Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Sistemas Eletrônicos - PSI

PSI-2553- Projeto de Sistemas Integrados

O Processador Plasma - Parte 2
(Compilação no Quartus e simulação no ModelSim)

Objetivos

Esta experiência visa familiarizar o estudante com a arquitetura e funcionamento do processador MLITE. Será realizado um mapeamento deste processador sobre uma FPGA comercial e depois será realizado o estudo do programa compilado e sua simulação numa implementação específica do processador que adota a técnica de segmentação (*pipelining*).

Parte prática

Preparando as pastas e arquivos

1. Criar o diretório **X:\psi2553\exp2\plasma_2.**
2. Copiar para esta pasta todos os arquivos que estão em **Rede\NEWSERVERLAB\psi2553\exp2\projeto.** Eles correspondem aos arquivos VHDL do sistema plasma e os arquivos de simulação.
3. Observe os códigos VHDL, plasma.vhd, mlite_CPU.vhd e RAM.vhd. Procure entender as interfaces com o auxílio do diagrama da apostila teórica.
4. No Windows, selecione **Start→All Programs→Acessories→Command Prompt**. Uma janela de terminal (prompt de comando) se abrirá.
5. Tecle X: <enter>.
6. Digite o comando **cd X:\psi2553\tut_mips\tools** para acessar a pasta utilizada na exp1.
7. Digite **set path=.** <enter>.
8. Gere os arquivos de programa com o comando **gmake tohex**

Isto gerou os arquivos code0.hex, code1.hex, code2.hex e code3.hex. Eles integrarão as memórias RAM, em formato hex, como descrito na parte teórica da exp1. A interpretação exata do formato hex é dada no Apêndice.

9. Copie os 4 arquivos code_0.hex, code_1.hex, code_2.hex e code_3.hex da pasta **X:\psi2553\tut_mips\tools** para **X:\psi2553\plasma_2.**

10. Digite o comando **cd X:\psi2553\plasma_2** para acessar a pasta desta exp2.

11. Edite o arquivo RAM.vhd para incluir os 4 arquivos hex. Para isto, localize a descrição das 4 memórias da Altera e faça as substituições necessárias (basta acrescentar os nomes dos arquivos nas linhas **lpm_file => ???.hex**). Atenção: O arquivo *code0.hex* corresponde ao conteúdo do *byte* mais significativo da memória.

Mapeamento e Simulação no Quartus

1) Carregue o programa *Quartus II* da Altera clicando ***Iniciar->Programas->Altera->Quartus II 9.1*** que será usado para fazer a compilação do projeto e as simulações**.**

2) Crie um projeto como o nome *plasma*.

a. Clique em ***New Project Wizard***.

b. Atribuir o nome "**plasma**" ao projeto

c. Incluir todos os arquivos *.vdh

d. Assinalar o dispositivo na caixa de diálogo seguinte (EP2C35F672C6)

e. No campo de simulação, adicionar em ***Tool Name***: ***ModelSim-Altera*** e em *Format*: **VHDL**

Após finalizar esta etapa é possível observar as características do projeto na janela ***Project Navigator***. Clicando duas vezes sobre o *plasma.vhd* (em ***Files***) poderá visualizar o código VHDL do projeto. Lembre-se que os arquivos *.hex já estão incluídos pela referência feita nas memórias RAM.

O simulador ***ModelSim*** será utilizado pela facilidade de se acessar os sinais internos do projeto. Para configurar o ***Quartus*** para este objetivo:

3) Clique em ***Assignments > EDAToolSettings > Simulation***.

4) Na opção "Compile test bench":

a. Clique em testbench

b. Clique em NEW

c. Completar: "*Test bench name*" = *plasma_tbw*; "*Top level module in test bench*" = *plasma_tbw*; "*Design instance name in test bench*" = *behavior*.

d. No campo "*Test bench files*" adicionar o arquivo "*plasma_tbw.vhd*"

5) Na opção "*Use script to set up simuation*" indique a rota do arquivo "*inicio.txt*", que será o *script* para a simulação no ModelSim.

6) Abra os arquivos ***plasma_tbw.vhd*** e ***inicio.txt*** e observe os seus conteúdos. Use a Figura 2 da apostila teórica para associação com os sinais de *inicio.txt*. A Figura 3 desta apostila mostra os sinais entre os módulos que compõem o processador MLITE. A Figura 4 ilustra as alternativas de execução em pipeline.

Anote no seu roteiro

- **Nomes dos sinais presentes no arquivo plasma_tbw.vhd**
- **Valor do período de relógio**

7) Para compilar o arquivo *plasma.vhd*, clique em:

a. ***Processing*** > ***Start > Start Analysis & Synthesis***.

O resultado da compilação é apresentado numa janela tipo *pop-up*.

b. Pressione ***OK***.

Uma janela com o relatório da compilação é aberta automaticamente ou pode ser acessada pela opção do menu

c. ***Processing*** > ***Compilation report***.

Anote no seu roteiro

- **código do componente**
- **número de elementos lógicos utilizados**
- **número de registradores utilizados**

8) Realize a simulação funcional do arquivo plasma. Para realizar a simulação funcional:

a. ***Tools > Run EDA Simulation Tool > EDA RTL Simulation***.

O ModelSim deverá abrir apresentando o resultado da simulação especificada no *testbench*.

ATENÇÃO: lembrar que estão sendo observados os sinais em cada estágio de um *pipeline*. Portanto existe uma defasagem no tempo entre os sinais que correspondem a cada instrução. Por exemplo, se o resultado do estágio *fetch* ocorrer num ciclo *i* o resultado do estágio *decode/control* correspondente à mesma instrução deverá ocorrer num ciclo posterior *i+n*.

9) Na simulação, você deverá:

a. Identificar todos os sinais que aparecem na simulação e correlaciona-los com o modelo que aparece na apostila de teoria.

b. Identificar a sequência dos códigos das instruções (*opcodes*) mostrados na simulação e verificar que são os que aparecem no código *assembly* gerado na aula plasma_1. Para isto observe o conteúdo do arquivo ***test.lst***:

c. Identificar a sequência dos endereços das instruções mostradas na simulação e verificar que são as que aparecem no arquivo ***test.lst***.

10) Salve o resultado da simulação, nos trechos de interesse (veja a seguir os momentos da simulação que devem ser observados).

Anote no seu roteiro

Dica: modifique o tipo de apresentação dos sinais no ModelSim de binário para inteiro e hexadecimal (Radix) para facilitar a análise da simulação.

- **Identificar no ModelSim os nomes dos seguintes sinais:**
  1. **O registrador de instruções**
  2. **O registrador do *opcode***
  3. **Os registradores do *Register file***
  4. **O endereço de escrita na memória**
  5. **O valor dos dados**
- **Quais registradores do *Register File* foram utilizados durante a simulação (anotar os números e nomes destes registradores)**
- **Quantas ocorrências de *write_enable* ocorreram ao longo de sua simulação? O que significam estas ocorrências?**
- **Anote os *opcodes*, endereços e instantes de tempo de simulação dos seguintes eventos:**
  1. **Início da execução do Main**
  2. **Fim da execução do Main**
  3. **Início da execução da função Fibonacci**
  4. **Fim da execução da função Fibonacci**
  5. **Chamada da função Fibonacci no Main**
  6. **Passagem do argumento para a função Fibonacci**
  7. **Retorno do valor calculado na função Fibonacci para o Main**
  8. **Qual foi o tempo total de computação do número de Fibonacci?**

APÊNDICE

1. Intel HEX-record Format (retirado de http://www.lucidtechnologies.info/)

INTRODUCTION

Intel's Hex-record format allows program or data files to be encoded in a printable (ASCII) format. This allows viewing of the object file with standard tools and easy file transfer from one computer to another, or between a host and target. An individual Hex-record is a single line in a file composed of many Hex-records.

HEX-RECORD CONTENT

Hex-Records are character strings made of several fields which specify the record type, record length, memory address, data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first ASCII character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 6 fields which comprise a Hex-record are defined as follows:

Field	Characters	Description
1	Start code	1
2	Byte count	2
3	Address	4
4	Type	2
5	Data	0-2n
6	Checksum	2

Each record may be terminated with a CR/LF/NULL. Accuracy of transmission is ensured by the byte count and checksum fields.

HEX-RECORD TYPES

There are three possible types of Hex-records.

00	A record containing data and the 2-byte address at which the data is to reside.
01	A termination record for a file of Hex-records. Only one termination record is allowed per file and it must be the last line of the file. There is no data field.
02	A segment base address record. This type of record is ignored by Lucid programmers.

HEX-RECORD EXAMPLE

Following is a typical Hex-record module consisting of four data records and a termination record.

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
```

```
:00000001FF
```

The first data record is explained as follows:

: Start code.

10 Hex 10 (decimal 16), indicating 16 data character pairs, 16 bytes of binary data, in this record.

0100 Four-character 2-byte address field: hex address 0100, indicates location where the following data is to be loaded.

00 Record type indicating a data record.

The next 16 character pairs are the ASCII bytes of the actual program data.

40 Checksum of the first Hex-record.

The termination record is explained as follows:

: Start code.

00 Byte count is zero, no data in termination record.

0000 Four-character 2-byte address field, zeros.

01 Record type 01 is termination.

FF Checksum of termination record.