ARP: Um Gerenciador de Pacotes para Sistemas Embarcados com Processadores Modelados em ArchC

Rodolfo Azevedo, Bruno Albertini, Sandro Rigo

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6172 – 13083-852 – Campinas – SP – Brazil

{rodolfo,balbertini,sandro}@ic.unicamp.br

Resumo. *Este artigo apresenta a ferramenta ARP, criada para facilitar o gerenciamento de projetos de sistemas embarcados. A ARP permite uma melhor organização dos componentes do sistema, isolando-os de forma a facilitar o reuso e a exploração de espaço de projeto dos sistemas. Na versão atual, ela foi desenvolvida para utilizar a linguagem SystemC. O artigo apresenta dois exemplos dos casos de uso principais: um com finalidade didática e outro focado em pesquisa de memórias transacionais. A ARP está disponível para download através da página da internet do ArchC.*

1. Introdução

A análise de espaço de projeto dos sistemas embarcados requer a criação de múltiplas versões do mesmo sistema, que são diferentes entre si desde pequenos parâmetros até grandes componentes. Assim, o gerenciamento das múltiplas versões desta plataforma, seus parâmetros e componentes é uma tarefa essencial no desenvolvimento dos novos sistemas embarcados.

Este trabalho apresenta as linhas gerais da ferramenta ARP, algumas de suas plataformas de exemplo e casos de uso. O foco principal da ferramenta foi a facilidade de uso, com suporte para o gerenciamento, simultâneo, de diversas configurações do sistema. Desta forma, a ARP facilita a exploração do espaço de projeto, bem como a validação de parâmetros como escalabilidade do número de componentes do sistema.

Do ponto de vista da ARP, um projeto é denominado plataforma, que possui diversos componentes. A versão atual da ARP está desenvolvida em SystemC, embora os conceitos possam ser utilizados em outras linguagens de descrição de sistemas se necessário.

Este artigo está organizado da seguinte forma: A Seção 2 apresenta os trabalhos relacionados à ARP, seguida pela Seção 3 que apresenta uma visão geral da ferramenta. A Seção 4 descreve as funcionalidades fornecidas pela ARP, seguida pelas as Seções 5 e 6 que descrevem dois casos de uso da ARP. Por fim, a Seção 7 conclui o artigo.

2. Trabalhos Relacionados

O gerenciamento de plataformas virtuais tem obtido destaque em diversas ferramentas comerciais da atualidade, sejam elas produtos de fabricantes de EDA como as aquisições da Virtio e CoWare pela Synopsys, que passou a oferecer os produtos Innovator, Platform Architect, CoMet e METeor [Synopsys 2010]. A Altera, na linha de FPGAs, oferece o

SOPC Builder integrado ao Quartus II [Altera 2010]. No outro extremo, a GreenSocs [GreenSocs 2010] oferece um conjunto de ferramentas para gerenciar plataformas baseadas no modelo de distribuição do Debian Linux.

As duas primeiras (Synopsys e Altera) oferecem ferramentas com interfaces gráficas, com suporte a diversos componentes proprietários, que podem ser acoplados aos componentes desenvolvidos pelos usuários. Uma particularidade da Synopsys é o suporte ao padrão TLM 2.0, que permite maior integração entre os componentes. No caso da Altera, a grande facilidade de integração com a ferramenta de sintese para FPGA é o destaque.

Já a GreenSocs, por ter foco na oferta de serviços de projeto, fornece uma infraestrutura mais focada na distribuição de componentes e na integração entre eles.

Um ponto fraco dos projetos acima é a flexibilidade de gerenciamento dos componentes da plataforma. As versões comerciais amarram os componentes às suas próprias plataformas enquanto que a GreenSocs foca todo o seu trabalho ao redor do GreenBus, para fornecer a interconexão.

A ARP tem por objetivo apenas gerenciar as plataformas e seus componentes, dando flexibilidade ao projetista de escolher que componentes utilizar, bem como a forma de utilizá-los.

3. Visão Geral da Ferramenta

Após instalar a configuração básica da ARP, o usuário terá uma estrutura de diretórios com os componentes indicados abaixo:

processors: Diretório que contém modelos de processadores. Embora este artigo foque nos processadores descritos em ArchC, esta não precisa ser a única fonte de processadores para a plataforma.

is: Diretório que contém os elementos de interconexão disponíveis para construção de plataformas, como barramentos, roteadores, NoCs, etc.

ip: Diretório que contém os IPs disponíveis no momento. Cada novo IP, colocado neste diretório, poderá ser utilizado por qualquer plataforma disponível.

sw: Diretório que contém os softwares que serão executados nas plataformas. Em linhas gerais, um mesmo software pode ser utilizado em mais de uma plataforma. No entanto, como é comum a mudança de parâmetros na plataforma, o software terá que ser programado adequadamente para satisfazer a esta variedade de configurações.

wrappers: Diretório que contém os conversores de protocolos que podem ser utilizados para conectar componentes distintos das plataformas.

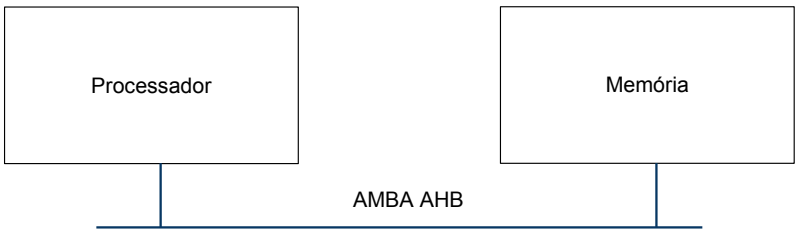
platforms: Diretório que contém as plataformas desenvolvidas. Cada versão da plataforma pode/deve ter um diretório diferente, facilitando a exploração do projeto e também o gerenciamento das configurações.

Junto com estes diretórios também há o utilitário `arp.py`[1], cujas funcionalidades serão descritas adiante e um `Makefile` que fornece os comandos de compilação, execução e gerência necessários.

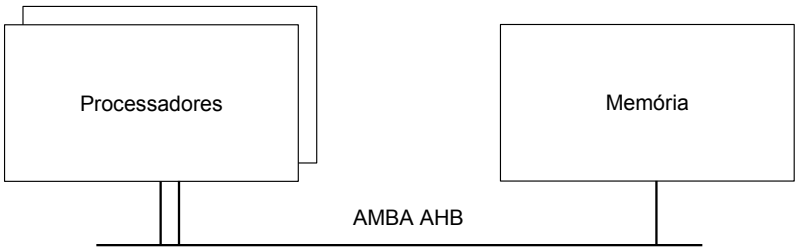
[1] A opção por manter o sufixo `.py` se deve ao conflito de nomes entre o gerenciador de plataformas e o software de rede de mesmo nome.

Embora, num primeiro momento, possa parecer uma organização não usual para uma plataforma, visto que os componentes ficam espalhados em diretórios distintos, compartilhados com outros componentes de outras plataformas, esta abordagem permite a uniformização dos componentes utilizados em mais de uma plataforma, gerando um ganho de organização ao não forçar o desenvolvedor a manter diversas cópias dos mesmos arquivos. Além disto, a ferramenta ARP possui um utilitário com funcionalidades de empacotar e desempacotar uma plataforma, permitindo a montagem de um arquivo com todos os componentes necessários para a execução da plataforma, transferência entre computadores e sua posterior restauração.

Para melhor exemplificar esta organização, suponha uma plataforma simples, como ilustrada na Figura 1(a), com um processador, um barramento e uma memória. Para armazenar esta plataforma, o diretório `processors` conteria um diretório com o código do processador (por exemplo: `sparc`), o diretório `is` conteria um diretório com o código do barramento (por exemplo: `amba_ahb`) e o diretório `ip` conteria um diretório com o código da memória. Como estes três componentes podem ser reutilizados por diversas plataformas, deverá ser criada um novo diretório, dentro de `platforms` com a instanciação dos componentes desejados, neste exemplo, este diretório poderia se chamar `simple_platform` e conteria o código em SystemC para instanciar os componentes necessários. Além dos diretórios mencionados, o software que será executado na plataforma deve ser desenvolvido e incluído dentro do diretório `sw`.



(a) Plataforma Simples



(b) Plataforma *Dual Core*

Figura 1. Duas plataformas de exemplo

Se for necessário, em algum momento posterior, criar uma versão com dois cores desta plataforma, como ilustrado na Figura 1(b), basta criar um novo diretório dentro de `platforms`, que conterá o código SystemC que instancia os dois processadores. Neste exemplo simples, como o código SystemC será muito semelhante ao da

simple_platform, basta copiar o arquivo principal do diretório anterior e realizar as modificações necessárias. Não é preciso replicar nenhum código dos componentes caso eles já possuam o suporte necessário a esta plataforma.

4. Facilidades Fornecidas pela ARP

As duas primeiras funcionalidades a destacar da ferramenta arp.py estão relacionadas com o armazenamento das plataformas. O parâmetro --pack empacota uma plataforma, gravando em um único lugar todos os arquivos utilizados durante o desenvolvimento e execução da mesma. O nome da plataforma deve ser fornecido como parâmetro. O arquivo gerado pode ser enviado para outro computador que, através das mesmas ferramentas, será capaz de reconstruir toda a árvore de diretórios e reexecutar a plataforma. Para desempacotar a plataforma, basta utilizar o parâmetro --unpack, fornecendo o arquivo criado anteriormente.

Além da facilidade de empacotar e desempacotar plataformas, que permite a transferência fácil dos componentes de um computador para outro, a organização dos componentes na ARP permite um melhor gerenciamento do processo de compilação: cada diretório de cada componente tem um arquivo Makefile, que contém instruções de compilação específicas com a finalidade de criar uma biblioteca com este componente. O diretório da plataforma contém um Makefile que compila todos os seus arquivos e realiza sua ligação com as bibliotecas dos componentes necessários. O diretório principal da ARP contém também um Makefile que executa os comandos de compilação apenas nos componentes necessários da plataforma, utilizando o mesmo conceito das bibliotecas já mencionado. Apenas o Makefile principal e o da plataforma precisam ser alterados quando da construção de uma nova plataforma. A listagem dos componentes que a plataforma requer é feita num arquivo chamado defs.arp, que fica dentro do diretorio da plataforma. Este arquivo é incluído no Makefile principal, ajudando no gerenciamento do processo de compilação. A listagem da Figura 2 mostra um arquivo defs.arp de exemplo indicando os componentes necessários para executar um programa "Hello World" numa plataforma simples como a da Figura 1(a).

```
1 IP := ac_tlm_mem
2 IS := ac_tlm_bus
3 PROCESSOR := mips1
4 SW := hello_world
5 WRAPPER :=
```

Figura 2. Exemplo do arquivo defs.arp

A primeira linha do arquivo de exemplo contém o nome do IP que implementa uma memória (ac_tlm_mem). O dispositivo de interconexão, neste caso, é um barramento genérico (ac_tlm_bus), o processador é um MIPS (mips1) descrito através da ADL ArchC e o software é o hello_world. Caso mais de um componente da mesma categoria seja necessário, basta colocar todos na mesma linha, com os nomes separados por espaços.

Uma vez que um componente possui, em seu Makefile, todos os comandos necessários para sua compilação e uso pelas plataformas, sua reutilização tende a aumentar.

Exemplos destes `Makefile` são fornecidos junto com o pacote ARP. Os comandos que devem ser suportados são:

all: Este é o padrão para todo `Makefile` e significa compilar todo o componente.
clean: Apaga os arquivos gerados pela compilação do componente. Todos estes arquivos podem ser regerados ao executar um novo `make all`.
distclean: Apaga os arquivos gerados por ferramentas auxiliares, como ArchC. Como exemplo, todos os arquivos C++ gerados pelo gerador de simuladores (`acsim`) são apagados. Eles também podem ser reconstruídos através do comando `make all`.
lib: Compila o componente atual e armazena numa biblioteca. Assim fica mais fácil reutilizá-lo. O próprio `Makefile` principal chama esta regra para os diretórios de componentes.
run: Executa o programa na plataforma atual. Para isto, se necessário, o programa sera compilado e a plataforma será executada recebendo o programa como parâmetro.

A vantagem de se criar bibliotecas com cada um dos componentes é a facilidade de passá-las como parâmetros para as plataformas através das opções `-l` e `-L` do `gcc`, como é feito por padrão pelo `Makefile` principal.

Outro aspecto importante para facilitar a interconexão dos componentes é a utilização de um padrão de comunicação como o TLM [Ghenassia 2005, Black et al. 2009] da OSCI [OSCI 2010]. Embora o suporte ou não à TLM não seja diretamente ligado às funcionalidades da ARP, todos os componentes distribuídos adotam este modelo de interface.

As próximas duas seções abordam dois casos de uso da ARP: uma plataforma multicore com finalidade educacional e outra plataforma, mais elaborada, com finalidade de pesquisa na área de memórias transacionais. Ambas as plataformas são distribuídas sob demanda.

5. Plataforma multicore básica e seu uso educacional

Esta seção ilustra como transformar a plataforma da Figura 1(a) numa plataforma multicore básica, como a versão dual-core da Figura 1(b) e como esta atividade está, atualmente, sento utilizada para fins educacionais. Será fornecida uma descrição mais detalhada da implementação do componente de gerenciamento de concorrência, por ser de grande reuso e depois serão mostradas as atividades que são fornecidas aos alunos na montagem da plataforma multicore.

5.1. Primitivas para controle de concorrência

Um dos primeiros requisitos importantes de uma plataforma multicore é o fornecimento de mecanismos para controle de concorrência. Os mais simples e efetivos são instruções atômicas como *test-and-set*, *load-and-increment* e *compare-and-swap*. Estas instruções precisam ser implementadas nos processadores mas, devido às restrições de atomicidade, precisam também de um suporte da plataforma. Uma outra alternativa, com boa efetividade, é trabalhar com um periférico (IP) que implemente um mecanismo atômico em hardware, como no caso da plataforma MPARM [Loghi et al. 2004]. O MPARM implementa um número restrito de *locks*, que podem ser utilizados na implementação de

uma biblioteca de controle de concorrência geral. Devido aos requisitos da plataforma alvo, serão utilizados como exemplo, sem perda de generalidade, os componentes básicos (processador e protocolo de interconexão) já disponíveis na ARP.

Não seria uma boa prática de projeto escolher uma das instruções atômicas mencionadas e implementá-la diretamente num processador sem antes verificar o suporte do seu conjunto de instruções. Como as três instruções mencionadas são equivalentes[2], apenas a instrução *test-and-set* será descrita a seguir.

A instrução *test-and-set* possui dois parâmetros: valor a comparar e endereço de memória. Uma versão primitiva dela pode operar, atomicamente, da seguinte forma: primeiro lê o valor armazenado no endereço. Depois compara este valor com o fornecido pela instrução, se eles forem diferentes, grava o novo valor na memória e retorna o valor antigo. Se os valores forem iguais, retorna o valor sem afetar a memória. Com base no valor de retorno é possível saber se a instrução foi bem sucedida ou não e implementar um *mutex*. O grande requisito, neste momento, é o suporte a operações atômicas no barramento e, potencialmente, os demais componentes entre o processador e a memória.

A implementação desta operação atômica pode ser feita através da implementação TLM disponível nos modelos de processador gerados por ArchC, que suportam a operação *lock* para bloquear um endereço dos componentes onde os processadores estão conectados. Assim, é necessario adquirir um *lock* através da interface TLM e, depois, efetuar as operações descritas anteriormente, liberando o *lock* a seguir.

A segunda alternativa, implementação de um periférico para gerenciar um *lock* em hardware. Uma especificação possivel para um único *lock* é a seguinte: o periférico contém um registrador de *lock*, inicialmente com valor zero. Toda leitura retorna o valor disponível no registrador de *lock*, alterando seu valor para 1. Toda escrita grava o valor fornecido no registrador. A Figura 3 ilustra os passos necessários para utilizar efetivamente o *lock* em hardware. O primeiro passo é a declaração de um ponteiro para a posição de memória onde o *lock* está localizado. A seguir, o laço aguarda pela liberação do *lock* (no nosso exemplo, o *lock* está liberado quando tiver valor 0). A seguir é executada a região crítica e o *lock* é liberado através da escrita do valor zero nele liberando, potencialmente, outras *threads* concorrentes.

```
1 volatile int *lock = (int *) ENDERECO_LOCK; // Ponteiro para o
    lock em hardware
2
3 while (*lock); // Obtem o lock
4   // trecho do programa a executar
5 *lock = 0; // Libera o lock
```

Figura 3. Exemplo de uso de um lock em hardware

Através desta implementação, é possível criar duas primitivas `AquireGlobalLock` e `ReleaseGlobalLock`, como mostrada na Figura 4, que são fragmentos do código de exemplo da Figura 3.

Estas duas funções são suficientes para gerenciar uma outra posição de memória

[2]A partir de qualquer uma delas, é possível implementar a funcionalidade das outras duas.

```
1  volatile int *lock = (int *) ENDERECO_LOCK;
2
3  void AquireGlobalLock()
4  {
5    while (*lock);
6  }
7  void ReleaseGlobalLock()
8  {
9    *lock = 0;
10   }
```

Figura 4. Implementação das funções de acesso ao *lock* global

de forma similar a um lock, como mostram as implementações da Figura 5. Tanto a função `AquireLock` quanto a `ReleaseLock` primeiro obtém o *lock* global e depois fazem a alteração no *lock* local. Assim, garantidamente somente uma *thread* do programa terá acesso, por vez, às variáveis dos *locks* locais.

```
1  void AquireLock(int *l)
2  {
3    int lockValue = 0;
4
5    while (!lockValue) {
6      AquireGlobalLock();
7        if (! *l)
8          lockValue = *l = 1;
9      ReleaseGlobalLock();
10   }
11 }
12 void ReleaseLock(int *l)
13 {
14   AquireGlobalLock();
15   *l = 0;
16   ReleaseGlobalLock();
17 }
```

Figura 5. Implementação das funções de *lock* local baseadas no *lock* globall

Tanto o componente quanto o software gerados são de grande reuso, fornecdendo um bom mecanismo de controle de concorrência no sistema.

5.2. Atividades seguintes

Geralmente a criação dos componentes de hardware (IP) e software (biblioteca) para controle de concorrência são feitas como atividades iniciais para adaptação ao uso da ARP. A seguir, as atividades de uso de plataformas e montagens de sistemas dedicados tomam forma na disciplina. A seguir são descritas, brevemente, algumas das atividades:

Definição do espaço de endereçamento dos componentes: Dado um conjunto de IPs, incluindo a memória, definir o espaço de endereçamento dos periféricos, configu-

rar o barramento para ativá-los nos momentos corretos e demonstrar o funcionamento através de um software que verifica as funcionalidades implementadas. É neste momento que o IP de gerenciamento de *locks* é inserido na plataforma.

Ampliação do número de processadores: O próximo passo é aumentar o número de processadores da plataforma, tornando-a realmente multicore. Para isto, são tratadas questões de individualização das pilhas, separação de linhas de execução (criação de *threads* sem infraestrutura de sistema operacional, divisão do software em partes paralelas, etc).

Testes de escalabilidade: O aumento no número de processadores permite testar a escalabilidade de todo o software desenvolvido e também avaliar os gargalos de concorrência por recursos globais compartilhados.

Migração de funcionalidades de software para hardware: Aqui o aluno é exposto às primeiras noções de hardware/software *co-design*. Após realizar um *profile* do código em execução, ele deve escolher funções importantes a serem migradas para hardware. Estas funções precisam ser implementadas como periféricos em alto nível de abstração. Tambem cabe ao aluno definir o protocolo de comunicação entre o software restante e o periférico, bem como encapsular as chamadas necessárias para obter a mesma funcionalidade no programa. É comum que os alunos encontrem e resolvam problemas relacionados com a diferença de endian das arquiteturas.

Uma característica importante do gerenciador de plataformas, bastante utilizada nos exercícios acima, é a facilidade de fornecer soluções parciais e também trocar componentes entre diversos computadores/usuários. Desta forma, os alunos podem trabalhar em grupos e garantir que as funcionalidades desenvolvidas por um deles será aproveitada corretamente pelo outro, além de garantir o completo funcionamento das plataformas no computador do docente que irá avaliá-la. Como exemplo, é comum a entrega de uma plataforma básica, como a versão de um único processador, através de um pacote ARP e a solicitar a resposta como um outro pacote ARP, agora com todos os componentes.

6. Plataforma multicore e seu uso na pesquisa de memórias transacionais

Dentre os projetos que contaram com suporte da ARP podemos destacar a criação de uma plataforma de prototipagem e simulação de sistemas de memória transacional com suporte específico em hardware.

A plataforma de prototipagem desenvolvida nesta pesquisa foi inicialmente baseada na proposta original de Herlihy e Moss [Herlihy and Moss 1993] e estendida para dar suporte a transações aninhadas. Neste trabalho, utilizamos modelos de processadores descritos em ArchC, um IP adicional para implementar as *caches* transacional e normal, um barramento para ligar as várias instâncias de processadores e alguns adaptadores.

Uma das tarefas necessárias ao desenvolvimento da plataforma foi a adição de novas instruções aos modelos funcionais das arquiteturas MIPS, SPARC e PowerPC. Este processo é bastante simples uma vez que os modelos são descritos em um nível de abstração bastante alto e revelam a flexibilidade obtida ao empregarmos uma ADL. Também introduzimos novos tipos de mensagem TLM para permitir a comunicação adequada entre processador e o controlador da cache, uma vez que a responsabilidade da implementação do comportamento das novas instruções é compartilhada entre processador e cache. Os modelos de processadores emitem apenas uma instrução a cada ciclo e

as instruções são executadas em sua ordem de emissão. Uma vez tendo os modelos de processadores disponíveis dentro do arcabouço, a tarefa de trocar o processador usado na plataforma se resume a uma simples alteração no arquivo de definições da mesma, como explicado anteriormente.

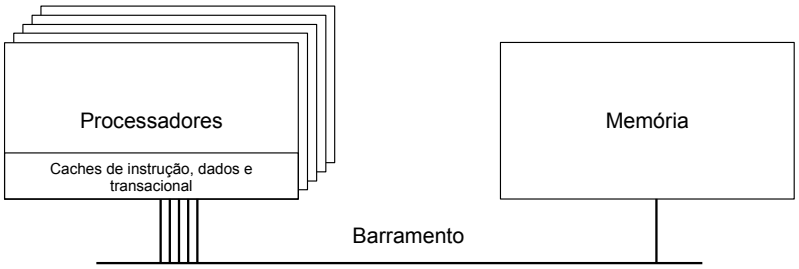


Figura 6. Plataforma de Prototipação de TM

A plataforma de prototipação está ilustrada na Figura 6. Neste ambiente, partimos de uma plataforma dual core e montamos diversas plataformas com números variados de processadores para explorar a escalabilidade do sistema de memória transacional modelado, avaliando de 2 até 256 processadores, com suas respectivas *caches* com suporte a memória transacional em hardware. Como o foco não era no mecanismo de interconexão, colocamos um barramento funcional para interligar os processadores. Essa plataforma pode ser facilmente instrumentada para colher estatísticas relacionadas ao sistema de TM, como número de abortos, confirmações, etc, e foi base do estudo publicado em [Kronbauer et al. 2007].

Partindo deste trabalho, algumas extensões podem ser feitas à plataforma como:

- Avaliação de mecanismos de interconexão como NoCs. O uso de um barramento real certamente degradaria o desempenho do sistema, como mencionado no próprio artigo [Kronbauer et al. 2007].
- Avaliação de protocolos de coerência de *caches*. Variações nos protocolos podem ser de extrema importância em situações de grandes conflitos das transações.
- Avaliação de *tradeoffs* entre memórias transacionais em software e hardware.

7. Conclusões

Gerenciar as diversas versões de um sistema embarcado pode levar a erros de inconsistências entre os componentes, seja de modificações necessárias ou desnecessárias. A ferramenta ARP surge neste contexto para fornecer um repositório organizado em diretórios e arquivos auxiliares para compilação, empacotamento e desempacotamento. Este artigo mostrou os dois principais ramos de uso da ARP atualmente: educacional, dando suporte a atividades de disciplinas de laboratórios e científico, dando suporte a pesquisas avançadas, como na área de memória transacional.

Alguns exemplos iniciais de plataformas, juntos com o pacote de gerenciamento, estão disponíveis no site da linguagem ArchC (http://www.archc.org). Um repositório com mais componentes está em fase de criação e a participação de uma comunidade pode ajudar no seu crescimento.

Como trabalhos futuros nesta direção estão: a criação de uma interface gráfica para gerenciamento dos pacotes, o suporte a padrões de empacotamento como SPIRIT e maior disponibilidade de componentes básicos para iniciar novas plataformas no repositório da ARP.

8. Agradecimentos

Este trabalho foi financiado com recursos da FAPESP, CNPq e CAPEs.

Referências

Altera (2010). Altera design software. online. `http://www.altera.com/products/software/sfw-index.jsp`.

Black, D., Donovan, J., Bunton, B., and Keist, A. (2009). *SystemC from the Ground Up*. Springer.

Ghenassia, F., editor (2005). *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer.

GreenSocs (2010). Green projects overview. online. `http://www.greensocs.com/node/1564`.

Herlihy, M. and Moss, J. E. B. (1993). Transactional memory: architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300.

Kronbauer, F., Baldassin, A., Albertini, B., Centoducatte, P., Rigo, S., Araujo, G., and Azevedo, R. (2007). A flexible platform framework for rapid transactional memory systems prototyping and evaluation. In *RSP '07: Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pages 123–129, Washington, DC, USA. IEEE Computer Society.

Loghi, M., Poncino, M., and Benini, L. (2004). Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 410–406, New York, NY, USA. ACM.

OSCI (2010). Open systemc initiative. online. `http://www.systemc.org/home/`.

Synopsys (2010). Synopsys system-level design tools. online. `http://www.synopsys.com/Tools/SLD/Pages/default.aspx`.