Simulação de Sistemas Embarcados utilizando ArchC*

Andréia Aparecida Barbiero
Universidade Federal do Paraná
Departamento de Informática
CEP 81.531-990 – Curitiba, PR, Brasil
andreia@c3sl.ufpr.br

Roberto André Hexsel
Universidade Federal do Paraná
Departamento de Informática
CEP 81.531-990 – Curitiba, PR, Brasil
roberto@inf.ufpr.br

Resumo

O ritmo do mercado de produção de hardware e software para sistemas embarcados exige que o tempo de projeto seja cada vez mais curto, e isso determina o uso de ferramentas para auxiliar os projetistas na escolha do hardware mais adequado para uma determinada aplicação. Este trabalho descreve um conjunto de simuladores que permite escolher o processador mais adequado ainda na fase inicial de um projeto. Até o momento, foram desenvolvidos simuladores para os processadores Motorola DSP56827, Rabbit R2000 e Atmel Atmega8515 utilizando a Linguagem de Descrição de Arquiteturas ArchC. As ferramentas dos simuladores permitem estimar, através de simulações precisas, métricas de desempenho como tempo de execução e memória utilizada em código, pilha e conjunto de dados.

1. Introdução

Um relatório publicado pela *Intel* em 2000 mostrava que apenas 2% de todos os processadores fabricados no mundo eram destinados a aplicações *desktop*. Os 98% restantes eram divididos em aplicações de robótica (6%), sistemas automotivos (12%), e 80% para sistemas embarcados [18]. As aplicações mais comuns deste último mercado são telefones celulares, brinquedos, eletrodomésticos e outros equipamentos eletrônicos. Estes produtos são projetados sob restrições de recursos como energia consumida, utilização de memória, tamanho do circuito integrado, e principalmente custo. No custo deve ser incluído, além dos componentes de hardware, o desenvolvimento do hardware e de software aplicativo.

Não é raro que os projetos neste segmento possuam o seguinte perfil: (i) tratam-se de aplicações "novas" com a especificação do sistema ainda incompleta; (ii) existem ao menos duas opções de microprocessadores; e (iii) alguns componentes de software já encontram-se disponíveis, tais como uma implementação da pilha TCP/IP ou um algoritmo de criptografia. Neste caso, questões como *"qual processador atenderá aos requisitos de desempenho e/ou consumo?"* e *"quanta memória será necessária?"* devem ser respondidas no início do processo de desenvolvimento.

Quando apenas a experiência dos projetistas é utilizada para responder àquelas perguntas alguns problemas podem ocorrer. Dentre eles salientam-se o sub- ou o superdimensionamento de recursos, perda de tempo e o conseqüente aumento no custo de desenvolvimento. Há superdimensionamento quando são utilizados recursos além do necessário para determinada aplicação, como a utilização de um processador poderoso e caro para executar uma aplicação simples. No sub-dimensionamento ocorre o contrário, e a capacidade do hardware não é suficiente para satisfazer aos requisitos de desempenho, como em uma aplicação que demanda mais memória do que o disponível no microcontrolador escolhido. Neste caso pode ocorrer a inutilização de placas de circuito impresso para inclusão de mais memória. Em ambos os exemplos, é evidente o aumento no custo e no tempo de desenvolvimento.

Este trabalho descreve um conjunto de simuladores que serão utilizados para auxiliar os projetistas na escolha de um processador que atenda aos requisitos de sua aplicação. Os simuladores permitem prever com razoável precisão, no início do ciclo de desenvolvimento, (1) se determinado processador tem desempenho suficiente para a aplicação, (2) qual será o tamanho do código compilado, e (3) qual deve ser a capacidade da memória de dados. Com estimativas precisas e referidas à aplicação é mais provável que a escolha do processador ou microcontrolador seja a acertada. A maioria dos fabricantes não fornece ferramentas adequadas para este tipo de comparação, ao contrário, as ferramentas são dirigidas às suas linhas de produtos, na forma de software proprietário e com custo elevado, como é o caso de [4], por exemplo.

Os simuladores foram implementados com a *linguagem de descrição de arquitetura* ArchC [14, 2], que por sua vez é

*WSCAD 2006, Ouro Preto, MG, agosto de 2006

baseada em SystemC [8]. ArchC permite descrever a arquitetura desejada de forma eficiente e rápida, e a partir da descrição gera automaticamente o simulador apropriado, que é capaz de estimar o tempo de execução e o número de acessos à memória.

Correntemente, estão implementados modelos de três microcontroladores amplamente utilizados no mercado, que são: Motorola DSP56827 [12], Rabbit R2000 [13], e Atmel Atmega8515 [3]. Estes foram escolhidos por conta do envolvimento dos autores em projetos em que estes microcontroladores seriam possíveis candidatos [11, 19]. Além dos três modelos citados acima, os simuladores podem ser facilmente expandidos para aceitar a simulação de outros processadores da mesma família, já que na maioria das vezes o conjunto de instruções não sofre grandes alterações, apenas adiciona ou remove algumas funções. Por exemplo, o simulador do microprocessador R2000 pode facilmente simular o microprocessador R3000.

Para agregar mais informações úteis ao conjunto de métricas disponibilizado pela linguagem ArchC foi desenvolvido uma medida para o tamanho, ou utilização, da memória de dados e de programa. A métrica da utilização da memória de dados inclui a medição dinâmica, e precisa, dos tamanhos máximos de *heap* e pilha.

O texto está organizado como segue. Alguns conceitos sobre desenvolvimento de sistemas embarcados e trabalhos relacionados são discutidos na Seção 2. O desenvolvimento do projeto e a avaliação de desempenho são vistos, respectivamente, nas Seções 3 e 4. Finalmente, a conclusão e as propostas para a expansão do ambiente de desenvolvimento, estão na Seção 5.

2. Definições e Trabalhos Relacionados

O processo de desenvolvimento de sistemas embarcados é dividido nas seguintes fases: (i) Especificação e Modelagem, (ii) Validação, e (iii) Síntese [7]. Na Especificação e Modelagem são investigados os requisitos e as restrições do sistema, métricas de desempenho são estabelecidas, e então um modelo baseado nas relações entre os componentes é construído. Na fase de Validação o modelo é implementado e os requisitos do sistema são re-avaliados. É nesta fase que a simulação é utilizada. Na fase de Síntese, é efetuado o particionamento entre as funções de hardware e as de software, e o sistema é efetivamente implementado.

O processo de simulação consiste em se utilizar um modelo do sistema para, por exemplo, obter uma melhor compreensão do problema, das metas que devem ser atingidas, verificar se os requisitos de projeto serão atingidos pela solução sob estudo. Para auxiliar na construção de um modelo várias linguagens podem ser utilizadas, cada qual com suas vantagens e desvantagens.

Com o objetivo de manter um alto nível de abstração e possibilitar a descrição do hardware foi desenvolvida a ligugem *SystemC* [8]. SystemC combina as vantagens da popularidade do C++ e de sua adequação ao processo de geração de software, com capacidade semântica adicional e apropriada para a descrição de hardware, através de construções como portas, sinais e relógios. Estas construções e abstrações são disponibilizadas numa biblioteca de funções. O desenvolvimento de SystemC iniciou-se em 1999, através da cooperação entre Synopsys, CoWare/IMEC e Universidade da Califórnia, Invine. Atualmente, várias empresas e instituições compõem a Open System Initiative (OCSI) e disponibilizam ferramentas e código fonte na página do projeto [17].

ArchC é uma especialização do SystemC e foi desenvolvida com o objetivo de geração automática de simuladores [14]. ArchC utiliza as bibliotecas do SystemC para modelar o hardware a ser simulado, e inclui ferramentas capazes de automatizar o processo de construção do simulador e a geração de montadores.

ArchC permite gerar três tipos de simuladores: (a) funcional, (b) multi-ciclo e (c) segmentado *(pipeline)*. Com a simulação funcional pode-se verificar e testar o comportamento das instruções e a funcionalidade básica do sistema modelado. A descrição das instruções para os simuladores multi-ciclo e segmentado deve conter informações detalhadas sobre o tempo de execução de cada instrução. No simulador de uma implementação multi-ciclo declara-se o número de ciclos dispendidos ao executar cada tipo de instrução, enquanto que no simulador de uma implementação segmentada a descrição deve conter um modelo detalhado dos segmentos do processador bem como de circuitos de adiantamento ou de bloqueios [10]. Evidentemente, a precisão das medidas de tempo depende do nível de detalhe com que o modelo é elaborado.

Os simuladores gerados com ArchC permitem medir diretamente grandezas como (a) o número que ciclos executados, (b) tempo de execução, (c) número de instruções executadas, (d) número de chamadas de E/S, e (e) número de acessos a cada instrução e à memória [15]. Uma medida importante para os projetistas de sistemas embarcados, e que não é diretamente produzida pelos simuladores, é a quantidade de memória necessária para executar determinada aplicação. Os simuladores desenvolvidos neste trabalho foram estendidos de forma a serem capazes de medir o tamanho do conjunto de dados, determinando dinamicamente, e com precisão, o tamanho da pilha e do *heap*.

Além da ArchC, outras Linguagens de Descrição de Arquitetura ou *Architectute Description Languages (ADLs)* já foram desenvolvidas. A *Language for Instruction Set Architecture* (LISA) [20] foi uma das primeiras linguagens a permitir a descrição de *pipelines* e de modelos seqüenciais. Algumas desvantagens de LISA são a complexidade da implementação para descrever formatos de instruções com-

plexos e a falta de ferramentas de domínio público para manipulação da linguagem. A EXPRESSION [9] é uma Linguagem de Descrição de Arquitetura de código livre focada na exploração do espaço arquitetural para *systems on chips* (SoCs) e na geração automática de compiladores e simuladores. O ponto forte da linguagem é o suporte à especificação de hierarquia de memória. ABSCISS [1] é um gerador de simuladores compilados de conjunto de instruções que simula a execução de código escrito em *assembly*. Um inconveniente da abordagem no nível de montador é a baixa precisão na simulação de caches. Isto ocorre devido ao ABSCISS atribuir arbitrariamente os endereços de dados e de instruções, possivelmente de forma diferente daquela de um ligador convencional.

Não existem ferramentas de domínio público disponíveis para a simulação de dois dos microprocessadores utilizados neste trabalho, o Motorola DSP56827 e o Rabbit R2000. A plataforma *CodeWarrior*, da Metrowerks [4] é a ferramenta mais utilizada por desenvolvedores dos microprocessadores da família 56800, à qual pertence o DSP56827. Esta plataforma permite a implementação do software, execução passo-a-passo e também simulação. É possível estimar o número de ciclos executados e a utilização de memória. A desvantagem da ferramenta é o seu custo, em torno de R$ 2000, que inviabiliza sua aquisição apenas para fins de teste da arquitetura. Para o microcontrolador Rabbit R2000, não há uma ferramenta de simulação no mercado. As plataformas de desenvolvimento, *WinIDE* da Softools [16] e *Dynamic C* [13] da Rabbit, são aplicativos proprietários e só executam quando conectadas diretamente a um módulo de hardware. Para o microprocessador Atmega8515, a ferramenta *AVR Studio* da Atmel [3] é gratuita, porém esta não disponibiliza métricas de desempenho.

Os simuladores aqui descritos permitem que os desenvolvedores de sistemas baseados nos três processadores modelados utilizem os simuladores para iniciar o desenvolvimento do software antes que os protótipos em hardware estejam disponíveis. Isso permite o levantamento antecipado das características dos algoritmos e de detalhes obscuros da especificação. Além disso, os projetistas poderão utilizar os simuladores como ferramentas de um ambiente de desenvolvimento e simulação para decidir qual processador melhor atende as necessidades e requisitos de projeto.

3. Desenvolvimento do Ambiente de Simulação

No que se segue são descritos, de maneira sucinta, a implementação dos simuladores e a técnica de obtenção das métricas relacionadas à capacidade de memória naqueles.

3.1 Implementação dos Simuladores

O desenvolvimento de versões Multi-Ciclo dos três modelos, para o Motorola DSP56827, Rabbit R2000 e Atmel Atmega8515, utilizando ArchC está concluído. O processo de modelagem foi relativamente trabalhoso, principalmente para o processador da Motorola, que possui inúmeras funcionalidades e modos de execução. Ao todo, foram necessárioas mais de 26.000 linhas de código para descrever os três conjuntos de instruções.

Os modelos foram inicialmente escritos para a simulação Multi-Ciclo, que produz as métricas de desempenho com uma precisão adequada. Testes preliminares, comparando os resultados de simulação com medidas em módulos de hardware do DSP56827 e Atmega8515, indicam que as estimativas de tempo ficam a $\pm 80\%$ dos valores medidos com o hardware. As versões dos modelos com segmentação já estão sendo desenvolvidas e a expectativa é de que os modelos segmentados apresentem precisão da ordem de $\pm 95\%$ com relação a valores medidos diretamente nos processadores.

Para gerar um simulador de uma CPU com ArchC é necessário, primeiramente, descrever as características básicas da organização da CPU, como mostrado na Figura 1, e a sintaxe do conjunto de instruções, como mostrado na Figura 2. As figuras mostram um trecho da descrição do Motorola DSP56827. Arquivos com a descrição da organização (arquitetura), e do conjunto de instruções são submetidos ao pré-processador do ArchC, que gera automaticamente a estrutura do simulador e um terceiro arquivo no qual deve ser descrito o comportamento das instruções, como mostrado na Figura 3.

```
AC_ARCH(dsp56827){
   // CPU de 16 bits
   ac_wordsize 16;
   // 128K de código, 64K de dados
   ac_mem PM:128k;    ac_mem PM:64k;
   // registradores visíveis
   ac_reg SR;         ac_reg OMR;
   ac_reg PC;
   ac_reg LC;         ac_reg LA;
   ac_reg HWS0;       ac_reg HWS1;
   ac_reg M01;        ac_reg N;
   ac_reg SP;         ac_reg X0;

   ARCH_CTOR(dsp56827) {
      ac_isa("dsp56827_isa.ac");
      set_endian( "big" );
   };
};
```

Figura 1. Organização do DSP56827.

```
AC_ISA(dsp56827){
 // Ex: add A,B
 Type_1W  = "%op4:16";
 // Ex: inc X:0000
 ac_format Type_2W = "%op4:16 %imm16:16";
 // Ex: bfclr #0001,X0
 ac_format Type_2W_Bit1 = "%op2:8 %mod:3
reg:5 %imm16:16";

 ac_instr<Type_1W> add_b_a, tfr_b_a;
 ac_instr<Type_2W> dec_imm16, inc_imm16;
 ac_instr<Type_2W_Bit1> bfclr_imm_d5, bfset_imm_d5;

 ISA_CTOR(dsp56827){
  add_b_a.set_asm("add B,A");
  add_b_a.set_decoder(op4=0x6400);
  add_b_a.set_cycles(2);

  tfr_b_a.set_asm("tfr B,A");
  tfr_b_a.set_decoder(op4=0x6c00);
  tfr_b_a.set_cycles(2);

  bfclr_imm_d5.set_asm("bfclr %imm16,%d5");
  bfclr_imm_d5.set_decoder(op2=0x81,mod=0x6);
  bfclr_imm_d5.set_cycles(4);
    ...
   };
  };
```

Figura 2. Descrição do conjunto de instruções (sintaxe) do DSP56827 (parte).

```
//!Instruction or_a1_x0 behavior method.
void ac_behavior( or_a1_x0 )
{
  int res = 0;
  switch (cycle)
  {
    case 1:
      printf("@@@ OR a1 x0 @@@\n");
      //Efetua a operação lógica
      res = A1 | X0.read();
      //Atualiza o SR
      SR_ccr_v = 0;
      SR_ccr_z = (res == 0);
      SR_ccr_n = (res >> 15);
      X0.write(res & 0x0000ffff);
      break;
    case 2:
      break;
  }
  ac_cycle++;
}
```

Figura 3. Descrição do comportamento de uma instrução (semântica) do DSP56827.

3.2 Métricas de Memória

Com o objetivo de complementar o conjunto de métricas disponibilizado pela linguagem ArchC e oferecer uma ferramenta mais completa para os projetistas, foram implementadas neste trabalho duas métricas relacionadas à quantidade de memória utilizada pela aplicação. As métricas permitem determinar a quantidade de memória, em bytes, do código (estático) e dos dados (estático e dinâmico). Esta foi uma contribuição inédita do trabalho ao conjunto de funcionalidades da ArchC.

Para incluir a métrica de utilização de memória foi necessário alterar o código fonte do ArchC. Como a linguagem já produz um arquivo com estatísticas de número de ciclos/instruções executados, número de acessos à memória e às instruções, as métricas de utilização de memória são geradas a partir de duas novas variáveis, *im_length*, que armazena a utilização da memória de programa, e *dm_length* que verifica a utilização da memória de dados.

A variável *im_length* é incrementada no método de leitura do arquivo com o binário/executável e essencialmente conta o número de bytes do programa. A variável *dm_length* conta os bytes de memória estática e dinâmica do programa. A quantidade de dados estáticos também é obtida do método de leitura do programa, enquanto que a utilização dinâmica de memória é medida nos métodos de escrita na memória.

Para evitar que escritas sucessivas no mesmo endereço de memória sejam contabilizados, um vetor mantém um histórico de todos os acessos à memória. Quando ocorre uma escrita na memória, é testado se este endereço já foi acessado. Se sim, a variável *dm_length* não é incrementada; senão, a variável é incrementada e o acesso registrado no vetor de histórico. A Figura 4 mostra um trecho de código que executa o registro no histórico da gravação de dados de 16 bits.

```
if (mem_control[address] == 0)
{
   ac_sim_stats.dm_length += 2;
   mem_control[address] = 1;
   mem_control[address+1] = 1;
}
```

Figura 4. Trecho do código que computa o tamanho da memória de dados.

4. Avaliação de Desempenho

Nesta seção são apresentados alguns resultados do uso dos simuladores. As simulações descritas adiante foram úteis para a aferição da qualidade dos modelos bem como para a depuração dos mesmos. São descritas as características dos três processadores modelados, os testes realizados e são apresentados os resultados obtidos. Devido a problemas com o compilador e o montador do Rabbit R2000, não foi possível utilizar o simulador para todos os programas, somente para os mais simples e que efetuam poucos acessos a memória. Este problema está sendo verificado junto ao fabricante do compilador.

Os três microprocessadores possuem diferenças significativas em termos de recursos e de custo, como mostra a Tabela 1. O DSP56827 da Motorola (DSP) é um processador

de 16 bits e instruções de tamanho variável, enquanto que o Atmel Atmega8515 (AT) é um processador de 8 bits de dados, com instruções de 16 bits. O Rabbit R2000 (RB) é um processador de 8 bits com instruções de tamanho variável. A escolha de processadores tão diferentes foi proposital, pois através dos testes será possível efetuar a medição e a comparação direta da capacidade de processamento de cada um deles. Comparações como estas permitirão aos projetistas decidir com base em dados concretos, e quando possível, a decisão será baseada na execução do código da aplicação para a qual os processadores são candidatos.

Processador	*DSP*	*AT*	*RB*
Freqüência [MHz]	80*	16	30
Memória Flash [KBytes]	128	8	1024**
RAM Interna [Bytes]	4092	512	1 M**
RAM externa [KBytes]	64	64	6000
Custo [US$]	11,11	4,84	12,75

* A freqüência do relógio interno é o dobro da externa.

** Endereça 1Mbytes de memória, para dados e código.

Tabela 1. Principais características dos microprocessadores.

Os modelos foram validados através da execução dos programas de teste disponibilizados pelo *Dalton Project* [5], programas clássicos de ordenação e um gerador de números primos, todos descritos sucintamente na Tabela 2. Os programas do *Dalton Project* foram utilizados pela equipe de desenvolvedores do ArchC para testar o modelo do microprocessador Intel 8051.

Programa	*Descrição*
negcnt.c	Conta 1000 números negativos
int2bin.c	Traduz núm. de 16 bits (hexa) para binário
gcd.c	Compara, soma e subtrai 2 números
cast.c	Transforma `long` em `char`
fib.c	Gera 100 elementos da Série de Fibonacci
bubble.c	Ordena 1000 números pelo Bubble Sort
insertion.c	Ordena 1000 números pelo Insertion Sort
quick.c	Ordena 1000 números pelo Quick Sort
selection.c	Ordena 1000 números pelo Selection Sort
shell.c	Ordena 1000 números pelo Shell Sort
crivo.c	Gera primos $<$1000 pelo Cr. de Eratóstenes

Tabela 2. Programas de teste.

Os testes foram executados num PC com sistema operacional Linux *Slackware 10.1*, com ArchC versão 1.6. Os arquivos de testes foram gerados pelos compiladores *Metrowerks Code Warrior 7.2* [4], *AVR Studio 4.0* [3] e *Softools WinIde 1.58.03* [16] para os processadores da Motorola, da Atmel e da Rabbit, respectivamente.

4.1 Comparação entre os Processadores

A Figura 5 mostra a utilização dinâmica de memória de dados dos processadores quando executam os programas de teste. No gráfico, o eixo das ordenadas está em escala logarítmica. Nota-se que a utilização é praticamente a mesma para os três processadores, já que todos os programas utilizam números inteiros. Caso fossem utilizados vetores do tipo `char`, o DSP56827 apresentaria desvantagem pois o compilador deste processador considera que o tamanho do `char` é 16 bits, o que duplicaria sua utilização de memória.

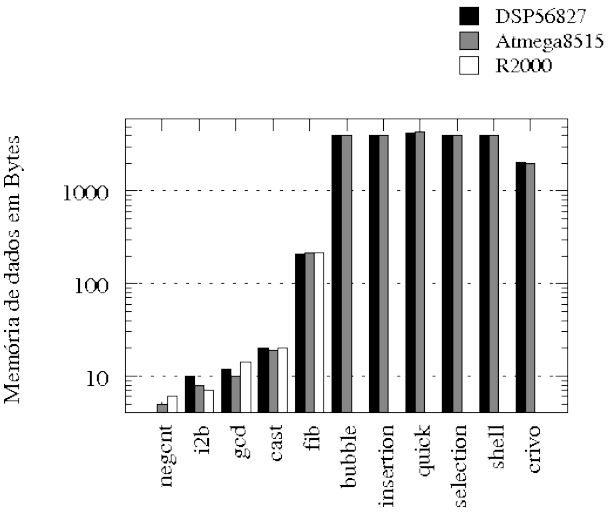

Figura 5. Utilização da Memória de Dados.

A Figura 6 compara tamanho do código (estático) gerado para os processadores. No gráfico, o eixo das ordenadas está em escala logarítmica. Esta medida é relacionada à eficiência do conjunto de instruções e do compilador, que é capaz de gerar código compacto a partir de um programa escrito em C. Neste caso, os pares processador-compilador da Motorola e da Rabbit são mais eficientes: a relação entre o tamanho dos executáveis produzidos pelos pares compilador/processador Atmega e DSP57827, *com estes programas simples de teste*, é de de 1,4 a 7,4 vezes maior para o Atmega. comparando-se com o DSP57827.

As Figuras 7 e 8 mostram os resultados para número de ciclos e número de instruções executadas. Comparando-se o número de ciclos com estes programas de teste, o processador da Motorola executa 1,2 vezes mais ciclos que o Atmega somente com o programa *cast*. Nos demais, o Atmega executa 1,23 a 1,62 vezes mais ciclos que o Motorola. Já o R2000 é consideravelmente mais lento, executa cerca de 4 vezes mais ciclos que o DSP56827. Isso ocorre porque, ao contrário do DSP56827 e do Atmega, o R2000 não

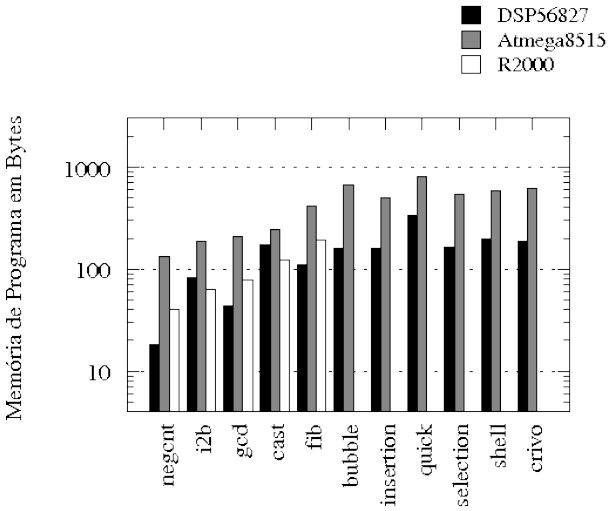

Figura 6. Utilização da Memória de Programa.

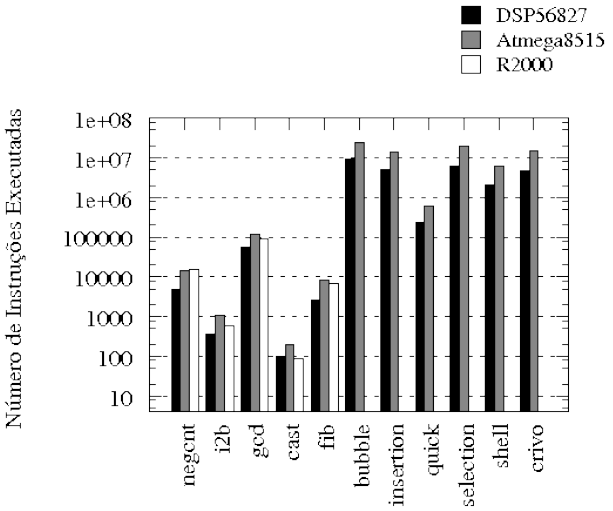

Figura 8. Número de Instruções Executadas.

é segmentado, e cada instrução consome de 2 a 15 ciclos para ser executada. Note que o eixo vertical dos gráficos é logarítmico. Quanto ao número de instruções executadas, o Atmega executa de 1,24 a 1,50 mais instruções que o Motorola. A eficiência do código gerado para o Rabbit depende do programa, sendo que o tamanho fica entre aqueles dos outros dois processadores, exceto para *negcnt*, no qual o R2000 possui o maior executável dos três.

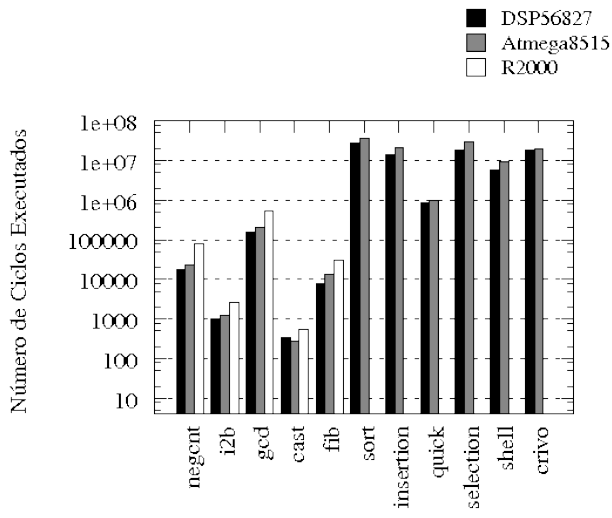

Figura 7. Número de Ciclos Executados.

Esta análise preliminar dos resultados indica que para os programas com mais dados ("maiores"), o processador da Motorola possui um desempenho substancialmente melhor, mas para os programas "menores" fica evidente o desperdício de recursos ao utilizar este processador. Contudo, deve-se considerar que a medida de número de ciclos executados *não reflete o tempo de execução* destes programas porque as faixas de freqüência dos três processadores são *muito diferentes*: a freqüência máxima do Atmega é 16 MHz, enquanto que a freqüência máxima do Motorola é 80 MHz. A decisão por um ou por outro processador deve, necessariamente, considerar a natureza e os requisitos do projeto —especialmente o custo e o tempo de resposta dos aplicativos.

Outro ponto que deve ser considerado refere-se à memória interna ao processador, especialmente quanto a esta permitir a execução da aplicação sem a necessidade de inclusão de memória externa. Os custos da adição de memória externa a um processador mais simples/barato podem ultrapassar o custo de um processador mais complexo e com mais memória interna.

4.2 Precisão dos Modelos

Com o objetivo de avaliar a precisão dos simuladores foram executadas comparações entre o simulador e um módulo de hardware do processador Motorola DSP56827. A Tabela 3 mostra os resultados para o tempo de execução, medido em ciclos. A coluna marcada "S/H" mostra a razão entre os valores produzidos pelo simulador (SIM) e as medidas no processador (HW).

O processador executa até 20% mais ciclos que o estimado pelo simulador. Isto ocorre porque o simulador ignora alguns dos ciclos adicionais devidos a desvios tomados. Para corrigir esta deficiência, é necessário escrever um

Programa	*# Ciclos*		
	SIM	*HW*	*S/H*
negcnt	18022	20036	0,90
int2bin	2144	2566	0,84
gcd	162542	171110	0,95
cast	344	422	0,82
fib	74122	86022	0,86
bubble	$27{,}2{\cdot}10^6$	$32{,}8{\cdot}10^6$	0,83
insertion	$14{,}5{\cdot}10^6$	$17{,}6{\cdot}10^6$	0,82
quick	875233	897174	0,97
selection	$18{,}1{\cdot}10^6$	$22{,}1{\cdot}10^6$	0,81
shell	$5{,}9{\cdot}10^6$	$6{,}7{\cdot}10^6$	0,88
crivo	$12{,}6{\cdot}10^6$	$14{,}6{\cdot}10^6$	0,86

Tabela 3. Comparação entre modelo e processador DSP56827.

modelo mais detalhado do processador que simule exatamente o comportamento dos desvios. A versão segmentada dos modelos está em desenvolvimento e nossa espectativa é que a precisão das medidas seja melhor do que 95%.

4.3 Estudo de Caso: Conversor Serial-IP

Umas das motivações para o desenvolvimento deste trabalho foi a participação no projeto do *Conversor Serial-IP* (CSI) [19], em execução no Lactec [11]. O CSI consiste de uma plataforma de hardware, que utiliza o processador Motorola DSP56F827, e um módulo de comunicação celular com a tecnologia *General Packet Radio Services* (GPRS). Este dispositivo é conectado a uma Unidade Terminal Remota (UTR) e funciona como uma ponte entre a UTR e o Centro de Operação da Distribuição de Energia (COD). A UTR e o COD comunicam-se através do protocolo serial DNP3 [6] e utilizam o CSI para enviar as mensagens pela Internet sobre o protocolo TCP/IP [19].

Devido ao crescimento da utilização de protocolos de comunicação via Internet em sistemas embarcados e ao gargalo de desempenho que estes protocolos provocam, foi efetuado um teste com a pilha TCP/IP/PPP, utilizada no projeto CSI, com o objetivo de comparar os simuladores dos processadores DSP57F827 (DSP) e Atmega8515 (AT) com o hardware do projeto CSI (baseado no DSP56F827).

O teste consiste em empacotar e desempacotar um conjunto de dez inteiros utilizando os protocolos TCP/IP e PPP (O protocolo *Point-to-Point Protocol*, PPP, é utilizado para efetuar a conexão GPRS para transmissão de dados via através da rede de celular.), sem considerar as funções de conexão e retransmissão de dados. A Tabela 4 mostra os resultados obtidos.

Os resultados seguem o padrão dos testes anteriores, o simulador atinge 85% de precisão na métrica de número de ciclos em relação ao hardware. O tamanho do programa em bytes e o número de instruções executadas pelo Atmega8515 é 1,7 e 2,24 vezes, respectivamente, do que para o DSP56F827. Isto se deve a eficiência do conjunto de instruções do processador da Motorola. A utilização da memória de dados praticamente não se altera entre os dois processadores. O número de ciclos executados no Atmega8515 é menor que no DSP56F827, devido a esta aplicação utilizar as instruções mais simples do Atmega8515, que consomem apenas um ciclo de relógio.

Métricas	*HW*	*DSP*	*AT*
# Instruções	7132	7239	16160
# Ciclos	33382	27830	24077
Dados [bytes]	–	666	634
Código [bytes]	4282	4404	7660

Tabela 4. Resultado Obtido com um Trecho da Pilha TCP/IP/PPP.

A medida do consumo de memória de dados não é explícita na ferramenta *CodeWarrior*, que gerencia o módulo de hardware do CSI. Ao invés disso o *CodeWarrior* mostra o número de acessos a memória RAM que é aproximadamente 13440 acessos. A ferramenta ArchC disponibiliza esta informação no arquivo de estatísticas, e para a pilha TCP/IP/PPP utilizando o simulador DSP56F827 fica em torno de 15028 acessos, cerca de 11% a mais que o medido no módulo de hardware.

De acordo com os testes, a escolha do processador DSP56F827 para o projeto CSI foi a mais acertada, pois o mesmo possui o melhor desempenho em termos de tempo de execução e utilização de memória. O Atmega8515 só poderia ser utilizado para esta aplicação com a inserção de memória externa, já que apenas um trecho da pilha TCP/IP utilizou 95% da memória Flash interna.

5. Conclusão e Trabalhos Futuros

O mercado de desenvolvimento de aplicações embarcadas sofre de alguma carência de ferramentas que auxiliem na redução de custos através de melhorias no aproveitamento dos recursos. Além disso, é indiscutível a utilidade de ferramentas que permitam que a codificação dos aplicativos possa ser iniciada antes da conclusão do hardware, reduzindo-se assim o tempo de realização do projeto.

Ao utilizar o conjunto de simuladores descrito aqui, o projetista é capaz de escolher com segurança o microprocessador ou microcontrolador que melhor se adapta a sua aplicação, nos quesitos de custo, capacidade de processamento e quantidade de memória utilizada. Além disso, os simuladores podem ser utilizados para antecipar a codificação dos aplicativos antes que haja hardware disponível. A versão atual do simulador do Motorola

DSP56F827 possui precisão melhor que 80%, nas medidas de tempo de execução, e 89% a 97% nas medidas de utilização de memória quando comparado com uma implementação em hardware. Isto permite um alto grau de confiabilidade na escolha do processador.

Os testes realizados permitiram a comparação dos microprocessadores em relação ao tempo de execução e memória utilizada. O processador DSP56F827 apresentou a melhor relação custo/benefício, devido principalmente à eficiência do seu conjunto de instruções que possibilita uma melhor utilização da memória de programa. A eficiência do conjunto de instruções aliada ao fato de que cada instrução necessita em média dois ciclos de relógio para executar faz com que o DSP56F827 seja a melhor opção para aplicações de "médio" e "grande" porte.

A avaliação dos resultados também permite concluir que no momento da escolha do microprocessador para um projeto não basta apenas o levantamento das características básicas do processador, como quantidade de memória e freqüência do relógio, é necessária uma avaliação mais cuidadosa da arquitetura, como verificação de segmentação e do conjunto de instruções. Por exemplo, o processador Atmega8515 disponibiliza uma quantidade razoável de memória e é mais barato, porém os programas gerados ocupam bastante memória de programa, o que pode requerer inclusão de memória externa, aumentando o custo e diminuindo o desempenho do sistema final.

Dentre as atividades previstas para o futuro próximo estão a conclusão dos testes com o microprocessador Rabbit R2000 com a correção do problema entre o ambiente de desenvolvimento e o módulo de hardware, a modelagem da versão segmentada dos três processadores, e a coleta e avaliação de um conjunto maior de programas de teste. Com estas tarefas concluídas, será possível efetuar comparações precisas do desempenho dos microprocessadores modelados.

As versões multi-ciclos dos simuladores serão enviadas para avaliação da equipe de desenvolvimento ArchC para possível disponibilização dos modelos na página do projeto. No médio prazo, os simuladores serão adaptados para a versão 2 do ArchC, que permite a aferição de métricas como a utilização de memória através de *estimadores* definidos pelo programador.

Referências

[1] R. Amicel and F. Bodin. Mastering startup costs in assembler-based compiled instruction-set simulation. In *Sixth Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT.02)*, 2002.

[2] The ArchC Arquitecture Description Language, fev 2005. `http://www.archc.org`.

[3] ATMEL Avr microcontrollers, mar 2005. `http://www.atmel.com`.

[4] Metrowerks Code Warrior IDE, mar 2005. `http://www.metrowerks.com/MW/Products/CodeWarrior+Technology.htm`.

[5] CS.UCR.edu. Dalton project, fev 2006. `http://www.cs.ucr.edu/~dalton/i8051/i8051syn`.

[6] Distributed network protocol, fev 2006. `http://www.dnp.org`.

[7] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation and synthesis. *Proceedings of the IEEE*, 85(3):366–390, March 1997.

[8] T. Groetker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*, volume 1. Webman, 2002.

[9] A. Halambi, P. Grun, V. Ganesh, A. Khare, N.Dutt, and A. Nicolau. Expression: A language for architecture exploration through compiler/simulator retargetability. In *European Conference on Design, Automation and Test*, 1999.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3rd edition, 2003.

[11] LACTEC Instituto de Tecnologia para o Desenvolvimento, mar 2005. `http://www.lactec.org.br`.

[12] Motorola DSP family, mar 2005. `http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=012795`.

[13] RABBIT Semiconductor, mar 2005. `http://www.rabbitsemiconductor.com/products/dc/index.shtml`.

[14] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. ArchC: A systemc-based architecture description language. In *16th Symposium on Computer Architecture and High Performance Computing*, Foz do Iguacu, Brazil, Outubro 2004.

[15] S. Rigo, R. Azevedo, and G. Araujo. The ArchC architecture description language. Technical report, IC - Unicamp, Junho 2003.

[16] Rabbit WinIDE, mar 2006. `http://www.softools.com/`.

[17] SystemC.org. SystemC community, mar 2005. `http://www.systemc.org`.

[18] D. Tennenhouse. Proactive computing. *Communications ACM*, 43(5):43–50, 2000.

[19] V. Zambenedetti, R. P. Siqueira, F. R. Coutinho, A. A. Barbiero, J. Pereira, and R. A. Hexsel. Uso de comunicação celular digital utilizando a tecnologia 2.5G para sistemas de automação de energia elétrica. In *VI SIMPASE, São Paulo, Brasil*, Julho 2005.

[20] V. Zivojnovic, S. Pees, and H. Meyr. Lisa - machine description language and generic machine model for hw/sw codesign. In *IEEE Workshop on VLSI Signal Processing, San Francisco*, 1996.