algumas informações de propósito geral do barramento, como por exemplo: freqüência máxima e total de área ocupada no FPGA. Todas as compilações do projeto foram feitas para o dispositivo da família Stratix EP1S10F780C6ES (da Altera).

A tabela 6.3 mostra o total de elementos lógicos necessários em cada tipo de compilação, assim como sua freqüência máxima alcançada, referentes ao sistema da figura 6.10. Existem dois tipos de compilação que visam otimização de espaço e velocidade e um terceiro que procura um balanceamento de tais otimizações. A figura 6.15 mostra a área ocupada (mais escura), pelo sistema, no chip, utilizando compilação balanceada. Informações mais detalhadas sobre a área ocupada pelo barramento (árbitro e pontes APB e APB3) podem ser vistas na tabela 6.4. É importante ressaltar que a área ocupada no chip e a velocidade do barramento (freqüência) estão diretamente ligadas ao número de mestres e escravos do barramento, já que isso influencia diretamente no processo de arbitragem e decodificação de endereços. Mais detalhes sobre área ocupada e velocidade

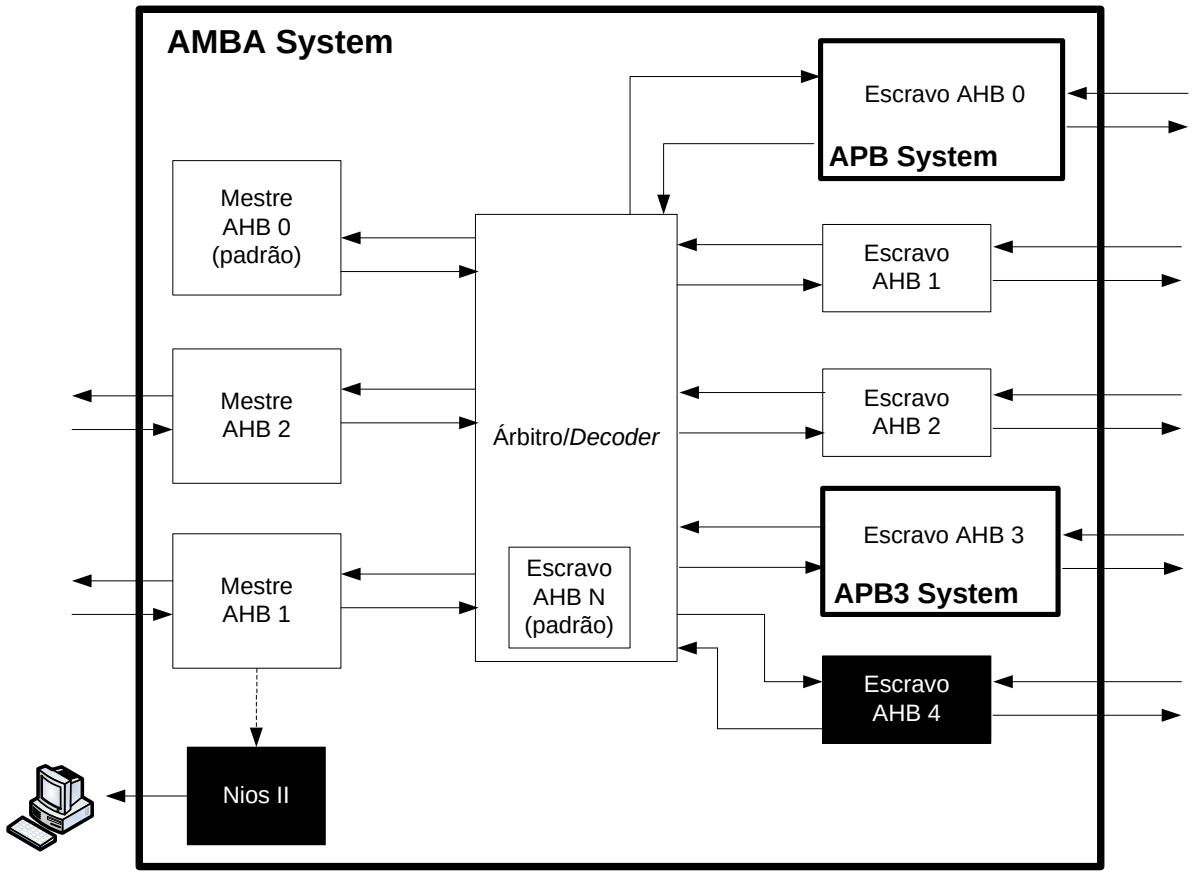

Figura 6.13: Sistema de validação com o processador Nios II

atingida pelo barramento (com diferentes números de escravo e mestres) não puderam ser analisados devido às limitações de entrada e saída do dispositivo EP1S10F780C6ES e da ferramenta Quartus II. Informações complementares sobre a implementação podem ser vistas em (Queiroz et al., 2004) e (Queiroz et al., 2005).

Compilação	**Elem. Lóg.**	**Freqüência**
Espaço	1018 (9%)	110,40 MHz
Velocidade	1465 (13%)	117,25 Mhz
Balanceado	1392 (13%)	110,02 Mhz

Tabela 6.3: Área ocupada e freqüência atingida, pelo sistema de validação, usando os 3 tipos de compilação

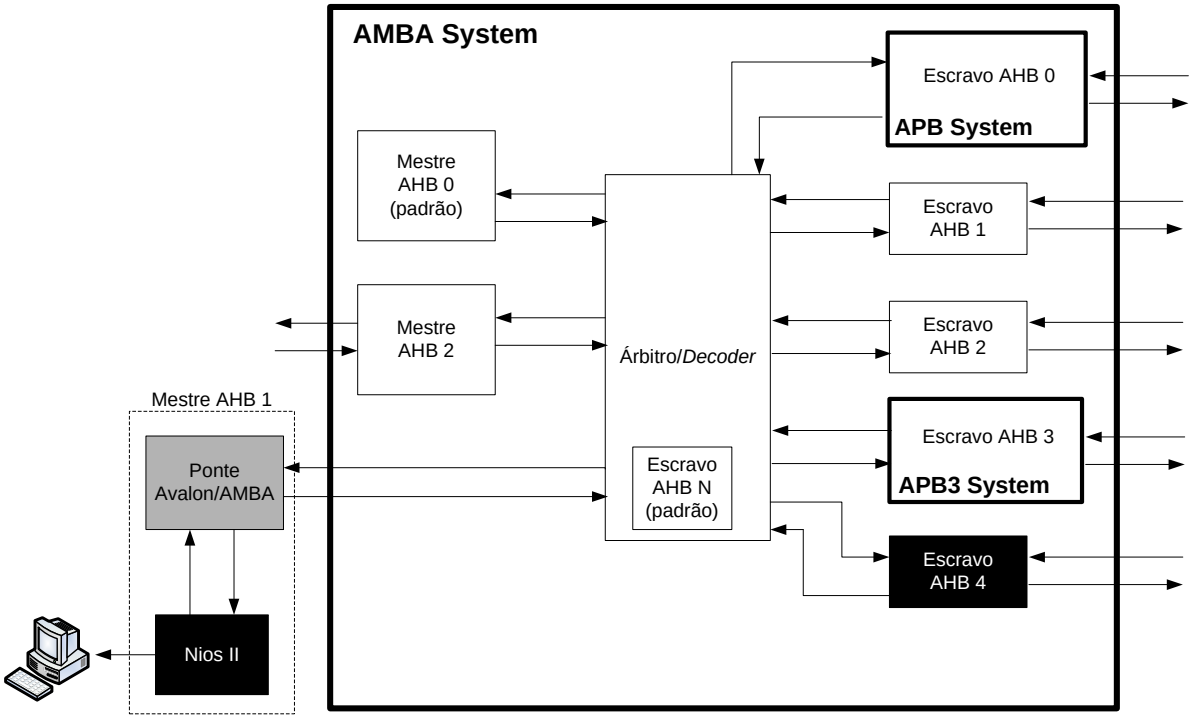

Figura 6.14: Sistema de validação com o Nios II utilizando a ponte Avalon/AMBA

Tipo de Compilação	**Elem. Lóg (Árbitro)**	**Elem. Lóg (Ponte APB)**	**Elem. Lóg (Ponte APB3)**	**Total**
Espaço	297 (2,8%)	27 (0,2%)	25 (0,2%)	349 (3,3%)
Velocidade	239 (2,2%)	86 (0,8%)	90 (0,8%)	415 (3,9%)
Balanceado	239 (2,2%)	86 (0,8%)	120 (1,1%)	445 (4,2%)

Tabela 6.4: Área ocupada somente pelos componentes que formam o barramento

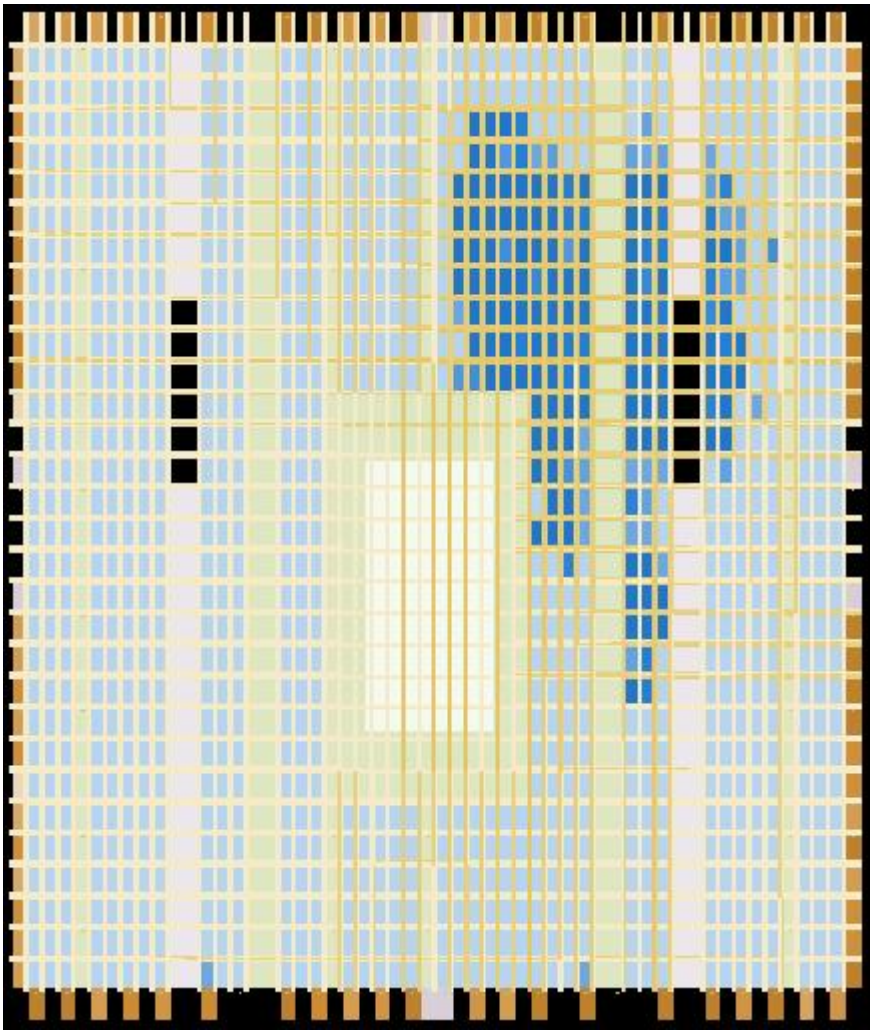

Figura 6.15: Área ocupada, pelo sistema, no FPGA EP1S10F780C6ES

Capítulo 7

Conclusão

A utilização de um protoloco padrão é essencial para a comunicação entre módulos em um SoC ou entre diversos dispositivos de um robô móvel, por exemplo. Isso torna possível a interligação de dispositivos construídos por diferentes fabricantes e/ou desenvolvedores, além de tornar possível a reutilização dos mesmos.

A utilização de ferramentas EDA foi de crucial importância para o desenvolvimento do projeto, já que auxiliou bastante no processo de simulação e depuração do mesmo. Todo o código-fonte está escrito em VHDL, o que torna a implementação independe do hardware em que se deseja usar.

Foram implementados três tipos de barramento AMBA, que são: AHB e as versões 2 e 3 do barramento APB. Isso quer dizer que pode-se criar um SoC completo (com barramento de sistema e periféricos) utilizando tal implementação. Além disso, a implementação possui um desempenho satisfatório e pode ser totalmente configurada pelo usuário.

O Laboratório de Computação Reconfigurável (LCR) absorveu a importância de construir *cores* baseados em algum padrão, para que os mesmos possam ser reutilizados em outros projetos e até mesmo por outras pessoas. Desta maneira, este trabalho permite

que todos os dispositivos construídos seguindo o barramento AMBA possam ser ligados facilmente ao sistema on-chip AMBA que se deseja construir.

Todos os objetivos do trabalho foram atingidos, e o mesmo pode ser utilizado tanto no projeto do robô móvel reconfigurável (facilitando a conexão de todos os seus módulos), como em qualquer SoC baseado em AMBA, já que o trabalho não está restrito ao projeto do robô. No caso do robô, foi incluído ao seu ambiente de projeto o controlador AMBA desenvolvido neste trabalho. Todos os componentes (mestres e escravos) do robô são ligados diretamente ao controlador, como pode ser visto na figura 7.1.

Além disso, para o uso da plataforma Nios II, ficou fácil a incorporação dessa estrutura de barramento a esse processador, pois na ferramenta SoPC Builder da Altera está disponível uma ponte Avalon/AMBA agilizando o tempo de conexão do Nios II a este meio de comunicação para a criação de sistemas on-chip baseados no barramento AMBA.

7.1 Trabalhos Futuros

Alguns dos trabalhos futuros que podem ser realizados para a continuidade desta pesquisa são:

- Passar todos os *cores* desenvolvidos no LCR para o padrão AMBA, para que os mesmos possam ser reutilizados em vários projetos e ligados facilmente ao barramento.
- Eventualmente passar os *cores* do LCR para o padrão OCP e depois criar adaptadores OCP/AMBA. Essa abordagem possibilita que os *cores* do laboratório possam ser ligados, através de adaptadores, a qualquer tipo de barramento on-chip.
- Criação de um componente para utilização na ferramenta SoPC Builder, que funcione como o controlador do barramento AHB e que também possa se comunicar com dispositivos Avalon, que são suportados por padrão na ferramenta da Altera.

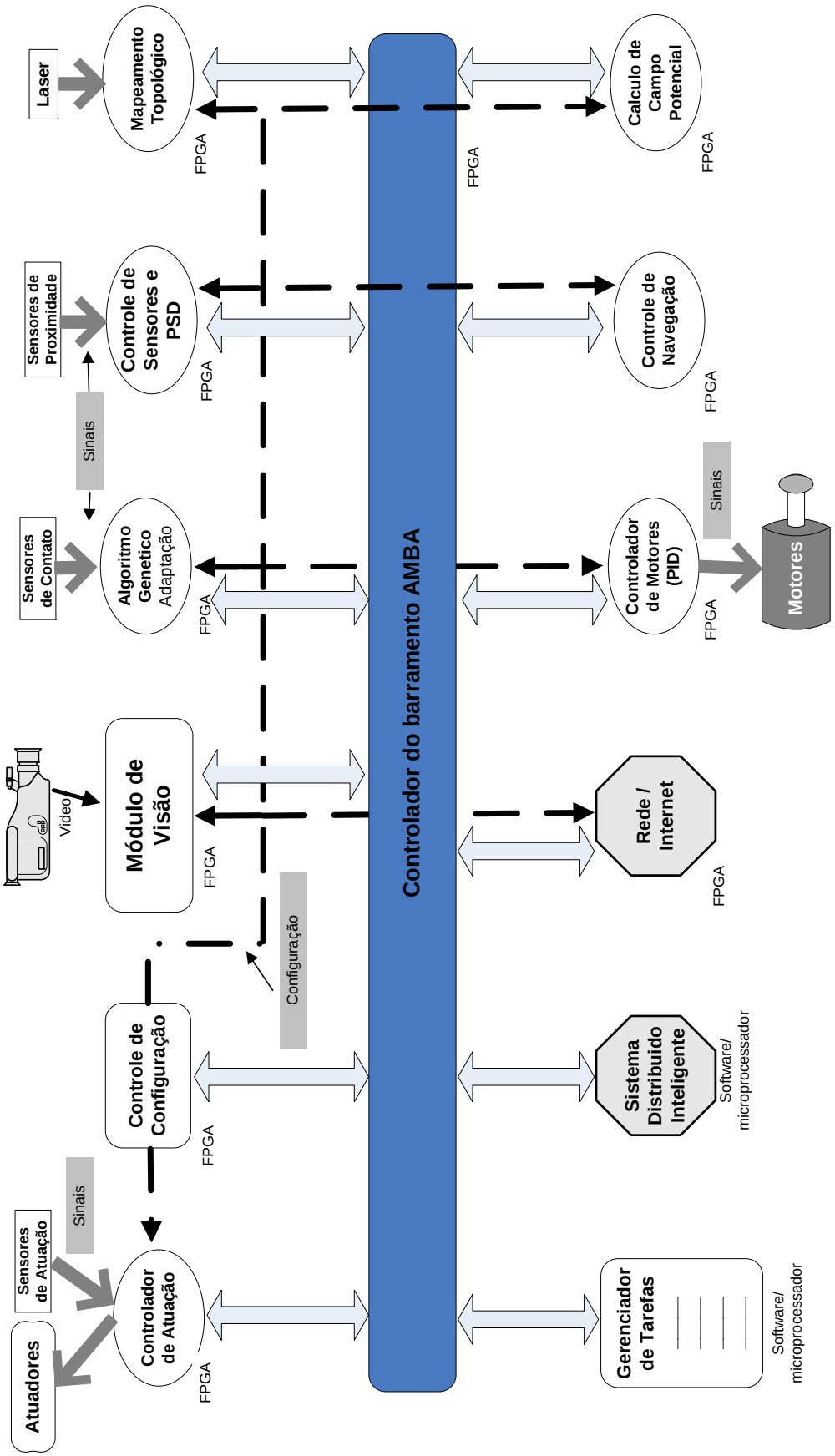

Figura 7.1: Ambiente do Robô Móvel Reconfigurável com o controlador do barramento AMBA.

Referências Bibliográficas

ALDEC Aldec - products - activehdl. 2005.
Disponível em http://www.aldec.com/ActiveHDL (Acessado em Janeiro de 2005)

ALTERA Avalon bus specification: Reference manual. 2003a.
Disponível em http://www.altera.com/literature/manual/mnl_avalon_bus.pdf (Acessado em Novembro de 2003)

ALTERA Nios development board: Reference manual, stratix edition. 2003b.
Disponível em http://www.altera.com/literature/manual/mnl_nios_board_stratix_1s10.pdf (Acessado em Outubro de 2003)

ALTERA Stratix devices: New levels of system integration. 2003c.
Disponível em http://www.altera.com/products/devices/stratix/stx-index.jsp (Acessado em Novembro de 2003)

ALTERA Altera corporation. 2004a.
Disponível em http://www.altera.com (Acessado em Fevereiro de 2004)

ALTERA Nios ii integrated development environment. 2004b.
Disponível em http://www.altera.com/products/software/products/nios2/emb-nios2_ide.html (Acessado em Novembro de 2004)

ALTERA Quartus ii software - #1 in performance. 2004c.
Disponível em http://www.altera.com/products/software/products/quartus2/qts-index.html (Acessado em Novembro de 2004)

ALTERA Sopc builder. 2004d.
Disponível em http://www.altera.com/products/software/products/sopc/sop-index.html (Acessado em Dezembro de 2004)

ALTERA Stratix ii device handbook, volume 1. 2004e.
Disponível em http://www.altera.com/literature/hb/stx2/stx2_sii5v1.pdf (Acessado em Fevereiro de 2004)

ARAGÃO, A. C. O. S.; MARQUES, E. A tecnologia fpga. Relatório Técnico nº 60-ICMC/USP, 1997.

ARAGÃO, A. C. O. S.; ROMERO, R.; MARQUES, E. Computação reconfigurável aplicada à robótica. *Computação Reconfigurável: Experiências e Perspectivas*, p. 184–188, 2000.

ARM Amba specification (rev. 2.0). 1999.
Disponível em http://www.arm.com/download.nsf/htmlall/3ACB322746FA8F5C80256AA2005017A5/$File/IHI0011A_AMBA_SPEC.pdf?OpenElement (Acessado em Maio de 2003)

ARM Ahb-lite: Overview. 2001a.
Disponível em http://www.arm.com/armtech/5DADW3/$File/AHB_Lite_Overview.pdf (Acessado em Maio de 2003)

ARM Multi-layer ahb: Overview. 2001b.
Disponível em http://www.arm.com/armtech/5DADVQ/$File/Multi_Layer_AHB_Overview.pdf (Acessado em Maio de 2003)

ARM Amba axi protocol: Specification. 2003a.
Disponível em http://www.arm.com/axi.nsf/20dedca13ec8c77b80256bbb00591c05/$FILE/AMBAaxi.pdf (Acessado em Outubro de 2003)

ARM Amba: The de facto standard for on-chip bus. 2003b.
Disponível em http://www.arm.com/armtech/AMBA?OpenDocument (Acessado em Maio de 2003)

ARM Amba 3 apb protocol. 2004.
Disponível em http://www.arm.com/amba_axi/AMBA3apb.pdf (Acessado em Novembro de 2004)

BECKER, J.; VORBACH, M. Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (csoc). *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, p. 6, 2003.

BERGER, A. *Embedded systems design*. Kansas: CMP Books, 2002.

CARRO, L.; WAGNER, F. R. Sistemas computacionais embarcados. In: *XXII Jornadas de Atualização em Informática - JAI´03*, Campinas, Brazil, 2003, p. Capítulo 2.

GAISLER, J. The leon-2 processor user's manual (version 1.0.21). 2003.
Disponível em http://www.gaisler.com/doc/leon2-1.0.21-xst.pdf (Acessado em Novembro de 2003)

GONÇALVES, R. A. *Architect-r: Uma ferramenta para o desenvolvimento de robôs móveis reconfiguráveis.* Dissertação de Mestrado, ICMC-USP, 2000.

HELLMICH, H. H.; ERDOGAN, A. T.; ARSLAN, T. Re-usable low power dsp ip embedded in an arm based soc architecture. *IEE Colloquium on Intellectual Property*, p. 5, 2000.

IBM The coreconnect bus architecture. 1999.
Disponível em http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF778525699100500COFB/$file/crcon_wp.pdf (Acessado em Abril de 2003)

IDLAB Embedded systems design - an idlab interactive course. 2004.
Disponível em http://www.idlab.dal.ca/Products/Courses/EmbeddedSystemsDesign/mod_01/es1-2.htm

KALTE, H.; LANGEN, D.; VONNAHME, E.; BRINKMANN, A.; RÜCKERT, U. Dynamically reconfigurable system-on-programmable-chip. *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP 2002)*, p. 9, 2002.

LEE, A. S.; BERGMANN, N. W. On-chip communication architectures for reconfigurable system-on-chip. *Proceedings International Conference on Field-Programmable Technology*, p. 4, 2003.

LEIBSON, S. Reduce soc simulation and development time by designing with processors, not gates. 2002.
Disponível em http://www.us.design-reuse.com/articles/article4463.html (Acessado em Novembro de 2004)

MACCABE, A. B. *Computer systems: Architecture, organization, and programming.* Richard D. Irwin, 1993.

MARQUES, E. *Projeto de uma rede local de computadores de alta velocidade.* Dissertação de Mestrado, Universidade de São Paulo, 1988.

MIPS Mips technologies inc. 2004.
Disponível em http://www.mips.com (Acessado em Fevereiro de 2004)

MOCH, S.; BEREKOVIC, M.; STOLBERG, H. J.; FRIEBE, L.; KULACZEWSKI, M. B.; DEHNHARDT, A.; PIRSCH, P. Hibrid-soc: A multi-core architecture for image and

video applications. *Proceedings MEDEA Workshop at The Twelfth International Conference on Parallel Architectures and Compilation Techniques (PACT 2003)*, p. 7, 2003.

OCP-IP Socket-centric ip core interface maximizes ip applications. 2004a.
Disponível em http://www.ocpip.org/data/wp_pointofview_final.pdf (Acessado em Novembro de 2004)

OCP-IP Welcome to the ocp-ip. 2004b.
Disponível em http://www.ocpip.org (Acessado em Dezembro de 2004)

PACT Smexpp smart media processor. 2003.
Disponível em http://www.pactxpp.com/xneu/px_SMeXPP.html (Acessado em Fevereiro de 2004)

PACT Pact xpp technologies. 2004.
Disponível em http://www.pactxpp.com (Acessado em Fevereiro de 2004)

Palma, J. C.; Moraes, F.; Calazans, N. Métodos para desenvolvimento e distribuição de ip cores. *SCR'2001 - Seminário de Computação Reconfigurável*, p. 8, 2001.

Patterson, D. A.; Hennessy, J. L. *Computer organization and design: the hardware/software interface*. 2 ed. Morgan Kaufmann, 1998.

Prosilog Prosilog - sailing over complexity - ips portfolio. 2004.
Disponível em http://www.prosilog.com/products/ips/products_ips.html (Acessado em Novembro de 2004)

Queiroz, D. C.; Holanda, J. A. M.; Marques, E. Implementação do barramento amba baseada em computação reconfigurável. *Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Vídeo, Visão Computacional e Robótica*, 2004.

Queiroz, D. C.; Holanda, J. A. M.; Marques, E.; Sanches, A. K. Implementação do barramento on-chip amba baseada em computação reconfigurável. *Jornadas sobre Sistemas Reconfiguráveis*, (A ser apresentado), 2005.

Research, G. Leon3 processor core. 2004.
Disponível em http://www.gaisler.com/doc/Leon3%20Grlib%20folder.pdf (Acessado em Novembro de 2004)

Ribeiro, A. A. L.; Marques, E. Reconfiguração remota de hardware. *Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Vídeo, Visão Computacional e Robótica*, 2004.

ROMERO, R. A. Aprendizado em robôs móveis via software e hardware - armosh. Projeto de Pesquisa, 2000.

SAMSUNG Ics for mobile solution. 2003.
Disponível em http://www.samsung.com/Products/Semiconductor/Support/ebrochure/2003/ics_for_mobile_solution_200302.pdf (Acessado em Fevereiro de 2004)

SHANLEY, T.; ANDERSON, D. *Pci system architecture*. PC System Architecture, 4 ed. Addison-Wesley, 1999.

SILICORE Wishbone specification. 2001.
Disponível em http://www.silicore.net/pdfiles/wishbone/specs/wbspec_b2_chgtrk.pdf (Acessado em Abril de 2003)

SPURGEON, C. E. *Ethernet: The definitive guide*. O'Reilly, 2000.

STALLINGS, W. *Computer organization and architecture*. 5 ed. Prentice-Hall, 2000.

TSAI, W.-C.; HAUNG, C.-M.; WANG, J.-J.; LEE, C.-Y. Infrastructure for education and research of soc/ip in taiwan. *2003 International Conference on Microelectronics Systems Education (MSE'03)*, p. 2, 2003.

WAKERLY, J. F. *Digital design: Principles and practices*. 3 ed. Prentice-Hall, 2001.

WILSON, R. Design reuse is now a market necessity. 2003.
Disponível em http://www.eetimes.com/story/OEG20030501S0003 (Acessado em Dezembro de 2004)

WOLF, W. *Computers as components*. McGraw-Hill, 2001.

XILINX An introduction to xilinx products. 1999.
Disponível em bwrc.eecs.berkeley.edu/Research/Pico_Radio/Test_Bed/Hardware/Documentation/Xilinx/intro.pdf (Acessado em Janeiro de 2005)

XILINX Xilinx and gibson deliver the first true digital guitar. 2002a.
Disponível em http://www.xilinx.com/publications/xcellonline/news/xc_pdf/xc_gibson44.pdf (Acessado em Novembro de 2004)

XILINX Xilinx virtex-ii pro: Q&a (version 1.7). 2002b.
Disponível em http://www.xilinx.com/products/virtex2pro/virtex2pro_qanda.pdf (Acessado em Novembro de 2003)

XILINX Programmable logic design quick start hand book. Xilinx University Program, 2003.
Disponível em http://www.xilinx.com/univ/Beginners_Bk.pdf

XILINX Fpgas on mars. 2004a.
Disponível em http://www.xilinx.com/publications/xcellonline/xcell_50/xc_pdf/xc_mars50.pdf (Acessado em Novembro de 2004)

XILINX Virtex-4 family overview. 2004b.
Disponível em http://direct.xilinx.com/bvdocs/publications/ds112.pdf (Acessado em Dezembro de 2004)

ZHOU, J. P. *Ocp complianr adapter for network-on-chip.* Dissertação de Mestrado, Technical University of Denmark, 2004.
Disponível em http://www.imm.dtu.dk/pubdb/views/edoc_download.php/3341/pdf/imm3341.pdf (Acessado em Dezembro de 2004)

Apêndice

A

Kit de desenvolvimento Nios - Edição Stratix

O kit de desenvolvimento Nios (edição Stratix) é composto basicamente por uma placa de desenvolvimento e por ferramentas EDA (*Electronic Design Automation*) para síntese, simulação, testes e implementação de hardware.

A placa de desenvolvimento do kit oferece uma plataforma de desenvolvimento para SoC (*Systems on-Chip*) baseado em dispositivos da Altera. A placa possui um FPGA EP1S10F780C6ES, da Altera, que possui 10.570 elementos lógicos, além de 920Kbits de memória on-chip. Também já vem pré-programado na placa um processador Nios de 32bits (Altera, 2003b).

A placa é composta pelos seguintes componentes:

- 8 Mbytes de memória flash;
- 16 Mbytes de memória SDRAM (RAM Dinâmica Síncrona);
- 1 Mbyte de memória SRAM (RAM Estática Assíncrona);
- Lógica para configuração da Stratix a partir da memória Flash;
- Conector para cartão de memória FlashTM;

- Conector *Mictor* para executar debug de hardware and software;
- Duas portas DB9 de comunicação RS-232;
- Dispositivo *ethernet* MAC/PHY;
- Quatro botões do tipo *push-button*, oito *LEDs* e dois display de 7-segmentos para uso geral;
- Conector e cabo JTAG para download de *datastream* (dados de configuração do FPGA);
- Circuito de *Power-on reset* (circuito que é acionado toda vez que o sistema for energizado);
- Oscilador de 50 MHz ligado ao FPGA;
- Tensões 5Vcc, 3,3Vcc e 1,5Vcc disponíveis nos barramento de expansão;

A figura A.1 mostra o diagrama de blocos da placa de desenvolvimento. As figuras A.2 e A.3 mostram mais detalhes da placa.

Além da placa Nios, o kit possui uma ferramenta da Altera chamada Quartus II. Tal ferramenta auxilia o desenvolvedor de hardware em todas as fases de projeto e construção do mesmo, fornecendo meios para a realização de síntese, descrição lógica, análise, simulação, testes e implementação de hardware.

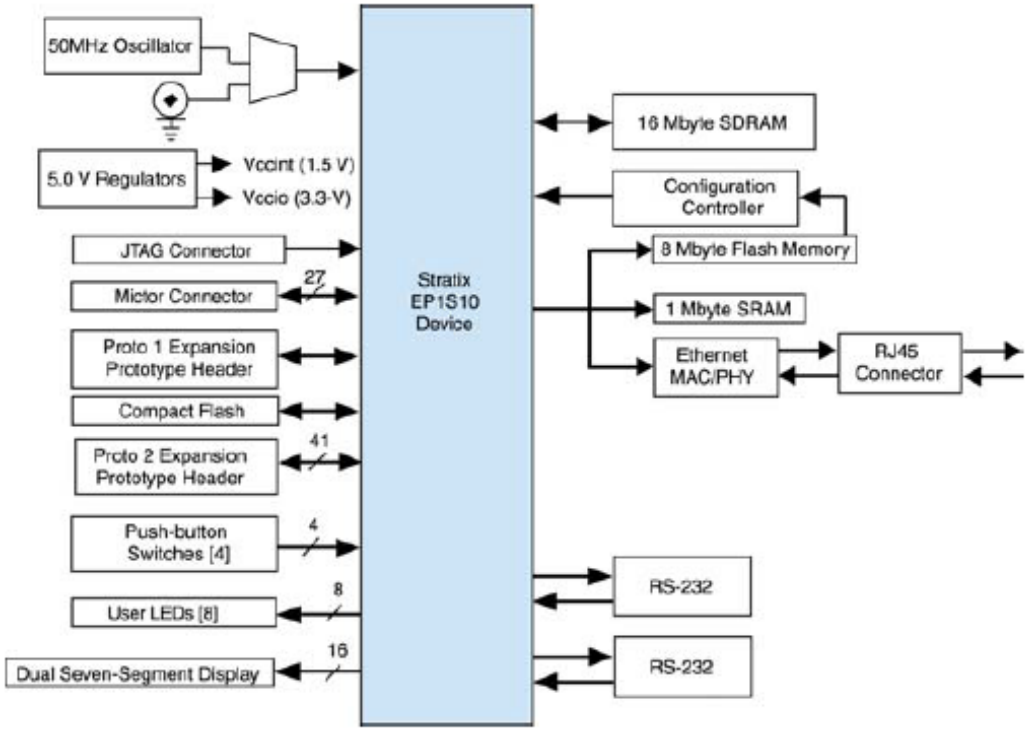

Figura A.1: Diagrama de bloco da placa Nios (Altera, 2003b).

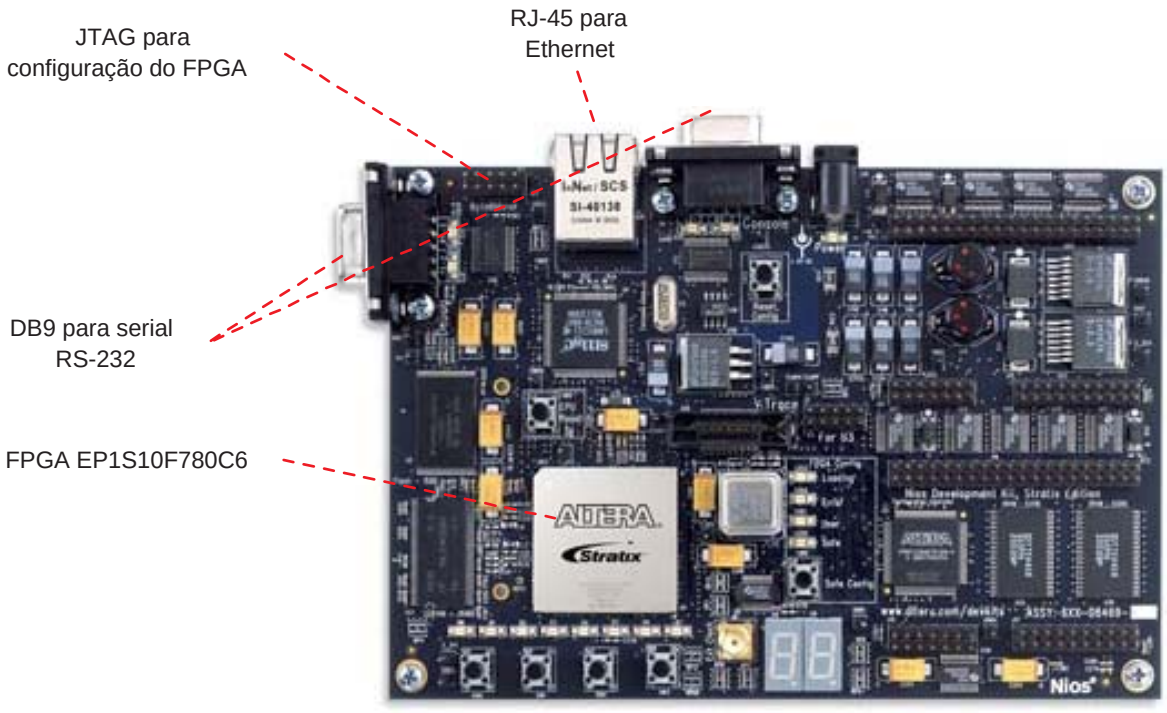

Figura A.2: Foto da placa Nios.

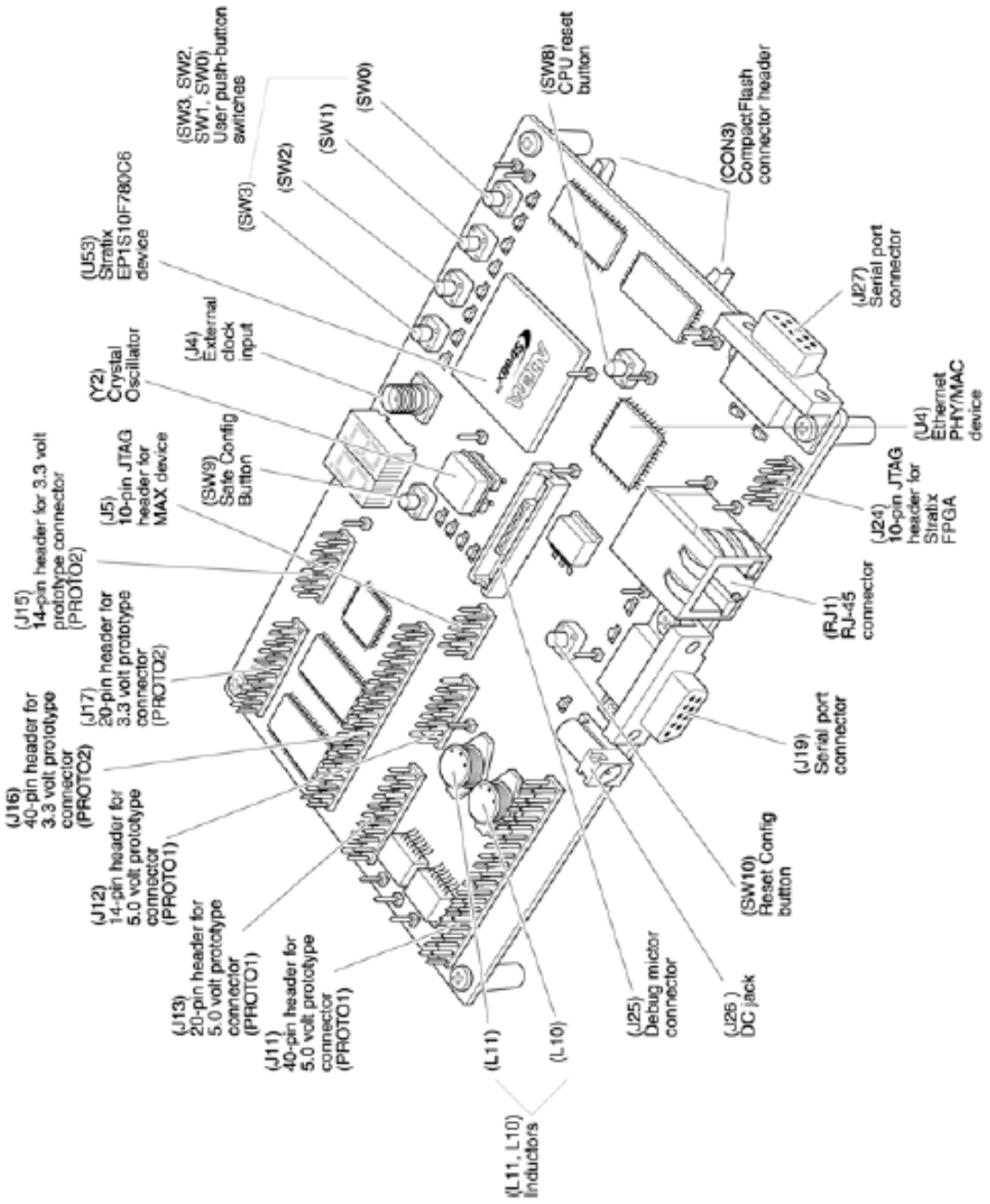

Figura A.3: Componentes da placa Nios (Altera, 2003b).

Apêndice

B

Resultados das Simulações

São apresentadas, a seguir, algumas simulações realizadas com os componentes do sistema de validação desenvolvido. Todas elas foram realizadas utilizando a ferramenta Quartus II da Altera. Cabe ressaltar, que em alguns casos os endereços contidos nas simulações não estão de acordo com o endereçamento do sistema. Isso ocorre, pois em alguns casos os endereços reais não são necessários. Um exemplo pode ser visto na simulação de um escravo APB, pois neste caso não é necessário que o mesmo esteja ligado ao sistema, para que possa ser simulado.

A figura B.1 mostra o resultado da simulação com um escravo APB. Ele realiza funções de leitura e escrita em sua única posição de memória. Caso o endereço de entrada termine com x"FF"(ex: x"000002FF") e a operação seja de leitura, o mesmo irá retornar o valor x"88888888"como resposta. Nos outros casos irá retornar o valor contido na memória.

A figura B.2 mostra o resultado da simulação com um escravo APB 3. Ele realiza funções de leitura e escrita em sua única posição de memória. Caso o endereço de entrada termine com x"FF"(ex: x"000002FF") e a operação seja de leitura, o mesmo irá retornar o valor x"22222222"como resposta. Quando terminar em x"EE"retornará um sinal de erro. Se terminar em x"AA"irá gerar um wait state. Nos outros casos irá retornar o valor

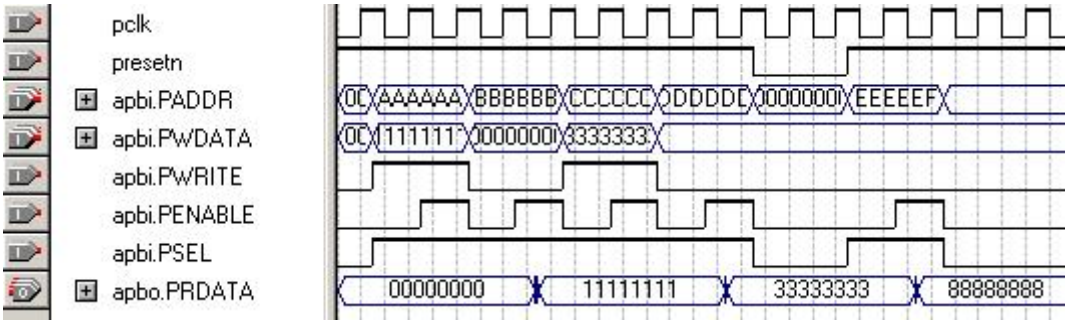

Figura B.1: Simulação de um escravo APB.

contido na memória (sem wait state).

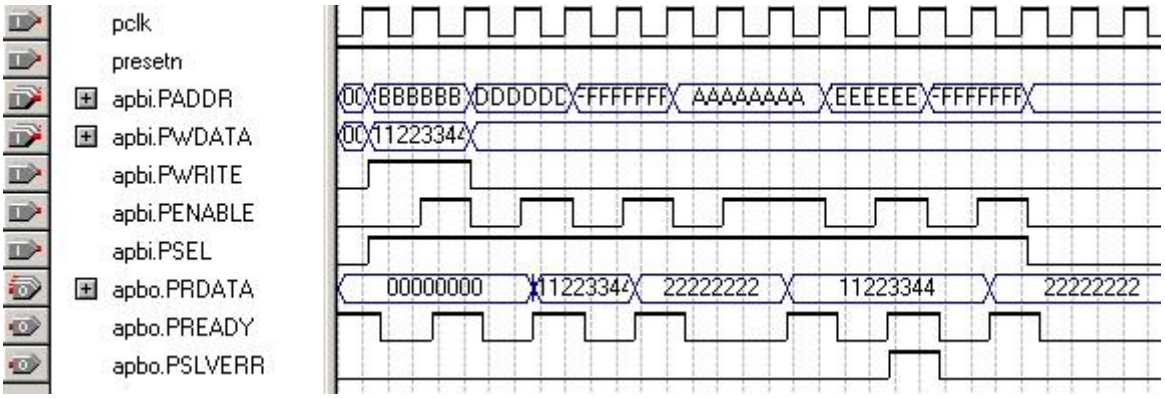

Figura B.2: Simulação de um escravo APB 3.

As figuras B.3 e B.4 mostram os resultados das simulações de leitura e escrita, respectivamente, com a ponte AHB/APB (mestre APB). A ponte é responsável por converter as transferências AHB em transferências APB.

As figuras B.5 e B.6 mostram os resultados das simulações de leitura e escrita, respectivamente, com a ponte AHB/APB3 (mestre APB 3). A ponte é responsável por converter as transferências AHB em transferências APB 3.

A figura B.7 mostra o resultado da simulação com o APB_System. São mostradas algumas operações de leitura e escrita com o escravos APB. O APB_System funciona como se fosse um escravo AHB.

A figura B.8 mostra o resultado da simulação com o APB3_System. São mostradas algumas operações de leitura e escrita com o escravos APB3. O APB3_System funciona como se fosse um escravo AHB.

A figura B.9 mostra o resultado da simulação com o escravo 1 AHB. Ele vai enviar

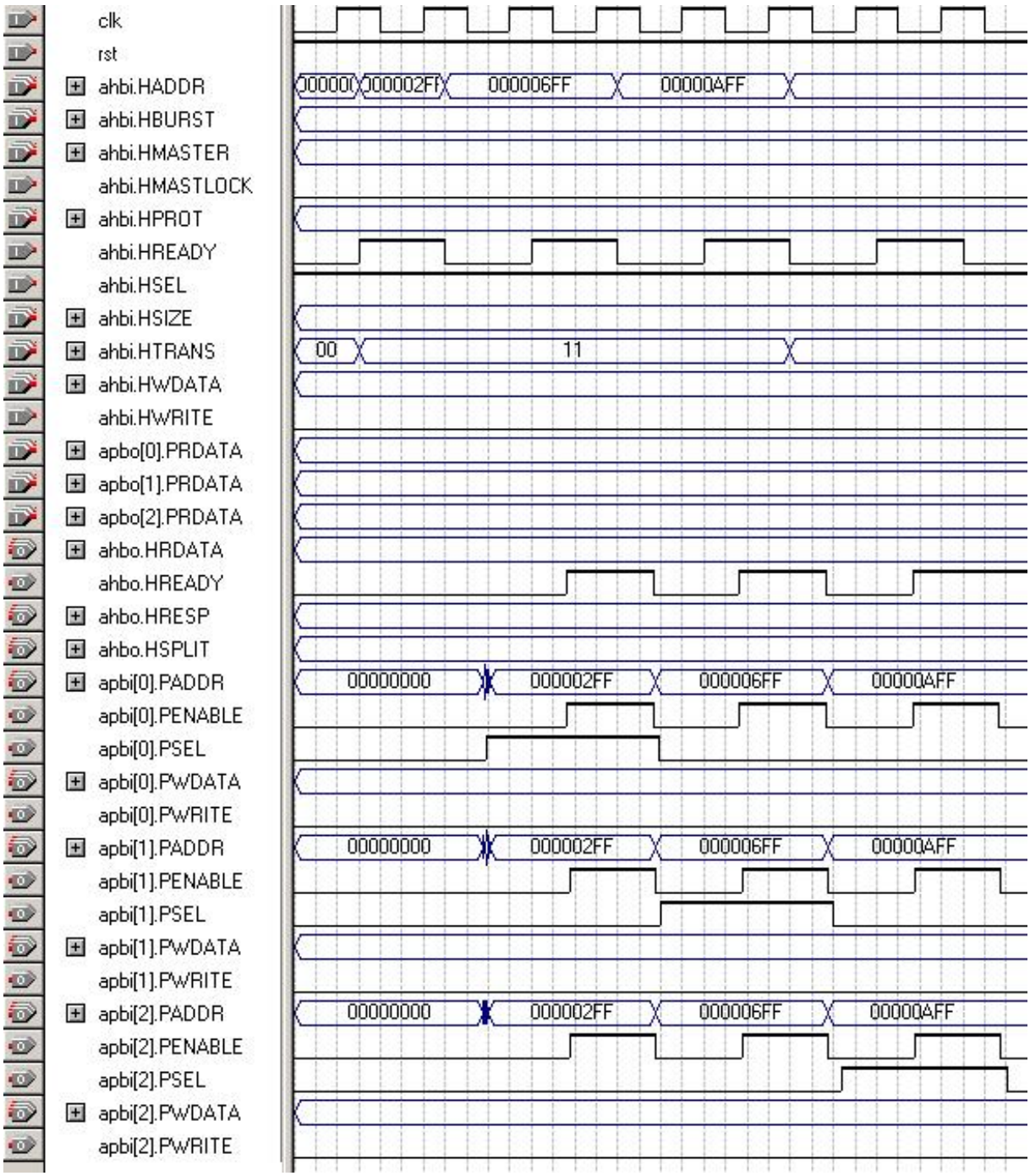

Figura B.3: Operações de leitura da ponte AHB/APB.

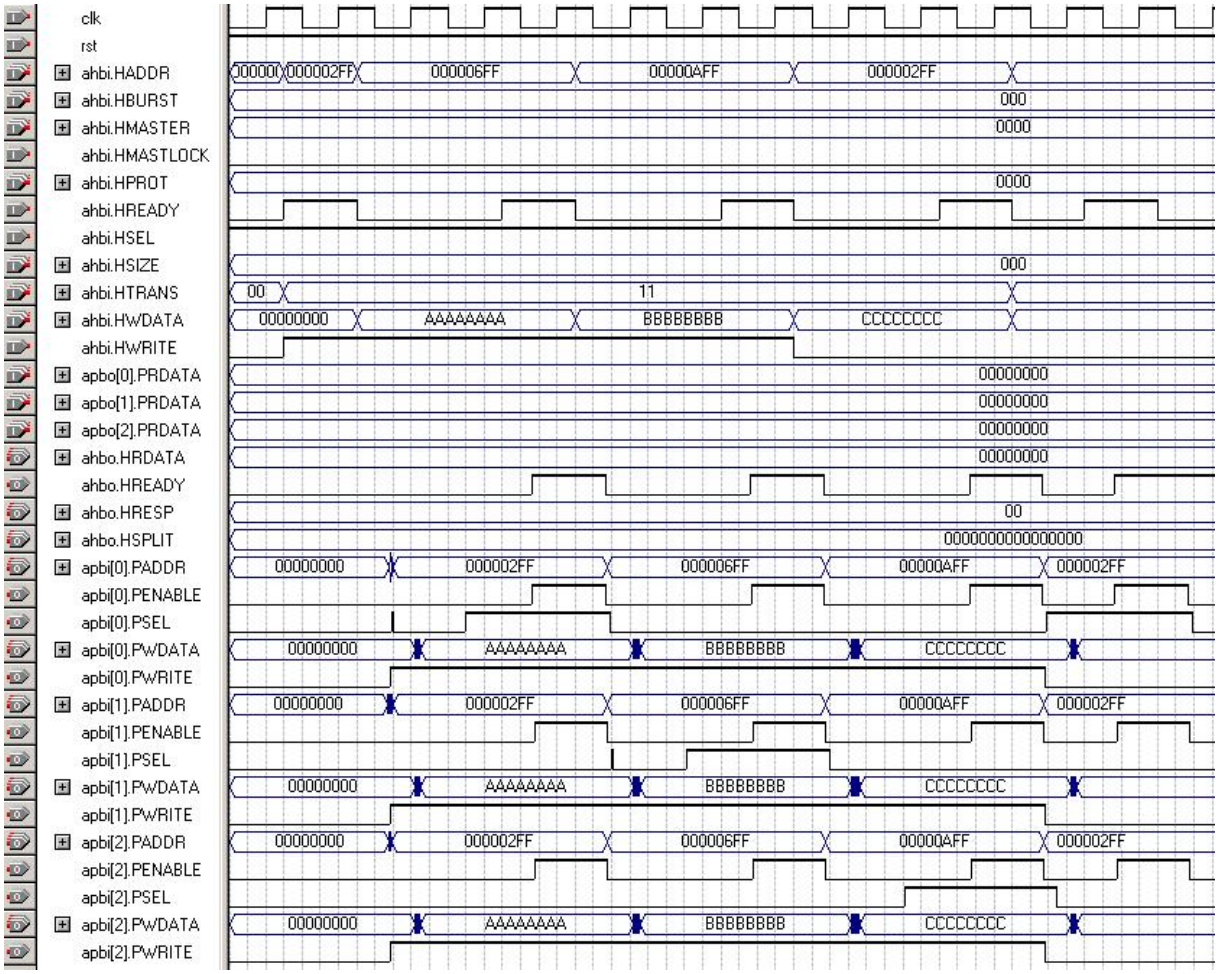

Figura B.4: Operações de escrita da ponte AHB/APB.

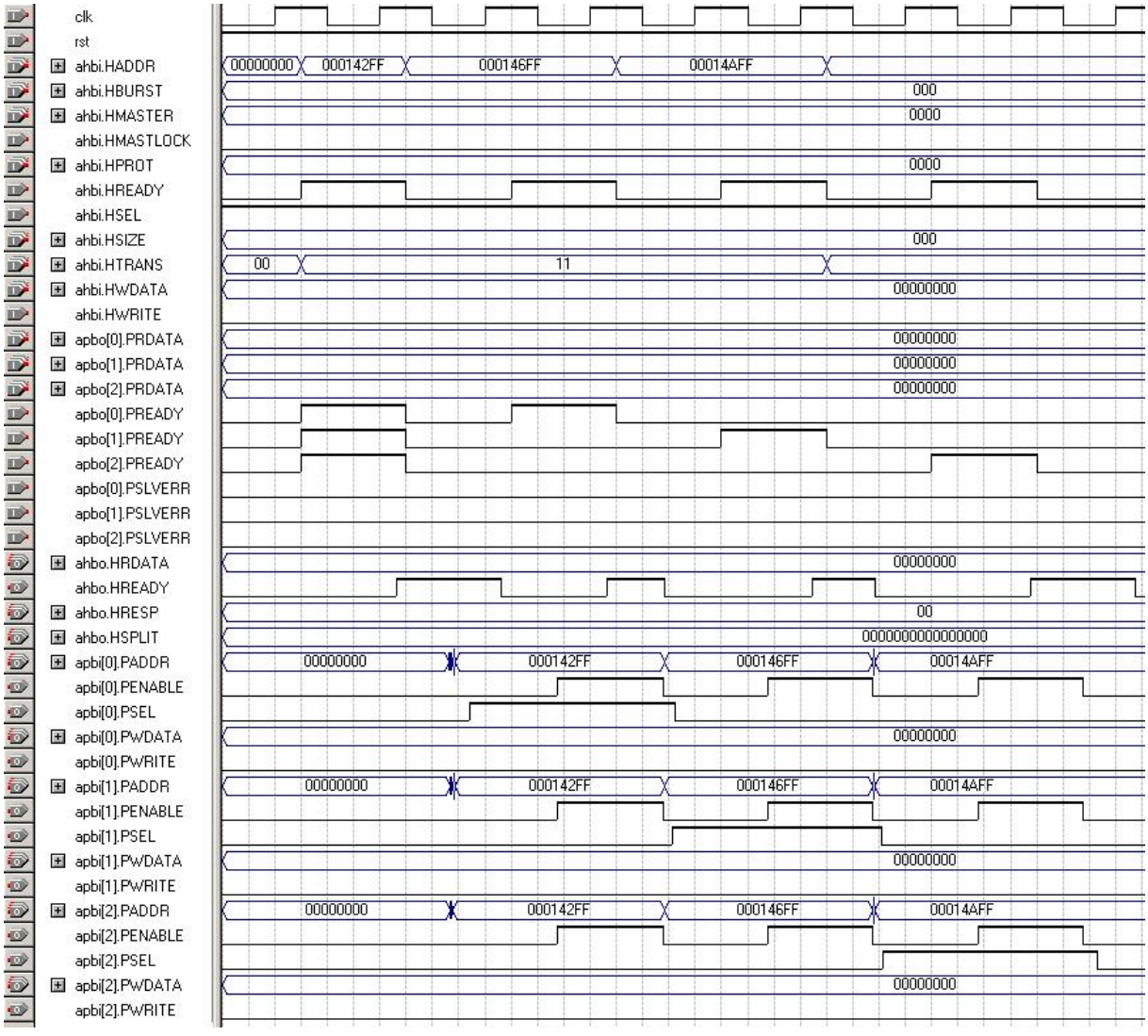

Figura B.5: Operações de leitura da ponte AHB/APB3.

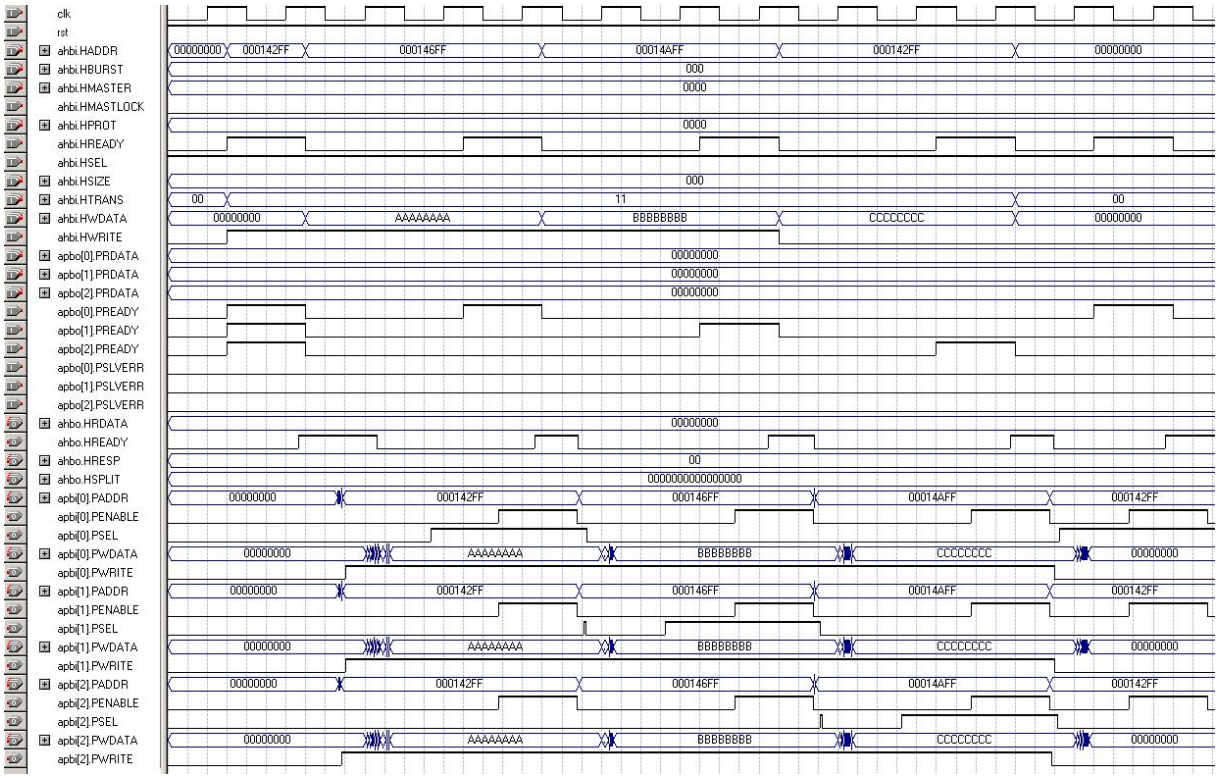

Figura B.6: Operações de escrita da ponte AHB/APB3.

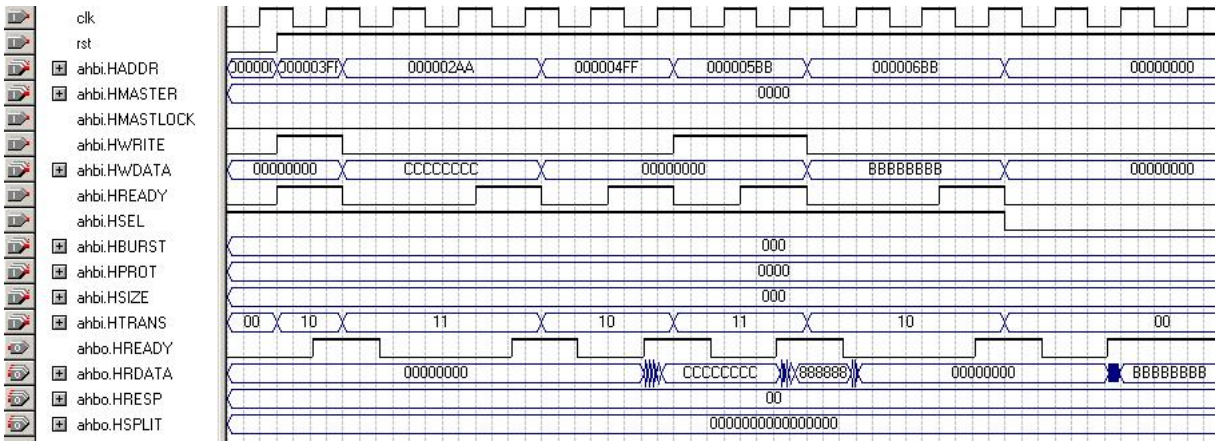

Figura B.7: Simulação do APB_System.

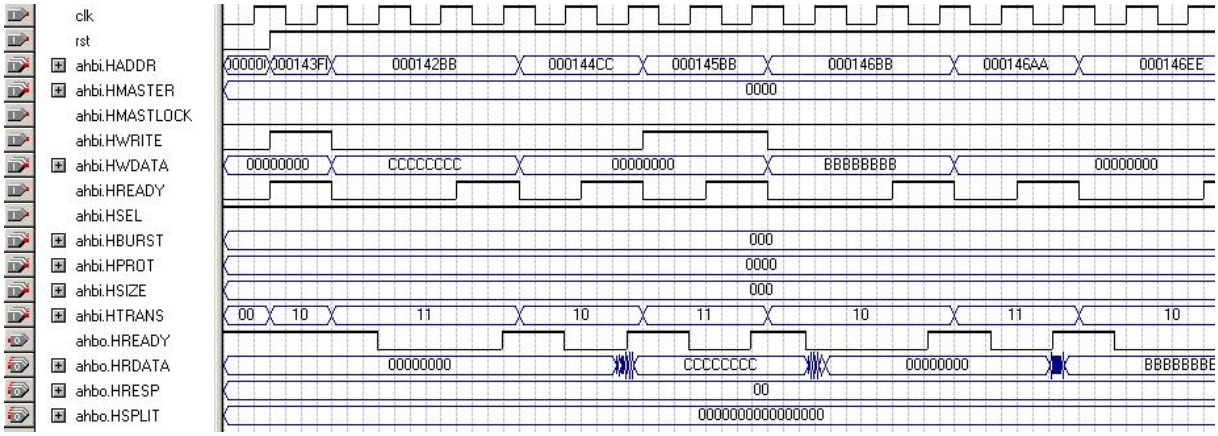

Figura B.8: Simulação do APB3_System.

diferentes tipos de respostas de acordo com o endereço que for recebido. Ele sempre irá colocar o endereço recebido como dado de leitura. A tabela B.1 mostra as respostas relativas a cada endereço.

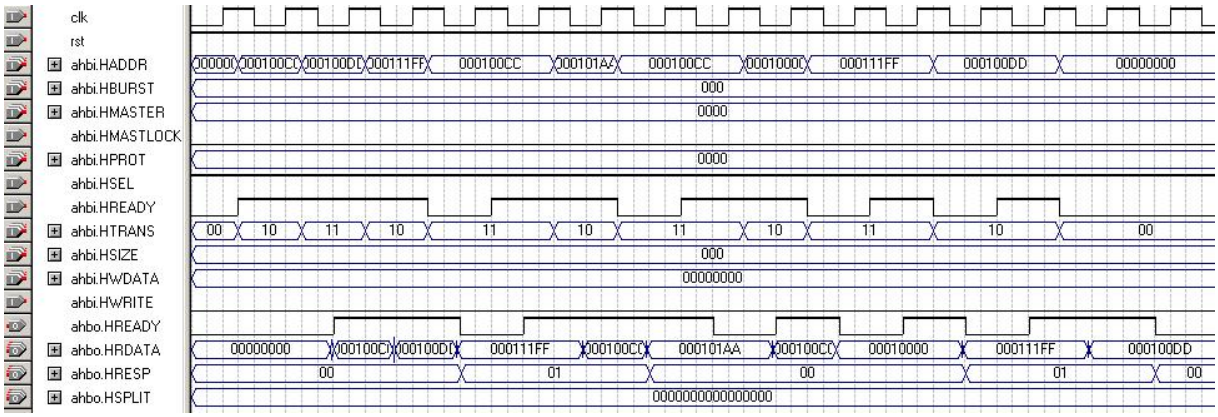

Figura B.9: Simulação do escravo 1 AHB.

Endereço	**Resposta**
0x000100--	OKAY (menos no caso 0x00010000)
0x00010000	Wait State
0x000101--	RETRY
0x000110--	SPLIT
0x000111--	ERROR

Tabela B.1: Respostas do escravo 1 AHB atribuídas aos endereços.

A figura B.10 mostra o resultado da simulação com o escravo 2 AHB (que possui uma memória de leitura/escrita e uma memória somente leitura). Ele seleciona a posição de memória que deve ser acessada de acordo com o endereço de entrada. Caso alguma operação de escrita aconteça na memória ROM, o escravo envia uma resposta de erro

(HRESP = ERROR). A tabela B.2 mostra as possíveis ações em cada endereço.

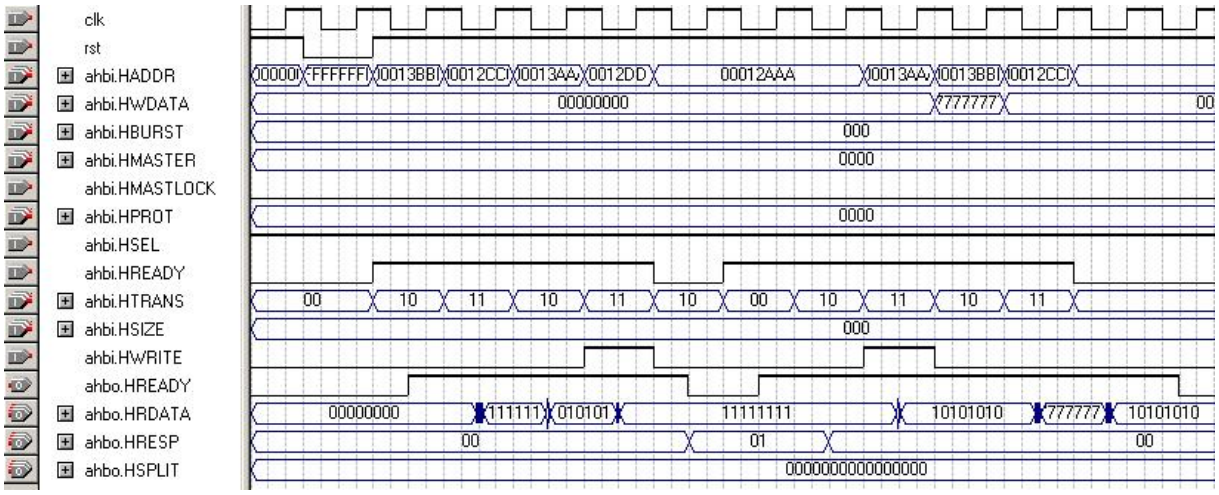

Figura B.10: Simulação do escravo 2 AHB.

Endereço	**Tipo de memória**
0x00012---	somente leitura
0x00013---	leitura/escrita

Tabela B.2: Tipo de memória do escravo 2 AHB de acordo com o endereço.

A figura B.11 mostra o resultado da simulação com o mestre 1 AHB. Ele repassa os sinais de entrada para o barramento. Há algumas ações realizadas automaticamente, como: enviar o HTRANS certo dependendo das respostas recebidas pelo mestre. Ele funciona como se fosse uma interface mestre.

As figuras B.12, B.13 e B.14 mostram o funcionamento do árbitro do sistema. Alguns sinais foram omitidos, pois não influenciam no resultado.

As figuras B.15, B.16, B.17, B.18, B.19 e B.20 mostram simulações feitas com o AMBA_System, onde várias operações de leitura e escrita são realizadas com os escravos AHB e APB.

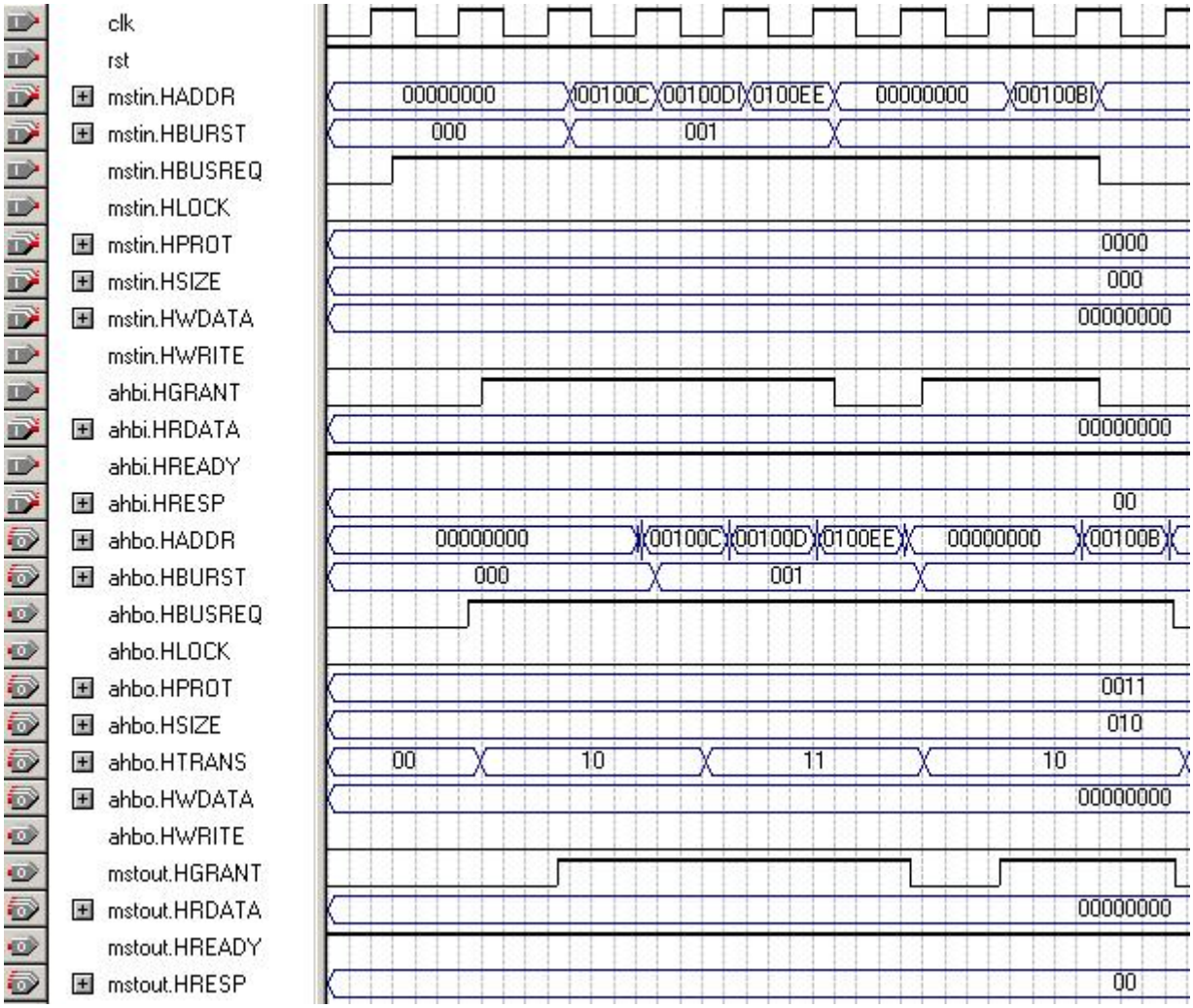

Figura B.11: Simulação do mestre 1 AHB.

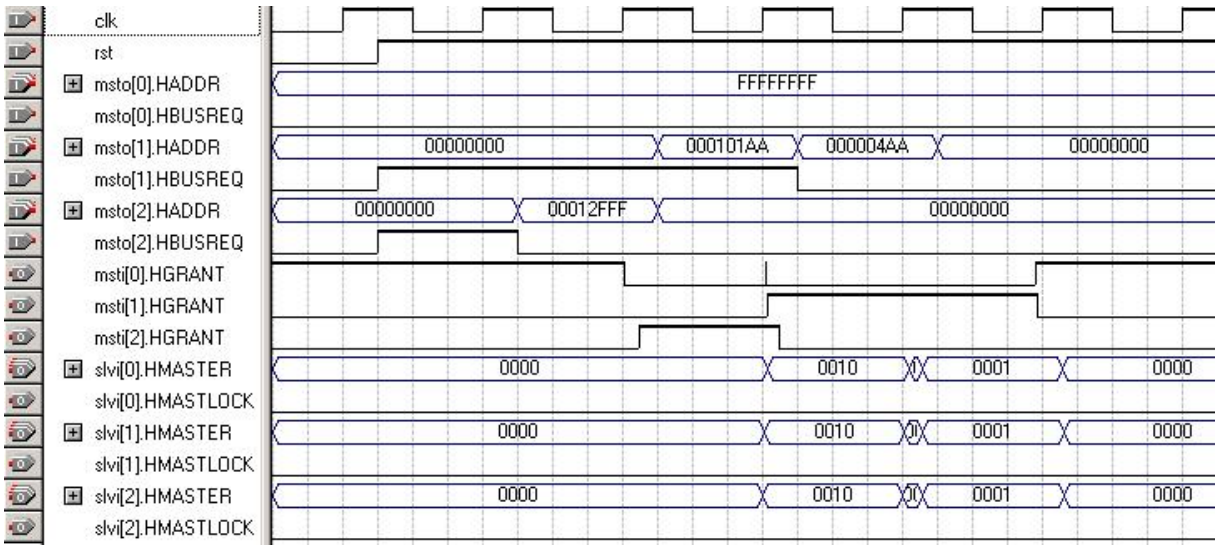

Figura B.12: Simulação da arbitragem por prioridade.

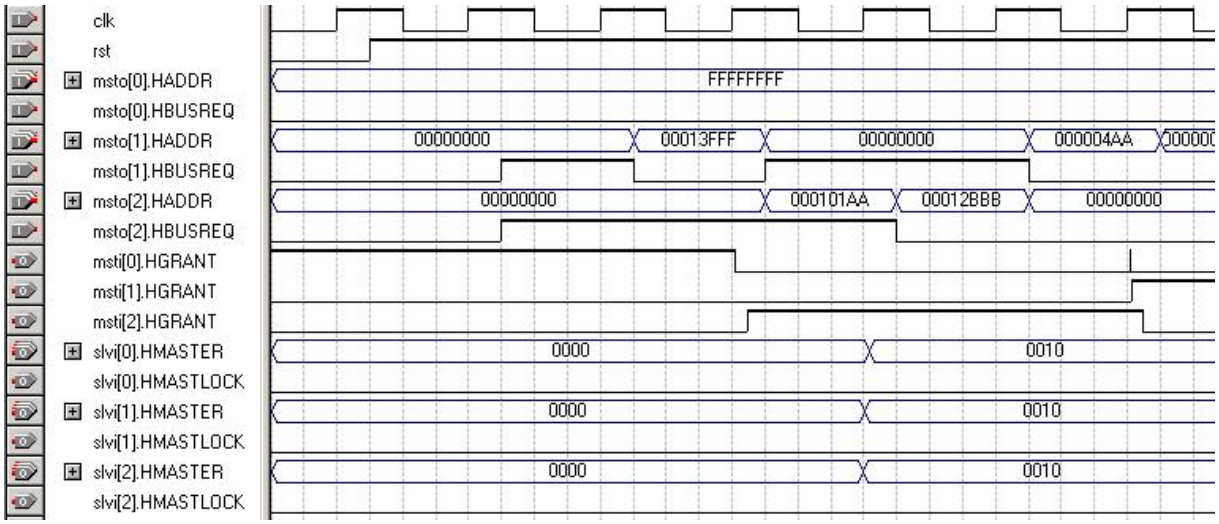

Figura B.13: Simulação usando a arbitragem Round Robin.

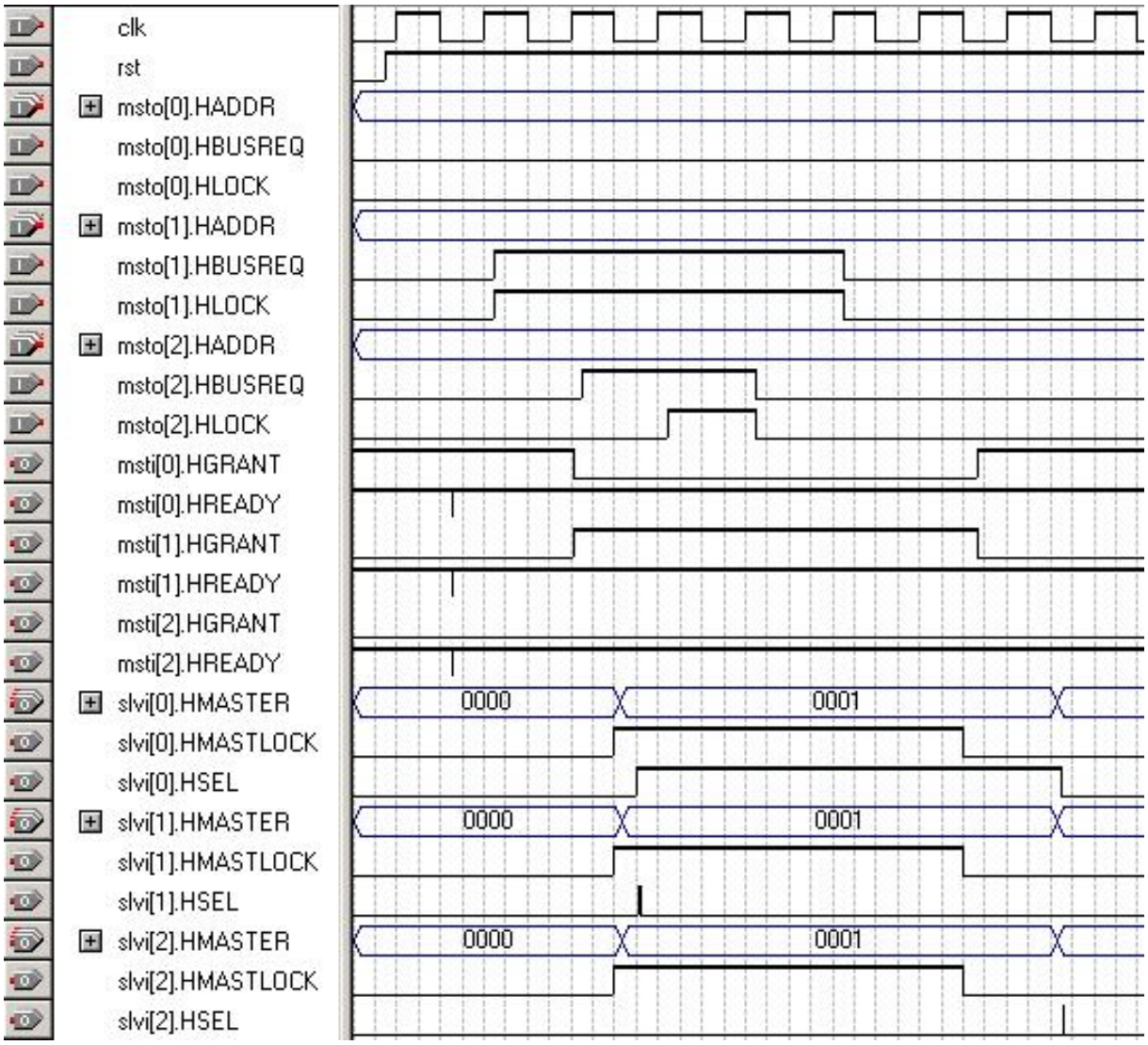

Figura B.14: Simulação da arbitragem com transferências travadas (*locked*).

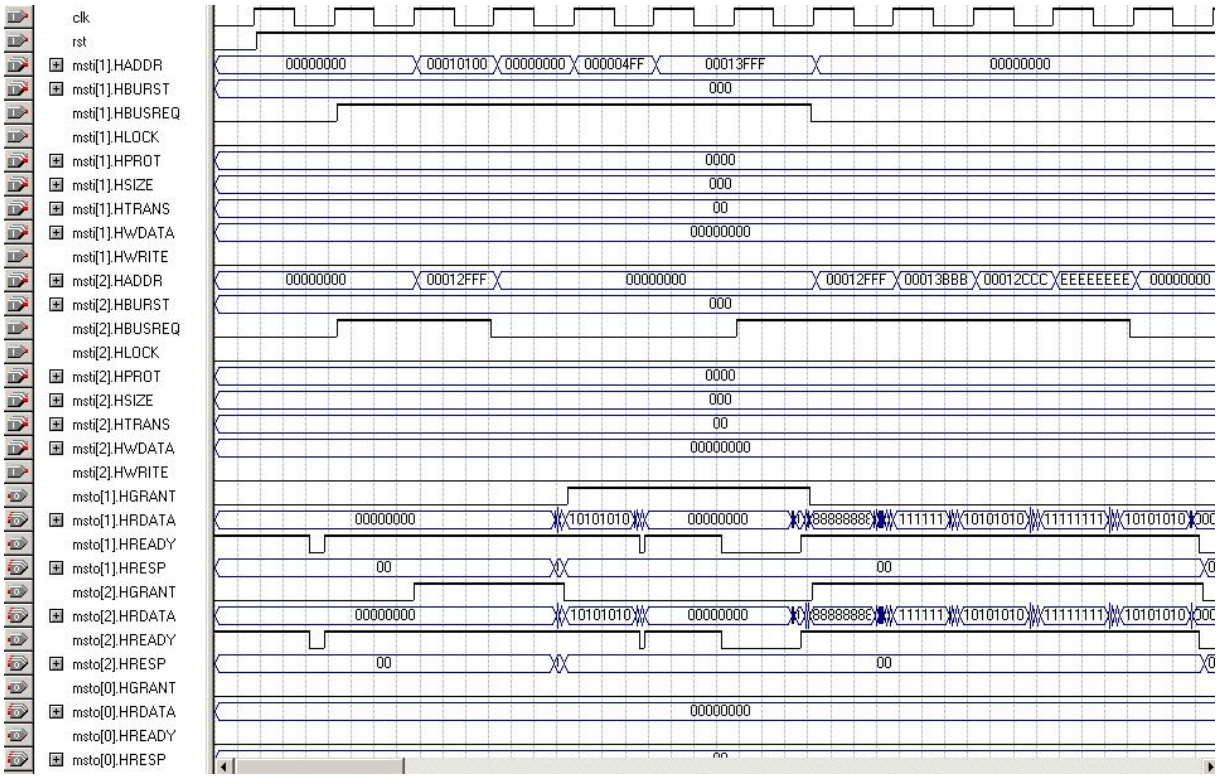

Figura B.15: Operações de leitura e escrita, através da arbitragem por prioridade, com os escravos do sistema.

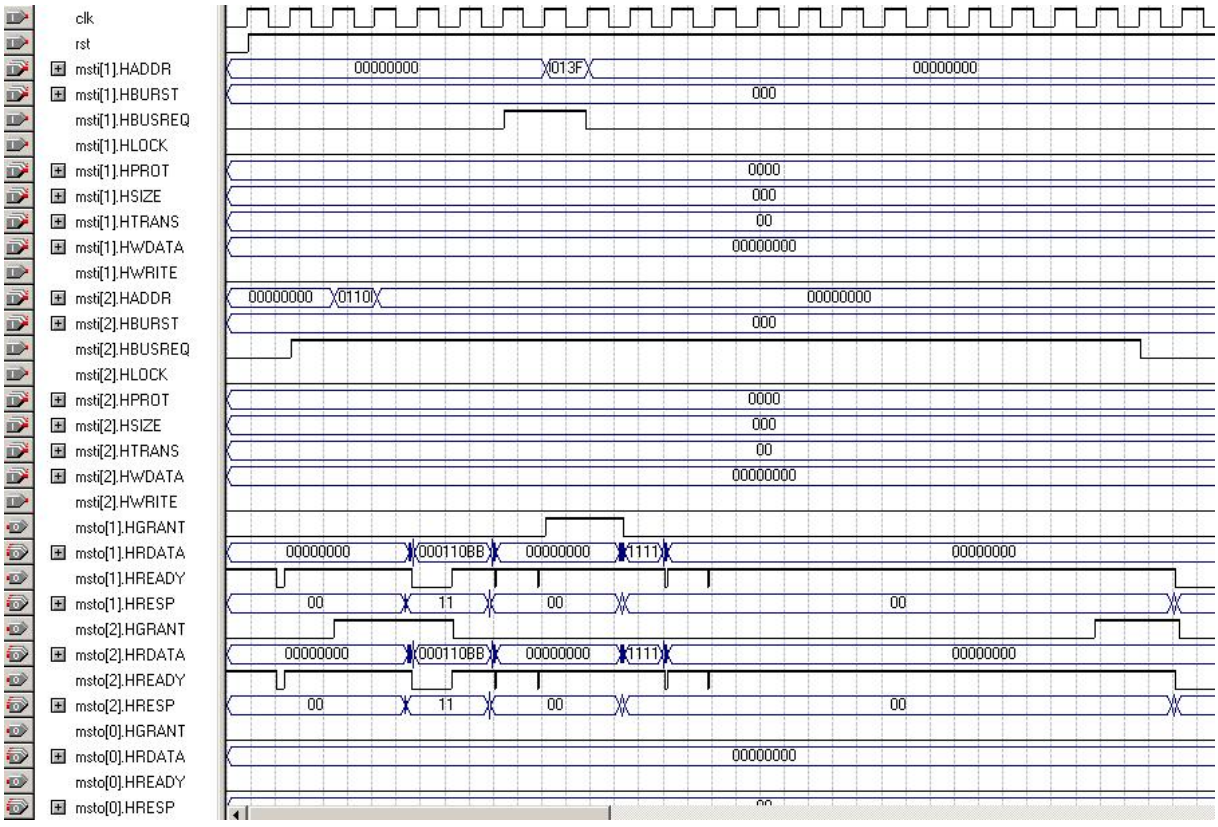

Figura B.16: Simulação de transferência *split* por parte de um dos mestres e arbitragem por prioridade.

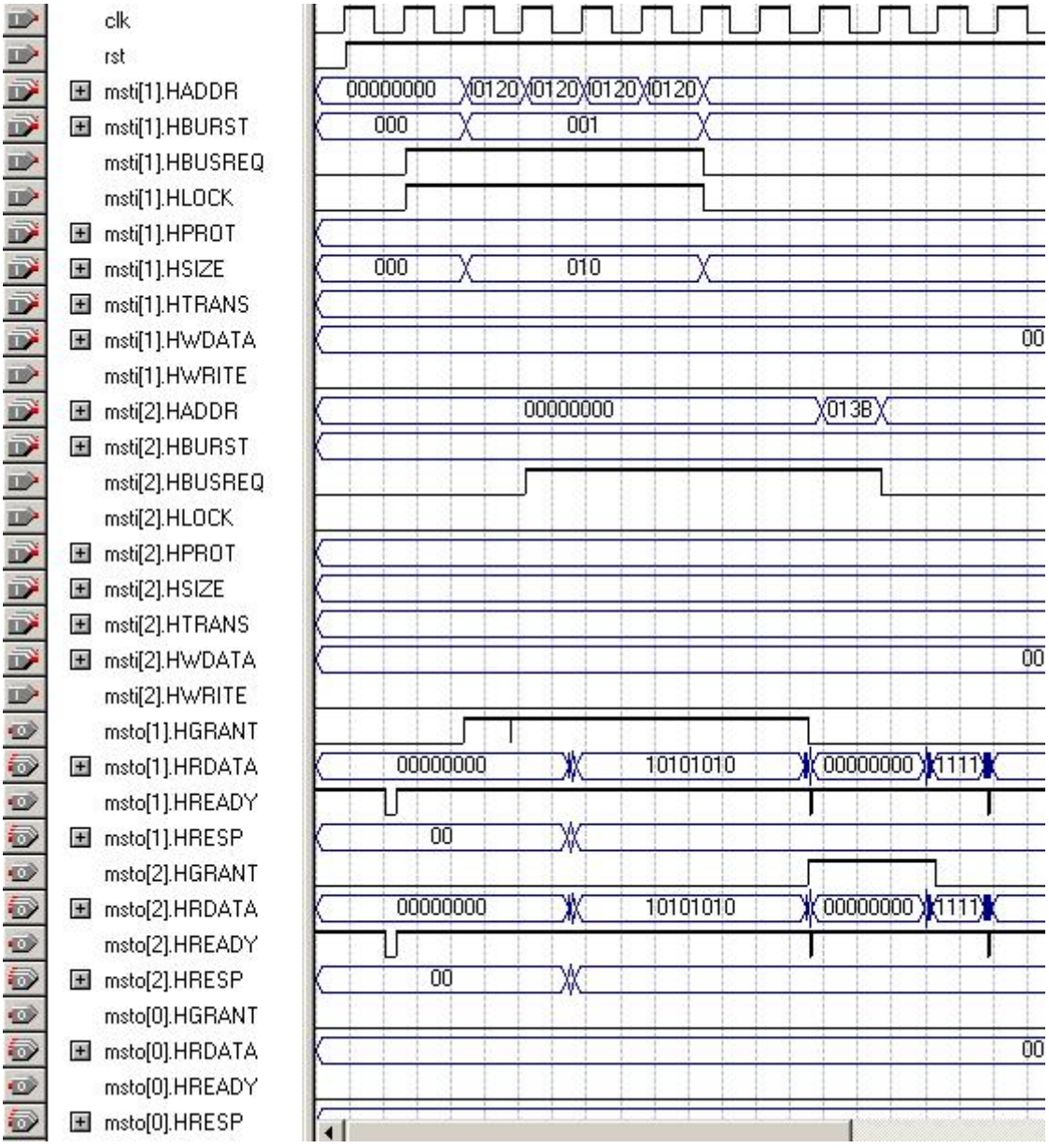

Figura B.17: Simulação de transferência *burst* e arbitragem por prioridade.

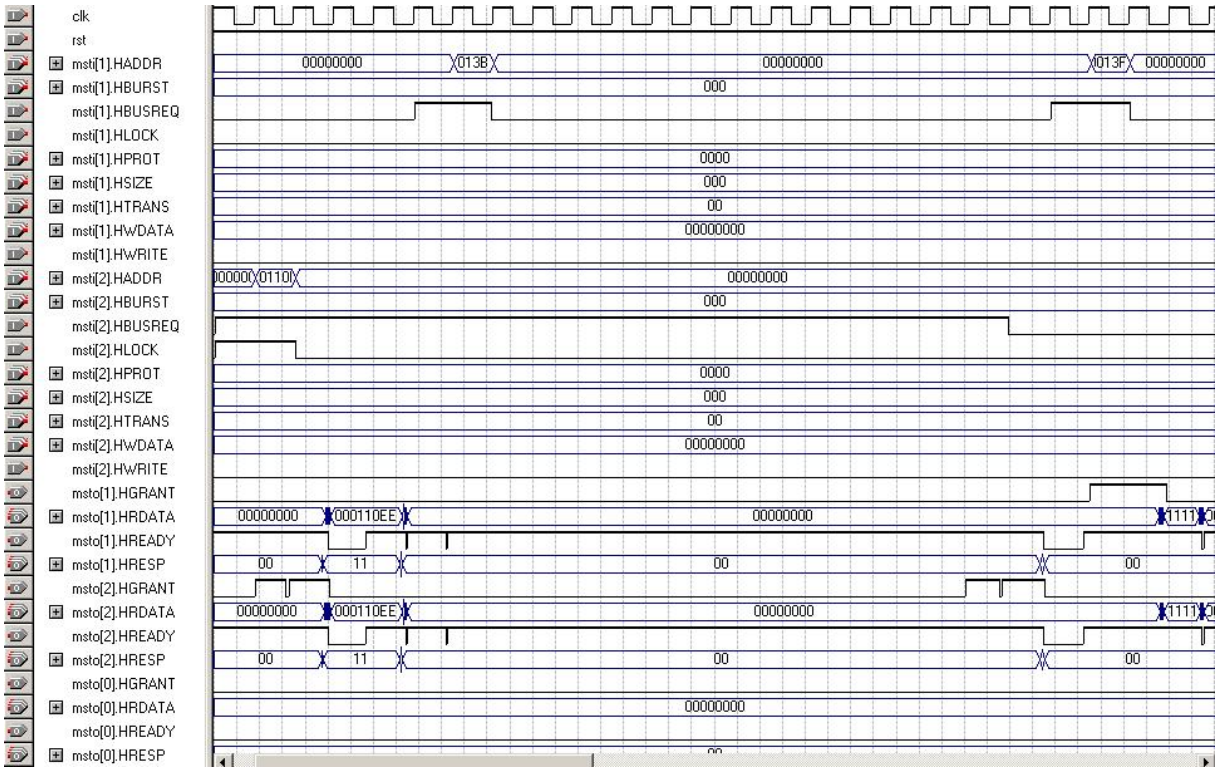

Figura B.18: Simulação de transferência travada (*locked*) que recebe uma resposta *split*.

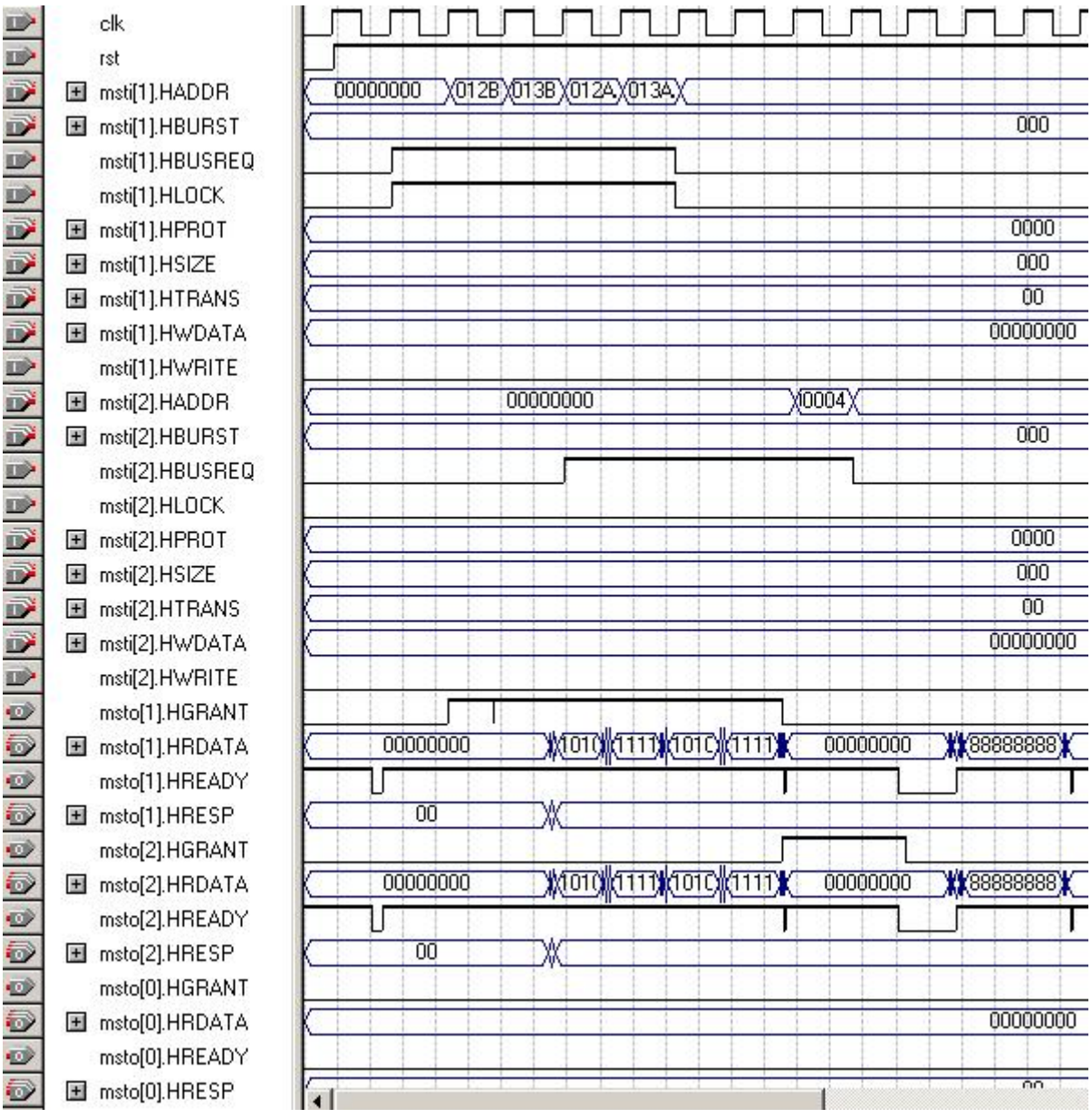

Figura B.19: Simulação com transferência travada (*locked*) e arbitragem por prioridade.

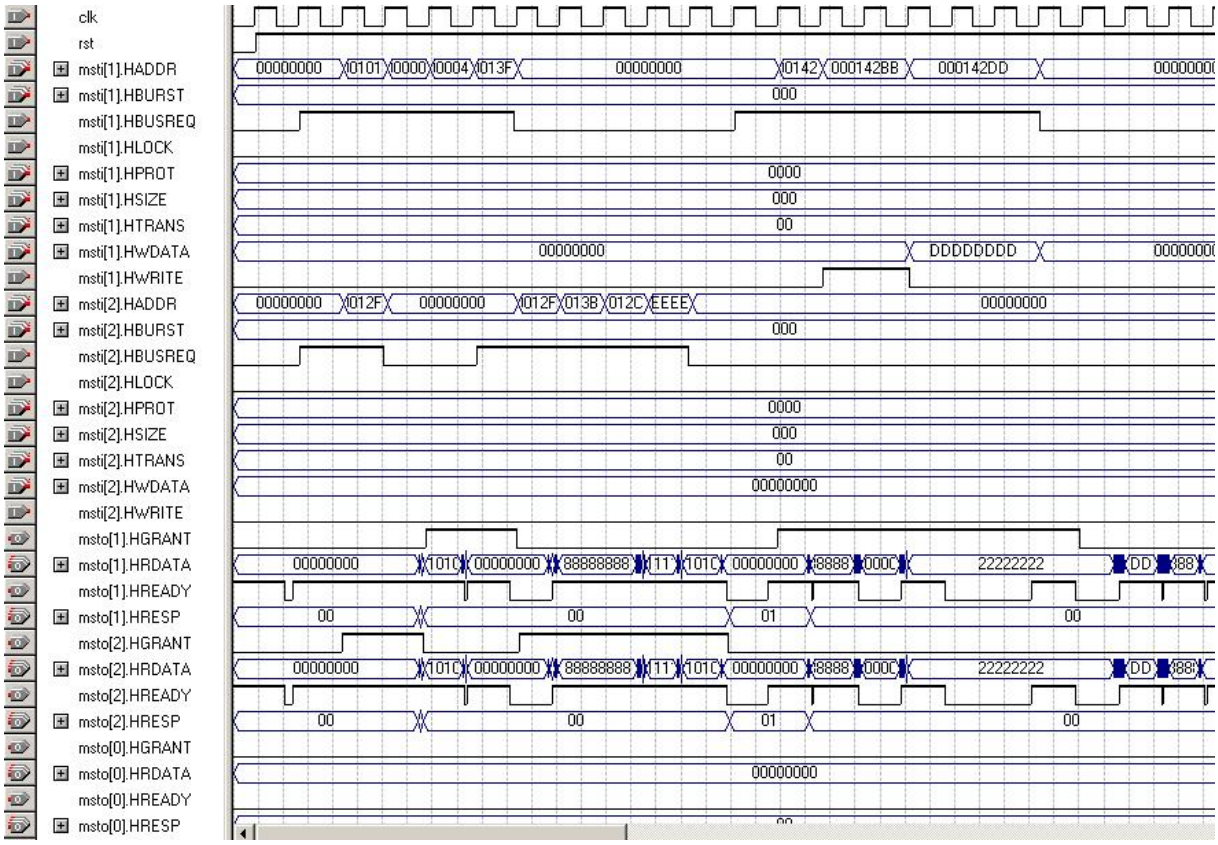

Figura B.20: Transferências com os escravos do sistema (incluindo APB3_System).