BananaCore - Processador em VHDL

Rogiel Sulzbach Jefferson Johner Matheus Oliveira

4 de Dezembro de 2015

Conteúdo

Capítulo 1

Introdução

Escrever capítulo de introdução

Capítulo 2

Especificação

A especificação inicial para o nosso projeto consiste em um processador de arquitetura do tipo Von Neumann[1] com instruções básicas do tipo:

- Operações de memória: Carregar dado da memória, armazenar dado na memória
- Operações aritméticas: adição, subtração, multiplicação e divisão
- Operações de IO: Write port e Read port
- Operações bit-a-bit: AND, NAND, OR, NOR, XOR e NOT

Como o objetivo do trabalho é desenvolver apenas um processador – não estamos interessados em como armazenar o programa no FPGA – escolhemos por gravar o programa de forma fixa; isto é, o programa é armazenado direto na memória RAM como um valor inicial.

Sabendo da limitação desta implementação, o design deve permitir que seja fácil substituir esta implementação inicial por outra mais funcional e completa.

[1] A arquitetura Von Neumann consiste em uma única memória compartilhada tanto para dados como para programas.

Capítulo 3

Implementação

3.1 Controlador de Memória

O controlador de memória é a unidade que faz o intermédio ao acesso a memória do processador; nele, a operação de escrita é validada e então enviada para um respectivo memory bank (ver seção 3.1.1).

Descrever o funcionamento básico do controlador

3.1.1 Memory Bank

O memory bank é uma entidade simples e serve como uma abstração para um módulo de memória RAM genérico. O otimizador do Quartus II, ao detectar a presença de um grande bloco de dados, automaticamente executa uma otimização e substitui este por um bloco de memória RAM.

3.2 Controlador de Registrador

Para evitar que fosse necessário injetar um grande número de sinais de acesso a dados, controle e status dos registradores, uma abstração semelhante ao acesso à memória foi criada para simplificar este desenvolvimento. Detalhes desta implementação serão omitidos, pois são muito semelhantes ao barramento de memória.

3.3 Controlador de Instruções

O controlador de instruções é, sem dúvida, uma das partes com maior quantidade de código descrevendo hardware; isto se dá devido a uma dificuldade

que encontramos ao implementar um barramento único de acesso à memória e aos registradores.

3.3.1 Decodificador de instruções

Detalhar a implementação atual

3.3.2 Executor de instruções

Cada instrução foi dividida em uma entidade chamada de *executor*. Esta entidade é responsável por fazer o carregamento de dados, execução da instrução e armazenamento do resultado final. Devido a repetição de código nestes executores, utilizamos geradores para gerar grande parte do código de forma automática e simples (ver seção **??**).

3.3.3 Acesso à memória

Inicialmente, pretendíamos implementar o acesso global a memória por via de um barramento delimitado por *buffers tri-state*, contudo, esta implementação se mostrou muito complexa, pois ao incrementar as implementações de instruções do processador o barramento entrava em um estado inválido pois mais de dois sinais tentavam ser escritos no barramento em simultâneo. Acreditamos que estes problemas são devidos a falhas de design da arquitetura e que poderiam ser resolvidas escolhendo uma forma alternativa de implementação das instruções.

A solução desde problema, embora não seja ideal, foi simples: um grande MUX foi implementado de forma a fazer o "controle"de acesso ao barramento principal. Esta solução tem um grave problema: a necessidade de escrever código cresce muito em função da quantidade de instruções implementadas. Para um processador simples como o BananaCore isto pode não ser um problema muito relevante; mas para implementações maiores isto pode ganhar uma faceta muito mais adversa. Como forma de solucionar, parcialmente, este empecilho, fizemos uso de artefatos de geração de código para gerar os muxes e demais condições do decodificador de instruções.

Detalhar a implementação atual

3.3.4 Geração de código

Para gerar os códigos repetitivos e forma algorítmica, fizemos usos de duas ferramentas distintas: Cog[1] e um script personalizado de geração de executores de instruções.

Mais detalhes da geração de código podem ser extraídas do código fonte do projeto disponível no GitHub.

3.4 Especificação final

Escrever seção de especificação final

[1] Cog é um aplicativo que executa códigos Python contidos em comentários do código fonte e substitui sua saída ao final da execução. O programa está disponível publicamente em `http://nedbatchelder.com/code/cog/`

Capítulo 4

Testes

Escrever capítulo de testes

Capítulo 5

Conclusão

Escrever capítulo de conclusão