이 문서는 MIPS design project (Phase 1) 의 최종 단계인 Step 10 (MIPS single-cycle implementation) 을 수행하는 과정에 대한 도움 정보를 제공합니다. 우선 Step 10 에서는 Step 1 에서 Step 9 까지 에서 설계한 VHDL component 들을 연결하여 더 큰 component 를 만들게 됩니다. 이 과정은 pipelined MIPS implementation 에서의 하나의 stage 를 설계하는 과정과 원리적으로 동일합니다. 따라서 여기서는 이미 설계된 컴포넌트를 이용하여 MIPS 의 IF stage 를 만드는 과정을 소개하겠습니다. 클래스 홈페이지에 IF stage 에 대한 프로젝트 파일이 제공되었기 때문에, 여러분이 아래의 설명을 따라서 IF stage 설계를 수행해 보면, MIPS single-cycle implementation 을 어떻게 하면 되는지에 대해 충분한 단서를 잡을 수 있을 것입니다.

How to perform the “IF” project

1. 우선 IF stage 설계에 필요한 미리 만들어진 component 들을 클래스 홈페이지에서 다운로드 합니다.

- add4.zip, imem.zip, mux32.zip, pc.zip

2. “여기서 압축풀기” 하여 다음의 디렉토리를 만듭니다.

- Add4, Imem, MUX32, PC

3. IF stage 에 대한 프로젝트파일 (“if.zip”) 을 클래스 홈페이지에서 다운로드하고 압축 풀기를 하여 “IF” 디렉토리를 만듭니다.

(여기서는 IF 디렉토리에 있는 VHDL 소스파일 만을 이용하여 IF stage 를 새로 설계할 것입니다.)

- IF stage 에 대한 VHDL 소스파일 (“eIF.vhd”)을 읽어 보십시오.

- 그리고 또 다른 VHDL 소스 파일인 “eJMUX.vhd” 를 읽어 보십시오. (IF project 에서는 홈페이지에서 제공하는 component 들 외에도 Jmux 라는 component 가 하나 더 필요합니다. Jmux 는 jump instruction 에 들어 있는 26-bit offset 을 처리하기 위한 multiplexer 입니다.)

4. IF stage 를 새로 설계하기 위하여 기존의 “IF” 디렉토리를, 예를 들어, “oldif” 라고 이름을 바꿉니다.

5. run Quartus software

6. select “file->new project wizard”

7. click “next”

8. 제공된 IF project file 과 동일하게 하려면 다음과 같이 입력

- type in the name of the working directory: C:\...\IF

- Type in project name: IF

- Type in the top-level design entity: eIF

9. Click “next” and 팝업창에서 click “yes”

(이제부터는 우리가 미리 설계해 둔 컴포넌트들을 이용하기 위해, 필요한 design file파일들을 include 하겠습니다: Step 1 through 9 에서는 미리 설계해 둔 컴포넌트를 이용한다는 개념이 없었기 때문에 이러한 과정이 없었습니다.)

10. Click “…” button, then move to “oldif” directory

- double click “eIF.vhd”

- click “Add” (이 파일을 이렇게 가져옴으로써 IF stage 에 대한 VHDL 소스파일을 type in 할 필요가 없어졌습니다.)

11. 다시 click “…” button, 그리고 Add4 디렉토리로 갑니다.

- double click “eAdd4.vhd”

- click “Add”

12. 다시 click “…” button, 그리고 Imem 디렉토리로 갑니다.

- double click “eImem.vhd”

- click “Add”

13. 다시 click “…” button, 그리고 MUX32 디렉토리로 갑니다.

- double click “eMUX32.vhd”

- click “Add”

14. 다시 click “…” button, 그리고 PC 디렉토리로 갑니다.

- double click “PC.vhd”

- click “Add”

(IF project 에서는 위의 component 들 외에도 Jmux 라는 component 가 하나 더 필요합니다. 그 설계 파일, 즉 eJMUX.vhd 는 “oldif” 디렉토리에서 찾아 불러올 수 있습니다; 아직 그렇게 하지 않았다면 나중에 불러오기가 끝난 후 “File->open” 하여 oldif 디렉토리로 가서 Jmux 에 대한 VHDL 소스 파일을 읽어 그 기능을 확인하십시오; Jmux 는 jump instruction 에 들어 있는 26-bit offset 을 처리하기 위한 multiplexer 입니다.)

15. 다시 click “…” button, 그리고 oldif 디렉토리로 갑니다.

- double click “eJMUX.vhd”

- click “Add”

16. click “next”, “next”, “next” and “finish”

17. Select “Processing-start compilation”

(소스 파일들이 컴파일 됩니다.)

(다음은 instruction memory 를 초기화 하기 위하여 mif 파일을 정의할 차례입니다.)

18. After the compilation, select “file->new”

19. Select “memory initialization file” and click OK

- 앞의 instruction memory project 에서 설명한 대로 mif file 을 만듬

- 단, 클래스 홈페이지에 보이는 test output 을 나오게 하려면 다음과 같이 mif 파일의 내용을 채움:

0 번지: 8C020002

1 번지: 8C030003

2 번지: 00432020

1. 번지: AC044004

그외의 모든 번지는 ‘00000000’ 으로 채움

* 그리고 파일이름을 ‘imem” 으로 저장

(자 이제 마지막으로 test input and output 을 설정하여 “eIF.vwf” 파일로 저장하는 단계입니다.)

20. select “file->new”

21. Select “Vector waveform file” and click OK

- 앞의 Step 1 through Step 9 의 예에서 습득한 대로, 클래스 홈페이지의 test input and output 에 표시된 대로 입력 신호 (pClock, pReset, pJSelect, pSelect, pNextPC, pJPC) 를 정의함; 출력 신호 (pIncpc, pInstruction)는 선언만 함; 그리고 모두 저장함

(이제 시뮬레이션을 수행하여 IF stage 가 제대로 되었는지 확인하는 단계입니다.)

22. Select “processing-> start simulation”

The simulation results appears in a window

You can examine whether your design makes sense

\*\* 여기서 IF stage 에 대한 VHDL source file 과 test input & output 을 비교하며, 이미 존재하는 component 들을 이용하여 더 복잡한 component 를 만든다는 것이 어떤 의미인지 음미하실 것.

* 참고사항: 위의 test output 에서 PC 값이 00000400 일 때 실제 메모리 address 로는 끝의 두 자리 즉 hex ‘00’ 만이 입력됨 – 그 이유는 우리가 정의한 mif file 에서 메오리에는 256 번지까지만 존재하기 때문임)

(we are done.)

여러분이 Step 10 에서 해야 할 일 중에서 가장 핵심적인 것이 MIPS single-cycle implementation 에 대한 VHDL 소스 파일 (예를 들어 “eSingle.vhd”) 을 만드는 것입니다. 여러분은 “eIF.vhd” 파일에 대한 세밀한 분석을 통해, “eSingle.vhd” 을 어떻게 작성할지에 대한 거의 모든 단서를 발견할 수 있습니다.

우선 “eIF.vhd” 의 구조를 살펴보면:

1. 맨 먼저 entity 가 선언되어 있고, 그 안에 모든 입출력 신호들이 선언되어 있습니다.

(이들 입출력 신호들은 나중에 vwf 파일에 들어가게 됩니다. 그리고 입력신호들에 대해서는 test input 값을 정의해 주게 됩니다.)

2. 그 다음의 architecture body 를 살펴보면:

- 우리가 Step 1 through 9 까지의 예에서는 predefined component 를 사용하지 않으면서 새로운 component 를 설계하였기 때문에, architecture body 에 주로 ‘동작적 특성’ 을 정의하였습니다

- 그러나 IF stage 처럼 여러 predefined component 들을 사용해서 더 복잡한 새로운 component 를 설계하는 경우에는, architecture body 에 predefined component 들을 어떻게 연결할 것인가 하는 ‘구조적 특성’ 에 대한 표현이 필요하게 됩니다.

- 이를 위해 ‘eIF.vhd” 에서는 컴포넌트들을 연결할 신호 (전선; wire) 를 signal 이란 이름으로 선언하였고, 우리가 사용할 predefined component 들을 선언하였으며, 마지막의 “begin … end” 사이에서 다음을 수행합니다:

- IF stage 의 외부 입출력 신호들과 내부연결용 signal (전선; wire) 들과의 연결

- 혹시 추가적인 ‘동작적 특성’ 이 있으면 이를 추가함

- predefined component 들의 입출력 신호와 내부연결용 signal (전선; wire) 들과의 연결

- 예를 들어, 다음과 같은 표현이 있는데

cadd4:eAdd4 port map (sCpc, sIpc);

이것의 의미를 살펴보면,

- cadd4 는 eAdd4 의 한 instance 이고

(여러분이 OOP 에서 익숙하겠지만, eAdd4 의 instance 를 여러 개 만들어 쓸 수도 있음)

- cadd4 의 pIn 은 sCpc 와, pOut 은 sIpc 와 연결됨

\*\* 이러한 구조는 MIPS single-cycle implementation 에서도 그대로 적용됩니다.

종합하여 정리하면, MIPS single-cycle implementation 프로젝트 (Step 10) 는 다음과 같은 순서로 접근하면 됩니다:

How to do MIPS single-cycle implementation project

1. 우선 MIPS single-cycle implementation 에 대한 VHDL source code 를 작성하는 것이 필요함

2. 이것을 하기 위해 predefined-component 들을 이용하여 전제적인 single-cycle implementation 의 block diagram 을 그려야 함

- 그러면 각 predefined component 가 몇 개가 필요한지,

- 각 instance 의 이름을 어떻게 할 것인지

- 이들을 연결할 신호들은 어떤 것이 필요한지, 또 실제 연결은 어떻게 되는지

- 내부 연결용 신호 (signal) 들의 이름은 어떻게 지을 것인지

- 외부 입출력 신호들의 이름은 어떻게 할 것인지

- 외부 입출력 신호와 내부 연결용 신호 (signal) 들의 연결은 어떻게 하는지

- “eSingle.vhd” 에 추가적으로 정의할 ‘동작적 특성’ 이 있는지 .. 등등을 결정할 수 있음

3. 그 다음에는 VHDL 소스 파일 (“eSingle.vhd”) 작성; 소스 파일의 구조는 앞의 “eIF.vhd” 의 그것과 같은 바, 다시 요약하면:

- 우선, include 부분 선언

- entity 선언하고, 이 entity 의 외부 입출력 신호 선언

- architecture body 에서

- predefined-component 들을 선언

- 내부 연결용 신호 (signal) 선언

- 마지막으로 “begin … end” 사이에서 다음을 수행

- 외부 입출력신호와 내부연결용 signal 들을 연결

- 내부연결용 signal 들을 이용하여 component 들을 연결

- 필요하면 ‘동작적 특성’

4. 클래스 홈페이지에서 저공된 Bubblesort 프로그램으로 “imem.mif” 파일 만듬

- Bubblesort 프로그램은 assembler 및 hexadecimal 형태로 주어졌음

- hexadecimal 형태를 이용하여 instruction memory 의 0 번지부터 채우면 됨

5. 클래스 홈페이지의 시뮬레이션 결과에 보면, bubble sort 프로그램이 data memory 의 0번지, 1번지, 2번지에 각각 저장된 세 개의 숫자 (hexadecimal 로 00000002, 00000005, 00000009) 를 sort 한다.

- 따라서 이 세 숫자를 “dmem.mif” 파일의 0, 1, 2 번지에 차례로 넣음. (sorting 이 끝나면 0, 1, 2 번지에는 각각 00000009, 00000005, 00000002 가 들어 있을 것임)

6. 클래스 홈페이지의 시뮬레이션 결과를 참조하여, Test input and output (vwf 파일)을 정의한 후, Quartus simulation 을 수행하여, sorting 이 제대로 되었는지 확인

- sorting 의 결과는 bubble sort 프로그램의 마지막에 있는 세 개의 “lw” instruction 이 읽는 데이터 값을 확인하면 알 수 있음

- 주어진 세 개의 숫자의 값과 순서를 바꾸어도 sorting 이 되는지 확인하세요.

이로써 여러분은 “MIPS single-cycle implementation” 프로젝트 수행을 완료 하였습니다. 한 가지 남은 것은. 클래스 홈페이지에는 bubble sort 프로그램이 (여러분의 숙제 부담을 줄여 주기 위해) hexadecimal form 으로도 변환되어 주어졌는데, 원래는 이 프로그램을 assembler 형태로만 주었었습니다. 만일 Assembler 형태로만 주어진다면, 주어진 assembler 프로그램을 binary 로 변환하여 mif 파일로 만들어야 합니다. Assembler 프로그램을 binary 로 변환하는 것은 SPIM 이라고 불리는 MIPS CPU instruction-level simulator 를 이용하여 수행할 수 있습니다. (CPU instruction-level simulator 가 무엇인지 호기심을 느끼는 사람을 위해 변환 과정을 아래에서 설명합니다 – 따라서 해 보신 후, CPU simulator를 탐험해 보시기 바랍니다. 혹시 나중에 professional programmer 로서 필요에 의해 CPU simulator 를 만들게 될 수도 있습니다.)

How to transform assembler program to binary program

1. Go to <http://spimsimulator.sourceforge.net/>

2. Click “Download SPIM”

3. Click and download latest version for Windows (예: QtSpim\_9.1.9\_Windows.zip) or Linux or Mac

4. 압축을 풀고 install 함

5. 바탕화면에 생긴 icon 을 클릭하여 QtSPIM 을 run

6. Select “File->Load file” (화면 왼쪽 최상단)

- 클래스홈페이지로부터 다운 받은 bubblesort\_without\_j 파일 (assembler form) 선택 (txt 파일)

- Text 창의 User Text Segment 를 자세히 보면 bubblesort 프로그램이 어느 위치에 있는지를 확인할 수 있고, hexadecimal 변환 결과를 볼 수 있음 – 이 hexadecimal 결과를 “imem.mif” 파일의 0 번지부터 채우면 됨

\*\* 주어진 Bubble sort 프로그램은 데이터 메모리의 0, 1, 2 번지에 있는 세 숫자를 bubble sort 를 이용하여 내림차 순으로 sorting 한다 – 실행한 후 데이터 파일의 0, 1, 2 번지를 보면 sorting 이 제대로 되었는지 확인 가능하다. (마지막 세 개의 “lw” 인스트럭션에서 정렬 결과 확인 가능함.)

\*\* 참고사항: SPIM 은 branch 인스트럭션에서 offset 을 결정할 때 4만큼 크게 잡음 (이유는 MIPS CPU 의 delayed branch 와 관계 있음) – 이 때문에 Phase 1 의 Step 4 에서 ‘-4’ 를 추가하여 이를 보상하였음.

(end-of-SPIM)

추가적인 참고자료: “eSingle,.vhd” 의 개략적인 구조의 한 예

library ieee; -- includes

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_signed.all;

use ieee.numeric\_std.all;

entity single is -- top entry ports

port (

pReset : in std\_logic; - test input

pClock : in std\_logic;

pPC : out std\_logic\_vector(31 downto 0); - test output

.

.

.

);

end single;

architecture behavior of single is -- behavior for top entry

------------------------------------- define components

component eAdd4

port (

pIn : in std\_logic\_vector(31 downto 0);

pOut : out std\_logic\_vector(31 downto 0)

);

end component;

.

.

.

-------------------------------- define signals

signal sINS : std\_logic\_vector(31 downto 0);

signal sCPC : std\_logic\_vector(31 downto 0);

.

.

-------------------------------- begin behavior

begin

-------------------------------- output port

pPC <=sCPC; -- signals you want to observe

pNextPC <=sNPC;

.

.

------------------------------- component connect

PC : ePC

port map(

pIn=>sNPC,

pOut=>sCPC,

pReset=>pReset,

pClock=>not pClock

);

.

.

.

end behavior;

(This is the complete end of MIPS single-cycle implementation project!)