

LABORATÓRIO DE SISTEMAS DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO
INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Porte de um Sistema Operacional para uma Arquitetura Reconfigurável

Projeto de Iniciação Científica

Aluno:	Gustavo Svezut Barbieri <i>gustavo.barbieri@ic.unicamp.br</i>
Orientador:	Prof. Rodolfo Jardim de Azevedo <i>rodolfo@ic.unicamp.br</i>
Conteúdo:	1. Introdução e Motivação 2. Projeto 3. Cronograma 4. Bibliografia

1 Introdução e Motivação

A crescente demanda por dispositivos portáteis e de controle embutidos, combinados com o aumento na disponibilidade de transistores por unidade de área, vem resultando num crescente interesse por arquiteturas especializadas dentro da computação, entendendo-se por arquiteturas especializada aquela otimizada para alcançar requisitos de desempenho, consumo, etc, para uma dada aplicação ou conjunto de aplicações de uma mesma área. O projeto ChameLeon[2], do qual esse é um sub-projeto, tem como objetivo a especialização de processadores para sistemas dedicados e pretendemos com isso desenvolver um ambiente que sirva como base para a realização de pesquisa pioneira em técnicas de especialização de arquiteturas e de compilação para arquiteturas dedicadas e reconfiguráveis. A experiência internacional (projetos SPAM, SUIF e IMPACT) tem demonstrado que sem uma infra-estrutura sólida, os resultados da pesquisa ficam restritos ao meio acadêmico. Estes resultados frequentemente são ignorados pela indústria quer seja por não terem sido validados em sua plenitude, através de benchmarks realistas, quer seja por não virem acompanhados de uma análise criteriosa de sua interação com os outros componentes do sistema. Assim sendo, o objetivo fundamental do ChameLeon é garantir que as novas arquiteturas e compilador resultantes sejam confiáveis, de modo a permitir uma base sólida sobre a qual novos algoritmos possam ser pesquisados.

Nesse projeto de Iniciação Científica, pretendemos portar o sistema operacional μ CLinux[10] para o processador Leon[3, 4] que implementa a arquitetura SPARC V8[14] e executa-lo num *kit* de desenvolvimento XESS XSV800[15] com FPGA¹ da Xilinx[7]. Atualmente, todos os testes do projeto ChameLeon que são executados no *kit* são implementados de forma a não precisar de nenhum sistema operacional. Isso exige que sejam feitas mudanças no código fonte de cada um dos programas a testar ou do compilador. Pretendemos com essa implementação do μ CLinux, subir o nível de teste e permitir a utilização das funcionalidades fornecidas pelo sistema operacional como: medição do tempo real de execução de um processo, sistema de arquivos, comunicação via rede, entre outros. O trabalho do Gustavo será de grande importância no desenvolvimento do projeto ChameLeon e a participação dele no grupo de pesquisa também será muito importante para a sua formação. Atualmente o grupo de participantes do projeto ChameLeon possui: 3 professores doutores, 2 alunos de doutorado, 3 alunos de mestrado e 1 aluno de iniciação científica. O Gustavo já começou a assistir as reuniões do grupo e também já está se familiarizando com o ambiente de trabalho.

¹*Field Programmable Gate Array*

Embora parte do projeto ChameLeon envolva o projeto e implementação de hardware, o Gustavo vai trabalhar apenas na parte de software. Acreditamos que focar o trabalho neste momento vai permitir um maior avanço na direção desejada.

1.1 O Projeto ChameLeon

O objetivo do projeto ChameLeon é a especialização de processadores para sistemas dedicados. Para isso, trechos de código freqüentemente utilizados por uma aplicação são transformados em hardware e acessados através de novas instruções que são incluídas no *datapath* do processador. Dado o código de uma aplicação e uma versão básica de um processador RISC, identificam-se conjuntos de instruções (denominados padrões) que são executados com maior freqüência através da realização de *profiling*. Cria-se em seguida uma descrição HDL que implemente o padrão em unidades funcionais a serem inseridas no processador, levando à obtenção de novas instruções. Finalmente, modifica-se o *back-end* de um compilador para que reconheça e possa gerar código que faça uso das novas instruções, levando a executáveis altamente otimizados. Este processo inteiro, idealmente, seria realizado de forma automatizada. O fluxo de desenvolvimento pode ser descrito através das seguintes fases:

1. O código fonte de uma aplicação arbitrária, escrito em linguagem de alto nível, deve ser fornecido de modo a poder ser executado de forma não interativa;
2. O código é compilado utilizando o GCC, com opções de instrumentação para realização de *profiling*;
3. Gera-se um binário para arquitetura SPARC V8, cuja especificação é seguida pelo processador Leon;
4. Realiza-se o *profiling* executando a aplicação, sendo assim possível escrever em disco os arquivos com as estatísticas de execução (contagem do número de vezes que cada bloco básico foi executado);
5. Os arquivos resultantes do *profiling* (item 4) são analisados numa nova compilação do código fonte (item 2). Com isso, é possível gerar uma representação intermediária do programa anotada com as estatísticas de execução. No caso do GCC, a representação intermediária se chama RTL (Register Transfer Language). Cada instrução RTL é anotada com o número de vezes que foi executada;

6. Padrões (pequenos trechos) de código típicos e altamente executados são manualmente identificados em aplicações que compõem benchmarks para sistemas dedicados, como MediaBench[8]. Uma descrição otimizada em VHDL é também escrita (manualmente numa versão inicial) para implementação daqueles padrões em hardware. Inicialmente, buscam-se padrões que se sabe que, se implementados em hardware, dão grande ganho de velocidade de execução a um custo muito baixo de hardware.
7. Os padrões identificados no item acima, com suas implementações VHDL, são adicionados a uma biblioteca de padrões;
8. O objetivo da fase de seleção de padrões é decidir quais os padrões que trarão maior ganho em velocidade de execução para a aplicação se implementados em hardware. Um processo automático faz o casamento dos padrões da biblioteca sobre os padrões da aplicação, buscando sobreposições. Mas também padrões da própria aplicação podem ser casados sobre outros padrões da aplicação;
9. Os padrões selecionados são unidos num procedimento chamado *merging*. Trata-se basicamente de tentar reutilizar unidades funcionais e especialmente suas interconexões. Uma das alunas de doutorado do LSC, Nahri Moreano, em conjunto com o Prof. Guido Araújo e pesquisadores da universidade de Princeton, EUA, publicaram recentemente um artigo a respeito do problema[11];
10. O resultado do *merging*, representado na forma de um grafo de fluxo de dados e de controle (CDFG), ou possivelmente na forma de uma versão simplificada das instruções RTL da representação intermediária do GCC, é automaticamente convertido em código VHDL.
11. O código VHDL original do processador Leon será alterado (apenas 1 vez) para incluir interfaces configuráveis, porém pré-definidas, com a unidade de inteiros do processador, assim como o preparo para a inclusão de sinais de controle adicionais. Estas interfaces permitirão a anexação de unidades funcionais adicionais e a decodificação de novas instruções para o processador;
12. O código VHDL dos padrões é automaticamente mesclado com o código do Leon, utilizando as interfaces citadas no item 11;

13. O código VHDL é sintetizado em hardware. Nos protótipos de laboratório, utilizamos FPGAs para fins de validação, mas na indústria a síntese poderia ser feita em máscaras para confecção de ASICs;
14. O processo de inserção de unidades funcionais no Leon para execução de n padrões deverá produzir n novas instruções, para as quais serão alocados novos opcodes e mnemônicos;
15. A representação intermediária da qual foram extraídos os padrões será editada para substituir as instruções (RTL) que os compõem pelas novas instruções especializadas (item anterior). Além disso, a *machine description* do GCC, que é uma linguagem de descrição de máquina capaz de descrever o conjunto de instruções de um processador, deve ser alterada para que o *back-end* do processador (incluindo o montador) possa ser capaz de gerar código binário para as novas instruções. Com o reconhecimento de novas instruções, uma recompilação do código fonte pode permitir que a etapa de geração de código identificasse novas possibilidades de cobertura (tiling) de padrões, gerando código mais eficiente;
16. Obtém-se o novo arquivo binário, adaptado para execução no novo processador. Um simulador que reconhecesse as novas instruções permitiria a simulação do novo arquivo binário sem a necessidade da implementação física do processador;
17. O novo processador resulta da síntese discutida no item 13.

O trabalho desta Iniciação Científica ajudará na realização do item `refitem:profiling` do projeto ChameLeon. Atualmente os programas são compilados sem o suporte a sistema operacional, o que exige a modificação do código fonte ou dos programas do *benchmark* ou mesmo do compilador. Com um sistema operacional disponível, o desenvolvimento e teste de novos aplicativos será um processo mais eficiente.

1.2 O Processador Leon

O processador utilizado no projeto ChameLeon foi o Leon[3, 4], que é um *core* escrito em VHDL e sintetizável para várias tecnologias de FPGAs e ASICs. O Leon implementa o conjunto de instruções do padrão SPARC V8[14], e foi desenvolvido inicialmente para aplicações

da Agência Espacial Européia, sendo colocado em domínio público posteriormente. O processador Leon possui as seguintes características:

- *Pipeline* de 5 estágios que implementa o conjunto de instruções SPARC V8;
- Unidades de multiplicação, divisão e MAC (*Multiply and Accumulate*) em hardware;
- *Cache* de dados e instruções separadas, ambas *direct mapped*;
- Implementação completa dos barramentos AHB e APB para interligação de periféricos internos seguindo o padrão AMBA-2.0 [?];
- *Cache* de dados capaz de efetuar *snooping* do barramento AHB;
- Controlador programável de memória RAM e ROM externas com barramentos de 8, 16 ou 32 bits;
- Unidade de depuração interna para inspeção do estado do processador sem afetar a execução do programa;
- Diversos periféricos como UART, temporizadores, controlador de interrupção, uma porta de entrada e saída de 16 bits e um controlador Ethernet;
- Interface com a unidade de ponto flutuante Meiko e um co-processador definido pelo usuário;
- Unidade de ponto flutuante IEEE-754 própria que permite a soma, subtração e comparação (outras operações têm que ser implementadas por software);
- Modo de economia de energia.

No LSC existem 2 *kits* de desenvolvimento XESS XSV800 [1, 15] (Figura ??), nos quais o processador Leon é implementado. Os principais componentes do *kit* são:

- FPGA Virtex XCV800 [16] da Xilinx [7], que é o módulo principal da placa e onde o Leon é implementado;
- CPLD XC95108 que é utilizado para controles e programação da FPGA Virtex;
- Dois bancos de memória SRAM de 512K x 16 e 16 Mbit de memória Flash;

- Oscilador de 100MHz com divisor programável (entre 1 e 2052);
- Conectores para interface paralela e serial;
- Conector RJ-45 para implementação de interface Ethernet;

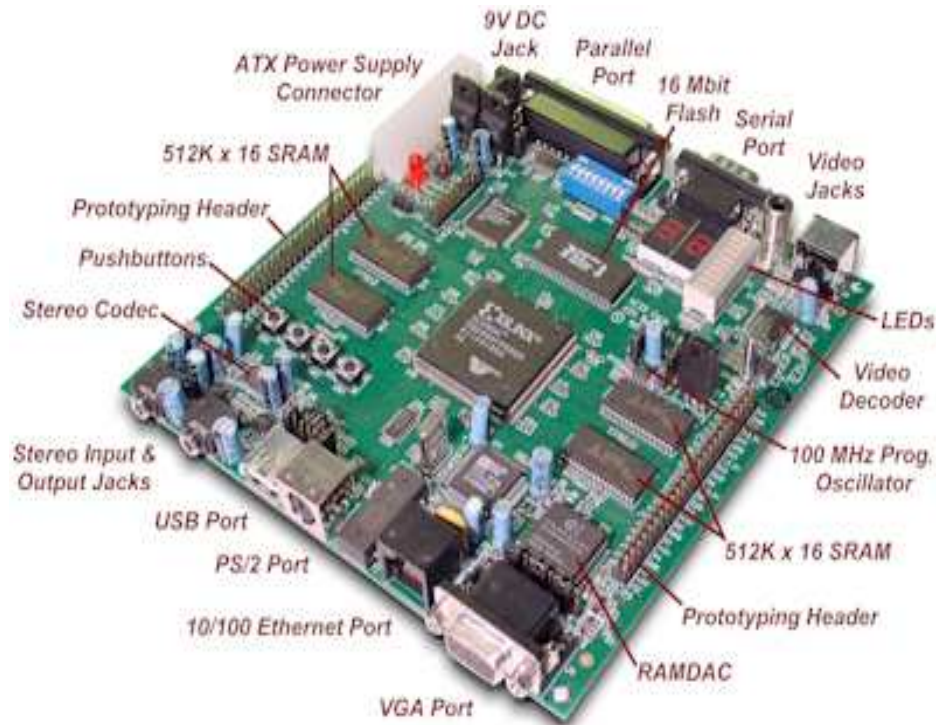


Figura 1: Placa XESS XSV800

2 Projeto

Pretendemos ao final desse projeto ter um porte do μ CLinux capaz de ser executado no Leon e no *kit* de desenvolvimento.

A escolha do μ CLinux no lugar de uma distribuição convencional do Linux é devido à pequena quantidade de memória existente no *kit* de desenvolvimento (apenas 2Mb, sem dispositivo de armazenamento secundário). Como o μ CLinux foi desenvolvido a partir do

Linux e direcionado para dispositivos dedicados, ele ocupa muito menos memória que o Linux convencional. Outra característica do μ CLinux é que ele foi desenvolvido para processadores sem MMU², o que permite a sua implementação utilizando a distribuição principal do Leon visto que a implementação de uma MMU ainda não está integrada ao processador, nem possui suporte neste momento.

Na fase de familiarização com o ambiente, pretendemos ampliar o suporte do compilador para o processador Leon, mantendo sempre a versão mais atual. As duas formas de simulação de entrada/saída, para aquisição e escrita dos *benchmarks*, que estamos considerando, são:

- Emular a entrada/saída através do canal de comunicação já disponível, que é a interface serial RS-232. Por ser um dispositivo lento, as transferências de dados demoram muito tempo. O Gustavo trabalhará, no início do projeto, na implementação de um canal paralelo de comunicação, através do qual pretendemos ter um aumento substancial de desempenho. Essa abordagem pode ser utilizada pelos programas que não rodarão sobre um sistema operacional, portanto o código a ser gerado vai ser ligado à biblioteca básica do compilador, a qual gera as instruções de entrada e saída para a arquitetura desejada. A emulação da entrada/saída de dados será através a biblioteca básica do compilador que será alterada de modo que ao ser realizada uma chamada `open`, `read`, `write` ou `close`, esta chamada passará as instruções ao computador ao qual a FPGA está conectado, sendo neste executadas e os resultados serão retornados à aplicação residente no FPGA. Pela pequena quantidade de memória do *kit* de desenvolvimento, emular a entrada/saída através da memória não será possível;
- O porte do sistema operacional será a segunda fase do projeto. Para isso, o μ CLinux foi escolhido por não exigir a presença da MMU e também por executar em pouca memória. Cabe ressaltar que o objetivo final do ChameLeon é ter um ambiente para a execução de programas e se o sistema operacional ocupar toda a memória disponível, não será possível executar os *benchmarks*. A adoção do μ CLinux integra perfeitamente no ambiente de desenvolvimento com GCC, além de facilitar o ambiente de testes provendo interface para rede, pilha TCP/IP, suporte a arquivos em memória Flash, além de podermos utilizar um dos vários escalonadores disponíveis: pré-emptivos de propósito geral, tempo-real baseado em *deadline* ou baseado em entrada/saída, com os quais

²Memory Management Unit

também podemos controlar a latência deste na aplicação, atendendo aos propósitos desejados. Como tarefa adicional ao porte do sistema operacional, também será necessária a escrita de *drivers* de dispositivos para periféricos com maior capacidade de transferência de dados, como o controlador Ethernet. O projeto LEOX[9] já trabalha na adaptação do μ CLinux para o Leon.

Com a junção do ChameLeon com o μ Linux pode-se produzir, então, processadores especializados já com sistema operacional embutido, sendo uma solução altamente viável para sistemas embarcados, principalmente sistemas multimídia, nos quais existem padrões no código que ao serem otimizados pelo ChameLeon podem ter tempo de execução reduzidos drasticamente.

3 Cronograma

O cronograma completo até julho de 2004 encontra-se na tabela 1 e possui as seguintes fases:

1. Estudo da biblioteca básica do GCC;
2. Implementação da comunicação paralela na biblioteca básica do GCC;
3. Estudo do μ CLinux;
4. Porte do μ CLinux para o Leon utilizando o *kit* XESS XSV800;
5. Criação/inclusão dos drivers dos novos periféricos no μ CLinux (Ethernet e outros);
6. Relatório;

Atividade	2003					2004						
	08	09	10	11	12	01	02	03	04	05	06	07
1	•	•										
2		•	•	•								
3				•	•	•	•					
4					•	•	•	•	•	•		
5										•	•	•
6						•						•

Tabela 1: Cronograma de Atividades: agosto/2003 até julho/2004

Referências

- [1] XESS Corporation. <http://www.xess.com>.
- [2] Laboratório de Sistemas de Computação. Projeto chameleon. In <http://www.lsc.ic.unicamp.br>.
- [3] Jiri Gaisler. *The LEON Processor User's Manual*. Gaisler Research, novembro 2001.
- [4] Research Gaisler. Leon – processador compatível com SPARC V8. In <http://www.gaisler.com>.
- [5] The GNU Compiler Collection. Compiladores e ferramentas de desenvolvimento feitos em código aberto. In <http://www.gnu.org/software/gcc>.
- [6] The GNU Lesser General Public License. Licença lgpl. In <http://www.gnu.org/licenses/licenses.html>.
- [7] Xilinx Inc. <http://www.xilinx.com>.
- [8] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.
- [9] O time LEOX. Leox – **LE**on toll**BO**X. In <http://www.leon.org>.
- [10] Linux. Implementação unix livre, de código aberto, compatível com padrões posix. In <http://www.linux.org>.

- [11] Nahri Moreano, Guido Araujo, Zhining Huang, and Sharad Malik. Datapagh merging and interconnection sharing for reconfigurable architectures. In *Proc. ACM/IEEE Int'l Symp. on System Synthesis*, outubro 2002.
- [12] μ CLinux. Implementação unix livre, de código aberto, compatível com padrões posix que executa em máquinas sem mmu. In <http://www.uclinux.org>.
- [13] Corporação International Sparc. Sparc – Scalable Processor **ARC**chitecture. In <http://www.sparc.org>.
- [14] SPARC International Inc. *SPARC V8 Architecture Manual*.
- [15] XESS Corporation. *XSV Board V1.1 Manual*, setembro 2001.
- [16] Xilinx inc. *Virtex Field Programmable Gate Arrays Specification*, abril 2001.