ArchC: Uma linguagem de descrição de arquiteturas

Javier A. Montoya Zegarra-Farach R.A: 041468
Instituto de Computação - Unicamp
E-mail: javier.montoya@ic.unicamp.br

ABSTRACT

Linguagens de descrição de arquiteturas (ADLs) estão sendo cada vez mais difundidas e utilizadas, devido a que estas permitem modelar e validar antecipadamente a arquitetura de uma aplicação. A geração automática de ferramentas a partir de descrições de ADLs proporciona uma maior flexibilidade aos projetistas de *hardware* durante a fase de exploração do projeto.

Este trabalho apresenta a linguagem de descrição de arquiteturas ArchC, a qual foi desenvolvida no Laboratorio de Sistemas de Computação (LSC) do Instituto de Computação da Universidade Estadual de Campinas (Unicamp). ArchC permite descrever arquiteturas em um alto nível de abstração e gera diversas ferramentas entre elas montadores, simuladores e interfaces de co-verificação.

Keywords

Linguagens de descrição de Arquiteturas, ArchC

1. INTRODUÇÃO

Avanços na tecnologia de semicondutores, assim como a disponibilidade de bibliotecas de IP de *hardware* e *software*, têm contribuído no desenvolvimento de sistemas dedicados em *chips* (SOCs Systems-On-Chip) [5]. Em comparação com sistemas dedicados tradicionais, a tecnologia SOC possibilita a integração de: processadores (ASIPs e DSPs), memórias (SRAM, DRAMs e memória *flash*), circuitos de interfaces de entrada/saída e ASICS (*Application Specific Integrated Circuits*) em um único *chip*, de forma que as arquiteturas dos sistemas dedicados possam ser customizadas para aplicações específicas [10]. Contudo, devido à peculiaridade de cada aplicação (desempenho, consumo de energia, etc) e à complexidade dos SOCs, a escolha da arquitetura ideal pode ser demorada, prolongando-se dessa forma o tempo de projeto e o ingresso do sistema no mercado [2]. Além disso, uma variação do processador invalidará o conjunto de ferramentas (compiladores, montadores e simuladores) utilizado para criar e testar o software [3]. Projetistas de sistemas dedicados precisam porém de um conjunto básico de ferramentas (compiladores, montadores e simuladores de ISAs) que considerem esses aspectos.

Neste contexto, as linguagens de descrição de arquiteturas (ADLs - *Architecture Decription Languages*) são utilizadas para antecipar a modelagem e a validação das arquiteturas dos SOCs. A arquitetura típica de um SOC inclui: (i) blocos/componentes que residem no SOC, (ii) interconexões entre os mesmos (bus do sistema) e (iii) funcionalidades de cada bloco/componente [10]. A descrição desta arquitetura é utilizada para geração automática de ferramentas de software específicas para o modelo descrito. Estas ferramentas incluem compiladores, simuladores, montadores, etc. Dessa forma, quando o ISA (*Instruction Set Architecture*) do processador precisar ser alterado, bastará mudar seu modelo na ADL para que as ferramentas de software sejam automaticamente geradas e dessa forma, refletir as últimas mudanças realizadas na arquitetura [3].

Este trabalho apresenta a linguagem de descrição de arquiteturas ArchC, a qual é baseada em SystemC. ArchC permite descrever arquiteturas em alto nível de abstração, possibilitando uma maior flexibilidade e uma maior exploração da arquitetura [9], além de permitir a geração automática de ferramentas de software como montadores, simuladores e interfaces de co-verificação [8].

As seções restantes deste trabalho estão organizadas da seguinte maneira. A Seção 2 apresenta alguns trabalhos correlatos na literatura. A Seção 3 introduz a linguagem ArchC para a modelagem de um conjunto de instruções. As Seções 4 e 5 descrevem a implementação do ArchC, destacando as suas principais ferramentas.

2. TRABALHOS CORRELATOS

Nesta seção apresenta-se inicialmente uma proposta de classificação das ADLs. Posteriormente, os trabalhos mais relevantes nas áreas de ADL e simulação são apresentados. Finalmente é realizada uma comparação entre as principais características da linguagem ArchC e de algumas outras ADLs.

Considerando o tipo de informação que as ADLs podem descrever, elas podem ser classificadas em três grupos: (i) ADLs baseadas em comportamento; (ii) ADLs baseadas em estrutura e finalmente (iii) ADLs híbridas [5].

Segundo Trabalho de MO401 2005 UNICAMP, Campinas BRA

O primeiro tipo de ADLs considera o comportamento do conjunto de instruções de um processador para dessa forma, obter informações da arquitetura e gerar automaticamente ferramentas de software. Devido ao alto nível de abstração nas descrições, este tipo de linguagem pode ter deficiências em descrições com precisão de ciclos de arquiteturas complexas [6]. Exemplos de ADLs baseadas em comportamento incluem: nML, ISDL, ValenC e CSDL.

O segundo tipo de ADLs baseia-se em informações de mais baixo nível sobre o *hardware* do processador. Estas ADLs provêem diagramas de blocos ou *net-lists* da arquitetura do SOC. MIMOLA e COACH são exemplos deste tipo de ADLs.

Finalmente, o terceiro tipo de ADLs utiliza informações comportamentais (conjunto de instruções) e estruturais (estrutura interna do processador) para facilitar a geração automática das ferramentas de software. Entre as ADLs híbridas encontram-se: MDes, Flexware, EXPRESSION, LISA e RADL.

As sintaxes das linguagens nML e ISDL são baseadas em gramáticas com atributos. Além de possuir um estilo de sintaxe pouco intuitivo para projetistas de hardware, elas podem acarretar algumas restrições nas descrições comportamentais [6]. No caso de nML essas restrições incluem: declaração de variáveis locais, contador de programa implícito, etc. No caso de ISDL não é possível descrever instruções multiciclo de tamanho variável.

A linguagem MIMOLA é baseada na estrutura do processador. Pelo fato das arquiteturas serem descritas em um nível muito baixo, esta linguagem não é aplicada para exploração de novas arquiteturas em projetos em nível de sistema [6].

As linguagens LISA e EXPRESSION consideram tanto a estrutura interna do processador, como o conjunto de instruções para descrever a arquitetura. LISA possui uma sintaxe própria, semelhante às declarações de estruturas ou classes em C++ e quando inicialmente foi lançada, sua principal contribuição foi a descrição em nível operacional do *pipeline* e do modelo de seqüenciamento [3]. Contudo, formatos de instruções complexos requerem codificação extensiva [8]. Por outro lado, a linguagem EXPRESSION enfatiza a exploração da arquitetura (*architecture design space exploration*) e a geração automática de uma ferramenta de software composta por um compilador e um simulador. A principal vantagem desta linguagem é o seu suporte para especificar subsistemas de memória [8].

ArchC é uma ADL híbrida e sua sintaxe é totalmente baseada em SystemC e C++. SystemC é composta por um conjunto de classes C++ e seu principal objetivo é elevar o nível de abstração no projeto e verificação de hardware. Em comparação com outras ADLs, ArchC possui um conjunto de características próprias, como por exemplo: descrições comportamentais em diferentes níveis de abstração, hierarquia de comportamentos, etc [1]. Além disso, ArchC possui uma metodologia de co-verificação baseada em dispositivos de armazenamento, uma hierarquia de classes de dispositivos de armazenamento e permite a integração com outros IPs SystemC. A Tabela 1 [6] compara as principais características de ArchC com as outras linguagens de descrição de arquiteturas apresentadas nesta seção.

Característica	1	2	3	4	5
Conjunto de instruções	•	•	•	•	•
Precisão de ciclos	•	•			•
Suporte a Multi-ciclo	•	•			•
Suporte a *pipeline*	•	•			•
Hierarquia de Memória	•	•			•
Emulação de S.O					•
Hierarquia de comportamentos					•
Co-verificação					•
Geração de Simuladores SystemC alto nível					•
Geração de montadores	•	•		•	F
Simulação Compilada	•	•			•
Redirecionamento de Compiladores	•	•		•	F

1 - LISA; 2 - EXPRESSION; 3 - nML;
4 - ISDL; 5 - ArchC.

Table 1: Comparação entre ADLs.

3. A LINGUAGEM DE DESCRIÇÃO DE ARQUITETURAS ARCHC

A descrição de uma arquitetura em ArchC pode ser dividida em duas partes: a descrição dos recursos da arquitetura (AC_ARCH) e a descrição do conjunto de instruções da arquitetura (AC_ISA) [1]. A descrição AC_ARCH fornece informações ao ArchC referentes à estrutura da arquitetura, como por exemplo: dispositivos de armazenamento, estrutura de pipeline, número de registradores, etc.

A descrição AC_ISA fornece informações ao ArchC referentes a cada uma das instruções, tais como: (i) formato, tamanho e sintaxe em linguagem *assembly*; (ii) informação do *opcode* usada na decodificação de instruções e (iii) comportamento da instrução em C/C++ [1]. Desta forma, um modelo para uma arquitetura em ArchC possui três arquivos: (i) o de recursos da arquitetura, por exemplo, mips.ac; (ii) o de declaração do conjunto de instruções, por exemplo mips_isa.ac; e (iii) o de comportamento das instruções, por exemplo, mips-isa.cpp. A Figura 1 [2] apresenta a estrutura de uma descrição para uma arquitetura em ArchC.

A partir destas descrições, ArchC gera automaticamente um simulador para a arquitetura. A seguir, as Subseções 3.1 e 3.2 apresentam em maior nível de detalhes, as etapas de descrição dos recursos da arquitetura e de descrição do conjunto de instruções.

3.1 Descrição dos recursos da arquitetura

Como acima mencionado, os recursos da arquitetura são definidos na descrição AC_ARCH. A Figura 2 ilustra uma declaração de recursos em ArchC para um modelo funcional da arquitetura MIPS.

Conforme a Figura 2, a descrição dos recursos da arquitetura começa sempre com a palavra reservada AC_ARCH

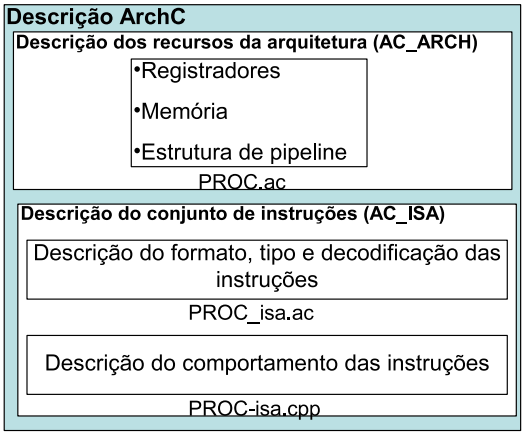

Figure 1: Estrutura de uma descrição para uma arquitetura em ArchC.

```
AC_ARCH(mips){
  ac_cache   MEM:4M;
  ac_regbank RB:34;
  ac_wordsize 32;

  ARCH_CTOR(mips){
    ac_isa("mips_isa.ac");
    set_endian("big");
  };
};
```

Figure 2: Descrição de recursos da arquitetura para MIPS.

seguido entre parênteses pelo nome do projeto. A seguir uma série de palavras reservadas tais como: *ac_cache*, *ac_regbank* e *ac_wordsize* são utilizadas. A palavra reservada *ac_cache* é utilizada para declarar um dispositivo de armazenamento do tipo cache com seu respectivo tamanho, o qual pode ser expresso em *bytes*, *kilobytes* (K ou k), *megabytes* (M ou m) ou *gigabytes* (G ou g). A palavra reservada *ac_regbank* é utilizada para instanciar o banco de registradores indicando o número total de registradores a serem utilizados, enquanto isso, a palavra *ac_wordsize* é utilizada para definir o tamanho de palavra da arquitetura, neste exemplo é de 32 bits.

A seção ARCH_CTOR inicializa a declaração do constructor de AC_ARCH utilizando o nome do arquivo indicado por *ac_isa* par compor o modelo. A palavra reservada *set_endian* configura o *endianness* da arquitetura e os possíveis valores a serem utilizados são: big ou little. Além das palavras reservadas acima apresentadas existem outras que podem ser utilizadas na declaração dos recursos da arquitetura. Estas palavras reservadas são: *ac_pipe*, *ac_icache*, *ac_dcache*, *ac_mem*, *ac_reg* e *ac_format*. Uma descrição mais aprofundada pode ser encontrada em [7].

3.2 Descrição do conjunto de instruções

A descrição do conjunto de instruções da arquitetura AC_ISA possui toda a informação que ArchC precisa para gerar automaticamente o decodificador de instruções no código objeto, assim como o comportamento de cada uma das instruções da arquitetura. Neste sentido, a descrição AC_ISA é dividida em dois arquivos: o primeiro contendo os formatos possíveis das instruções junto com a declaração das mesmas. O segundo arquivo contém o comportamento das instruções descrito em C++/SystemC. A Figura 3 aprensenta o primeiro destes arquivos para o modelo MIPS.

```
AC_ISA(mips){
  ac_format TypeR  = "%op:6 %rs:5 %rt:5 %rd:5 %shamt:5 %func:6";
  ac_format TypeI  = "%op:6 %rs:5 %rt:5 %offset:16";

  ac_instr<TypeR> add;
  ac_instr<TypeI> lw;

  ISA_CTOR(mips){
    add.set_asm("add %rs, %rt, %rd");
    add.set_decoder(op=0x00, func=0x20);

    lw.set_asm("lw %rt, %offset(%rs)");
    lw.set_decoder(op=0x23);
  };
};
```

Figure 3: Descrição de ISA para MIPS.

Conforme a Figura 3, a palavra reservada AC_ISA é utilizada para indicar o começo da descrição do conjunto de instruções para um dado projeto, que neste caso é o MIPS. Seguidamente, deve-se declarar os formatos possíveis das instruções utilizando a palavra chave *ac_format*. A declaração de cada um dos campos de um formato começa com o caracter "%", assim "%op:6" declara um campo de 6 bits denominado *op*. Uma vez declarados os tipos de formatos, passa-se a declarar as instruções. Todas as instruções devem estar associadas a um dos formatos definidos. A palavra reservada *ac_instr* é utilizada para declarar uma instrução e associá-la com um dos formatos predefinidos. No casso da Figura 3, a instrução cujo nome é *add* é associada ao formato de tipo R (TypeR).

A seção ISA_CTOR da declaração IC_ISA é utilizada para construir as instruções a partir dos seus formatos predefinidos. Para cada uma das instruções declaradas, utiliza-se o caracter "." seguido de duas possíveis palavras reservadas: *set_asm* e *set_decoder* [3]. A palavra reservada *set_decoder* especifica a forma de decodificação da instrução. A chamada ao método *add.setdecoder* determina que para que uma instrução seja reconhecida como uma instrução **add**, os campos **op** e **funct** devem possuir os valores 0x00 e 0x20 respectivamente [6]. As informações obtidas das palavras reservadas *set_asm* e *set_decoder* podem ser utilizadas para sintetizar o assembler do processador e o decodificador para o simulador, respectivamente.

As informações fornecidas ao ArchC nas descrições de recursos de arquitetura (AC_ARCH) e do conjunto de instruções (AC_ISA) são utilizadas pelo gerador de simuladores ArchC (acsim) para criar um arquivo esqueleto (project_name-isa.cpp), o qual será preenchido para definir o comportamento de cada uma das instruções declaradas. A Figura 4 ilustra o comportamento das instruções: *add* e *load*.

A principal vantagem na descrição de comportamento em ArchC é sua flexibilidade, pois é possível descrever com-

```
void ac_behavior( add )
{
  RB[rd] = RB[rs] + RB[rt];
};

void ac_behavior( lw )
{
  RB[rt] = DM.read(RB[rs]+ offset);
};
```

Figure 4: Descrição de comportamento funcional em ArchC.

portamento em vários níveis de abstração [9]. Esta característica baseia-se no fato de existir um conjunto de operações que podem ser compartilhadas entre instruções. O ArchC permite definir três tipos de comportamento: (i) comportamento genérico de instrução, (ii) comportamento para formatos e (iii) comportamento específico de instrução [6]. Dessa forma, o comportamento genérico de instrução é primeiramente executado, seguido pelo comportamento do formato da instrução e finalmente pelo comportamento específico da instrução. Um comportamento genérico de instrução é válido para todas as instruções do ISA. A Figura 5 apresenta um comportamento genérico na familia de processadores MIPS, na qual, para todas as instruções, o incremento do contador do programa (*ac_pc*) é realizado.

```
void ac_behavior( instruction )
{
  ac_pc = ac_pc +4;
};
```

Figure 5: Descrição de comportamento genérico de MIPS em ArchC.

O domínio de um comportamento para formatos é restrito às instruções do mesmo formato, por exemplo, no caso do processador MIPS, todas as instruções do tipo R executam o mesmo conjunto de testes para verificar de onde devem-se obter os operandos *rs* e *rt* é realizado para cada uma das instruções deste tipo [6]. O comportamento específico de instrução é válido somente no contexto da própria instrução. A figura 4 mostra dois comportamentos específicos das instruções *add* e *load*. Mais informações sobre descrição de comportamentos estão disponíveis em [7].

4. SUPORTE PARA O PROJETO EM ARCHC

ArchC possui um conjunto de ferramentas para prover mecanismos de suporte ao projeto. Estas incluem co-verificação baseada em dispositivos de armazenamento, suporte GDB para simuladores e finalmente emulação de sistema operacional. Estas ferramentas são descritas nas seguintes subseções.

4.1 Co-verificação baseada em dispositivos de armazenamento

Um projeto novo de uma arquitetura começa com um modelo em alto nível de abstração, o qual é definido gradualmente até a fase de síntese. No caso de ArchC, um novo modelo pode começar com a abstração de um modelo funcional e uma vez validado o seu conjunto de instruções, este pode ser refinado. Um exemplo desse refinamento é a de *pipeline*. Devido ao fato do processo de refinamento ser realizado manualmente, ele está sujeito a erros. Nesse contexto, ArchC fornece uma ferramenta denominada *ac_verifier* para a co-verificação de dois modelos diferentes de uma mesma arquitetura. Um dos modelos, normalmente o funcional, é utilizado como referência para verificar o comportamento de um modelo refinado (DUV - *Device Under Verification*). Para tais fins, ambos os modelos executam a mesma aplicação e os dispositivos de armazenamento a serem testados devem possuir o mesmo nome.

A metodologia de verificação de ArchC é baseada em transações e é denominada "co-verificação baseada em dispositivos de armazenamento". A idéia desta metodologia consiste em monitorar cada atualização realizada nos dispositivos de armazenamento em ambos os modelos. Para cada uma das atualizações realizadas um *timestamp* é atribuído, para dessa forma, saber quando elas aconteceram. A ferramenta *ac_verifier* compara as seqüencias dessas atualizações através da execução para determinar a consistência entre ambos modelos. No caso de ocorrer alguma inconsistência, um relatório é emitido, o qual informa o nome do dispositivo em que a inconsistência ocorreu e um *log* das atualizações realizadas por ambos os modelos depois da occorrência do erro [6]. O algoritmo de co-verificação foi projetado para funcionar em duas possíveis situações. Na primeira situação, busca-se garantir que as atualizações realizadas nos dois modelos sejam executadas no mesmo ciclo. Entretanto, na segunda situação busca-se garantir que as seqüencias das atualizações geradas por ambos sejam as mesmas. Para gerar o simulador no modo de co-verificação é necessário informar o gerador através de um parâmetro em linha de comando. Dessa forma, não há a necessidade de modificar a especificação da arquitetura.

Por outro lado, é importante ressaltar que ambos os modelos, juntamente com a ferramenta *ac_verifier* executam em paralelo utilizando três processos diferentes do sistema operacional Linux. Para fins de comunicação, eles utilizam algumas funcionalidades do sistema operacional denominadas IPC (*Interprocess comunication*). A Figura 6 [1] ilustra a metodologia de co-verificação adotada por ArchC.

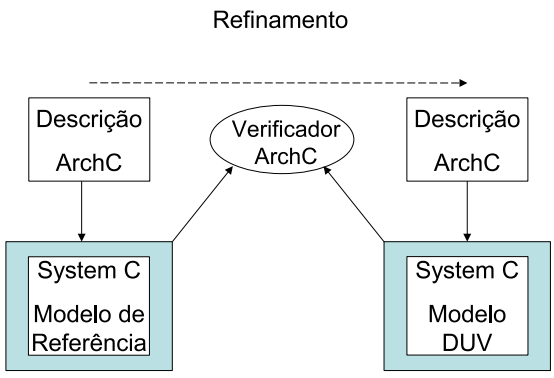

Figure 6: Metodologia de co-verificação em ArchC.

4.2 Suporte GDB para Simuladores

Os simuladores gerados por *acsim* para modelos funcionais podem utilizar o protocolo do GDB facilmente [1]. Para tais fins, métodos dependentes do processador devem ser implementados para intermediar a comunicação entre a arquitetura do processador e o GDB. Além disso, a implementação desses métodos possibilita o mapeamento dos registradores e da memória para o formato GDB desejado. Dessa forma, os usuários poderão realizar *debugs* de *software* dentro do simulador.

Os métodos são implementados em um arquivo denominado *PROC_gdb_funcs.cpp*, como por exemplo *mips_gdb_funcs.cpp*. Executando o comando *acsim* com o parâmetro *-gdb* será gerada a classe do simulador que herda de *AC_GDB_Interface* e, além disso, ele utilizará as funções definidas em *PROC_gdb_funcs.cpp*.

Para que o simulador possa enviar os pacotes de registradores de leitura e escrita para o GDB, os seguintes três métodos precisam ser implementados: *nRegs()*, *reg_read()* e *reg_write()*. No método *nRegs()* define-se o número de registradores que GDB pode receber e enviar. As funções *reg_read()* e *reg_write()* possuem informações sobre os registradores de leitura/escritura a serem utilizados. A Figura 7 ilustra uma implementação destes 3 métodos para um banco de registradores de propósito geral (RB_GP).

```
int mips1::nRegs(void) {
   return 32;
}

ac_word mips1::reg_read( int reg ) {
  if ( ( reg >= 0 ) && ( reg < 32 ) )
    return RB.read( reg );
  return 0;
}

void mips1::reg_write( int reg, ac_word value ) {
  /* general purpose registers */
  if ( ( reg >= 0 ) && ( reg < 32 ) )
    RB.write( reg, value );
}
```

Figure 7: Métodos para manipulação de registradores para suporte GDB.

Para determinar como as regiões de memória são lidas e escritas, torna-se necessário implementar as funções *mem_read()* e *mem_write()*, pertencentes à interface *AC_GDB_Interface*. A Figura 8 ilustra duas implementações de ambos métodos para um banco de memória de dados e instruções.

4.3 Emulação do Sistema Operacional

Os simuladores gerados por ArchC podem ser instrumentados com um mecanismo de emulação de chamadas de sistema operacional [4]. As rotinas de chamadas de sistema foram agrupadas em uma biblioteca *retargetable* definida em C. Dessa forma, basta uma recompilação da biblioteca para permitir a sua portabilidade a qualquer outra *target*. Utilizando um conjunto de funções ABI (*Application Binary Interface*), os projetistas de novos modelos em ArchC informam quais dispositivos de armazenamento (memória, banco

```
unsigned char mips1::mem_read( unsigned int address ) {
  return ac_resources::IM->read_byte( address );
}

void mips1::mem_write( unsigned int address, unsigned char byte )
{
  ac_resources::IM->write_byte( address, byte );
}
```

Figure 8: Métodos para manipulação de memória para suporte GDB.

de registradores, etc) são utilizados para passagem dos argumentos de chamadas de sistema operacional. Para que a ABI de uma dada arquitetura seja fornecida, cada novo modelo que é especificado, deve redefinir os métodos relacionados à emulação do sistema operacional. Estes métodos são definidos na classe *ac_syscall*. A informação obtida do manual ABI do processador é utilizada para definir estas implementações. A emulação de Sistema Operacional de ArchC permite que benchmarks como: MediaBench e MiBench sejam executados com dados reais [8].

5. FERRAMENTAS DE ARCHC

Nesta seção apresenta-se a ferramenta *acsim*, a qual possibilita a geração de simuladores em ArchC.

5.1 O Gerador de simuladores de ArchC

A ferramenta *acsim* (*ArchC Simulator Generator*) utiliza a descrição dos recursos de arquitetura (AC_ARCH) e a do conjunto de instruções (AC_ISA) para gerar o modelo de arquitetura. Internamente, acsim utiliza duas ferramentas não disponíveis ao usuário. Estas ferramentas são: (i) o pré-processador de ArchC (acpp) e (ii) o gerador de decodificadores [6].

O pré-processador de ArchC extrai as informações contidas nos arquivos de descrição e as armazena em estruturas de dados que são posteriormente utilizadas pela ferramenta *acsim* para criar todas as classes (C++) e/ou módulos SystemC necessários para a construção do simulador da arquitetura [8]. O pré-processador de ArchC é composto por um analisador léxico e um analisador sintático. Esses analisadores foram criados utilizando as ferramentas GNU Flex e GNU Bison, respectivamente.

O gerador de decodificadores se encarrega de identificar cada uma das instruções (campos da instrução, valores dos campos, etc.) a partir da cadeia de bits vinda da memória. Uma série de opções pode ser passada ao *acsim* por linha de comando, para obter um conjunto de características úteis na exploração da arquitetura, como: estatísticas de simulação, atualizações feitas a dispositivos de armazenamento, etc.

6. CONCLUSÕES

O presente trabalho apresentou a linguagem de descrição de arquiteturas ArchC, a qual é baseada em SystemC e possibilita a descrição de uma arquitetura em um alto nível de granularidade. Além disso, ArchC provê mecanismos de geração automática de ferramentas de *software*, tais como: montadores, simuladores e interfaces de co-verificação; reduzindo o esforço necessário para verificação da arquitetura

modelada.

Descrever a arquitetura de um processador através de uma linguagem de alto nível e obter ferramentas auxiliares é uma forma bastante eficaz de se garantir que um projeto não seja implementado sem que antes os projetistas de *hardware* estejam confiantes a respeito da satisfação dos requisitos. A linguagem ArchC apresenta uma forma simples de se representar uma arquitetura até mesmo para usuários menos experientes.

7. REFERENCES

[1] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros. The archc architecture description language and tools. *International Journal of Parallel Programming*, 33(5):453–484, October 2005.

[2] A. Baldassin and P. Centoducatte. Geração automática de montadores para modelos de arquiteturas escritos em archc. In *Proceedings of the 9th Brazilian Symposium on Programming Languages - Recife, Brazil*, May 2005.

[3] A. Baldassin, P. Centoducatte, and S. Rigo. Extending the archc language for automatic generation of assemblers. In *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing*, pages 60–67, October 2005.

[4] M. Bartholomeu, S. Rigo, R. Azevedo, and G. Araujo. Emulating operating system calls in retargetable isa simulators. Technical Report IC-03-15, Insitute of Computing, University of Campinas, December 2003.

[5] A. Halambi, P. Grun, H. Tomiyama, N. Dutt, and A. Nicolau. Automatic software toolkit generation for embedded systems-on-chip. In *Proceedings of the IEEE International Conference on VLSI and CAD*, pages 107–116, October 1999.

[6] S. Rigo. *ArchC: Uma Linguagem de Descrição de Arquiteturas*. PhD thesis, Instituto de Computação, Universidade Estadual de Campinas, 2004.

[7] S. Rigo. *The ArchC Architecture Description Language Reference Manual*. The ArchC Team, Avenida Albert Einstein, 1251 13084-971 PO Boxl 6176 - Campinas/SP - Brasil, July 2005.

[8] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. Archc: A systemc-based architecture description language. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing - Foz do Iguacu, Brazil*, pages 66–73, October 2004.

[9] S. Rigo, R. Azevedo, and G. Araujo. The archc architecture description language. Technical Report IC-03-15, Insitute of Computing, University of Campinas, June 2003.

[10] H. Tomiyama, A. Halambi, P. Grun, N. Dutt, and A. Nicolau. Architecture description languages for systems-on-chip design. In *Proceedings of the 6th Asia Pacific Conference on Chip Design Languages*, pages 109–116, October 1999.