이 문서는 MIPS design project (Phase 1) 의 steps 1에서 9 까지를 수행하는 방법을 알려 드립니다. 여기에 기록된 절차를 따라 수행하면 VHDL 및 Quartus software 를 사용해 보지 않은 학생이 다른 참고문헌을 읽지 않고서도 프로젝트를 진행할 수 있을 것입니다. 물론 관심이 있고 여건이 허용되는 학생들은 가능한 한 많은 참고문헌을 찾아보고 또 유사한 프로젝트를 수행할 수도 있을 것입니다.

How to test run the Add4 project (Step 1):

1. download “add4.zip” from the class homepage
2. 여기에 압축풀기 하면 “Add4” directory 가 만들어집니다
3. run Quartus software by clicking it

- “getting started …” 팝업창: 이 창을 제거합니다

- “upgrade ….” 팝업창: 이창이 없어질 때까지 잠시 기다립니다.

4. select “FILE->open project” (상변 메뉴바 참조)

5. “Add4” directory로 이동하여 Quartus project file (“Add4.qpf”)을 double click

- 팝업창에서 click “yes”

6. click “eAdd4” in the “Entity” pane on the top left of the screen

- the VHDL source file will appear

(it is the “eAdd4.vhd” file in the “Add4” directory)

- VHDL 소스 파일을 읽어보고, 홈페이지의 adder 그림을 비교해 보십시오.

(이 회로 모듈은 32-bit 데이터 값에 4를 더하는 기능을 수행합니다)

- VHDL 소스 파일의 첫 부분은 C 프로그램에서의 “include” 부분과 유사

- Entity declaration (둘째 부분): eAdd4라는 회로 컴포넌트의 외부 연결 (I/O 신호)

- Architecture body (셋째 부분): eAdd4라는 회로 컴포넌트의 내부적 동작 표시

- “File->print” (상변 메뉴바) 를 이용하여 소스파일 프린트 할 수 있음

- “File->save as” (상변 메뉴바) 하여 소스 파일을 text 파일로 저장 가능

(그러면 디렉토리에 “eAdd4.txt” 파일이 만들어짐)

- if you want, you can remove the source file window

7. select “Processing->start compilation” (상변 메뉴바 참조; 상변에 단축키도 있음)

- compilation will be finished

- click “확인” (warnings are okay)

- if you want, you can remove the compilation report window

9. select “Processing->start simulation” (상변 메뉴바 참조; 상변에 단축키도 있음)

(실제 시뮬레이션을 위해서는 test input and ouput 신호를 정의해 주어야 하나, “Adder4” directory 에는 이미 설계된 test input and output 이 “eAdder.vwf” 파일 안에 정의되어 있으며 일단 이것을 이용함; test input and output 을 정의하는 방법은 아래에서 별도로 다루겠음.)

- click “OK”

- the simulation report window appear

- click “확인”

- text input and output 을 관찰하며, 이 adder 가 의도된 대로 동작하는지 확인하실 것

- input: pIn

- output: pOut (pOut <= Pin + 4)

다음에는 위에서 설명한 adder component 를 홈페이지에서 제공된 파일을 이용하지 않고 처음부터 새로 만드는 과정을 설명하겠음.

How to perform the “Add4” project from the scratch

1. adder component 를 처음부터 새로 만들기 위해 이미 존재하는 “Adder4” directory 의 이름을 예를 들어 “a4” 라고 바꿈

2. run Quartus software

- “getting started …” 팝업창: 이 창을 제거합니다

- “upgrade ….” 팝업창: 이창이 없어질 때까지 잠시 기다립니다.

3. select “file->new project wizard” (상변 메뉴바)

4. click “next”

5. type in the name of the working directory (for example, C:\...\Add4)

- Type in project name: Add4

- Type in the top-level design entity: eAdd4

- click “next”

- 팝업창(“do you want to create directory?”)이 나타나면, click “yes”

6. Click “next” “next” “next” and “finish”

7. Select “File->new” (상변 메뉴바)

(Add4 component 에 대한 VHDL 소스파일 작성을 시작하는 것임)

8. 팝업창에서 select “VHDL file” and click “OK”

9. 팝업창에서 Add4 에 대한 VHDL 소스 파일을 type in

- 일일이 type 하기가 번거로우면, 앞의 과정에서 만들어 놓은 eAdd4.txt 파일을 copy-and-paste 해도 좋음

- 혹시 Add4 에 대한 VHDL 소스 파일을 text 파일로 저장해 놓지 않았다면 다음의 과정을 밟아도 좋음

- 일단 “File-Close” 합니다 (상변 메뉴바)

- ”File-open” 하고 홈페이지에서 다운로드한 파일들이 있는 “a4” 디렉토리로 이동하여 “eAdd4.vhd” 를 double click

- copy-and-paste 를 위해 소스파일 내용을 “copy” 함

- “File-close” (상변 메뉴바) 하여 열려진 파일을 닫음

- “File->new”

- 팝업창에서 select “VHDL file” and click “OK”

- 팝업창에서 “paste” 하여 소스파일 만듬

10. Select “file->close” (상변 메뉴바)

- 팝업창에서 click “yes”

11. 현재의 “a4” 디렉토리에서 “Add4” 디렉토리로 이동한 후, “eAdd4.vhd” 라는 이름으로 저장 (파일 형식은 vhd or vhdl)

(이로써 Add4 component 에 대한 VHDL 소스 파일이 만들어졌습니다.)

12. Select “Processing-start compilation” (상변 메뉴바 참조; 상변에 단축키도 있음)

- compilation will be finished

- click “확인” (warnings are okay)

- if you want, you can remove the compilation report window

13. Select “file->new”

(이제부터는 test input and output 을 정의하는 “eAdd4.vwf” 파일을 만든다)

14. Select “Vector waveform file” and click OK

- 팝업창에서 click “OK”

15. Waveform window 왼쪽의 “Name” field 아래에 있는 네모 형태의 dotted area 를 double click 함

- 팝업창에서 Click “Node Finder”

- Click “List”

- Select “pIn” and click “>” in the middle

(pIn 은 test input 으로서 우리가 값을 정해줄 것임)

(pIn 은 32-bit 신호로서 한꺼번에 선택함)

- Select “pOut” and click “>” in the middle

(pOut 은 test output 이며 그 값이 제대로 나오는지 관찰할 것임)

(pOut 은 32-bit 신호로서 한꺼번에 선택함)

- Click “OK”

(우리가 test input 신호를 정의해 주거나 또는 시뮬레이션 결과 확인을 위해 관찰하여야 하는 output 신호들을 모두 선택하여야 함; 홈페이지에 있는 각 회로 component에 대한 test input and output 신호들은 홈페이지의 시뮬레이션 결과 그림에서 확인할 수 있으므로 이를 참고하면 됨.)

- Click “OK” again to remove the pop-up window

16. Waveform 윈도우의 왼쪽 끝부분에서 “pIn” 왼쪽의 “0” 부분을 Double click

(그러면 “pIn” 신호가 선택되고 이 선택된 신호는 하늘색으로 나타남)

- 팝업창에서 “ASCII” 를 “hexadecimal” 로 바꾸고 click “확인”

(이렇게 하면 “pIn” 신호의 값은 hexadecimal 로 표시됨)

- “pIn” 이 선택된 상태에서 윈도우 왼쪽의 “C” 라는 symbol 을 click 하고, 팝업창에서 click “확인” 하면, 시간에 따라 점점 증가하도록 “pIn” 이 정의됨

(필요에 따라 “C” 팝업창에서 여러가지 옵션을 선택할 수도 있음)

(이 “C” 는 편리한 단축키이며, 이 경우에는 이용할 수 있으나, 일반적인 test input 을 정의하는데 사용하기는 어려움; test input 을 정의하는 일반적인 방법은 아래에 따로 설명하겠음)

17. Waveform 윈도우의 왼쪽 끝부분에서 “pOut” 왼쪽의 “33” 부분을 Double click

- 팝업창에서 “ASCII” 를 “hexadecimal” 로 바꾸고 click “확인”

(“pOut” 신호는 출력신호로서 시뮬레이션 결과를 확인하는데 사용됨; “pOut” 의 값은 시뮬레이터가 결정하므로 우리가 정의해 줄 필요가 없음)

18. Select “File->close” (상변 메뉴바)

(test input 파형을 정의했고, test output 신호를 선택하였으므로 이것을 파일로 저장함)

- 팝업창에서 Click “yes”

- “eAdd4” 라는 이름을 사용 (파일형식은 “vwf”)

- click “저장”

19. Select “processing-> start simulation” (상변 메뉴바)

- 팝업창에서 click “OK”

- the simulation report window appear

- 팝업창에서 click “확인”

- text input and output 을 관찰하며, 이 adder 가 의도된 대로 동작하는지 확인하실 것

- input: pIn

- output: pOut (pOut <= Pin + 4)

\*\* 여기서 waveform 을 관찰하기 위하여 time scale 을 조절하려면, 우선 왼쪽의 돋보기 심볼 (또는 “±” 심볼)을 마우스로 click 한 후, waveform 이 표지된 지역에 가서 마우스를 좌클릭 또는 우클릭 함. 그러면 이에 따라 time scale 이 바뀌게 됨. 적절한 time scale 을 선택한 후, 왼쪽의 맨 위에 있는 기울어진 화실표를 클릭하면 time scale 이 선택된 대로 고정됨

(we are done.)

\*\* 위의 16 항에서는 입력 신호 값을 정의할 때 왼쪽에 있는 “C” 를 눌러 순차적으로 증가하는 입력을 편하게 정의하였음; 여기서는 Add4 component에 대한 test input 을 클래스 홈페이지의 시뮬레이션 결과에 나와 있는 대로 정의하는 과정을 설명함으로써, test input 을 정의하는 일반적인 방법을 알아 본다. 이를 위해 위의 16 항에서 “C” 를 누르는 대신 다름과 같은 과정을 수행한다.

* Time scale 을 조절하여 0 부터 1 microsecond 까지 보이도록 함
* “pIn” 신호의 0 부터 100 ns 까지 선택 (마우스 클릭하고 drag 하면 됨)

(선택된 지역은 컬러로 표시됨)

* Waveform 윈도우의 왼쪽에 있는 “?” symbol 을 누른 후 뜨는 원도우에서 원하는 값을 (hexadecimal “00000000”) 입력함
* “pIn” 신호의 100 ns 부터 200 ns 까지 선택 (마우스 클릭하고 drag 하면 됨)

(선택된 지역은 컬러로 표시됨)

* Waveform 윈도우의 왼쪽에 있는 “?” symbol 을 누른 후 뜨는 원도우에서 원하는 값을 (hexadecimal “00000001”) 입력함
* 이같은 과정을 반복하여 “pIn” 의 값을 클래스 홈페이지에 나와 있는 대로 입력
* “pIn” 의 정의가 끝나면 (이 경우는 입력 신호가 하나 뿐인 경우임; Add4 가 아닌 다른 component를 설계할 때는 통상 여러 개의 입력 신호가 있으며, 이러한 입력 신호 각각을 다 정의해 주어야 함) 앞에서 설명한 요령대로 “eAdd4.vwf” 파일로 저장함
* 이제 시뮬레이션을 수행하여, 클래스홈페이지에 있는 대로 출력이 나오는지 확인하실 것. 또한 이 결과를 VHDL source file 과 비교하며, 내가 어떠한 논리회로를 정의하고 어떤 입력을 주어 어떤 출력을 얻었는지 음미하실 것

(여기까지 오신 것을 축하합니다)

위와 같은 요령으로 클래스 홈페이지에서 제공하는 다음과 같은 회로 component의 설계를 검토해 보십시오 (Steps 1 through 9). 구체적으로, 각 component에 대해 VHDL 소스 파일을 읽어보고, 홈페이지에 제공된 시뮬레이션 결과 (test input and output) 를 분석해 보십시오. 그 다음에 Quartus 소프트웨어를 이용하여 홈페이지의 그것과는 다른 새로운 test input 을 정의하여 시뮬레이션을 수행해 보십시오.

Step 1: adder (“PC + 4” 용도로 사용)

Step 2: sign extender

Step 3: two-to-one multiplexer (5-bit and 32-bit)

Step 4: 2-bit shift and add

Step 5: program counter

Step 6: data memory and instruction memory

Step 7: register file

Step 8: control unit

Step 9: ALU

단, 위의 과정 중에서 Step 4, Step 6, Step 8, Step 9 에 대해서는 다음과 같은 추가적인 추가적인 설명이 필요하니, 읽어 보십시오.

Step 4: 2-bit shift and add

1. 이 component는 branch instruction 에서 새로운 PC 값을 구하는데 사용됩니다.

2. Textbook 에서나 수업에서는 “-4” 하는 부분이 없는데, 홈페이지의 그림이나 VHDL 소스파일에 보면 “-4” 라는 기능이 추가되어 있습니다.

3. 이것은 MIPS 의 delayed branch 와 관련된 것입니다 – 아마도 delayed branch 는 수업에서 다루지는 않을 것 같습니다. 따라서 여기에 설명된 내용을 보고 그런 것이 있나 보다 라고 생각하시면 됩니다.

4. 나중에 Step 10 의 single-cycle implementation 에서, assembly 프로그램을 최종적으로 binary executable file 로 만들 때, SPIM 이라는 MIPS instruction-level simulator 를 사용하는데, 이 시뮬레이터는 branch instruction 의 offset 부분을 만들 때 실제보다 4 만큼 크게 만듭니다. (이것이 MIPS processor 의 delayed branch 와 관련된 것입니다.)

5. 따라서 이를 보상하기 위해 우리는 이 component 에서 “-4” 하는 기능을 추가하였습니다.

\*\* VHDL 소스 파일에 보면 “signal” 이라는 것이 있는데, 이는 C 프로그램에서의 external variable 과 유사함; chip 설계 관점에서 볼 때 signal 은 component 들을 연결하는 전선 (wire) 라고 보면 됨

\*\* 홈페이지에서 제공된 파일들에서는 신호 또는 컴포넌트 이름을 다음과 같이 지음

- 외부신호: p 로 시작 (p 는 port 를 의미)

- 내부신호 (wire): s 로 시작 (s 는 signal 을 의미)

- 회로 컴포넌트: e 로 시작 (e 는 entity 를 의미)

Step 4: instruction and data memory

1. 여기서는 VHDL 소스 파일, “vwf” 파일 (test input and output) 외에도 memory initialization file (“mif” file) 을 만들어 instruction memory 또는 data memory 를 원하는 값으로 초기화 해야 한다.

2. 우선 instruction memory 관련하여 VHDL 소스파일 (“eImem.vhd”)을 보면,

- memory 는 lpm library 를 이용하여 정의함

(여러분은 새로 메모리 component 를 만들지는 않고 제공된 메모리 component 를 사용만 하면 되기 때문에, 사용에 관련된 몇 가지 사항을 알면 됨)

- lpm library 를 이용하여 instruction memory 를 만들 때에는 data\_memory: lpm\_rom 으로 선언하고 generic\_map 및 port\_map 부분을 정의한다

- generic map 부분

- lpm\_widthad 는 8 로 선언: 이는 메모리 안에 28 = 256 개의 저장장소가 있음을 선언한 것이다 (따라서 메모리 어드레스는 8-bit)

- lpm\_file 은 “imem.mif” 로 선언: 따라서 나중에 “mif” 파일을 만들 때 file name 은 “imem” 으로 하여야 함 (아래 참조)

- lpm\_width 는 32 로 선언: 즉 메모리 내의 한 저장장소에는 32-bit data 가 들어 있음을 의미 (즉, 여기서 우리는 256\*32 bit memory 를 선언하였음)

- port map 부분

- address 는 외부신호 (pReadaddr) 의 8 bit 만을 사용 (가장 낮은 2-bit 은 사용하지 않았으므로, 이 어드레스는 word address 라는 의미가 됨; 우리가 선언한 메모리에서 한 저장장소에는 4 bytes 가 들어있으므로 word address 를 사용함)

3. Quartus 소프트웨어에서 “imem.mif” 파일을 정의하는 과정은 다음과 같음

- “File->new” (상변 메뉴바)

- select “memory initialization file” and click “OK”

- VHDL 소스파일에 정의한 것과 일치되도록 다음과 같이 입력

- Number of words: 256

- Word size: 32

- click “OK”

- Mif 윈도우의 맨 왼쪽에 있는 address column 에서, 마우스 right click 하여 address radix 와 memory radix 를 hexadecimal 로 설정

- 클래스 홈페이지에 있는 값으로 instruction memory 를 초기화한다고 가정하면

- 0 번지를 double click 하여 “8C020000” 입력

- 1 번지를 double click 하여 “8C030004” 입력

- 2 번지를 double click 하여 “00432020” 입력

- 3 번지를 …. 등등을 입력

(필요하면, 마우스 left click 후 drag 하여 많은 셀을 선택한 후, 마우스 right click 하여 원하는 값으로 한번에 정의할 수도 있음)

- “File->close” (상변 메뉴바)

- click “yes”

- 파일 이름은 “imem” 파일 형식은 “mif” 로 저장

(여기서 파일 이름을 VHDL 소스 파일의 “lpm\_file”에 선언한 이름을 그대로 사용하여야 함)

(이로써 instruction memory 를 초기화하는 과정을 마침.)  
(data memory 초기화도 같은 요령으로 수행하면 됨.)

Step 8: control unit

1. Textbook 에서는 ALUop 라는 신호를 2-bit 으로 잡아 설계했으나, 여기서는 pALUop 가 1-bit 으로 잡혀 있음. (따라서 일관성이 없음 – 여러분의 양해 바람.) pALUop 는 R-type instruction 이나 “bne” instruction 일 때 1 의 값을 가짐.

Step 9: ALU

1. 이 부분은 외부신호의 이름이 적절하게 지어지지 않았고, 따라서 여러분이 코드를 읽을 때 불편함을 느낄 것으로 예상됨. 우선 eALU entity 에 정의된 외부신호를 설명하면

– pOperator: R-type instruction 내부의 function field 를 의미

– pOperation: 위의 Step 8 에서 설명한 pALUop 와 같은 신호임

(R-type 또는 “bne” instruction 일 때 1의 값 가짐)

– pBranch: 위의 Step 8 에서의 pBranch 와 같은 신호임

* + 1. (“beq” or “bne” instruction 일 때 1의 값을 가짐)

– pBranch\_Out: conditional branch instruction 에서 조건이 맞아 branch 가 일어날 떄 1의 값을 가짐 (즉 “beq” 에서는 두 레지스터의 값이 같을 때 1의 값을 갖고, “bne” 에서는 두 레지스터의 값이 다를 때 1의 값을 가짐)

- pIn1, pIn2: 두 개의 32-bit ALU input operands

- pOut: 32-bit ALU output

2. VHDL 소스 코드를 100% 정확히 이해하지 않더라도 대략 이렇게 동작하는구나 하는 정도로 이해할 수 있으면 충분함

- architecture body 에는 두 개의 process 가 들어 있음

- 첫번째 process 는 “slt” instruction 을 처리하기 위한 것임

- 두번째 process 에서는 우리가 구현하지 않은 instruction 이 나오면 ALU 는 더하기 동작을 하도록 설계되었음

- 두번째 process 내의 case 문에서, 마지막 경우 (“when others”) 에 대한 코멘트가 “- - lw, sw” 라고 붙어 있는데, 이 코멘트는 틀림 (왜 틀렸는지 생각해 보셔도 좋음)

- sBranch 신호는 두 레지스터의 값이 같을 때 1의 값을 가짐

3. VHDL 에서 “process” 라는 개념을 보다 자세히 이해하고 싶은 학생은 “VHDL process 구문” 이라는 키워드로 Google 검색을 하면 자료를 찾으실 수 있습니다.

(Step 1 through 9 에 대한 도움말은 여기까지입니다.)