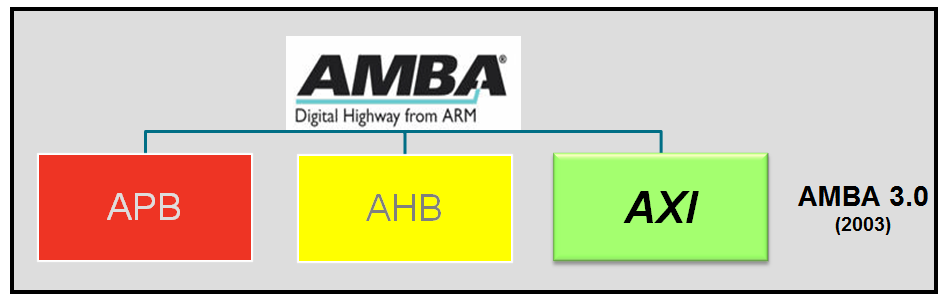
# AXI 이해하기

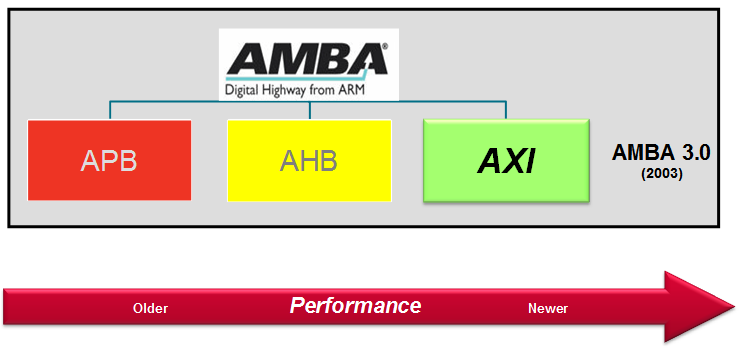
## AMBA ( Advanced Microcontroller Bus Architecture)



AMBA는 ARM프로세서가 사용하는 인터페이스 입니다.

ARM 프로세서와 주변 장치는 위 3가지 인터페이스 중 하나를 꼭 사용하고 있습니다.

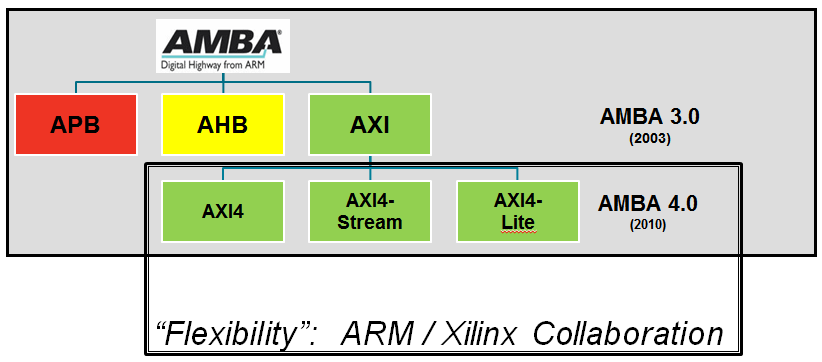
인터페이스를 이렇게 3가지로 나눈 이유는 마스터와 슬레이브간에 데이터 전송 속도에 차이가 있기 때문 입니다.



APB 는 상대적으로 전송 속도가 AHB나 AXI보다 낮습니다.

가장 최근에 나온 인터페이스가 AXI 입니다. 자일링스는 여러 인터페이스 중에 이 AXI를 지원하고 있고 APB는 AHB에 대해서는 Bridge 를 지원하고 있습니다.

이 인터페이스 중 자일링스 FPGA에서 관심을 가지고 있는 것이 바로 AXI 입니다.



AXI ( Advanced Extensible Interface)는 AMBA 4.0 규격에서 모두 3개의 서브 인터커넥터로 다시 확장 됩니다.

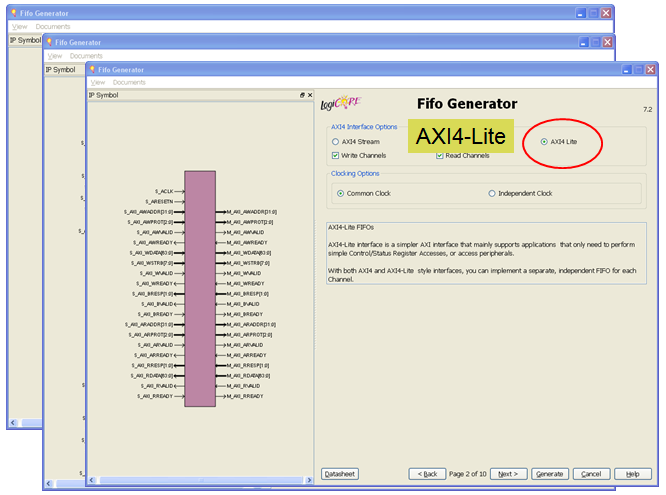
이 3개의 서브 인터커넥터가 바로 FPGA 내부에서 사용하는 인터페이스가 됩니다.

즉 자일링스에서 제공하는 일반적인 IP는 적어도 이 세가지 서브 인터페이스 중 하나를 지원한다는 것 입니다.

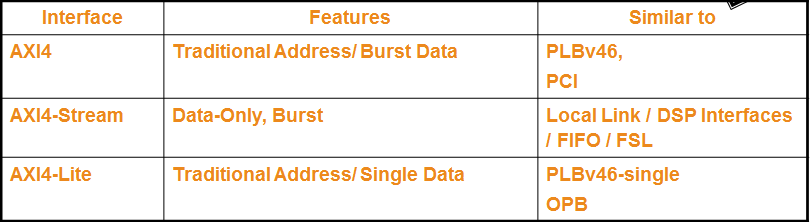
예를 들어 보겠습니다.

자일링스는 FPGA 내부에 여러 IP들을 생성할 때 Coregen이라는 프로그램을 사용 합니다.

다음 그림은 Coregen에서 FIFO를 만드는 모습을 보여 줍니다. 그림에서 보는 것 같이 AXI4Lite, AXI4Full, AXI4Stream 모두 지원하는 것을 알 수 있습니다.



FPGA에서 이렇게 AXI4Lite, AXI4Full, AXI4Stream 이렇게 3개의 인터커넥터를 나눈 이유는 다음과 같습니다.



### AXI4Stream

FPGA 를 설계하면 데이터를 주고 받는 마스터와 슬레이브를 많이 만들게 됩니다. 이때 마스터와 슬레이브간에 데이터를 주고 받는 방식을 생각해 보면 다음과 같은 특징이 있습니다.

가령 영상 데이터는 처리하는 로직을 설계할 경우, 해당 로직에 입력되는 영상 데이터는 그 흐름이 끊어지지 않고 계속 이어지게 됩니다.

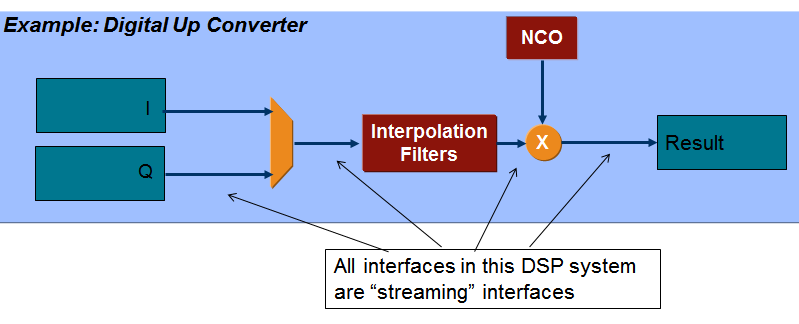
끊어지면 방송사고죠~~~

일반적으로 이런 데이터들은 스트리밍 방식이라고 부릅니다.

이런 인터페이스에는 어드레스는 의미가 없고 데이터들이 어떻게 끊어지지 않게 주고 받을 수 있는지에만 관심이 있습니다.

이때 사용하는 인터커넥터가 바로 AXI4Stream 입니다. 주로 FIFO를 사용해서 인터페이스 합니다.

다음 그림은 AXI4Stream을 사용한데 대표적인 어플리케이션 중 하나 입니다.

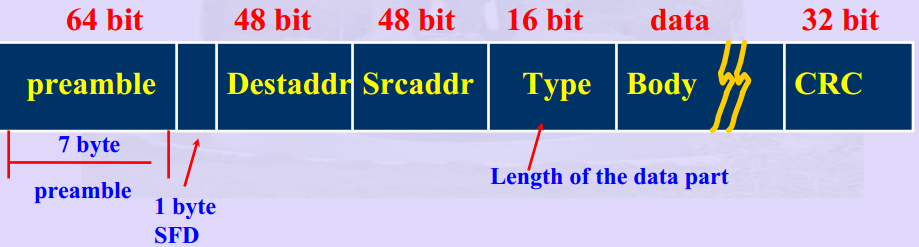


### AXI4Full

마스터와 슬레이브간에 데이터를 교환할 때 데이터 크기가 수십바이트에서 수키로 바이트 단위로 전송 됩니다.

가장 대표적인 프로토콜로는 이더넷 프레임이 있습니다.

다음 그림은 이더넷 프레임을 보여주고 있는데 여기서 관심을 가지려고 하는 것은 프레임의 구조가 아니고 전체 크기 입니다.



즉 이더넷 프레임은 데이터 전송이 시작되면 최소 64바이트에서 최대 1.5K정도까지 한꺼번에 쭉 전송 됩니다. 중간에 끊어지지 않습니다. 하지만 AXI4Stream과 달리 끝이 있습니다.

이런 종류의 프로토콜은 굉장히 많이 있습니다. 또다른 대표 선수로는 PCIe 가 있습니다.

이때 사용하는 인터페이스가 AXI4Full 입니다.

### AXI4Lite

마스터와 슬레이브간 데이터 전송량이 간헐적으로 이루어지고 전송되는 데이터 크기도 제한적일 때 사용하는 인터페이스 입니다.

간헐적이고 제한적이라는 말이 매우 주관적인 표현이지만 이런 인터페이스를 사용하는 경우는 매우 많습니다.

예를 들면 프로세서와 인터럽트 콘트롤러간에 데이터를 주고 받는 양은 얼마되지도 않고 속도가 빠를 필요도 없습니다.

주로 프로세서가 부팅되고 나서 주변 장치들을 초기활 때는 주변장치의 레지스터를 억세스 하기 때문에 AXI4Full이나 AXI4Stream 인터페이스를 사용할 이유가 없습니다.

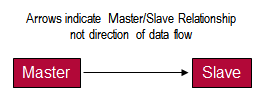
따라서 AXI4Lite는 프로세서와 주변장치를 연결할 때 많이 사용하는 인터페이스 입니다.

## AXI 에 대한 이해

### AXI는

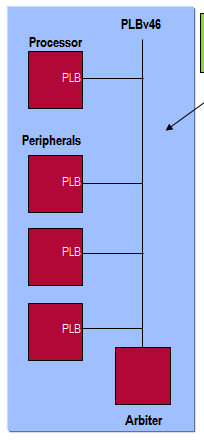
마스터 슬레이브간의 인터페이스이고 프로토콜 입니다.

흔히 Point to Point 인터페이스라고 하며 (프랑스 말로 뽕뚜뽕 이라고 합니다.)



하지만 버스는 아닙니다.

Point to Point 와 버스의 차이점은 다음과 같습니다.



버스는 마스터와 슬레이브들 연결할 때 어드레스, 데이터 및 각종 콘드롤 신호를 공유 합니다.

연결 신호선들을 공유하기 때문에 FPGA 입장에서는 라이팅 채널이라는 FPGA 자원을 적게 사용하는 잇점이 있지만 마스터와 슬레이브간 고속 데이터 전송에는 효율을 끌어 올리기가 어렵습니다.

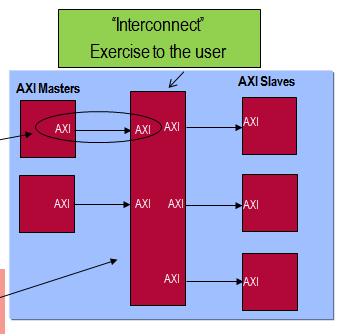
반면 Point to Point는 어느 한 순가 마스터와 슬레이브가 서로 1:1로 연결 됩니다.

가장 많은 질문 중에 하나가 바로 이 문제 때문에 생깁니다.

여러 개의 주변장치를 연결 할 때 어떻게 하느냐?

그래서 AXI는 여러 개의 마스터와 여러 슬레이브를 서로 연결 할 때 중간에 스위치를 둡니다.

이 스위치 이름이 바로 인터커넥터 입니다.

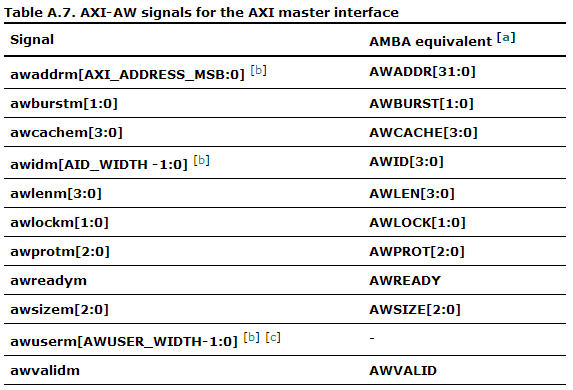


### Channel

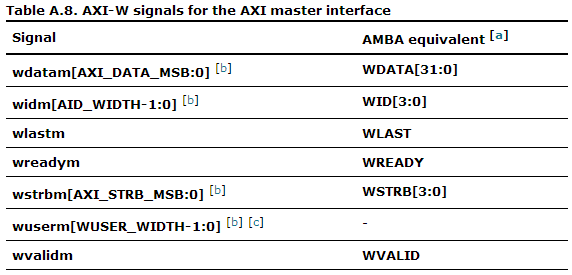
AXI 인터페이스를 구성하는 요소 중에 채널이라는 개념이 있습니다. AXI는 WriteAddress, ReadAddress, WriteData, ReadData, WriteResponse 5개의 채널로 구성 됩니다.

각 채널에 대한 신호를 살펴보면 다음과 같습니다.

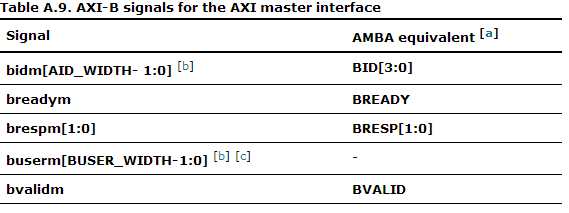
#### Write Address Channel



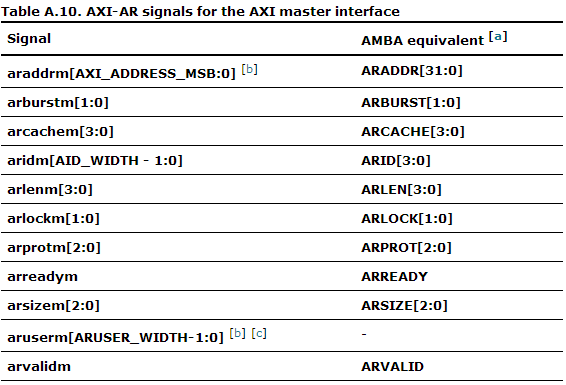
#### Write Data Channel



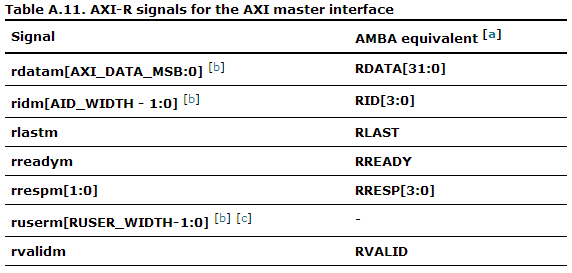
#### Write Response Channel



#### Read Address Channel



#### Read Data Channel



여기서 중요한 것은 각 채널의 신호 이름을 모두 이해하는 것이 아닙니다.

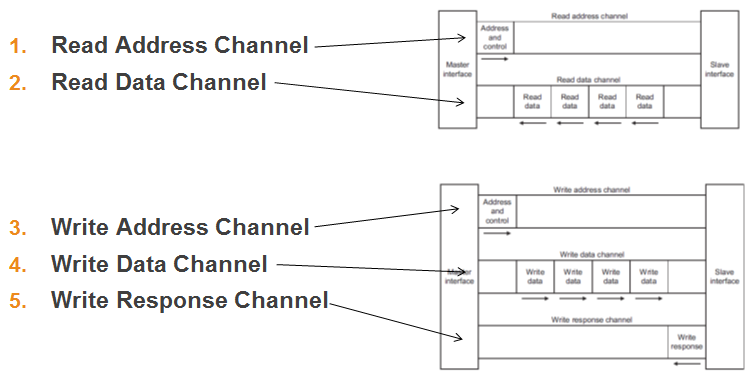
각 채널의 특징을 이해하는 것이 중요 합니다.

먼저 어드레스 채널을 보면 비슷하게 나오는 신호들이 있습니다. 예를 들면 Address, Burst, Cache, ID, LEN, LOCK, PROT, Ready, Size 가 있고 이 신호 이름 앞에 WA나 RA가 붙게 됩니다.

데이터 쪽도 만찬가지 입니다. Data, ID, LAST, Ready, STRB, Valid 신호들 앞에 WD, RD 가 붙어 있습니다.

Write Response을 보면 ID, READY, RESP, VALID 신호가 있고 앞에서 언급했던 신호 이름과 거의 유사 합니다.

이와 같이 AXI는 이렇게 5개의 채널을 통해 마스터와 슬레이브를 연결 합니다.



### AXI 인터페이스, AXI4Full

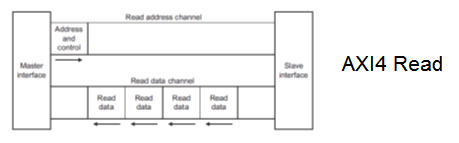
AXI4Full은 “Memory Mapped” 또는 Full AXI 라고 부르기도 합니다.

특징을 보면 다음과 같습니다.

처음 데이터 전송할 때 처음에만 어드레스가 나옵니다. 그 후에는 이 어드레스를 기준으로 차례대로 데이터가 쌓인다고 예상하고 데이터만 쭉 보내게 됩니다. 이런 방식을 버스트 방식이라고 하는데 최대 256번까지 보낼 수 있고 현재 자일링스는 데이터 폭 (Width)는 256bits이니까 32바이트가 됩니다.

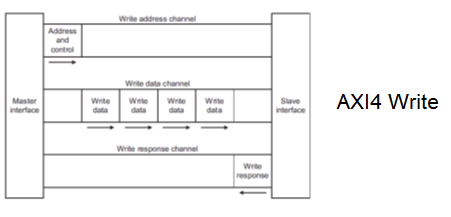
그러니까 한번 버스트 전송을 시작하면 32\*256 = 4K 바이트까지 보낼 수 있습니다. 그 이상되는 크기의 데이터는 버스트 방식으로 여러 번 전송해야 합니다.

다음 그림은 AXI4Full Read 블록도 입니다.



마스터가 슬레이브에게 데이터를 읽기 위한 어드레스와 콘트롤 신호를 보내면 슬레이브는 요청한 데이터를 버스트 방식으로 주루룩 전달 합니다

다음 그림은 AXI4Full Write 블록도 입니다.

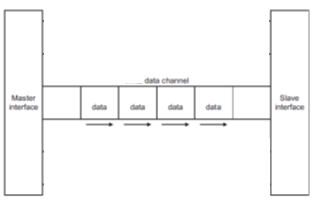


마스터가 슬레이브에게 데이터를 쓰려고 합니다. 이때 마스터와 슬레이브간에는 Write Response 채널을 통해 서로 핸드쉐이크 합니다. 핸드쉐이크는 서로 데이터 주고 받을 때 준비상태를 확인하는 프로토콜 입니다.

### AXI 인터페이스, AXI4Stream

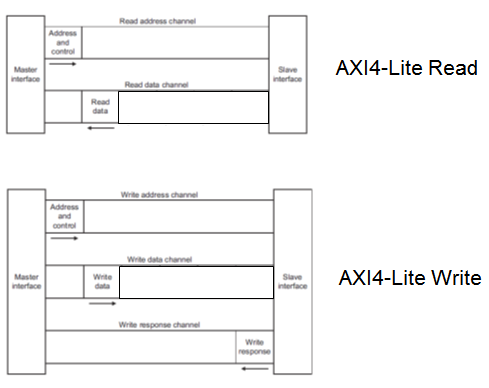
다음 그림은 AXI4Stream 인터페이스에 대한 블록도롤 보여주고 있습니다. 굉장히 간단하죠. 왜냐하면 AXI4Stream는 데이터의 방향이 단방향성인데다가 어드레스 채널이 존재하지 않기 때문에 5개의 채널 중 데이터 채널만 사용하게 됩니다.

버스트 크기의 제한도 없기 때문에 영상 데이터나 AD Convert에서 입력되는 신호를 연결하기에 좋은 인터페이스 입니다.



### AXI 인터페이스, AXI4Lite

다음 그림은 AXI4Lite에 대한 블록도 입니다.



데이터 채널을 보면 데이터가 버스트 방식이 아니고 한번에 하나씩 전달되는 것을 알 수 있습니다. 아까 말씀 드린대로 AXI4Lite는 프로세서와 주변장치를 인터페이스할 때 사용 합니다.

여기서 한가지 자일링스에서 AXI4Lite 인터페이스를 만든 이유에 대해서 말씀드리겠습니다.

AXI는 버스가 아니다. Point to Point 커넥션이다 라고 먼저 말씀드렸습니다. 자일링스는 이 AXI를 모든 IP 에서 지원할 수 있도록 대대적으로 수정하게 됩니다.

당연히 Coregen에서 만드는 IP나 프로세서에서 사용하는 IP가 AXI를 지원하기 시작 했는데 문제는 처음에는 AXI4Full과 AXI4Stream만 지원했죠 (제가 정확히 기억이 나지는 않습다만)

그런데 간단한 프로세서 블록을 설계하고 UART하나만 설계했는데도 FPGA에서 로직사이즈가 기존에 비해 엄청나게 커진거죠.

많은 엔지니어들이 깜짝 놀랐죠.

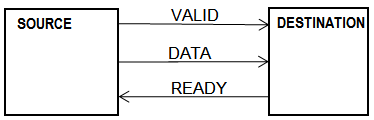
그리고 엄청난 컴플레인…

자일링스는 고민에 빠지게 됩니다. 그래서 만든 것이 AXI4Lite 버전이 입니다. 이 라이트 버전은 기존 AXI4Full에 비해서 굉장히 가볍게 설계되어 있습니다.

### AXI에서 알아두면 좋은 상식

#### Valid / Ready 핸드쉐이크

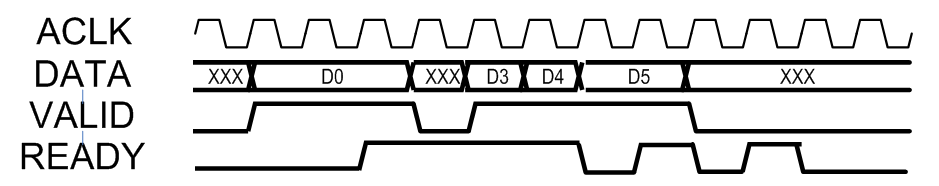
AXI 인터페이스를 구성하는 각 채널 안에는 Valid신호와 Ready 신호를 통해 마스터와 슬레이브간에 핸드쉐이크를 합니다.



Valid는 현제 데이터 (여기서는 어드레스일 수도 있고 데이터 일 수도 있습니다.) 에 실린 값이 의미가 있다라는 것을 알려 줍니다.

Ready는 슬레이브가 데이터를 받을 준비가 되어 있다는 것을 알려 줍니다.

다음 그림의 파형을 보면 마스터와 슬레이브간 데이터를 주고 받는 방식을 이해할 수 있을 것 입니다.



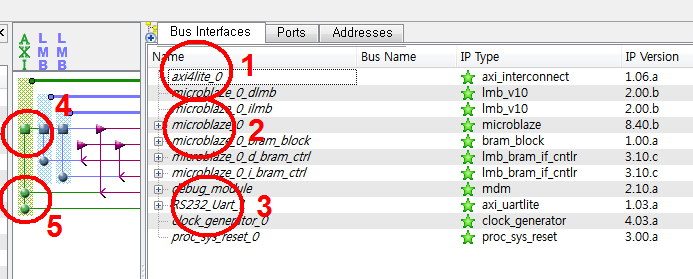
## Simulation을 통한 AXI4Lite 프로트콜 확인

### XPS 프로젝트, 버스 인터페이스

다음 그림은 자일링스에서 제공하는 SP605라는 데모보드를 대상으로 Microblaze 라는 프로세서와 UART를 연결상태를 보여 주고 있습니다.

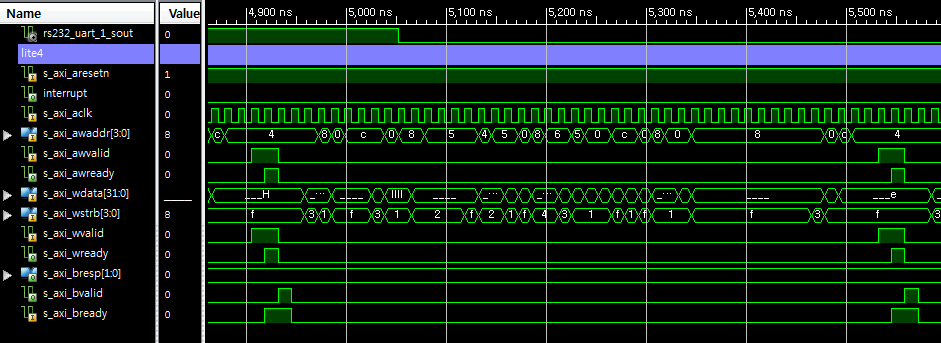
이 그림에서 1)은 AXI 인터커넥터에서 AXI4Lite 버전을 사용한 것을 보여 줍니다. 이 인터커넥터의 마스터로는 2) Microblaze가 연결되어 있고 3) Uart가 슬레이브로 연결되어 있습니다. 인터커넥터의 마스터 연결상태는 보통 4) 사각형으로 표시하고 슬레이브는 5) 원으로 표시 합니다.

이 인터커넥터에는 2개의 마스터와 2개의 슬레이브가 연결되어 있습니다.



## AXI4Lite 타이밍

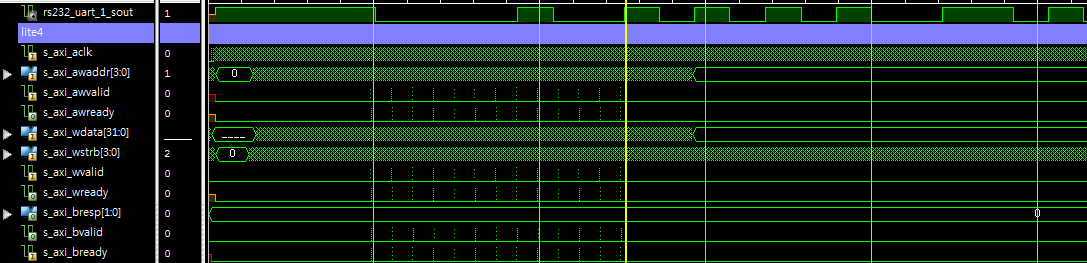
다음 그림은 Microblaze가 AXI 인터커넥터를 통해 UART에게 데이터를 쓰는 과정을 보여주고 있습니다.



신호가 잘 보이지는 않지만 awaddr[3:0], wavalid, awready, wdata, wstrb, wvalid, wready 가 정의된 프로토콜 대로 잘 움직이고 있습니다.

전체적으로 보면 Microblaze은 uart 내부에 있는 FIFO에 데이터를 전부 적고 나면 UART는 FIFO에 저장된 값을 차례대로 rs2323\_uart\_1\_sout 이라는 포트로 출력하고 있습니다.

다음 그림은 uart를 통해 데이터가 시리얼로 전송되는 것을 보여주고 있습니다.



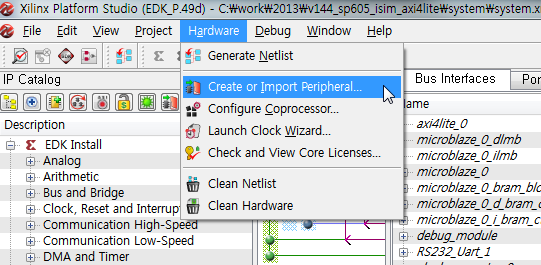
# CIP (Creat & Import Peripheral) Tool Flow

이번 장에서는 IP를 만드는 과정에 대해서 설명하도록 하겠습니다.

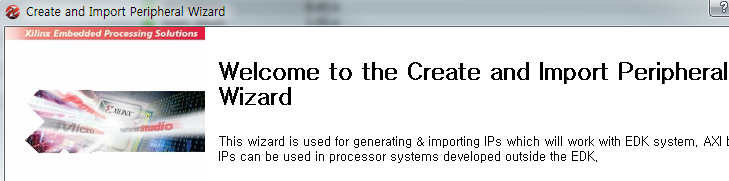
IP를 만들기 위해 필요한 툴 사용과정과 꼭 이해해야 하는 몇 개의 파일들, 수정하는 법을 배워보도록 하겠습니다.

## Tool Flow

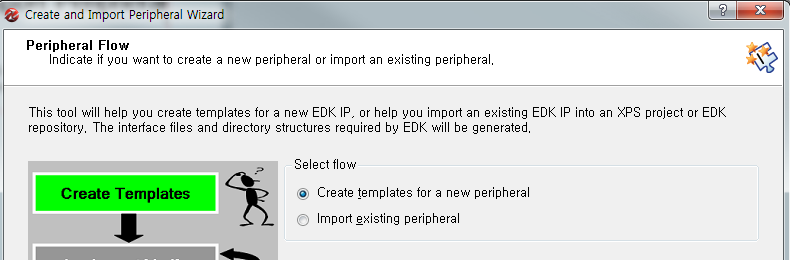
조금전에 시뮬레이션을 보여주었던 XPS 프로젝트에서 다음 메뉴를 선택 합니다.



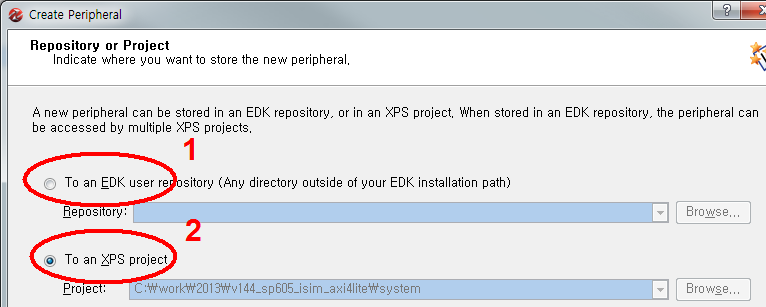
웰컴 메시지 나오고요



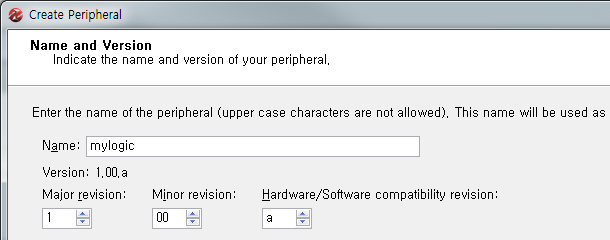
새로 만드는 과정을 거치기 때문에 첫번쨰 것을 선택 합니다.



다음 메뉴는 좀 생각할 만한 내용이 있습니다. 첫번쨰 것은 현재 만들려고 하는 IP를 다른 프로젝트에서도 이용할 수 있도록 공용 폴터에 저장하도록 합니다. 두번쨰 것은 현재 프로젝트에서만 사용할 수 있도록 로컬 폴더에 저장하도록 합니다. 여기서는 로컬 폴더에 저장 합니다.



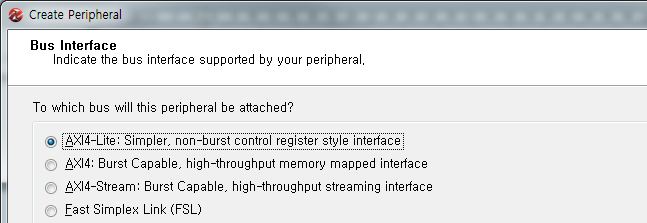
IP 이름을 정합니다. 이름 밑에 version 정보도 넣는 부분이 있는데 모두 기본 값을 선택 합니다.



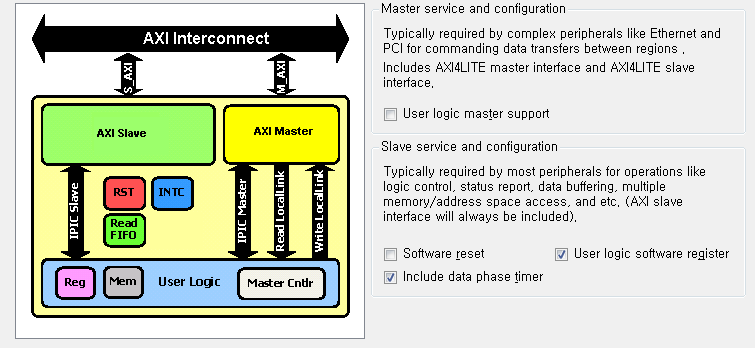
이제 이 IP는 어느 인터커넥터를 사용하는지 정해야 합니다.

현재 만드려고 하는 IP는 프로세서가 간단히 레지스터를 억세스 하는 용도로만 사용할 예정 입니다. 따라서 AXI4Lite를 선택 합니다.

제 경험상 개발자가 IP를 설계하는 경우는 2가지 경우가 있는데 지금과 같이 레지스터를 억세스하는 하는 슬레이브를 설계하거나 메모리를 억세스하기 위한 마스터를 설계할 때 입니다.



현재 설계하려고 하는 IP에 마스터 기능이 있어야 할 경우에는 “user logic master support”를 선택해야 하지만 현재 IP에는 해당사항이 없습니다. 기본 값을 선택 합니다.

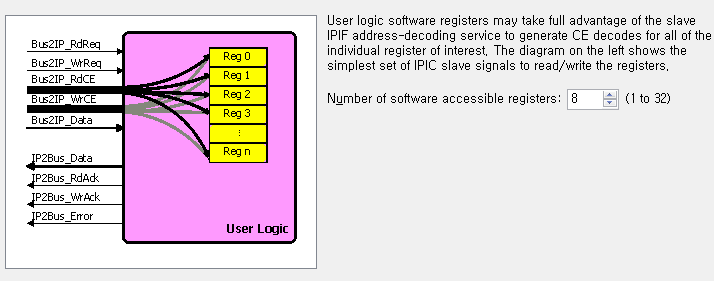


설계하려는 IP에 몇 개의 레지스터가 필요한지 선택 합니다.

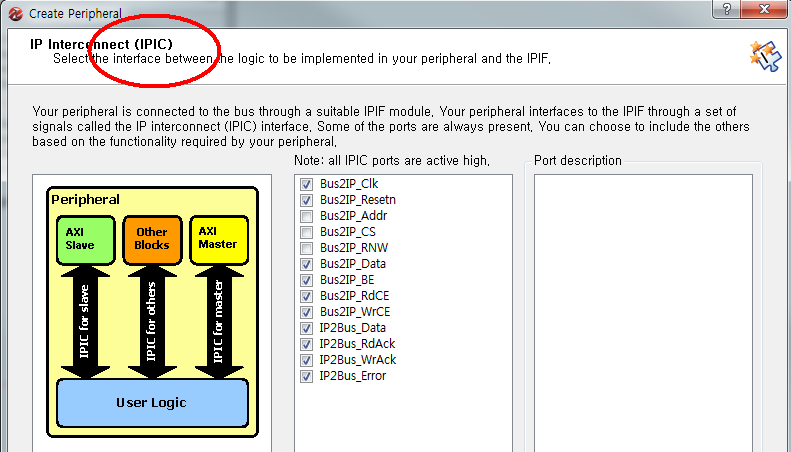
그런데 이 그림에서는 약간 다른 신호 이름이 보이는 군요. 앞에서 쭉 설명해 왔던 AXI에 대한 신호이름은 아닙니다.

Bus2IP\*\*\*, IP2Bus\*\*\*는 IPIC라는 인터커넥터인데 이 신호들은 개발자가 AXI라는 인터페이스 신호 전체를 다루지 않고도 쉽게 슬레이브를 억세스 할 수 있도록 도와 줍니다.

앞으로 개발하는 모든 IP는 특별한 경우가 아니면 Bus2IP\*\*\*, IP2Bus\*\*\*를 사용할 예정 입니다. 이 신호의 집합을 IPIC라고 합니다.



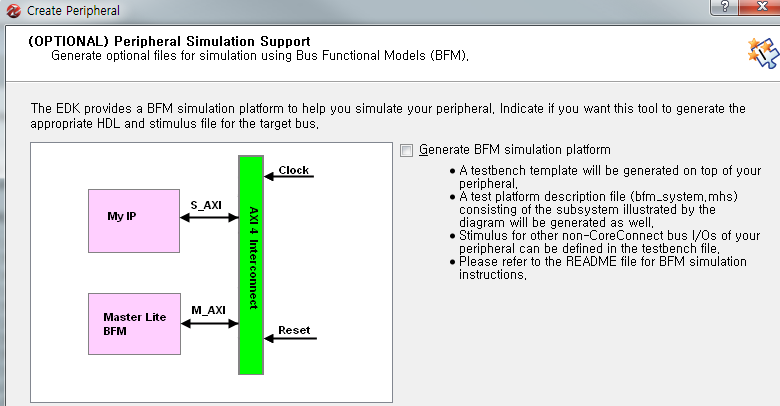
다음 그림은 IPIC중 사용중인 신호 이름을 보여주고 있습니다. 기본값을 선택 합니다.



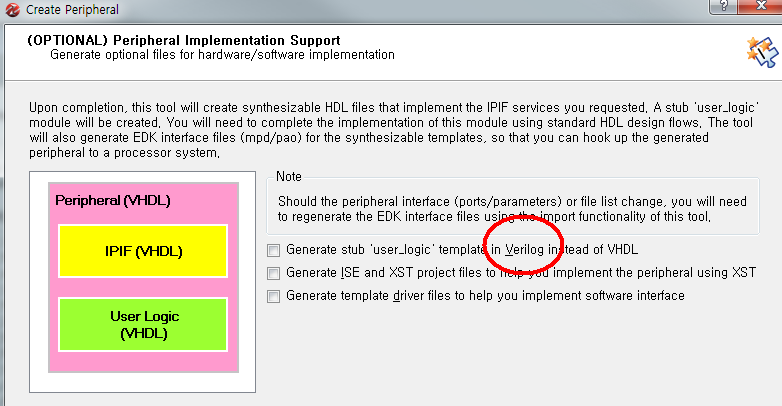
BFM 이라는 말은 Bus Function Modeling의 약자로 마스터와 슬레이브간에 데이터 전송이 제대로 이루어 지는지 검사하는 시뮬레이션 방법 중 하나 입니다.

내부 신호들에 대한 모든 시뮬레이션 과정을 거치지 않고 단순히 억세스가 제대로 되는지 안되는지에 관심을 가지는 시뮬레이션 입니다.

이 과정에서 이 부분은 다루지 않습니다.



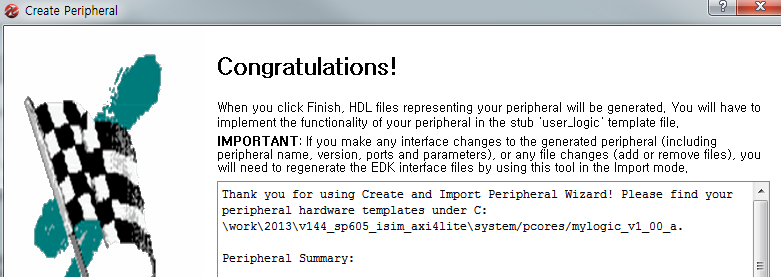
Verilog 개발자라면 아래 옵션을 선택 합니다.



IP를 생성하기 위한 모든 입력을 마쳤습니다.

한가지 기억해야 하는 것은 CIP 과정을 거쳐 만들어지는 IP는 기본적으로 앞에서 언급한 8개의 레지스터를 억세스 하는 과정에 대해서만 설계되어 있습니다.

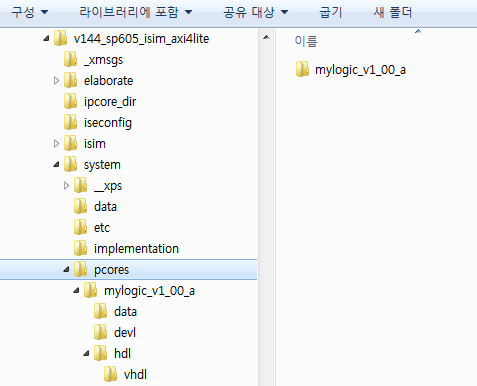
수정할 부분이 있는 떄는 어느 파일을 어떻게 수정해야 하는지 차례대로 살펴보겠습니다.



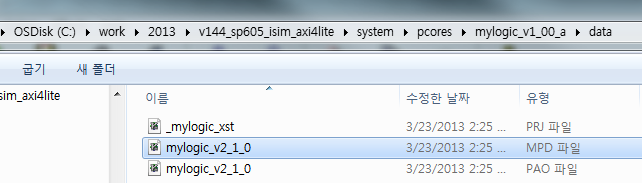
## 폴더 구조

다음 그림은 새로 추가된 IP 구성하는 파일 및 폴더 구조를 보여 줍니다. 여기서 중요한 폴더가 pcores라는 폴더 인데 이 폴더에는 mylogic이라는 이름과 버전 정보들이 구성된 폴더가 있습니다.

이 폴더 안에 data, devl, hdl 폴더가 있고 Verilog 옵션을 선택하지 않았을 때는 vhdl 폴더가 만들어 집니다.



먼저 data 폴더를 보면 MPD 파일과 PAO 파일이 있습니다.



### MPD File

다음 리스트는 MPD내용을 보여 주고 있습니다.

1 ###################################################################

2 ##

3 ## Name : mylogic

4 ## Desc : Microprocessor Peripheral Description

5 ## : Automatically generated by PsfUtility

6 ##

7 ###################################################################

8

9 BEGIN mylogic

10

11 ## Peripheral Options

12 OPTION IPTYPE = PERIPHERAL

13 OPTION IMP\_NETLIST = TRUE

14 OPTION HDL = VHDL

15 OPTION IP\_GROUP = MICROBLAZE:USER

16 OPTION DESC = MYLOGIC

17 OPTION ARCH\_SUPPORT\_MAP = (others=DEVELOPMENT)

18

19

20 ## Bus Interfaces

21 BUS\_INTERFACE BUS = S\_AXI, BUS\_STD = AXI, BUS\_TYPE = SLAVE

22

23 ## Generics for VHDL or Parameters for Verilog

24 PARAMETER C\_S\_AXI\_DATA\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

25 PARAMETER C\_S\_AXI\_ADDR\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

26 PARAMETER C\_S\_AXI\_MIN\_SIZE = 0x000001ff, DT = std\_logic\_vector, BUS = S\_AXI

27 PARAMETER C\_USE\_WSTRB = 0, DT = INTEGER

28 PARAMETER C\_DPHASE\_TIMEOUT = 8, DT = INTEGER

29 PARAMETER C\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, MIN\_SIZE = 0x100, PAIR = C\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

30 PARAMETER C\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

31 PARAMETER C\_FAMILY = virtex6, DT = STRING

32 PARAMETER C\_NUM\_REG = 1, DT = INTEGER

33 PARAMETER C\_NUM\_MEM = 1, DT = INTEGER

34 PARAMETER C\_SLV\_AWIDTH = 32, DT = INTEGER

35 PARAMETER C\_SLV\_DWIDTH = 32, DT = INTEGER

36 PARAMETER C\_S\_AXI\_PROTOCOL = AXI4LITE, TYPE = NON\_HDL, ASSIGNMENT = CONSTANT, DT = STRING, BUS = S\_AXI

37

38 ## Ports

39 PORT S\_AXI\_ACLK = "", DIR = I, SIGIS = CLK, BUS = S\_AXI

40 PORT S\_AXI\_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS = S\_AXI

41 PORT S\_AXI\_AWADDR = AWADDR, DIR = I, VEC = [(C\_S\_AXI\_ADDR\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

42 PORT S\_AXI\_AWVALID = AWVALID, DIR = I, BUS = S\_AXI

43 PORT S\_AXI\_WDATA = WDATA, DIR = I, VEC = [(C\_S\_AXI\_DATA\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

44 PORT S\_AXI\_WSTRB = WSTRB, DIR = I, VEC = [((C\_S\_AXI\_DATA\_WIDTH/8)-1):0], ENDIAN = LITTLE, BUS = S\_AXI

45 PORT S\_AXI\_WVALID = WVALID, DIR = I, BUS = S\_AXI

46 PORT S\_AXI\_BREADY = BREADY, DIR = I, BUS = S\_AXI

47 PORT S\_AXI\_ARADDR = ARADDR, DIR = I, VEC = [(C\_S\_AXI\_ADDR\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

48 PORT S\_AXI\_ARVALID = ARVALID, DIR = I, BUS = S\_AXI

49 PORT S\_AXI\_RREADY = RREADY, DIR = I, BUS = S\_AXI

50 PORT S\_AXI\_ARREADY = ARREADY, DIR = O, BUS = S\_AXI

51 PORT S\_AXI\_RDATA = RDATA, DIR = O, VEC = [(C\_S\_AXI\_DATA\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

52 PORT S\_AXI\_RRESP = RRESP, DIR = O, VEC = [1:0], BUS = S\_AXI

53 PORT S\_AXI\_RVALID = RVALID, DIR = O, BUS = S\_AXI

54 PORT S\_AXI\_WREADY = WREADY, DIR = O, BUS = S\_AXI

55 PORT S\_AXI\_BRESP = BRESP, DIR = O, VEC = [1:0], BUS = S\_AXI

56 PORT S\_AXI\_BVALID = BVALID, DIR = O, BUS = S\_AXI

57 PORT S\_AXI\_AWREADY = AWREADY, DIR = O, BUS = S\_AXI

58

59 END

먼저 라인 38~ 57까지 보면 AXI4Lite에서 사용하는 각종 채널 신호들을 확인할 수 있습니다.

라인 39에는 클럭포트가 비어 있는데 이 부분은 나중에 XPS에서 정의해 줘야 합니다.

39 PORT S\_AXI\_ACLK = "", DIR = I, SIGIS = CLK, BUS = S\_AXI

하지만 그 다음 라인 부터는 모두 신호들이 Mapping되어 있습니다.

두번쩌 라인 24부터 36까지는 현재 IP를 구성하는데 필요한 각종 파라미터를 정의한 부분 입니다.

라인 29,30 을 보면 IP의 어드레스를 정한 부분이 있는데 base는 0xffff\_ffff로 정의되어 있고 high는 0x0000\_0000으로 정의되어 있습니다.

이렇게 사용못하기 때문에 설계자는 이 IP를 억세스할 수 있는 어드레스 범위를 XPS에서 재정의 해야 합니다.

### PAO File

PAO파일에는 현재 IP를 구성하는 소스 파일과 합성 순서를 보여 줍니다.

이중에 개발자들이 주로 수정하는 파일은 user\_Logic.vhd 와 mylogic.vhd 파일 입니다.

##############################################################################

## Filename: C:/work/2013/v144\_sp605\_isim\_axi4lite/system/pcores/mylogic\_v1\_00\_a/data/mylogic\_v2\_1\_0.pao

## Description: Peripheral Analysis Order

## Date: Sat Mar 23 14:25:38 2013 (by Create and Import Peripheral Wizard)

##############################################################################

lib proc\_common\_v3\_00\_a all

lib axi\_lite\_ipif\_v1\_01\_a all

lib mylogic\_v1\_00\_a user\_logic vhdl

lib mylogic\_v1\_00\_a mylogic vhdl

### User\_logic.vhd

다음 리스트는 user\_logic.vhd 에 대한 내용 입니다.

1 ------------------------------------------------------------------------------

2 -- user\_logic.vhd - entity/architecture pair

3 ------------------------------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>

50

51 -- DO NOT EDIT BELOW THIS LINE --------------------

52 library ieee;

53 use ieee.std\_logic\_1164.all;

54 use ieee.std\_logic\_arith.all;

55 use ieee.std\_logic\_unsigned.all;

56

57 library proc\_common\_v3\_00\_a;

58 use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;

59

60 -- DO NOT EDIT ABOVE THIS LINE --------------------

61

62 --USER libraries added here

63

IPIC를 구성하는 신호들 입니다.

64 ------------------------------------------------------------------------------

65 -- Entity section

66 ------------------------------------------------------------------------------

67 -- Definition of Generics:

68 -- C\_NUM\_REG -- Number of software accessible registers

69 -- C\_SLV\_DWIDTH -- Slave interface data bus width

70 --

71 -- Definition of Ports:

72 -- Bus2IP\_Clk -- Bus to IP clock

73 -- Bus2IP\_Resetn -- Bus to IP reset

74 -- Bus2IP\_Data -- Bus to IP data bus

75 -- Bus2IP\_BE -- Bus to IP byte enables

76 -- Bus2IP\_RdCE -- Bus to IP read chip enable

77 -- Bus2IP\_WrCE -- Bus to IP write chip enable

78 -- IP2Bus\_Data -- IP to Bus data bus

79 -- IP2Bus\_RdAck -- IP to Bus read transfer acknowledgement

80 -- IP2Bus\_WrAck -- IP to Bus write transfer acknowledgement

81 -- IP2Bus\_Error -- IP to Bus error response

82 ------------------------------------------------------------------------------

Entity에는 generic 구문과 port 구문이 있는데 generic 구문에 있는 여러 파라미터는 MPD에 정의된 값이 이 genric 문에 전달 됩니다. 만약 MPD에 있는 것과 다른 파라미터를 사용하려면 MHS 파일에서 새로운 값을 정의 하면 됩니다.

즉 MPD 파일은 각종 파라미터의 기본값을 정의해 놓은 곳 입니다. 기본값과 다르게 사용하면 MHS 파일에서 재정의 하면 됩니다.

83

84 entity user\_logic is

85 generic

86 (

87 -- ADD USER GENERICS BELOW THIS LINE ---------------

88 --USER generics added here

89 -- ADD USER GENERICS ABOVE THIS LINE ---------------

90

91 -- DO NOT EDIT BELOW THIS LINE ---------------------

92 -- Bus protocol parameters, do not add to or delete

93 C\_NUM\_REG : integer := 8;

94 C\_SLV\_DWIDTH : integer := 32

95 -- DO NOT EDIT ABOVE THIS LINE ---------------------

96 );

다음은 포트 구성 입니다. 라인 101을 보면 새롭게 포트를 추가할 경우 101라인에 추가하라고 되어 있습니다.

IP를 설계할 때는 주로 포트들이 추가 되는 데 추가 되는 포트를 이곳에 정의하면 됩니다.

97 port

98 (

99 -- ADD USER PORTS BELOW THIS LINE ------------------

100 --USER ports added here

101 -- ADD USER PORTS ABOVE THIS LINE ------------------

102

103 -- DO NOT EDIT BELOW THIS LINE ---------------------

104 -- Bus protocol ports, do not add to or delete

나머지 IPIC에 대한 입출력 신호를 정의 했습니다.

105 Bus2IP\_Clk : in std\_logic;

106 Bus2IP\_Resetn : in std\_logic;

107 Bus2IP\_Data : in std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

108 Bus2IP\_BE : in std\_logic\_vector(C\_SLV\_DWIDTH/8-1 downto 0);

109 Bus2IP\_RdCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);

110 Bus2IP\_WrCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);

111 IP2Bus\_Data : out std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

112 IP2Bus\_RdAck : out std\_logic;

113 IP2Bus\_WrAck : out std\_logic;

114 IP2Bus\_Error : out std\_logic

115 -- DO NOT EDIT ABOVE THIS LINE ---------------------

116 );

117

118 attribute MAX\_FANOUT : string;

119 attribute SIGIS : string;

120

121 attribute SIGIS of Bus2IP\_Clk : signal is "CLK";

122 attribute SIGIS of Bus2IP\_Resetn : signal is "RST";

123

124 end entity user\_logic;

125

126 ------------------------------------------------------------------------------

127 -- Architecture section

128 ------------------------------------------------------------------------------

129

130 architecture IMP of user\_logic is

131

132 --USER signal declarations added here, as needed for user logic

133

134 ------------------------------------------

135 -- Signals for user logic slave model s/w accessible register example

136 ------------------------------------------

137 signal slv\_reg0 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

138 signal slv\_reg1 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

139 signal slv\_reg2 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

140 signal slv\_reg3 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

141 signal slv\_reg4 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

142 signal slv\_reg5 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

143 signal slv\_reg6 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

144 signal slv\_reg7 : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

145 signal slv\_reg\_write\_sel : std\_logic\_vector(7 downto 0);

146 signal slv\_reg\_read\_sel : std\_logic\_vector(7 downto 0);

147 signal slv\_ip2bus\_data : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

148 signal slv\_read\_ack : std\_logic;

149 signal slv\_write\_ack : std\_logic;

150

151 begin

152

153 --USER logic implementation added here

154

155 ------------------------------------------

156 -- Example code to read/write user logic slave model s/w accessible registers

157 --

158 -- Note:

159 -- The example code presented here is to show you one way of reading/writing

160 -- software accessible registers implemented in the user logic slave model.

161 -- Each bit of the Bus2IP\_WrCE/Bus2IP\_RdCE signals is configured to correspond

162 -- to one software accessible register by the top level template. For example,

163 -- if you have four 32 bit software accessible registers in the user logic,

164 -- you are basically operating on the following memory mapped registers:

165 --

166 -- Bus2IP\_WrCE/Bus2IP\_RdCE Memory Mapped Register

167 -- "1000" C\_BASEADDR + 0x0

168 -- "0100" C\_BASEADDR + 0x4

169 -- "0010" C\_BASEADDR + 0x8

170 -- "0001" C\_BASEADDR + 0xC

171 --

172 ------------------------------------------

Bus2IP\_WrCE(7 downto 0)라는 것은 앞에서 이 IP를 정의할 때 레지스터를 모두 8개 정의 했기 때문에 생긴 신호 입니다. 마찬가지로 Bus2IP\_RdCE도 같은 의미 입니다. Read, write할 때 8개의 레지스터 마다 모두 독립적인 CS를 가지고 있습니다.

173 slv\_reg\_write\_sel <= Bus2IP\_WrCE(7 downto 0);

174 slv\_reg\_read\_sel <= Bus2IP\_RdCE(7 downto 0);

ACK라는 신호는 현재 IP가 선택되었을 때 슬레이브가 올바로 디코딩이 됐다는 것을 버스에 알려줘야 합니다. 라인 176, 176은 마스터가 슬레이브 각각 read/write할 때 따로 따로 ack를 돌려주기 위한 설계를 보여주고 있습니다.

175 slv\_write\_ack <= Bus2IP\_WrCE(0) or Bus2IP\_WrCE(1) or Bus2IP\_WrCE(2) or Bus2IP\_WrCE(3) or Bus2IP\_WrCE(4) or Bus2IP\_WrCE(5) or Bus2IP\_WrCE(6) or Bus2IP\_WrCE(7);

176 slv\_read\_ack <= Bus2IP\_RdCE(0) or Bus2IP\_RdCE(1) or Bus2IP\_RdCE(2) or Bus2IP\_RdCE(3) or Bus2IP\_RdCE(4) or Bus2IP\_RdCE(5) or Bus2IP\_RdCE(6) or Bus2IP\_RdCE(7);

177

라인 178부터 247까지는 프로세서가 8개의 레지스터를 억세스 할 떄 필요한 디코딩 신호를 만들어 주고 있습니다. 중간에 for loop가 들어가 있는 것은 BE (Byte Enable)에 따라 억세스 되는 바이트를 선택하기 위해 사용되고 있습니다.

178 -- implement slave model software accessible register(s)

179 SLAVE\_REG\_WRITE\_PROC : process( Bus2IP\_Clk ) is

180 begin

181

182 if Bus2IP\_Clk'event and Bus2IP\_Clk = '1' then

183 if Bus2IP\_Resetn = '0' then

184 slv\_reg0 <= (others => '0');

185 slv\_reg1 <= (others => '0');

186 slv\_reg2 <= (others => '0');

187 slv\_reg3 <= (others => '0');

188 slv\_reg4 <= (others => '0');

189 slv\_reg5 <= (others => '0');

190 slv\_reg6 <= (others => '0');

191 slv\_reg7 <= (others => '0');

192 else

라인 192~246까지는 8개의 cs를 이용해서 8개의 slv\_reg# 중 하나를 선택해서 쓰기 동작을 하는 로직을 보여 줍니다.

프로세서가 byte accessable 하기 때문에 (즉 byte, half word, word) 단위로 쓸 수 있기 때문에 Bus2IP\_BE (Byte Enable) 를 이용하는 로직이 for loop에 추가 되어 있습니다.

193 case slv\_reg\_write\_sel is

194 when "10000000" =>

195 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

196 if ( Bus2IP\_BE(byte\_index) = '1' ) then

197 slv\_reg0(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

198 end if;

199 end loop;

200 when "01000000" =>

201 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

202 if ( Bus2IP\_BE(byte\_index) = '1' ) then

203 slv\_reg1(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

204 end if;

205 end loop;

206 when "00100000" =>

207 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

208 if ( Bus2IP\_BE(byte\_index) = '1' ) then

209 slv\_reg2(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

210 end if;

211 end loop;

212 when "00010000" =>

213 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

214 if ( Bus2IP\_BE(byte\_index) = '1' ) then

215 slv\_reg3(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

216 end if;

217 end loop;

218 when "00001000" =>

219 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

220 if ( Bus2IP\_BE(byte\_index) = '1' ) then

221 slv\_reg4(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

ㅋ1222 end if;

223 end loop;

224 when "00000100" =>

225 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

226 if ( Bus2IP\_BE(byte\_index) = '1' ) then

227 slv\_reg5(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

228 end if;

229 end loop;

230 when "00000010" =>

231 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

232 if ( Bus2IP\_BE(byte\_index) = '1' ) then

233 slv\_reg6(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

234 end if;

235 end loop;

236 when "00000001" =>

237 for byte\_index in 0 to (C\_SLV\_DWIDTH/8)-1 loop

238 if ( Bus2IP\_BE(byte\_index) = '1' ) then

239 slv\_reg7(byte\_index\*8+7 downto byte\_index\*8) <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

240 end if;

241 end loop;

242 when others => null;

243 end case;

244 end if;

245 end if;

246

247 end process SLAVE\_REG\_WRITE\_PROC;

248

다음 마스터가 슬레이브에서 레지스터를 읽을 때 사용되는 로직 구성 입니다. 라인 254부터 262까지 CS에 따라 모두 8개의 레지스터가 디코딩 되어서 버스에 실리는 것을 확인할 수 있습니다.

249 -- implement slave model software accessible register(s) read mux

250 SLAVE\_REG\_READ\_PROC : process( slv\_reg\_read\_sel, slv\_reg0, slv\_reg1, slv\_reg2, slv\_reg3, slv\_reg4, slv\_reg5, slv\_reg6, slv\_reg7 ) is

251 begin

252

253 case slv\_reg\_read\_sel is

254 when "10000000" => slv\_ip2bus\_data <= slv\_reg0;

255 when "01000000" => slv\_ip2bus\_data <= slv\_reg1;

256 when "00100000" => slv\_ip2bus\_data <= slv\_reg2;

257 when "00010000" => slv\_ip2bus\_data <= slv\_reg3;

258 when "00001000" => slv\_ip2bus\_data <= slv\_reg4;

259 when "00000100" => slv\_ip2bus\_data <= slv\_reg5;

260 when "00000010" => slv\_ip2bus\_data <= slv\_reg6;

261 when "00000001" => slv\_ip2bus\_data <= slv\_reg7;

262 when others => slv\_ip2bus\_data <= (others => '0');

263 end case;

264

265 end process SLAVE\_REG\_READ\_PROC;

슬레이브는 마스터의 요청가 읽기를 시도할 때만 데이터를 버스에 넘겨주고 나머지는 “0” 상태를 유지해야 합니다.

266

267 ------------------------------------------

268 -- Example code to drive IP to Bus signals

269 ------------------------------------------

270 IP2Bus\_Data <= slv\_ip2bus\_data when slv\_read\_ack = '1' else

271 (others => '0');

272

슬레이브에서 만들어진 ack 신호를 버스로 돌려 줍니다.

273 IP2Bus\_WrAck <= slv\_write\_ack;

274 IP2Bus\_RdAck <= slv\_read\_ack;

275 IP2Bus\_Error <= '0';

276

277 end IMP;

### Mylogic.vhd

다음 리스트 user\_logic.vhd와 AXI 인터커넥터를 연결시켜 주기 위한 mylogic.vhd에 대한 설명 입니다.

1 ------------------------------------------------------------------------------

2 -- mylogic.vhd - entity/architecture pair

>>>>>>>

58 ------------------------------------------------------------------------------

59

60 library ieee;

61 use ieee.std\_logic\_1164.all;

62 use ieee.std\_logic\_arith.all;

63 use ieee.std\_logic\_unsigned.all;

64

65 library proc\_common\_v3\_00\_a;

66 use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;

67 use proc\_common\_v3\_00\_a.ipif\_pkg.all;

68

69 library axi\_lite\_ipif\_v1\_01\_a;

70 use axi\_lite\_ipif\_v1\_01\_a.axi\_lite\_ipif;

71

72 library mylogic\_v1\_00\_a;

73 use mylogic\_v1\_00\_a.user\_logic;

>>>>>>>>>>>

Mylogic도 entity 부분과 generic문으로 나누어 집니다.

114 entity mylogic is

115 generic

116 (

117 -- ADD USER GENERICS BELOW THIS LINE ---------------

118 --USER generics added here

119 -- ADD USER GENERICS ABOVE THIS LINE ---------------

120

121 -- DO NOT EDIT BELOW THIS LINE ---------------------

122 -- Bus protocol parameters, do not add to or delete

123 C\_S\_AXI\_DATA\_WIDTH : integer := 32;

124 C\_S\_AXI\_ADDR\_WIDTH : integer := 32;

125 C\_S\_AXI\_MIN\_SIZE : std\_logic\_vector := X"000001FF";

126 C\_USE\_WSTRB : integer := 0;

127 C\_DPHASE\_TIMEOUT : integer := 8;

128 C\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

129 C\_HIGHADDR : std\_logic\_vector := X"00000000";

130 C\_FAMILY : string := "virtex6";

131 C\_NUM\_REG : integer := 1;

132 C\_NUM\_MEM : integer := 1;

133 C\_SLV\_AWIDTH : integer := 32;

134 C\_SLV\_DWIDTH : integer := 32

135 -- DO NOT EDIT ABOVE THIS LINE ---------------------

136 );

User\_logic에서 추가되었던 포트들이 여기까지 연결되어야 합니다.

137 port

138 (

139 -- ADD USER PORTS BELOW THIS LINE ------------------

140 --USER ports added here

141 -- ADD USER PORTS ABOVE THIS LINE ------------------

142

143 -- DO NOT EDIT BELOW THIS LINE ---------------------

144 -- Bus protocol ports, do not add to or delete

145 S\_AXI\_ACLK : in std\_logic;

146 S\_AXI\_ARESETN : in std\_logic;

147 S\_AXI\_AWADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);

148 S\_AXI\_AWVALID : in std\_logic;

149 S\_AXI\_WDATA : in std\_logic\_vector(C\_S\_AXI\_DATA\_WIDTH-1 downto 0);

150 S\_AXI\_WSTRB : in std\_logic\_vector((C\_S\_AXI\_DATA\_WIDTH/8)-1 downto 0);

151 S\_AXI\_WVALID : in std\_logic;

152 S\_AXI\_BREADY : in std\_logic;

153 S\_AXI\_ARADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);

154 S\_AXI\_ARVALID : in std\_logic;

155 S\_AXI\_RREADY : in std\_logic;

156 S\_AXI\_ARREADY : out std\_logic;

157 S\_AXI\_RDATA : out std\_logic\_vector(C\_S\_AXI\_DATA\_WIDTH-1 downto 0);

158 S\_AXI\_RRESP : out std\_logic\_vector(1 downto 0);

159 S\_AXI\_RVALID : out std\_logic;

160 S\_AXI\_WREADY : out std\_logic;

161 S\_AXI\_BRESP : out std\_logic\_vector(1 downto 0);

162 S\_AXI\_BVALID : out std\_logic;

163 S\_AXI\_AWREADY : out std\_logic

164 -- DO NOT EDIT ABOVE THIS LINE ---------------------

165 );

166

167 attribute MAX\_FANOUT : string;

168 attribute SIGIS : string;

169 attribute MAX\_FANOUT of S\_AXI\_ACLK : signal is "10000";

170 attribute MAX\_FANOUT of S\_AXI\_ARESETN : signal is "10000";

171 attribute SIGIS of S\_AXI\_ACLK : signal is "Clk";

172 attribute SIGIS of S\_AXI\_ARESETN : signal is "Rst";

173 end entity mylogic;

174

175 ------------------------------------------------------------------------------

176 -- Architecture section

177 ------------------------------------------------------------------------------

178

179 architecture IMP of mylogic is

180

181

>>>>>>>>>>>>>>>>>>>>>>>>

234

235 begin

236

237 ------------------------------------------

238 -- instantiate axi\_lite\_ipif

239 ------------------------------------------

IPIC와 AXI AXI4Lite를 연결하는 부분 입니다.

240 AXI\_LITE\_IPIF\_I : entity axi\_lite\_ipif\_v1\_01\_a.axi\_lite\_ipif

241 generic map

242 (

243 C\_S\_AXI\_DATA\_WIDTH => IPIF\_SLV\_DWIDTH,

244 C\_S\_AXI\_ADDR\_WIDTH => C\_S\_AXI\_ADDR\_WIDTH,

245 C\_S\_AXI\_MIN\_SIZE => C\_S\_AXI\_MIN\_SIZE,

246 C\_USE\_WSTRB => C\_USE\_WSTRB,

247 C\_DPHASE\_TIMEOUT => C\_DPHASE\_TIMEOUT,

248 C\_ARD\_ADDR\_RANGE\_ARRAY => IPIF\_ARD\_ADDR\_RANGE\_ARRAY,

249 C\_ARD\_NUM\_CE\_ARRAY => IPIF\_ARD\_NUM\_CE\_ARRAY,

250 C\_FAMILY => C\_FAMILY

251 )

252 port map

253 (

254 S\_AXI\_ACLK => S\_AXI\_ACLK,

255 S\_AXI\_ARESETN => S\_AXI\_ARESETN,

256 S\_AXI\_AWADDR => S\_AXI\_AWADDR,

257 S\_AXI\_AWVALID => S\_AXI\_AWVALID,

258 S\_AXI\_WDATA => S\_AXI\_WDATA,

259 S\_AXI\_WSTRB => S\_AXI\_WSTRB,

260 S\_AXI\_WVALID => S\_AXI\_WVALID,

261 S\_AXI\_BREADY => S\_AXI\_BREADY,

262 S\_AXI\_ARADDR => S\_AXI\_ARADDR,

263 S\_AXI\_ARVALID => S\_AXI\_ARVALID,

264 S\_AXI\_RREADY => S\_AXI\_RREADY,

265 S\_AXI\_ARREADY => S\_AXI\_ARREADY,

266 S\_AXI\_RDATA => S\_AXI\_RDATA,

267 S\_AXI\_RRESP => S\_AXI\_RRESP,

268 S\_AXI\_RVALID => S\_AXI\_RVALID,

269 S\_AXI\_WREADY => S\_AXI\_WREADY,

270 S\_AXI\_BRESP => S\_AXI\_BRESP,

271 S\_AXI\_BVALID => S\_AXI\_BVALID,

272 S\_AXI\_AWREADY => S\_AXI\_AWREADY,

273 Bus2IP\_Clk => ipif\_Bus2IP\_Clk,

274 Bus2IP\_Resetn => ipif\_Bus2IP\_Resetn,

275 Bus2IP\_Addr => ipif\_Bus2IP\_Addr,

276 Bus2IP\_RNW => ipif\_Bus2IP\_RNW,

277 Bus2IP\_BE => ipif\_Bus2IP\_BE,

278 Bus2IP\_CS => ipif\_Bus2IP\_CS,

279 Bus2IP\_RdCE => ipif\_Bus2IP\_RdCE,

280 Bus2IP\_WrCE => ipif\_Bus2IP\_WrCE,

281 Bus2IP\_Data => ipif\_Bus2IP\_Data,

282 IP2Bus\_WrAck => ipif\_IP2Bus\_WrAck,

283 IP2Bus\_RdAck => ipif\_IP2Bus\_RdAck,

284 IP2Bus\_Error => ipif\_IP2Bus\_Error,

285 IP2Bus\_Data => ipif\_IP2Bus\_Data

286 );

287

User\_logic과 연결되는 부분 입니다.

288 ------------------------------------------

289 -- instantiate User Logic

290 ------------------------------------------

291 USER\_LOGIC\_I : entity mylogic\_v1\_00\_a.user\_logic

292 generic map

293 (

294 -- MAP USER GENERICS BELOW THIS LINE ---------------

295 --USER generics mapped here

296 -- MAP USER GENERICS ABOVE THIS LINE ---------------

297

298 C\_NUM\_REG => USER\_NUM\_REG,

299 C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH

300 )

아래 부분에 user logic에서 추가된 부분을 연결하도록 합니다.

301 port map

302 (

303 -- MAP USER PORTS BELOW THIS LINE ------------------

304 --USER ports mapped here

305 -- MAP USER PORTS ABOVE THIS LINE ------------------

306

307 Bus2IP\_Clk => ipif\_Bus2IP\_Clk,

308 Bus2IP\_Resetn => ipif\_Bus2IP\_Resetn,

309 Bus2IP\_Data => ipif\_Bus2IP\_Data,

310 Bus2IP\_BE => ipif\_Bus2IP\_BE,

311 Bus2IP\_RdCE => user\_Bus2IP\_RdCE,

312 Bus2IP\_WrCE => user\_Bus2IP\_WrCE,

313 IP2Bus\_Data => user\_IP2Bus\_Data,

314 IP2Bus\_RdAck => user\_IP2Bus\_RdAck,

315 IP2Bus\_WrAck => user\_IP2Bus\_WrAck,

316 IP2Bus\_Error => user\_IP2Bus\_Error

317 );

318

319 ------------------------------------------

320 -- connect internal signals

321 ------------------------------------------

322 ipif\_IP2Bus\_Data <= user\_IP2Bus\_Data;

323 ipif\_IP2Bus\_WrAck <= user\_IP2Bus\_WrAck;

324 ipif\_IP2Bus\_RdAck <= user\_IP2Bus\_RdAck;

325 ipif\_IP2Bus\_Error <= user\_IP2Bus\_Error;

326

327 user\_Bus2IP\_RdCE <= ipif\_Bus2IP\_RdCE(USER\_NUM\_REG-1 downto 0);

328 user\_Bus2IP\_WrCE <= ipif\_Bus2IP\_WrCE(USER\_NUM\_REG-1 downto 0);

329

330 end IMP;

## Mylogic IP 수정

Mylogic IP는 레지스터를 8개 가지고 있습니다. IP외부와는 전혀 인터페이스가 있지 않습니다.

이 IP에 있는 8개의 레지스터 중 하나는 IP 외부로 입출력 하도록 수정해 보겠습니다.

### User\_logic.vhd 수정

라이 100, 101에 새로운 2개의 포트가 추가되었습니다. 라인 180에 slv\_reg7 의 출력이 my\_out으로 연결되고 라인 265에서 my\_in에 입력 값이 버스로 전달 됩니다.

97 port

98 (

99 -- ADD USER PORTS BELOW THIS LINE ------------------

100 --USER ports added here

101 my\_out : out std\_logic\_vector(31 downto 0);

102 my\_in : in std\_logic\_vector(31 downto 0);

103 -- ADD USER PORTS ABOVE THIS LINE ------------------

104

>>>>>>>>>>>

180 my\_out <= slv\_reg7;

>>>>>>>>>>>

254 SLAVE\_REG\_READ\_PROC : process( slv\_reg\_read\_sel, slv\_reg0, slv\_reg1, slv\_reg2, slv\_reg3, slv\_reg4, slv\_reg5, slv\_reg6, slv\_reg7 ) is

255 begin

256

257 case slv\_reg\_read\_sel is

258 when "10000000" => slv\_ip2bus\_data <= slv\_reg0;

259 when "01000000" => slv\_ip2bus\_data <= slv\_reg1;

260 when "00100000" => slv\_ip2bus\_data <= slv\_reg2;

261 when "00010000" => slv\_ip2bus\_data <= slv\_reg3;

262 when "00001000" => slv\_ip2bus\_data <= slv\_reg4;

263 when "00000100" => slv\_ip2bus\_data <= slv\_reg5;

264 when "00000010" => slv\_ip2bus\_data <= slv\_reg6;

265 when "00000001" => slv\_ip2bus\_data <= my\_in;

266 when others => slv\_ip2bus\_data <= (others => '0');

267 end case;

268

269 end process SLAVE\_REG\_READ\_PROC;

### Mylogic.vhd 수정

User\_logic.vhd 가 수정되면 그 상위 모듈인 my\_logic도 수정되어야 합니다.라인 141,142를 보면 2개의 포트가 그대로 추가되었고 라인 307, 308을 보면 user\_logic.vhd와 포트 맵이 되어 있습니다.

>>>>>>>>>>>

113

114 entity mylogic is

137 port

138 (

139 -- ADD USER PORTS BELOW THIS LINE ------------------

140 --USER ports added here

141 my\_in : in std\_logic\_vector(31 downto 0);

142 my\_out : out std\_logic\_vector(31 downto 0);

143 -- ADD USER PORTS ABOVE THIS LINE ------------------

144

145 -- DO NOT EDIT BELOW THIS LINE ---------------------

146 -- Bus protocol ports, do not add to or delete

147 S\_AXI\_ACLK : in std\_logic;

148 S\_AXI\_ARESETN : in std\_logic;

149 S\_AXI\_AWADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto

>>>>>>>>>>>>>

288 );

289

290 ------------------------------------------

291 -- instantiate User Logic

292 ------------------------------------------

293 USER\_LOGIC\_I : entity mylogic\_v1\_00\_a.user\_logic

294 generic map

295 (

296 -- MAP USER GENERICS BELOW THIS LINE ---------------

297 --USER generics mapped here

298 -- MAP USER GENERICS ABOVE THIS LINE ---------------

299

300 C\_NUM\_REG => USER\_NUM\_REG,

301 C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH

302 )

303 port map

304 (

305 -- MAP USER PORTS BELOW THIS LINE ------------------

306 --USER ports mapped here

307 my\_in => my\_in,

308 my\_out => my\_out,

309 -- MAP USER PORTS ABOVE THIS LINE ------------------

310

311 Bus2IP\_Clk => ipif\_Bus2IP\_Clk,

312 Bus2IP\_Resetn => ipif\_Bus2IP\_Resetn,

313 Bus2IP\_Data => ipif\_Bus2IP\_Data,

314 Bus2IP\_BE => ipif\_Bus2IP\_BE,

315 Bus2IP\_RdCE => user\_Bus2IP\_RdCE,

316 Bus2IP\_WrCE => user\_Bus2IP\_WrCE,

317 IP2Bus\_Data => user\_IP2Bus\_Data,

318 IP2Bus\_RdAck => user\_IP2Bus\_RdAck,

319 IP2Bus\_WrAck => user\_IP2Bus\_WrAck,

320 IP2Bus\_Error => user\_IP2Bus\_Error

321 );

322

### MPD 파일 수정

MPD 파일에서 라인 39, 40을 보면 2개의 포트가 추가되었습니다.

즉 이 부분은 나중에 MHS 파일에서 설계자가 어떻게 사용하는지 결정해야 합니다.

1 ###################################################################

2 ##

3 ## Name : mylogic

4 ## Desc : Microprocessor Peripheral Description

5 ## : Automatically generated by PsfUtility

6 ##

7 ###################################################################

8

9 BEGIN mylogic

10

>>>>>>>>

37

38 ## Ports

39 PORT my\_in = "", DIR = I, VEC = [31:0]

40 PORT my\_out = "", DIR = O, VEC = [31:0]

>>>>>>>>>

61 END

### PAO 파일 수정

Mylogic.vhd와 user\_logic.vhd만 수정을 했고 새로 추가한 파일이 없어서 따로 수정하지 않습니다.

##############################################################################

## Filename: C:/work/2013/v144\_sp605\_isim\_axi4lite/system/pcores/mylogic\_v1\_00\_a/data/mylogic\_v2\_1\_0.pao

## Description: Peripheral Analysis Order

## Date: Sat Mar 23 14:25:38 2013 (by Create and Import Peripheral Wizard)

##############################################################################

lib proc\_common\_v3\_00\_a all

lib axi\_lite\_ipif\_v1\_01\_a all

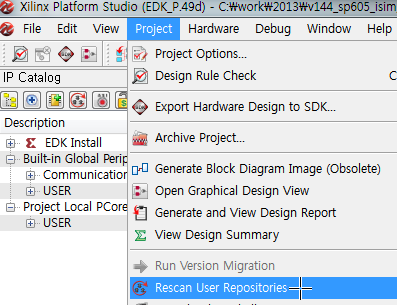
lib mylogic\_v1\_00\_a user\_logic vhdl

lib mylogic\_v1\_00\_a mylogic vhdl

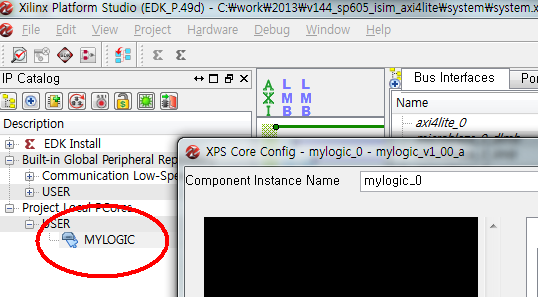
## 추가된 IP 억세스 하기

### XPS에서 IP 추가하기

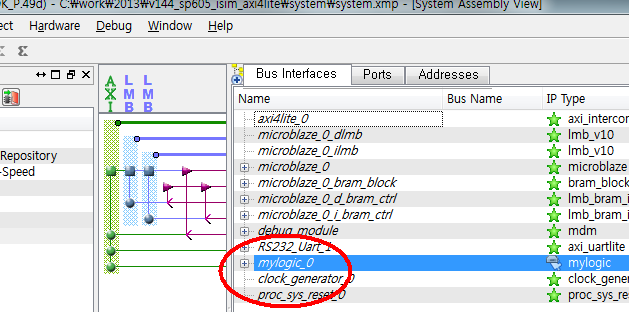
새롭게 추가된 포트들을 반영하기 위해서 Rescan을 합니다.



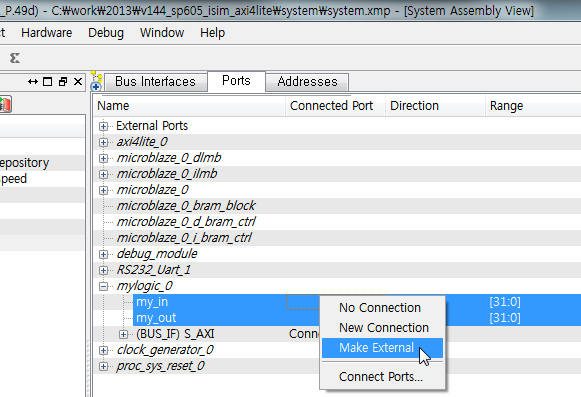
추가된 IP를 Drag & Drop을 하고 기본 값을 변경하기 않고 OK를 선택 합니다.



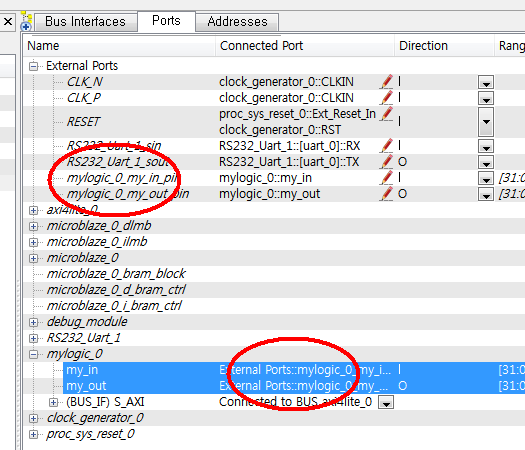
버스에 mylogic이 mylogic\_0이라는 이름으로 추가되었습니다.



mylogic에 추가된 2개의 포트를 exteral port로 연결 합니다.

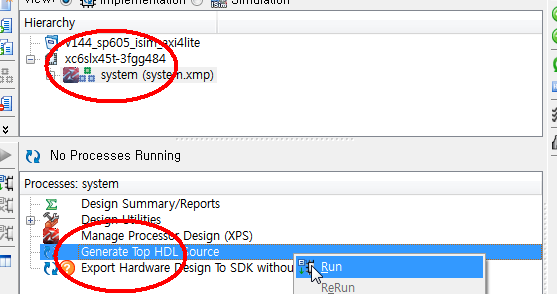


2개 포트가 추가되었습니다.



### ISE에서 top module, test bech 추가

MHS파일에서 external port 가 수정되었으면 새로운 top file이 필요 하기 때문에 다음 그림 처럼 새로 top file을 만들어 줍니다.



다음 리스트는 새로 만들어진 top file 입니다.

라인 17,18과 31,32,48,49 추가된 포트들이 어떻게 처리되는지 알 수 있습니다.

1 -------------------------------------------------------------------------------

2 -- system\_top.vhd

3 -------------------------------------------------------------------------------

4 library IEEE;

5 use IEEE.STD\_LOGIC\_1164.ALL;

6

7 library UNISIM;

8 use UNISIM.VCOMPONENTS.ALL;

9

10 entity system\_top is

11 port (

12 RS232\_Uart\_1\_sout : out std\_logic;

13 RS232\_Uart\_1\_sin : in std\_logic;

14 RESET : in std\_logic;

15 CLK\_P : in std\_logic;

16 CLK\_N : in std\_logic;

17 mylogic\_0\_my\_out\_pin : out std\_logic\_vector(31 downto 0);

18 mylogic\_0\_my\_in\_pin : in std\_logic\_vector(31 downto 0)

19 );

20 end system\_top;

21

22 architecture STRUCTURE of system\_top is

23

24 component system is

25 port (

26 RS232\_Uart\_1\_sout : out std\_logic;

27 RS232\_Uart\_1\_sin : in std\_logic;

28 RESET : in std\_logic;

29 CLK\_P : in std\_logic;

30 CLK\_N : in std\_logic;

31 mylogic\_0\_my\_out\_pin : out std\_logic\_vector(31 downto 0);

32 mylogic\_0\_my\_in\_pin : in std\_logic\_vector(31 downto 0)

33 );

34 end component;

35

36 attribute BOX\_TYPE : STRING;

37 attribute BOX\_TYPE of system : component is "user\_black\_box";

38

39 begin

40

41 system\_i : system

42 port map (

43 RS232\_Uart\_1\_sout => RS232\_Uart\_1\_sout,

44 RS232\_Uart\_1\_sin => RS232\_Uart\_1\_sin,

45 RESET => RESET,

46 CLK\_P => CLK\_P,

47 CLK\_N => CLK\_N,

48 mylogic\_0\_my\_out\_pin => mylogic\_0\_my\_out\_pin,

49 mylogic\_0\_my\_in\_pin => mylogic\_0\_my\_in\_pin

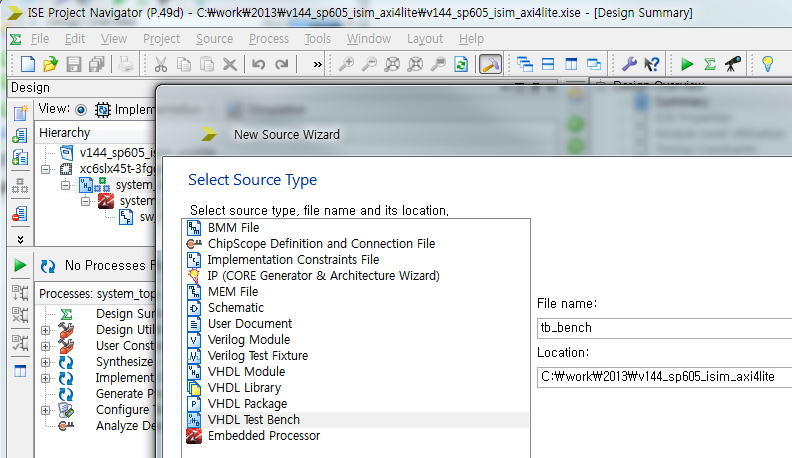
50 );

51

52 end architecture STRUCTURE;

53

Top file의 포트가 수정되었기 때문에 새로운 테스트 벤치가 필요합니다. ISE/Project/New Source를 선택한 후 테스트 벤치를 선택 합니다.



SP605 보드에 맞게끔 테스트 벤치를 수정 합니다. 라인 84~94까지 클럭, 리셋 그리고 추가된 포트에 대한 테스트 벤치를 참고 바랍니다.

>>>>>>>>>>>>>

28 LIBRARY ieee;

29 USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.ALL;

30

31 -- Uncomment the following library declaration if using

32 -- arithmetic functions with Signed or Unsigned values

33 --USE ieee.numeric\_std.ALL;

34

35 ENTITY tb\_bench IS

36 END tb\_bench;

37

38 ARCHITECTURE behavior OF tb\_bench IS

39

40 -- Component Declaration for the Unit Under Test (UUT)

41

42 COMPONENT system\_top

43 PORT(

44 RS232\_Uart\_1\_sout : OUT std\_logic;

45 RS232\_Uart\_1\_sin : IN std\_logic;

46 RESET : IN std\_logic;

47 CLK\_P : IN std\_logic;

48 CLK\_N : IN std\_logic;

49 mylogic\_0\_my\_out\_pin : OUT std\_logic\_vector(31 downto 0);

50 mylogic\_0\_my\_in\_pin : IN std\_logic\_vector(31 downto 0)

51 );

52 END COMPONENT;

53

54

55 --Inputs

56 signal RS232\_Uart\_1\_sin : std\_logic := '0';

57 signal RESET : std\_logic := '1';

58 signal CLK\_P : std\_logic := '1';

59 signal CLK\_N : std\_logic := '0';

60 signal mylogic\_0\_my\_in\_pin : std\_logic\_vector(31 downto 0) := (others => '0');

61

62 --Outputs

63 signal RS232\_Uart\_1\_sout : std\_logic;

64 signal mylogic\_0\_my\_out\_pin : std\_logic\_vector(31 downto 0);

65

66 -- Clock period definitions

67 constant CLK\_P\_period : time := 10 ns;

68 constant CLK\_N\_period : time := 10 ns;

69

70 BEGIN

71

72 -- Instantiate the Unit Under Test (UUT)

73 uut: system\_top

74 PORT MAP (

75 RS232\_Uart\_1\_sout => RS232\_Uart\_1\_sout,

76 RS232\_Uart\_1\_sin => RS232\_Uart\_1\_sin,

77 RESET => RESET,

78 CLK\_P => CLK\_P,

79 CLK\_N => CLK\_N,

80 mylogic\_0\_my\_out\_pin => mylogic\_0\_my\_out\_pin,

81 mylogic\_0\_my\_in\_pin => mylogic\_0\_my\_in\_pin

82 );

83

84 clk\_p <= not clk\_p after 2.5 ns;

85 clk\_n <= not clk\_p;

86 reset <= '0' after 100 ns;

87

88

89 process

90 begin

91 wait for 500 ns;

92 mylogic\_0\_my\_in\_pin <= mylogic\_0\_my\_in\_pin + 1;

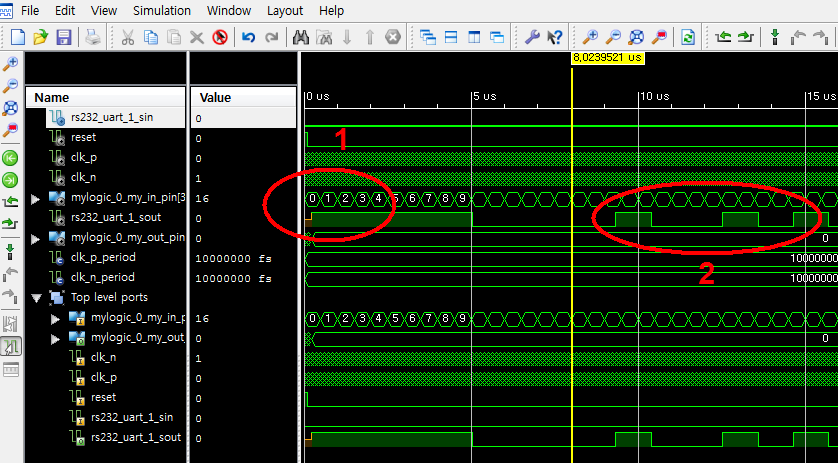
93

94 END process;

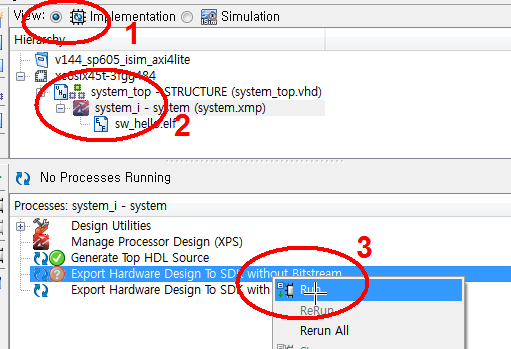
95

96 end architecture behavior;

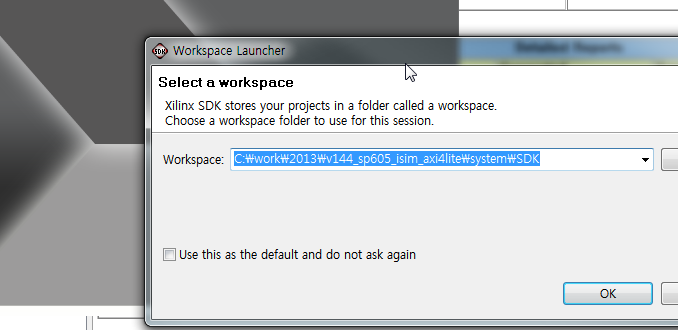
다음 그림은 시뮬레이션 결과 입니다. 1) 입력 값을 계속 변화 시키고 있고 2) 프로세서는 uart 출력 포트로 “hello world” 문자를 출력 하고 있습니다.



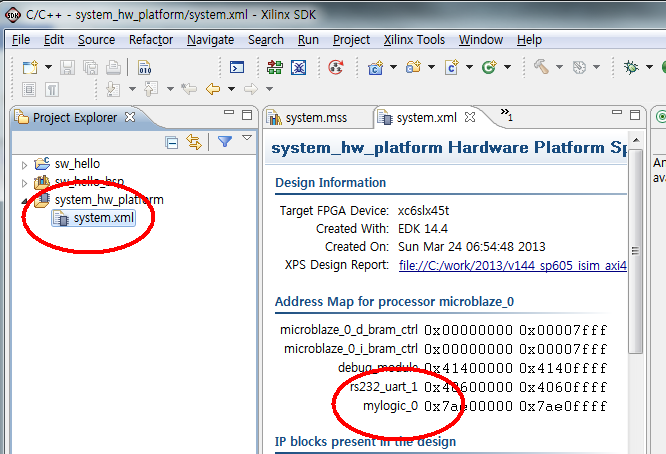
이제 추가된 로직을 읽고 쓰기 위한 프로그램을 작성해 보겠습니다. 다음 그림 처럼 메뉴를 선택해서 SDK를 실행 시킵니다.



Sdk가 실행되면 현재 프로젝트 밑에 system/sdk를 선택 합니다. 이렇게 하면 ISE는 SDK를 소프트웨어를 작성할 때 필요한 하드웨어 정보를 복사 합니다.



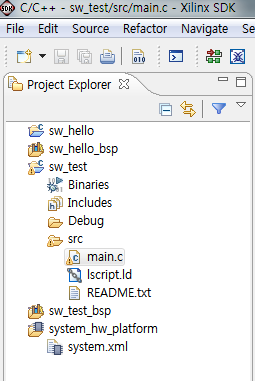
Ise가 sdk로 하드웨어 정보를 복사하는데 이 정보는 XML 이라는 파일에 저장 됩니다. 다음 그림은 조금전에 추가한 IP가 반영된 것을 보여 줍니다.



이 IP를 억세스하는 소프트웨어 코딩을 하려고 합니다.

### SDK/New application

다음 그림은 새로운 sw application인 sw\_test와 bsp가 추가되었습니다.



다음 리스트는 첫번째와 8번째 레지스터를 읽거나 쓰기 위한 코드 입니다. 0번부터 시작하니까 8번째 레지스터의 어드레스는 0x1C이 되었고 매크로 XPAR\_MYLOGIC\_0\_BASEADDR 는 xparameters.h 파일에 저장되어 있습니다.

/\*

\* main.c

\*

\* Created on: Mar 24, 2013

\* Author: hyukk

\*/

**#include** "xparameters.h"

**void** **main**()

{

**volatile** **int** \*add, \*addA, data;

**int** i = 0;

add = (**int** \*)(XPAR\_MYLOGIC\_0\_BASEADDR);

addA = (**int** \*)(XPAR\_MYLOGIC\_0\_BASEADDR + 0x1c);

**while**(1) {

data = \*add;

\*add = i++;

data = \*addA;

\*addA = i++;

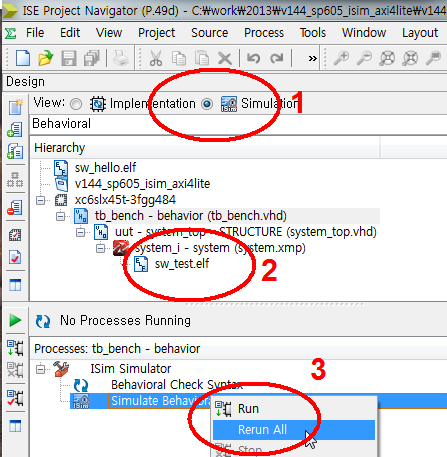
}

}

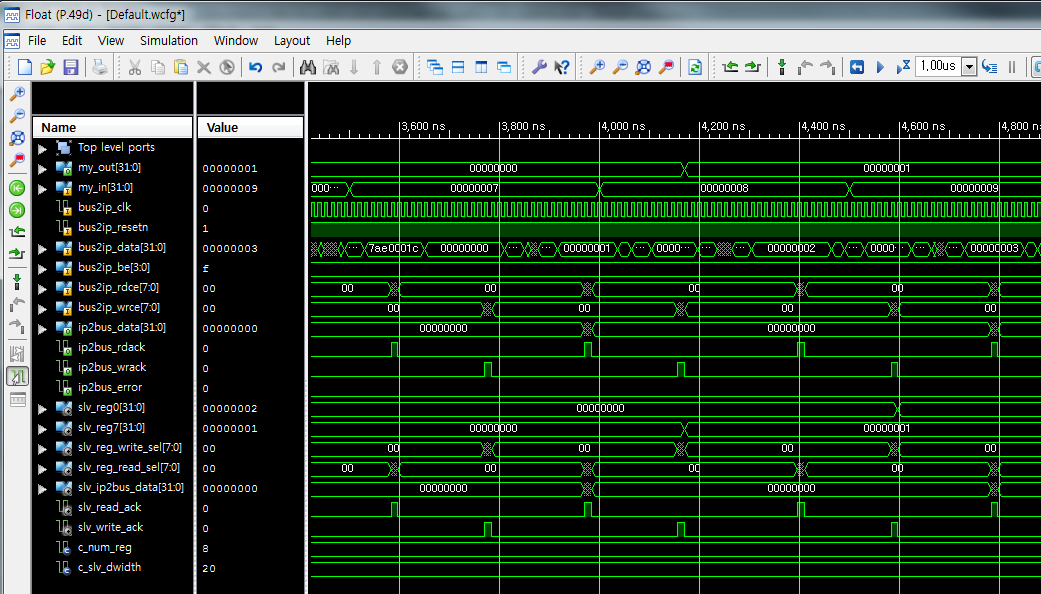
시뮬레이션을 하기 위해서는 2)처럼 sw\_test.elf 파일을 소스로 등록 시켜야 합니다.

현재 Microblaze는 FPGA 내부의 bram만을 사용하기 때문에 상관 없지만 만약 DDR 메모리 같이 external 메모리가 있는 경우에는 elf를 생성할 때 FPGA 내부의 bram을 사용하도록 링커스크립터를 수정해야 제대로 시뮬레이션이 됩니다.

시뮬레이션 라이브러리를 다시 만들어야 하기 때문에 3) 처럼 Rerun All을 선택 합니다.



다음 그림은 isim을 통해 시뮬레이션한 결과 입니다. 읽는 동작과 쓰는 동작에 의해서 ack 신호들과 my\_in, my\_out이 변하는 것을 참고 바랍니다.



## Memory Range가 있는 IP 설계

앞에서 말씀드린 ip는 8개의 레지스터만 억세스 합니다. 실제 FPGA에서 사용자 블록과 프로세서가 서로 데이터 교환하기 위해서는 8개 디코딩 공간 보다는 메모리를 억세스하는 것 처럼 어드레스 자체와 cs가 필요 합니다.

따라서 ip를 만들 때 레지스터 디코딩 공간이 아닌 메모리 디코딩 공간을 만들기 위한 방법이 필요 합니다.

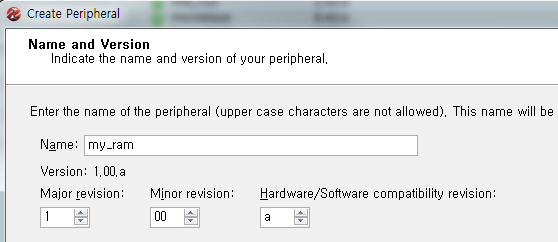
이 IP는 이미 저자가 만들어 놓은 상태이기 많은 분들이 사용하고 있지만 만드는 과정이 복잡해서 IP 형태로만 공급하고 있습니다.

하지만 앞에서 IP를 만드는 과정을 직접 경험해 보았기 때문에 이 부분을 설명하더라도 무리가 없으리라 생각 합니다.

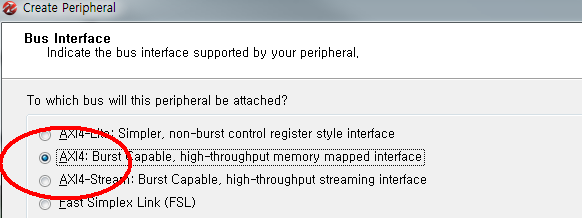
### CIP를 이용한 메모리 디코딩 IP 설계

XPS/HardWare/Creat and Import Peripheral을 선택하고 다음 그림이 나올 때 까지 Next를 선택 합니다.

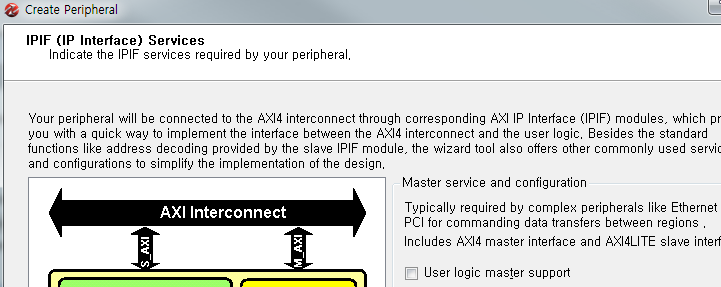
새로운 IP 이름을 정의 합니다.



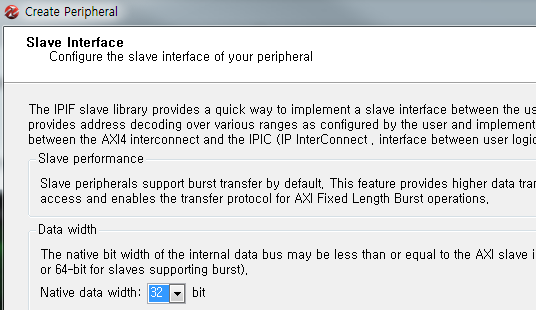
메모리 공간이 나오기 위해서는 AXI4 Burst 를 선택 합니다. 하지만 실제 인터커넥터에 연결할 때는 AXI4Lite에 연결할 겁니다. 왜냐하면 결국 프로세서가 레지스터를 억세스하는데 메모리 공간이 필요하기 때문 입니다.



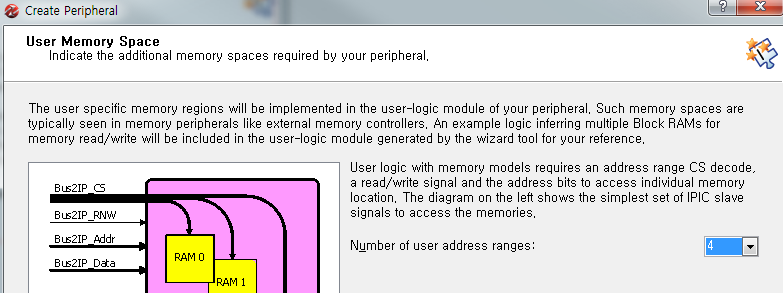
마스터 로직은 합성하지 않습니다.



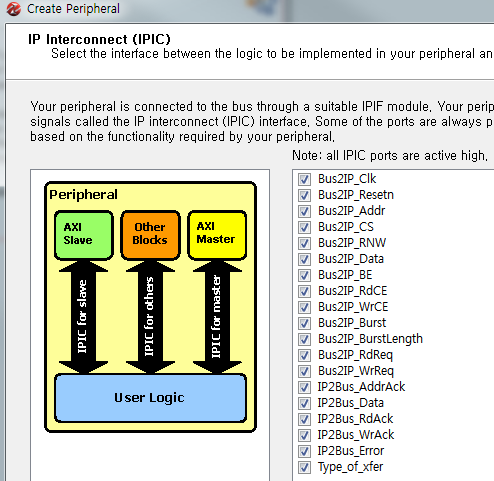
데이터 폭은 32비트로 고정 합니다.



메모리 공간이 몇 개 필요한지 선택 합니다. 이 숫자는 디코딩 숫자를 의미 하는 것이 아니고 메모리 CS 개수를 의미 합니다.



그러면 현재 사용 가능한 IPIC 신호를 보여 주고 있습니다.



### User\_logic.vhd

>>>>>>>>>

51 -- DO NOT EDIT BELOW THIS LINE --------------------

52 library ieee;

53 use ieee.std\_logic\_1164.all;

54 use ieee.std\_logic\_arith.all;

55 use ieee.std\_logic\_unsigned.all;

56

57 library proc\_common\_v3\_00\_a;

58 use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;

>>>>>>>>>>>

User logic에 대한 entity 입니다.

93

94 entity user\_logic is

>>>>>>>

108 port

109 (

110 -- ADD USER PORTS BELOW THIS LINE ------------------

111 --USER ports added here

112 -- ADD USER PORTS ABOVE THIS LINE ------------------

113

114 -- DO NOT EDIT BELOW THIS LINE ---------------------

115 -- Bus protocol ports, do not add to or delete

116 Bus2IP\_Clk : in std\_logic;

117 Bus2IP\_Resetn : in std\_logic;

라인 118, 119를 보면 어드레스와 CS가 표시되어 있습니다. C\_NUM\_MEM은 4로 정의되어 있습니다.

118 Bus2IP\_Addr : in std\_logic\_vector(C\_SLV\_AWIDTH-1 downto 0);

119 Bus2IP\_CS : in std\_logic\_vector(C\_NUM\_MEM-1 downto 0);

120 Bus2IP\_RNW : in std\_logic;

121 Bus2IP\_Data : in std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

122 Bus2IP\_BE : in std\_logic\_vector(C\_SLV\_DWIDTH/8-1 downto 0);

123 Bus2IP\_RdCE : in std\_logic\_vector(C\_NUM\_MEM-1 downto 0);

124 Bus2IP\_WrCE : in std\_logic\_vector(C\_NUM\_MEM-1 downto 0);

125 Bus2IP\_Burst : in std\_logic;

126 Bus2IP\_BurstLength : in std\_logic\_vector(7 downto 0);

127 Bus2IP\_RdReq : in std\_logic;

128 Bus2IP\_WrReq : in std\_logic;

129 IP2Bus\_AddrAck : out std\_logic;

130 IP2Bus\_Data : out std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

131 IP2Bus\_RdAck : out std\_logic;

132 IP2Bus\_WrAck : out std\_logic;

133 IP2Bus\_Error : out std\_logic;

134 Type\_of\_xfer : out std\_logic

135 -- DO NOT EDIT ABOVE THIS LINE ---------------------

136 );

137

138 attribute MAX\_FANOUT : string;

139 attribute SIGIS : string;

140

141 attribute SIGIS of Bus2IP\_Clk : signal is "CLK";

142 attribute SIGIS of Bus2IP\_Resetn : signal is "RST";

143

144 end entity user\_logic;

145

146 ------------------------------------------------------------------------------

147 -- Architecture section

148 ------------------------------------------------------------------------------

149

150 architecture IMP of user\_logic is

151

152 --USER signal declarations added here, as needed for user logic

153

라인 154~159은 메모리를 구현하는데 필요한 singal 선언 입니다. 8바이트짜리 array가 모두 256개 있고 이런 블록을 BYTE\_RAM\_TYPE으로 정했습니다.

BYTE\_RAM\_TYPE을 다시 DO\_TYPE으로 재정의 했는데 256 블록 4개가 다시 정의되었습니다.

154 ------------------------------------------

155 -- Signals for user logic memory space example

156 ------------------------------------------

157 type BYTE\_RAM\_TYPE is array (0 to 255) of std\_logic\_vector(7 downto 0);

158 type DO\_TYPE is array (0 to C\_NUM\_MEM-1) of std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

159 signal mem\_data\_out : DO\_TYPE;

160 signal mem\_address : std\_logic\_vector(7 downto 0);

161 signal mem\_select : std\_logic\_vector(0 to 3);

162 signal mem\_read\_enable : std\_logic;

163 signal mem\_ip2bus\_data : std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

164 signal mem\_read\_ack\_dly1 : std\_logic;

165 signal mem\_read\_ack\_dly2 : std\_logic;

166 signal mem\_read\_ack : std\_logic;

167 signal mem\_write\_ack : std\_logic;

168

169 begin

170

171 --USER logic implementation added here

172

173 ------------------------------------------

174 -- Example code to access user logic memory region

175 --

176 -- Note:

177 -- The example code presented here is to show you one way of using

178 -- the user logic memory space features. The Bus2IP\_Addr, Bus2IP\_CS,

179 -- and Bus2IP\_RNW IPIC signals are dedicated to these user logic

180 -- memory spaces. Each user logic memory space has its own address

181 -- range and is allocated one bit on the Bus2IP\_CS signal to indicated

182 -- selection of that memory space. Typically these user logic memory

183 -- spaces are used to implement memory controller type cores, but it

184 -- can also be used in cores that need to access additional address space

185 -- (non C\_BASEADDR based), s.t. bridges. This code snippet infers

186 -- 4 256x32-bit (byte accessible) single-port Block RAM by XST.

187 ------------------------------------------

메모리 선택하고 읽는 위한 신호와 쓰기 위한 신호가 표현 되어 있습니다. 특히 라인 194~208 까지는 ack를 만들기 위해 cs 신호를 조작하고 펄스를 하나 만들어서 read/write 동작할 때 active 시간을 조정하고 있습니다.

이렇게 조정하는 이유는 슬레이브 마다 억세스 하는 시간이 다르기 때문인데 라인 191을 보면 mem\_write\_ack는 쓰기 동작시 cs가 바로 ack로 되돌아 가지만 라인 190에서는 두 싸이클 뒤에 ack가 돌아가도록 되어 있습니다.

이 코드는 참고용 코드 이므로 ack를 언제 되돌려야 하는지는 개발자가 정해서 만들어도 상관이 없지만 한가지 유의할 것은 다음과 같습니다.

기본적으로 인터커넥터는 ack 신호가 8 클럭내에 돌아오지 않으면 에러 처리를 합니다.

188 mem\_select <= Bus2IP\_CS;

189 mem\_read\_enable <= ( Bus2IP\_RdCE(0) or Bus2IP\_RdCE(1) or Bus2IP\_RdCE(2) or Bus2IP\_RdCE(3) );

190 mem\_read\_ack <= mem\_read\_ack\_dly1 and (not mem\_read\_ack\_dly2);

191 mem\_write\_ack <= ( Bus2IP\_WrCE(0) or Bus2IP\_WrCE(1) or Bus2IP\_WrCE(2) or Bus2IP\_WrCE(3) );

192 mem\_address <= Bus2IP\_Addr(9 downto 2);

193

194 -- this process generates the read acknowledge 1 clock after read enable

195 -- is presented to the BRAM block. The BRAM block has a 1 clock delay

196 -- from read enable to data out.

197 BRAM\_RD\_ACK\_PROC : process( Bus2IP\_Clk ) is

198 begin

199

200 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

201 if ( Bus2IP\_Resetn = '0' ) then

202 mem\_read\_ack\_dly1 <= '0';

203 mem\_read\_ack\_dly2 <= '0';

204 else

205 mem\_read\_ack\_dly1 <= mem\_read\_enable;

206 mem\_read\_ack\_dly2 <= mem\_read\_ack\_dly1;

207 end if;

208 end if;

209

210 end process BRAM\_RD\_ACK\_PROC;

211

라인 212~246까지는 앞에서 정의한 메모리 구조체를 실제로 합성하는 과정을 보여 줍니다. 라인 213에서 4개의 메모리 구조체를 반복적으로 구현한다는 것을 보여 줍니다.

212 -- implement Block RAM(s)

213 BRAM\_GEN : for i in 0 to C\_NUM\_MEM-1 generate

214 constant NUM\_BYTE\_LANES : integer := (C\_SLV\_DWIDTH+7)/8;

215 begin

216

256바이트를 가지는 메모리에 대해서 읽는 동작과 쓰는 동작에 다른 회로를 구성 합니다.

217 BYTE\_BRAM\_GEN : for byte\_index in 0 to NUM\_BYTE\_LANES-1 generate

218 signal ram : BYTE\_RAM\_TYPE;

219 signal write\_enable : std\_logic;

220 signal data\_in : std\_logic\_vector(7 downto 0);

221 signal data\_out : std\_logic\_vector(7 downto 0);

222 signal read\_address : std\_logic\_vector(7 downto 0);

223 begin

224

225 write\_enable <= Bus2IP\_WrCE(i) and Bus2IP\_BE(byte\_index);

226

227 data\_in <= Bus2IP\_Data(byte\_index\*8+7 downto byte\_index\*8);

특히 라인 228에 있는 process는 클럭에 동기는 메모리를 구현한다는 것을 의미하고 라인 232, 235가 process내부에 구현되어 있기 때문에 읽는 동작에선 1 클럭 레이턴시가 발생 합니다.

이 블록은 FPGA 내에 있는 BRAM의 동작 특성과 똑깥기 때문에 이 코드는 BRAM으로 합성 됩니다.

228 BYTE\_RAM\_PROC : process( Bus2IP\_Clk ) is

229 begin

230

231 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

232 if ( write\_enable = '1' ) then

233 ram(CONV\_INTEGER(mem\_address)) <= data\_in;

234 end if;

235 read\_address <= mem\_address;

236 end if;

237

238 end process BYTE\_RAM\_PROC;

239

240 data\_out <= ram(CONV\_INTEGER(read\_address));

241

242 mem\_data\_out(i)(byte\_index\*8+7 downto byte\_index\*8) <= data\_out;

243

244 end generate BYTE\_BRAM\_GEN;

245

246 end generate BRAM\_GEN;

4개의 메모리 블록으로부터 나오는 출력을 IPIC에 연결해 줍니다.

247

248 -- implement Block RAM read mux

249 MEM\_IP2BUS\_DATA\_PROC : process( mem\_data\_out, mem\_select ) is

250 begin

251

252 case mem\_select is

253 when "0001" => mem\_ip2bus\_data <= mem\_data\_out(0);

254 when "0010" => mem\_ip2bus\_data <= mem\_data\_out(1);

255 when "0100" => mem\_ip2bus\_data <= mem\_data\_out(2);

256 when "1000" => mem\_ip2bus\_data <= mem\_data\_out(3);

257 when others => mem\_ip2bus\_data <= (others => '0');

258 end case;

259

260 end process MEM\_IP2BUS\_DATA\_PROC;

261

메모리를 읽을 때만 메모리 값을 IPIC에 돌려주고 나머지는 zero 상태로 만듭니다.

262 ------------------------------------------

263 -- Example code to drive IP to Bus signals

264 ------------------------------------------

265 IP2Bus\_Data <= mem\_ip2bus\_data when mem\_read\_ack = '1' else

266 (others => '0');

267

AddrAck, RdAck, WrAck 를 되돌려 줍니다.

268 IP2Bus\_AddrAck <= mem\_write\_ack or (mem\_read\_enable and mem\_read\_ack);

269 IP2Bus\_WrAck <= mem\_write\_ack;

270 IP2Bus\_RdAck <= mem\_read\_ack;

271 IP2Bus\_Error <= '0';

272

273 end IMP;

### AXI\_USER\_LOGIC\_32x8192\_4CS

앞에서 말씀 드린대로 메모리 공간을 가지고 user logic을 인터페이스하기 위한 IP는 이미 저자가 만들어서 많은 분들이 사용하고 있습니다.

다음 그림은 axi\_user\_logic\_32x8192\_4cs\_v1\_00\_b 와 user\_logic.vhd 의 관계를 보여 줍니다.



### AXI\_USER\_LOGIC\_32x8192\_4CS, User\_Logic.VHD

기본적으로 만들어지는 user\_logic.vhd에서 달라지는 부분만 설명 합니다.

>>>>

50

51 -- DO NOT EDIT BELOW THIS LINE --------------------

52 library ieee;

53 use ieee.std\_logic\_1164.all;

54 use ieee.std\_logic\_arith.all;

55 use ieee.std\_logic\_unsigned.all;

56

57 library proc\_common\_v3\_00\_a;

58 use proc\_common\_v3\_00\_a.proc\_common\_pkg.all;

59

>>>>>

93

94 entity user\_logic is

>>>>

라인 112~121에 새로운 포트를 추가 했습니다. 4개의 CS가 있기 때문에 읽는 동작에서는 data\_from\_user0, 1, 2,3 을 이용하고 쓰는 동작에서는 data\_to\_user라는 포트를 사용 합니다.

라인 117에는 메모리 공간을 확보하기 위한 어드레스 버스와 라인 121에는 4개 메모리 cs가 있습니다.

108 port

109 (

110 -- ADD USER PORTS BELOW THIS LINE ------------------

111 --USER ports added here

112 data\_from\_user0 : in std\_logic\_vector(31 downto 0);

113 data\_from\_user1 : in std\_logic\_vector(31 downto 0);

114 data\_from\_user2 : in std\_logic\_vector(31 downto 0);

115 data\_from\_user3 : in std\_logic\_vector(31 downto 0);

116 data\_to\_user : out std\_logic\_vector(31 downto 0);

117 user\_add : out std\_logic\_vector(11 downto 0);

118 user\_rd : out std\_logic;

119 user\_wr : out std\_logic;

120 user\_clk : out std\_logic;

121 user\_cs : out std\_logic\_vector(3 downto 0);

122 -- ADD USER PORTS ABOVE THIS LINE ------------------

123 >>>>>

144 Type\_of\_xfer : out std\_logic

145 -- DO NOT EDIT ABOVE THIS LINE ---------------------

146 );

147

148 attribute MAX\_FANOUT : string;

149 attribute SIGIS : string;

150

151 attribute SIGIS of Bus2IP\_Clk : signal is "CLK";

152 attribute SIGIS of Bus2IP\_Resetn : signal is "RST";

153

154 end entity user\_logic;

155

156 ------------------------------------------------------------------------------

157 -- Architecture section

158 ------------------------------------------------------------------------------

159

160 architecture IMP of user\_logic is

161

>>>>>>>

173 begin

IPIC에서 나오는 여러 버스 신호를 이용해 포트를 재정의 합니다.

174 user\_cs <= Bus2IP\_CS;

175 user\_rd <= mem\_read\_enable;

176 user\_wr <= mem\_write\_enable;

177 user\_clk <= Bus2IP\_Clk;

178 data\_to\_user <= Bus2IP\_Data;

179 user\_add <= Bus2IP\_Addr(13 downto 2);

180

181 mem\_select <= Bus2IP\_CS;

182 mem\_read\_enable <= (Bus2IP\_RdCE(0) or Bus2IP\_RdCE(1) or Bus2IP\_RdCE(2) or Bus2IP\_RdCE(3));

183 mem\_write\_enable<= (Bus2IP\_WrCE(0) or Bus2IP\_WrCE(1) or Bus2IP\_WrCE(2) or Bus2IP\_WrCE(3));

184 mem\_read\_ack <= mem\_read\_ack\_dly(0);

185 mem\_write\_ack <= mem\_write\_ack\_dly(0);

186

라인 187~202은 읽기/쓰기 동작에 따른 ack를 만들기 위해서 일단 펄스를 하나 만드는 과정을 보여 줍니다.

187 make\_read\_write\_pulse : process( Bus2IP\_Clk ) is

188 begin

189 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

190 if ( Bus2IP\_Resetn = '0' ) then

191 mem\_read\_enable\_dly1 <= '0';

192 mem\_write\_enable\_dly1 <= '0';

193 else

194 mem\_read\_enable\_dly1 <= mem\_read\_enable;

195 mem\_write\_enable\_dly1 <= mem\_write\_enable;

196 end if;

197 end if;

198 end process make\_read\_write\_pulse;

199

200

201 mem\_read\_req <= mem\_read\_enable and not(mem\_read\_enable\_dly1);

202 mem\_write\_req <= mem\_write\_enable and not(mem\_write\_enable\_dly1);

203

204

딜레이 로직을 사용해 ack를 돌려줍니다.

따라서 버스는 모두 10 클럭동안 유지 됩니다.

그러면 앞에서 언급한 대로 8개 버스 클럭이 넘으면 에러가 난다는 것을 알고 있기 때문에 이 부분은 특별히 하나의 파라미터를 수정해야 합니다. 이 부분은 MPD 파일을 설명할 때 말씀 드리겠습니다.

205 BRAM\_RD\_ACK\_PROC : process( Bus2IP\_Clk ) is

206 begin

207 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

208 if ( Bus2IP\_Resetn = '0' ) then

209 mem\_read\_ack\_dly <= (others => '0');

210 mem\_write\_ack\_dly <= (others => '0');

211 else

212 mem\_read\_ack\_dly(9) <= mem\_read\_req;

213 mem\_read\_ack\_dly(8) <= mem\_read\_ack\_dly(9);

214 mem\_read\_ack\_dly(7) <= mem\_read\_ack\_dly(8);

215 mem\_read\_ack\_dly(6) <= mem\_read\_ack\_dly(7);

216 mem\_read\_ack\_dly(5) <= mem\_read\_ack\_dly(6);

217 mem\_read\_ack\_dly(4) <= mem\_read\_ack\_dly(5);

218 mem\_read\_ack\_dly(3) <= mem\_read\_ack\_dly(4);

219 mem\_read\_ack\_dly(2) <= mem\_read\_ack\_dly(3);

220 mem\_read\_ack\_dly(1) <= mem\_read\_ack\_dly(2);

221 mem\_read\_ack\_dly(0) <= mem\_read\_ack\_dly(1);

222

223 mem\_write\_ack\_dly(9) <= mem\_write\_req;

224 mem\_write\_ack\_dly(8) <= mem\_write\_ack\_dly(9);

225 mem\_write\_ack\_dly(7) <= mem\_write\_ack\_dly(8);

226 mem\_write\_ack\_dly(6) <= mem\_write\_ack\_dly(7);

227 mem\_write\_ack\_dly(5) <= mem\_write\_ack\_dly(6);

228 mem\_write\_ack\_dly(4) <= mem\_write\_ack\_dly(5);

229 mem\_write\_ack\_dly(3) <= mem\_write\_ack\_dly(4);

230 mem\_write\_ack\_dly(2) <= mem\_write\_ack\_dly(3);

231 mem\_write\_ack\_dly(1) <= mem\_write\_ack\_dly(2);

232 mem\_write\_ack\_dly(0) <= mem\_write\_ack\_dly(1);

233 end if;

234 end if;

235

236 end process BRAM\_RD\_ACK\_PROC;

237

메모리로부터 읽어서 IPIC로 돌려주는 부분 입니다.

238 -- implement Block RAM read mux

239 MEM\_IP2BUS\_DATA\_PROC : process( data\_from\_user0, data\_from\_user1, data\_from\_user2, data\_from\_user3, mem\_select ) is

240 begin

241 case mem\_select is

242 when "0001" => mem\_ip2bus\_data <= data\_from\_user0;

243 when "0010" => mem\_ip2bus\_data <= data\_from\_user1;

244 when "0100" => mem\_ip2bus\_data <= data\_from\_user2;

245 when "1000" => mem\_ip2bus\_data <= data\_from\_user3;

246 when others => mem\_ip2bus\_data <= (others => '0');

247 end case;

248

249 end process MEM\_IP2BUS\_DATA\_PROC;

250

>>>>>>

261 end IMP;

### MPD 파일 수정

1 ###################################################################

2 ##

3 ## Name : axi\_user\_logic\_32x8192\_4cs

4 ## Desc : Microprocessor Peripheral Description

5 ## : Automatically generated by PsfUtility

6 ##

7 ###################################################################

8

9 BEGIN axi\_user\_logic\_32x8192\_4cs

10

11 ## Peripheral Options

12 OPTION IPTYPE = PERIPHERAL

13 OPTION IMP\_NETLIST = TRUE

14 OPTION HDL = VHDL

15 OPTION IP\_GROUP = MICROBLAZE:USER

16 OPTION DESC = AXI\_USER\_LOGIC\_32X8192\_4CS

17 OPTION ARCH\_SUPPORT\_MAP = (others=DEVELOPMENT)

18

19

20 ## Bus Interfaces

21 BUS\_INTERFACE BUS = S\_AXI, BUS\_STD = AXI, BUS\_TYPE = SLAVE

22

23 ## Generics for VHDL or Parameters for Verilog

24 PARAMETER C\_S\_AXI\_DATA\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

25 PARAMETER C\_S\_AXI\_ADDR\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

26 PARAMETER C\_S\_AXI\_ID\_WIDTH = 4, DT = INTEGER, BUS = S\_AXI

27 PARAMETER C\_RDATA\_FIFO\_DEPTH = 0, DT = INTEGER

28 PARAMETER C\_INCLUDE\_TIMEOUT\_CNT = 1, DT = INTEGER

라인 29에서 기본적으로 최대 8 클럭동안만 유지하도록 되어 있지만 이 부분을 12클럭을 바꿨습니다.

29 PARAMETER C\_TIMEOUT\_CNTR\_VAL = 12, DT = INTEGER

30 PARAMETER C\_ALIGN\_BE\_RDADDR = 0, DT = INTEGER

31 PARAMETER C\_S\_AXI\_SUPPORTS\_WRITE = 1, DT = INTEGER, BUS = S\_AXI

32 PARAMETER C\_S\_AXI\_SUPPORTS\_READ = 1, DT = INTEGER, BUS = S\_AXI

33 PARAMETER C\_FAMILY = virtex6, DT = STRING

34 PARAMETER C\_S\_AXI\_MEM0\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, CACHEABLE = TRUE, PAIR = C\_S\_AXI\_MEM0\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

35 PARAMETER C\_S\_AXI\_MEM0\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM0\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

36 PARAMETER C\_S\_AXI\_MEM1\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, CACHEABLE = TRUE, PAIR = C\_S\_AXI\_MEM1\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

37 PARAMETER C\_S\_AXI\_MEM1\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM1\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

38 PARAMETER C\_S\_AXI\_MEM2\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, CACHEABLE = TRUE, PAIR = C\_S\_AXI\_MEM2\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

39 PARAMETER C\_S\_AXI\_MEM2\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM2\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

40 PARAMETER C\_S\_AXI\_MEM3\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, CACHEABLE = TRUE, PAIR = C\_S\_AXI\_MEM3\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

41 PARAMETER C\_S\_AXI\_MEM3\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM3\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

42 PARAMETER C\_S\_AXI\_PROTOCOL = AXI4, TYPE = NON\_HDL, ASSIGNMENT = CONSTANT, DT = STRING, BUS = S\_AXI

43

라인 45~54까지 유저 로직에서 사용할 수 있도록 필요한 신호를 준비했습니다.

44 ## Ports

45 PORT data\_from\_user0 = "", DIR = I, VEC = [31:0]

46 PORT data\_from\_user1 = "", DIR = I, VEC = [31:0]

47 PORT data\_from\_user2 = "", DIR = I, VEC = [31:0]

48 PORT data\_from\_user3 = "", DIR = I, VEC = [31:0]

49 PORT data\_to\_user = "", DIR = O, VEC = [31:0]

50 PORT user\_cs = "", DIR = O, VEC = [3:0]

51 PORT user\_rd = "", DIR = O

52 PORT user\_wr = "", DIR = O

53 PORT user\_clk = "", DIR = O

54 PORT user\_add = "", DIR = O, VEC = [11:0]

55

56 PORT S\_AXI\_ACLK = "", DIR = I, SIGIS = CLK, BUS = S\_AXI

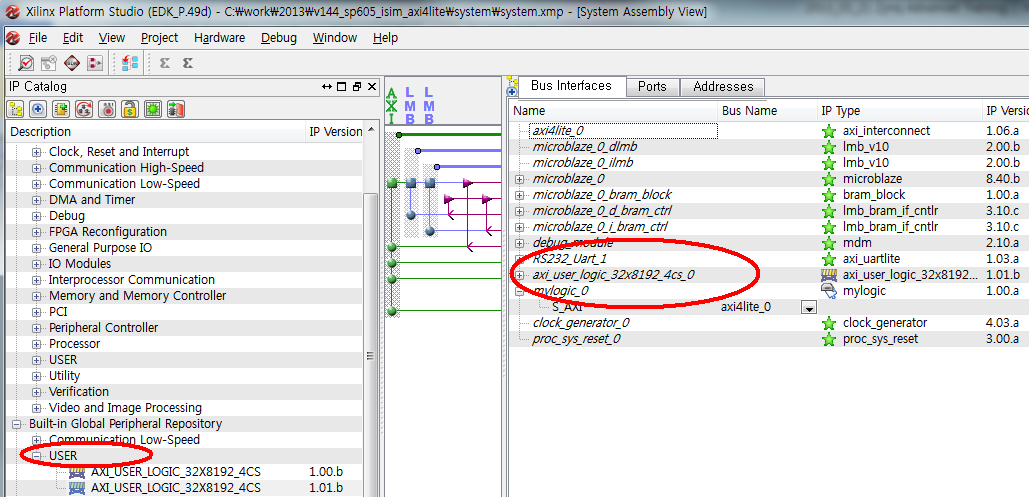
>>>>>>

92 PORT S\_AXI\_RLAST = RLAST, DIR = O, BUS = S\_AXI

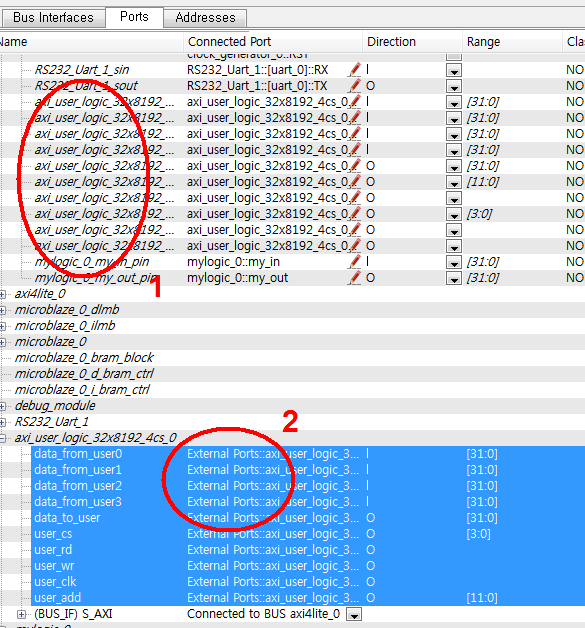
93

94 END

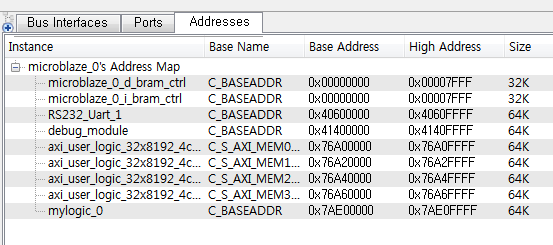
### 버스 연결



### 포트 연결



### 어드레스 할당



### 수정된 MHS FILE

PORT RS232\_Uart\_1\_sout = RS232\_Uart\_1\_sout, DIR = O

PORT RS232\_Uart\_1\_sin = RS232\_Uart\_1\_sin, DIR = I

PORT RESET = RESET, DIR = I, SIGIS = RST, RST\_POLARITY = 1

PORT CLK\_P = CLK, DIR = I, DIFFERENTIAL\_POLARITY = P, SIGIS = CLK, CLK\_FREQ = 200000000

PORT CLK\_N = CLK, DIR = I, DIFFERENTIAL\_POLARITY = N, SIGIS = CLK, CLK\_FREQ = 200000000

PORT mylogic\_0\_my\_out\_pin = mylogic\_0\_my\_out, DIR = O, VEC = [31:0]

PORT mylogic\_0\_my\_in\_pin = mylogic\_0\_my\_in, DIR = I, VEC = [31:0]

PORT user\_add = axi\_user\_logic\_32x8192\_4cs\_0\_user\_add, DIR = O, VEC = [11:0]

PORT user\_wr = axi\_user\_logic\_32x8192\_4cs\_0\_user\_wr, DIR = O

PORT user\_clk = axi\_user\_logic\_32x8192\_4cs\_0\_user\_clk, DIR = O

PORT data\_from\_user2 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user2, DIR = I, VEC = [31:0]

PORT user\_cs = axi\_user\_logic\_32x8192\_4cs\_0\_user\_cs, DIR = O, VEC = [3:0]

PORT user\_rd = axi\_user\_logic\_32x8192\_4cs\_0\_user\_rd, DIR = O

PORT data\_from\_user0 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user0, DIR = I, VEC = [31:0]

PORT data\_from\_user1 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user1, DIR = I, VEC = [31:0]

PORT data\_from\_user3 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user3, DIR = I, VEC = [31:0]

PORT data\_to\_user = axi\_user\_logic\_32x8192\_4cs\_0\_data\_to\_user, DIR = O, VEC = [31:0]

BEGIN axi\_user\_logic\_32x8192\_4cs

PARAMETER INSTANCE = axi\_user\_logic\_32x8192\_4cs\_0

PARAMETER HW\_VER = 1.01.b

PARAMETER C\_S\_AXI\_MEM0\_BASEADDR = 0x76a00000

PARAMETER C\_S\_AXI\_MEM0\_HIGHADDR = 0x76a0ffff

PARAMETER C\_S\_AXI\_MEM1\_BASEADDR = 0x76a20000

PARAMETER C\_S\_AXI\_MEM1\_HIGHADDR = 0x76a2ffff

PARAMETER C\_S\_AXI\_MEM2\_BASEADDR = 0x76a40000

PARAMETER C\_S\_AXI\_MEM2\_HIGHADDR = 0x76a4ffff

PARAMETER C\_S\_AXI\_MEM3\_BASEADDR = 0x76a60000

PARAMETER C\_S\_AXI\_MEM3\_HIGHADDR = 0x76a6ffff

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = clk\_75\_0000MHz

PORT user\_add = axi\_user\_logic\_32x8192\_4cs\_0\_user\_add

PORT user\_wr = axi\_user\_logic\_32x8192\_4cs\_0\_user\_wr

PORT user\_clk = axi\_user\_logic\_32x8192\_4cs\_0\_user\_clk

PORT data\_from\_user2 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user2

PORT user\_cs = axi\_user\_logic\_32x8192\_4cs\_0\_user\_cs

PORT user\_rd = axi\_user\_logic\_32x8192\_4cs\_0\_user\_rd

PORT data\_from\_user0 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user0

PORT data\_from\_user1 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user1

PORT data\_from\_user3 = axi\_user\_logic\_32x8192\_4cs\_0\_data\_from\_user3

PORT data\_to\_user = axi\_user\_logic\_32x8192\_4cs\_0\_data\_to\_user

END

### 수정된 top file

1 -------------------------------------------------------------------------------

2 -- system\_top.vhd

3 -------------------------------------------------------------------------------

4 library IEEE;

5 use IEEE.STD\_LOGIC\_1164.ALL;

6

7 library UNISIM;

8 use UNISIM.VCOMPONENTS.ALL;

9

10 entity system\_top is

11 port (

12 RS232\_Uart\_1\_sout : out std\_logic;

13 RS232\_Uart\_1\_sin : in std\_logic;

14 RESET : in std\_logic;

15 CLK\_P : in std\_logic;

16 CLK\_N : in std\_logic;

17 mylogic\_0\_my\_out\_pin : out std\_logic\_vector(31 downto 0);

18 mylogic\_0\_my\_in\_pin : in std\_logic\_vector(31 downto 0)

19 );

20 end system\_top;

21

22 architecture STRUCTURE of system\_top is

23

24 component system is

25 port (

26 RS232\_Uart\_1\_sout : out std\_logic;

27 RS232\_Uart\_1\_sin : in std\_logic;

28 RESET : in std\_logic;

29 CLK\_P : in std\_logic;

30 CLK\_N : in std\_logic;

31 mylogic\_0\_my\_out\_pin : out std\_logic\_vector(31 downto 0);

32 mylogic\_0\_my\_in\_pin : in std\_logic\_vector(31 downto 0);

33 user\_add : out std\_logic\_vector(11 downto 0);

34 user\_wr : out std\_logic;

35 user\_clk : out std\_logic;

36 data\_from\_user2 : in std\_logic\_vector(31 downto 0);

37 user\_cs : out std\_logic\_vector(3 downto 0);

38 user\_rd : out std\_logic;

39 data\_from\_user0 : in std\_logic\_vector(31 downto 0);

40 data\_from\_user1 : in std\_logic\_vector(31 downto 0);

41 data\_from\_user3 : in std\_logic\_vector(31 downto 0);

42 data\_to\_user : out std\_logic\_vector(31 downto 0)

43 );

44 end component;

45

46 attribute BOX\_TYPE : STRING;

47 attribute BOX\_TYPE of system : component is "user\_black\_box";

48

49 signal user\_rd, user\_wr, clk : std\_logic;

50 signal reg0, reg1, gnd32, data\_from\_user0, data\_to\_user : std\_logic\_vector(31 downto 0);

51 signal user\_cs : std\_logic\_vector(3 downto 0);

52 signal user\_add : std\_logic\_vector(11 downto 0);

53

54 begin

55

56 system\_i : system

57 port map (

58 RS232\_Uart\_1\_sout => RS232\_Uart\_1\_sout,

59 RS232\_Uart\_1\_sin => RS232\_Uart\_1\_sin,

60 RESET => RESET,

61 CLK\_P => CLK\_P,

62 CLK\_N => CLK\_N,

63 mylogic\_0\_my\_out\_pin => mylogic\_0\_my\_out\_pin,

64 mylogic\_0\_my\_in\_pin => mylogic\_0\_my\_in\_pin,

65 user\_add => user\_add,

66 user\_wr => user\_wr,

67 user\_clk => clk,

68 data\_from\_user2 => gnd32,

69 user\_cs => user\_cs,

70 user\_rd => user\_rd,

71 data\_from\_user0 => data\_from\_user0,

72 data\_from\_user1 => gnd32,

73 data\_from\_user3 => gnd32,

74 data\_to\_user => data\_to\_user

75 );

76

77 process(clk)

78 begin

79 if clk'event and clk = '1' then

80 if user\_cs(0) = '1' then

81 if user\_rd = '1' then

82 case user\_add is

83 when x"000" => data\_from\_user0 <= reg0;

84 when x"001" => data\_from\_user0 <= reg1;

85 when x"002" => data\_from\_user0 <= x"12345678";

86 when others => null;

87 end case;

88 end if;

89

90 if user\_wr = '1' then

91 case user\_add is

92 when x"000" => reg0 <= data\_to\_user;

93 when x"001" => reg1 <= data\_to\_user;

94 when others => null;

95 end case;

96 end if;

97 end if;

98 end if;

99 end process;

100 end architecture STRUCTURE;

101

## Software와의 연동

### Xparameters.h

/\* Definitions for peripheral AXI\_USER\_LOGIC\_32X8192\_4CS\_0 \*/

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM0\_BASEADDR 0x76A00000

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM0\_HIGHADDR 0x76A0FFFF

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM1\_BASEADDR 0x76A20000

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM1\_HIGHADDR 0x76A2FFFF

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM2\_BASEADDR 0x76A40000

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM2\_HIGHADDR 0x76A4FFFF

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM3\_BASEADDR 0x76A60000

**#define** XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM3\_HIGHADDR 0x76A6FFFF

/\* Definitions for peripheral MYLOGIC\_0 \*/

**#define** XPAR\_MYLOGIC\_0\_BASEADDR 0x7AE00000

**#define** XPAR\_MYLOGIC\_0\_HIGHADDR 0x7AE0FFFF

### C code

/\*

\* main.c

\*

\* Created on: Mar 24, 2013

\* Author: hyukk

\*/

**#include** "xparameters.h"

**void** **main**()

{

**volatile** **int** \*add, \*addA, data;

**volatile** **int** \*addB;

**int** i = 0;

add = (**int** \*)(XPAR\_MYLOGIC\_0\_BASEADDR);

addA = (**int** \*)(XPAR\_MYLOGIC\_0\_BASEADDR + 0x1c);

addB = (**int** \*)(XPAR\_AXI\_USER\_LOGIC\_32X8192\_4CS\_0\_S\_AXI\_MEM0\_BASEADDR);

\*addB = 0x1;

data = \*addB++;

\*addB = 0x3;

data = \*addB++;

data = \*addB++;

**while**(1) {

data = \*add;

\*add = i++;

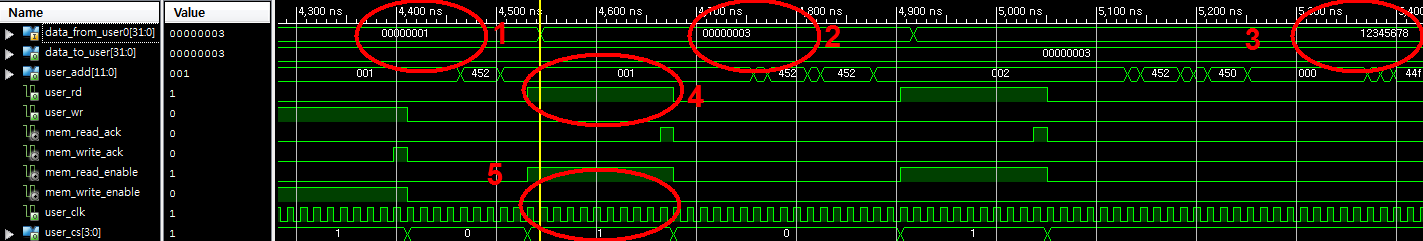
data = \*addA;

\*addA = i++;

}

}

### 시뮬레이션 결과



## ChipScope를 통한 동작확인

### IBA

### Trigger condition

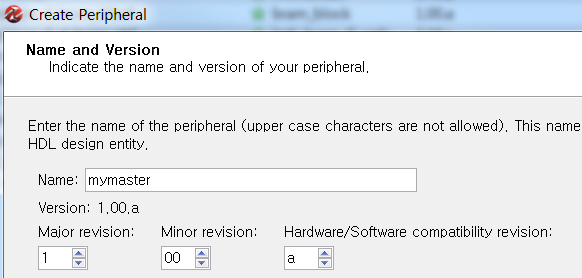
# Example AXI Master

이번장에서 자일링스는 마스터를 설계할 수 있는 기본적인 플랫폼을 이해하고 마스터가 AXI 인터커넥터와 어떤 프로토콜을 이용해 데이터 주고 받지는 살펴 보겠습니다.

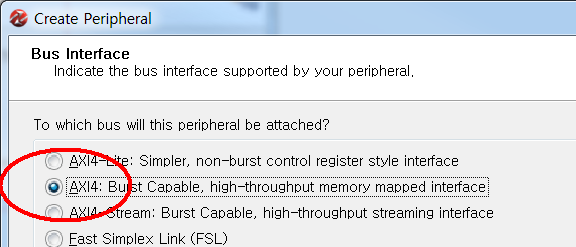
## CIP for Axi Master Example

CIP를 이용해서 마스터 example을 만드는 것은 앞장에서 IP를 만드는 과정과 몇가지 다른 점이 있는데 그중에 이해해야 하는 부분에 대해서 설명 하겠습니다.

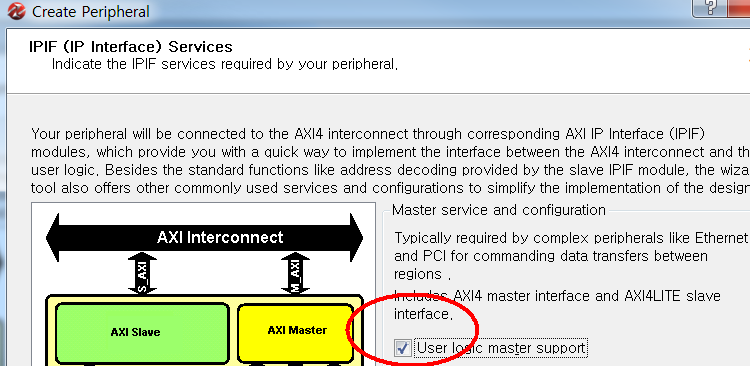
마스터 이름을 정합니다.



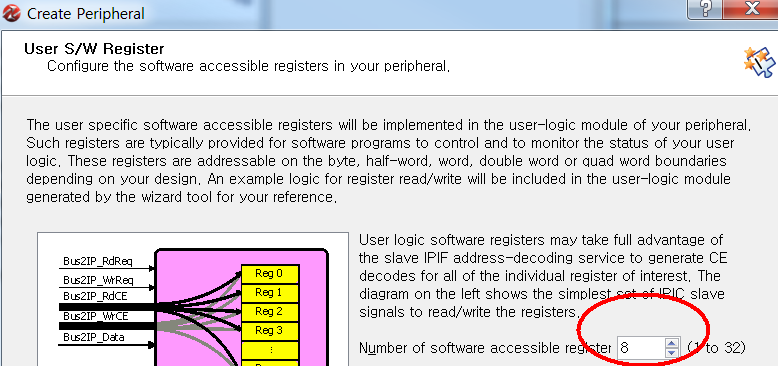
AXI4Full 방식으로 선택 합니다.



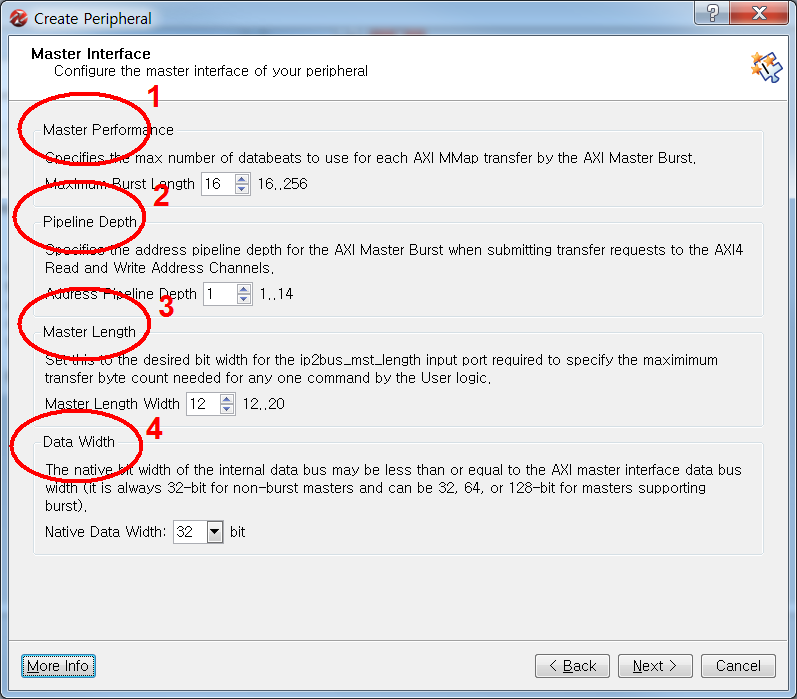
마스터 로직을 선택 합니다.



사용자 목적으로 사용할 레지스터의 개수를 정합니다. 경험상 8개 정도면 충분 합니다.



다음 그림은 마스터 로직이 동작할 때 필요한 여러 옵션을 선택하는 창 입니다.



#### Master Performance

이게 무슨 얘기인가 하면 마스터가 데이터를 전송할 때 버스트 방식으로 전송한다고 했습니다. 어드레스 한번 주고 다음에 데이터를 억세스할 때 8번을 억세스할 지, 16번 억세스할 지를 결정하는 옵션 입니다.

그런데 이게 왜 Performance와 관련이 있는가 하면 Payload라는 개념 땜에 그렇습니다.

차근히 설명해 보면 다음과 같습니다.

마스터가 데이터를 전송하려고 합니다. 그러면 마스터는 인터커넥터에게 먼저 어드레스를 날려주고 ack를 기다립니다.

Ack가 오면 읽기/쓰기 동작에 따라서 데이터를 쭉 억세스 합니다. 몇번 억세스하는가를 나타내는 용어로 databeats를 사용 합니다.

억세스 동작이 끝나면

마스터는 다시 어드레스를 인터커넥터에 날려주고 다시 ack를 기다리고 데이터를 억세스하는 과정을 반복하게 됩니다.

만약에 Master가 32비트 데이터 1024개를 전달하려고 가정 하면 기본 databeats가 16개 이니까 64번 버스트 데이터 전송을 해야 합니다.

버스트 데이터 전송을 64번 한다는 것은 어드레스를 64번 인터커넥터에 날려주고 ack를 기다리는 과정을 필요 하다는 것을 알 수 있습니다.

그러면 실제로 데이터를 전송하는데 사용하는 시간과 어드레스 날려주고 ack를 기다리는데 필요한 시간을 비교해 볼 때 어드레스를 억세스 하는 시간이 아쉬울 때가 있기 마련 입니다.

그래서 이 부분을 16 databeats 로 제한하지 않고 256 databeats로 선택할 수 있도록 했습니다.

256 databeats를 선택하면 1024/256 => 4번. 즉 어드레스를 인터커넥터에 날려주고 ack를 기다리는 횟수가 64번에서 4번으로 줄어 들게 되니까 상대적으로 데이터를 전송하는 비율이 올라가게 되는 것 입니다.

즉 전송율이 높아지게 되는 것이지요.

다른 말로 payload 비율이 높다고 말하는 것 입니다.

그렇다고 무조건 256을 쓰는게 좋은 것인가?

인터커넥터를 하나의 마스터가 오랫동안 점유하면 다른 마스터는 좀 쉬게 되는 경우가 있습니다.

최적화 된 값을 찾기 위해서는 마스터의 성능, 시스템의 성능, 슬레이브 (주로 메모리)의 성능을 모두 참조해서 결정해야 합니다.

무리가 없다면 기본값을 그대로 사용 합니다.

#### Pipeline Depth

#### Master Length

마스터가 한번에 전송하려는 최대 BYTE수를 지정할 수 있는 옵션 입니다. 기본 값은 12비트로 정해져 있으니까 2^12이면 4096개의 바이트를 전송할 수 있습니다.

#### Data Width

인터커넥터와 인터페이스 되는 데이터 버스의 폭을 의미 합니다. 기본 값은 32비트이고 최대 128비트까지 확장 가능 합니다.

만약 4KB 데이터가 주기적으로 입력되는 신호가 있다고 가정하고 모든 옵션을 기본 값으로 사용하다면

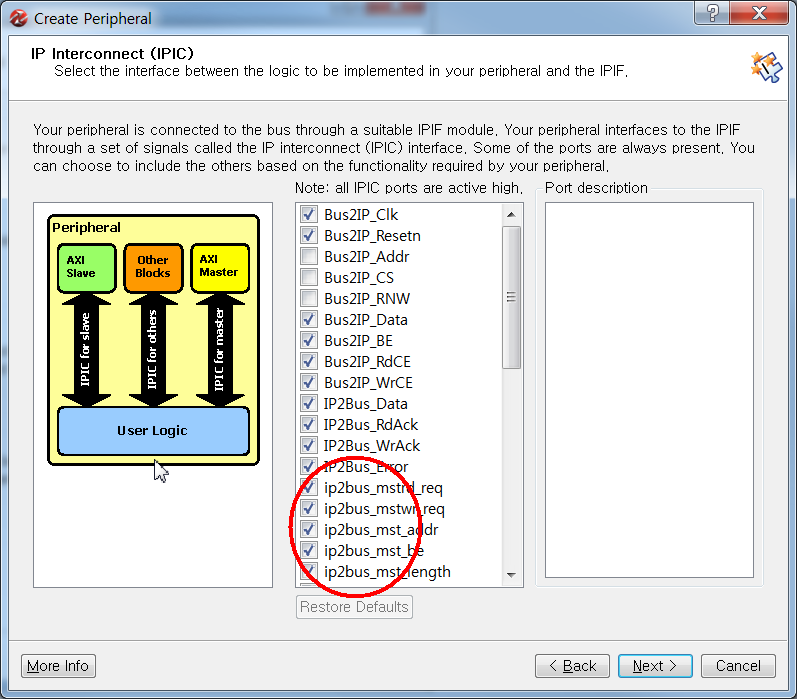
4096/4 (byte, 32비트) == 1024 번 전송을 해야 하는데

Master Length가 12비트이면 마스터 한번에 전송가능 한 숫자 입니다.

Master Performance가 현재 16databeats로 정해져 있으니까 1024/16 == 64.

그러면 IPIC에서 4KB를 전송하겠다고 요청 하면 AXI 마스터는 이것을 64번 나누어서 보내게 되는 것 입니다.

다음 그림은 IPIC 신호 중에 Master와 관련된 신호가 추가된 것을 보여 줍니다.



## DS844

CIP에서 제공하는 Example AXI 마스터를 이해하려면 IPIC 관련 신호들들 이해를 하고 있어야 합니다.

AXI 마스터와 IPIC와의 관계를 이해하려면 DS844 (datashee 844번, 자일링스 website에서 DS844를 검색하면 찾을 수 있습니다.) 를 살펴봐야 합니다.

<http://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf>

# Customer Axi Master

예를 들어 메모리에 있는 특정 데이터 블록을 계속 읽어야 할 필요가 있을 때 마스터를 설계해야 합니다.

앞에서 보았던 CDMA는 메모리와 메모리 간에 데이터 전송을 할 수는 있지만 메모리로 부터 특정 FIFO로 데이터를 전달할 필요가 있습니다.

모뎀 장비 같은 경우, 외부 메모리로 부터 IQ데이터를 중단없이 전달하기 위해서는 메모리와 FIFO를 연결해주는 DMA를 설계해야 하는데 이 부분이 그렇게 부담없이 해결할 수 있는 부분은 아닙니다.

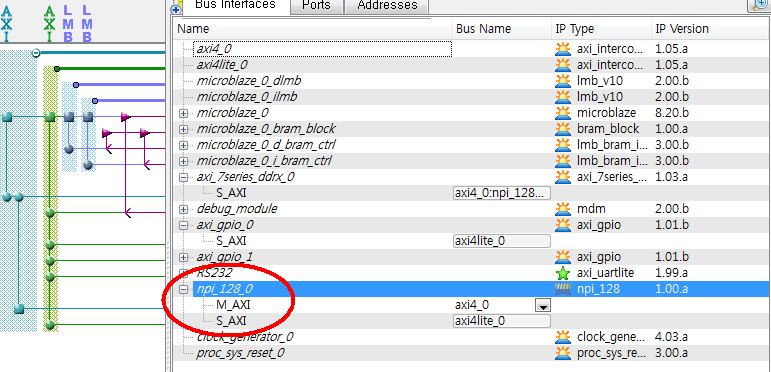
이 문제를 해결하기 위해서는 AXI 마스터를 설계하고 AXI 프로토콜을 제대로 이해해야 합니다.

먼저 기존에 임베디드 시스템에서 사용했던 IP를 살펴보도록 하겠습니다.

## 메모리 억세스 코어, npi\_128

이 IP는 npi\_128이라는 IP 입니다. 이 ip는 마스터 포트와 슬레이브 포트를 모두 가지고 있기 때문에 다음 그림과 같이 2개의 axi interconnect를 사용 합니다.

마스터 포트로는 메모리에서 데이터를 읽어내는 동작을 하고 슬레이브 포트는 프로세서로 부터 필요한 메모리 정보와 읽을 데이터 크기에 대한 정보를 전달 받게 됩니다.



이 IP의 포트는 다음과 같이 MPD파일에 저장되어 있습니다. 다음 리스트는 npi\_128에 대한 mpd file 입니다. 라인 47~52를 보면 이 포트를 통해 메모리의 데이터가 읽힌다는 것을 알 수 있습니다. 라인 61~111까지는 AXI4에 대한 인터페이스 입니다.

8

9 BEGIN npi\_128

10

45

46 ## Ports

47 PORT rd\_fifo\_clk = "", DIR = I

48 PORT rd\_fifo\_rd\_en = "", DIR = I

49 PORT rd\_fifo\_data = "", DIR = O, VEC = [31:0]

50 PORT rd\_fifo\_empty = "", DIR = O

51 PORT rd\_fifo\_full = "", DIR = O

52 PORT npi\_rd\_ready = "", DIR = O

60

61 PORT S\_AXI\_ACLK = "", DIR = I, SIGIS = CLK, BUS = S\_AXI

62 PORT S\_AXI\_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS = S\_AXI

63 PORT S\_AXI\_AWADDR = AWADDR, DIR = I, VEC = [(C\_S\_AXI\_ADDR\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

108 PORT m\_axi\_wlast = WLAST, DIR = O, BUS = M\_AXI

109 PORT m\_axi\_bready = BREADY, DIR = O, BUS = M\_AXI

110 PORT m\_axi\_bvalid = BVALID, DIR = I, BUS = M\_AXI

111 PORT m\_axi\_bresp = BRESP, DIR = I, VEC = [1:0], BUS = M\_AXI

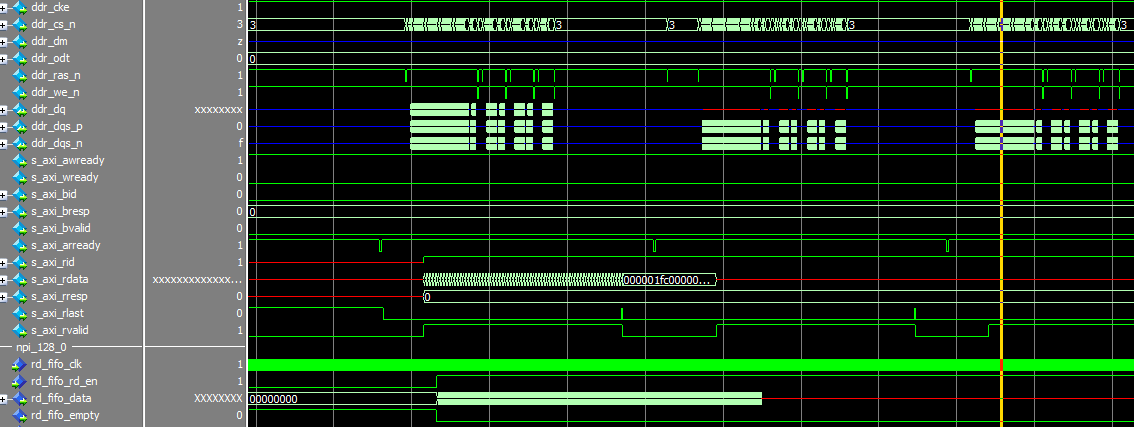
112

113 END

114

이 npi\_128은 프로세서가 적당한 값을 npi\_128 내부 레지터스에 기록하면 메모리로 부터 데이터를 계속 가져다가 npi\_128 내부에 있는 FIFO에 기록 합니다.

다음 그림과 같이 npi\_128는 메모리 콘트롤러에 read request를 하고 메모리 콘트롤러를 메모리로 부터 데이터를 읽어 npi\_128의 axi interface로 전달하게 됩니다.

ㅋ

다음 리스트는 npi\_128에 대한 mhs 파일 입니다.

BEGIN npi\_128

PARAMETER INSTANCE = npi\_128\_0

PARAMETER HW\_VER = 1.00.a

PARAMETER C\_BASEADDR = 0x71000000

PARAMETER C\_HIGHADDR = 0x7100FFFF

PARAMETER C\_INTERCONNECT\_M\_AXI\_READ\_FIFO\_DEPTH = 512

BUS\_INTERFACE M\_AXI = axi4\_0

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = clk\_100\_0000MHzPLLE0

PORT m\_axi\_aclk = clk\_100\_0000MHzPLLE0

PORT rd\_fifo\_clk = npi\_128\_0\_rd\_fifo\_clk

PORT rd\_fifo\_rd\_en = npi\_128\_0\_rd\_fifo\_rd\_en

PORT rd\_fifo\_data = npi\_128\_0\_rd\_fifo\_data

PORT npi\_rd\_ready = npi\_128\_0\_npi\_rd\_ready

PORT rd\_fifo\_full = npi\_128\_0\_rd\_fifo\_full

PORT rd\_fifo\_empty = npi\_128\_0\_rd\_fifo\_empty

END

여기서 axi는 100mhz 동작하고 있지만 npi\_128의 rd\_fifo\_clk은 외부에서 입력 받도록 되어 있습니다. 이렇게 만든 이유는 메모리를 억세스하는 콘트롤러의 클럭과 사용하고자 하는 클럭의 주파수가 항상 같지 않기 때문에 그렇습니다. 결과적은 npi\_128 내부에는 aync fifo를 구현했서 이렇게 서로 다른 클록 도메인간의 데이터 전송을 해결 했습니다.

## npi\_128의 user\_logic.vhd 구성

다음 리스트는 npi\_128 내부를 구성라는 로직에 대한 일부 코드 입니다. 라인 142부터 202까지는 npi\_128를 구성하는 포트에 대한 정의 입니다. 이 포트에는 FIFO 인터페이스와 AXI 인터페이스가 같이 있습니다.

1

72

125 entity user\_logic is

142 port

143 (

144 -- ADD USER PORTS BELOW THIS LINE ------------------

145 --USER ports added here

146 rd\_fifo\_clk : in std\_logic;

147 rd\_fifo\_rd\_en : in std\_logic;

148 rd\_fifo\_data : out std\_logic\_vector(31 downto 0);

149 rd\_fifo\_empty : out std\_logic;

150 rd\_fifo\_full : out std\_logic;

151 npi\_rd\_ready : out std\_logic;

159

160 -- ADD USER PORTS ABOVE THIS LINE ------------------

161

162 -- DO NOT EDIT BELOW THIS LINE ---------------------

163 -- Bus protocol ports, do not add to or delete

164 Bus2IP\_Clk : in std\_logic;

165 Bus2IP\_Resetn : in std\_logic;

166 Bus2IP\_Data : in std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

167 Bus2IP\_BE : in std\_logic\_vector(C\_SLV\_DWIDTH/8-1 downto 0);

168 Bus2IP\_RdCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);

169 Bus2IP\_WrCE : in std\_logic\_vector(C\_NUM\_REG-1 downto 0);

170 IP2Bus\_Data : out std\_logic\_vector(C\_SLV\_DWIDTH-1 downto 0);

171 IP2Bus\_RdAck : out std\_logic;

172 IP2Bus\_WrAck : out std\_logic;

200 ip2bus\_mstwr\_eof\_n : out std\_logic;

201 bus2ip\_mstwr\_dst\_rdy\_n : in std\_logic;

202 bus2ip\_mstwr\_dst\_dsc\_n : in std\_logic

203 -- DO NOT EDIT ABOVE THIS LINE ---------------------

204 );

라인 307에는 async fifo하나를 선언 했습니다. 입력은 128비트이고 출력은 32비트 입니다. 이렇게 정한 이유는 현재 메모리 콘트롤러의 PHY가 4:1 모드로 설계되어 있어서 메모리 콘트롤러가 4개의 데이터를 모아서 한꺼번에 데이터를 전송하는 방식으로 설계되어 있습니다.

따라서 병목 현상을 없애기 위해서는 메모리의 데이터 폭이 32비트 이므로 128비트가 되게 설계를 해야 합니다.

307 COMPONENT afifo\_128to32

308 PORT (

309 rst : IN STD\_LOGIC;

310 wr\_clk : IN STD\_LOGIC;

311 rd\_clk : IN STD\_LOGIC;

312 din : IN STD\_LOGIC\_VECTOR(127 DOWNTO 0);

313 wr\_en : IN STD\_LOGIC;

314 rd\_en : IN STD\_LOGIC;

315 dout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

316 full : OUT STD\_LOGIC;

317 empty : OUT STD\_LOGIC;

318 rd\_data\_count : OUT STD\_LOGIC\_VECTOR(2 DOWNTO 0)

319 );

320 END COMPONENT;

321

322

336

337 begin

338

426

라인 443~448까지는 입력 클럭에 값이 증가하는 카운터를 하나 준비 했습니다. 이 로직은 순수히 디버깅 용으로 외부에서 클럭이 제대로 공급되는지을 아는데 필요 합니다.

443 process(rd\_fifo\_clk)

444 begin

445 if rd\_fifo\_clk'event and rd\_fifo\_clk = '1' then

446 cnt\_rx <= cnt\_rx + 1;

447 end if;

448 end process;

라인 521부터 548까지는 프로세서와 npi\_128 서로 데이터를 주고 받기 위한 레지터스를 구성하는 부분 입니다. 특히 라인 547 에서는 메모리로 부터 데이터를 한꺼번에 몇 바이트를 읽는지를 결정 합니다.

메모리의 특징이 뭐냐하면 데이터를 한번 읽고 쓰는데 준비하는 시간이 너무 많이 걸린다는 것 입니다. 그래서 한번 읽을 때 많이 읽고 써야 제대로 메모리를 억세스 한다고 할 수 있습니다.

라인 542,3,4,5를 보면 0x100 바이트 읽는 610nsec 이 소요되고 0x400 바이트를 읽을 떄는 1090, 0x800바이트를 읽을 때는 모두 1730nsec이 소비 되었습니다. 읽는 시간은 약 3배로 증가했지만 읽는 데이터 양은 모두 8배 증가 했습니다.

521 -- rip control bits from master model registers

522 ------------ offset 0

523 mst\_cntl\_rd\_req <= mst\_reg(0)(0);

524 mst\_cntl\_wr\_req <= mst\_reg(0)(1);

525 mst\_cntl\_bus\_lock <= mst\_reg(0)(2);

526 mst\_cntl\_burst <= mst\_reg(0)(3);

527 ------------ offset 2

528 c\_cnt\_burst <= mst\_reg(3)(4 downto 0) & mst\_reg(2);

529 ------------ offset 4

530 -- mst\_ip2bus\_addr <= mst\_reg(7) & mst\_reg(6) & mst\_reg(5) & mst\_reg(4);

531 -- mst\_ip2bus\_addr <= mst\_reg(7)(7 downto 1) & cnt\_burst & x"00";

532 --- max pattern 307200

533 --- 307200/128 == 2400 == 0x960

534 --- every burst move 128byte ==> 0x80 ==> 0b1000\_0000

535 --- 24 + (12 + 1) + 7

536 --- mst\_ip2bus\_addr <= mst\_reg(7) & mst\_reg(6)(7 downto 3) & cnt\_burst & mst\_reg(5)(6 downto 0); -- when 0x80

라인 537에서는 프로세서가 메모리의 어느 번지를 읽을 것인지 결정 합니다.

537 mst\_ip2bus\_addr <= mst\_reg(7) & cnt\_burst & x"0" & mst\_reg(5)(6 downto 0); -- when 0x8000

538 --mst\_ip2bus\_be <= mst\_reg(9) & mst\_reg(8);

539 mst\_ip2bus\_be <= x"ffff";

0x800번씩 몇 번을 읽을 것인 결정 합니다.

540 c\_cnt\_repeat <= mst\_reg(11) & mst\_reg(10);

541 --mst\_xfer\_reg\_len <= mst\_reg(14)(3 downto 0) & mst\_reg(13) & mst\_reg(12);

542 -- mst\_xfer\_reg\_len <= x"00080";

543 -- mst\_xfer\_reg\_len <= x"00100"; -- 610 nsec between req

544 -- mst\_xfer\_reg\_len <= x"00400"; --1090 nsec between req

545 -- mst\_xfer\_reg\_len <= x"00800"; --1730 nsec between req

546 -- 0x800 ==> 0x1000\_0000\_0000

547 mst\_xfer\_reg\_len <= x"00800";

548 mst\_xfer\_length <= mst\_xfer\_reg\_len(C\_LENGTH\_WIDTH-1 downto 0 );

549

npi\_128는 모두 2개의 스테이트 머신이 동작을 합니다.

665 --implement master command interface state machine

666 MASTER\_CMD\_SM\_PROC : process( Bus2IP\_Clk ) is

667 begin

668

669 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

710 -- state transition

711 case mst\_cmd\_sm\_state is

최초의 상태는 CMD\_IDLE로서 프로세서 npi\_128에게 명령을 내릴 것을 기다립니다.

712 when CMD\_IDLE =>

프로세서로 부터 읽기 명령이 내려오면 CMD\_RUN을 상태를 바꾸게 됩니다.

713 if ( mst\_go = '1') then

714 if ( mst\_cntl\_rd\_req = '1' ) then

715 if rd\_fifo\_wr\_cnt\_s /= "111" then

716 mst\_cmd\_sm\_start\_rd\_llink <= '1';

717 mst\_cmd\_sm\_state <= CMD\_RUN;

718 cnt\_st <= x"2";

719 end if;

CMD\_RUN상태에서는 버스가 준비되어 있고 메모리를 읽는데 문제가 없는 경우일 때만 CMD\_DONE이라는 상태로 바꾸게 됩니다. 그렇지 않는 경우라면 메모리 콘트롤러 데이터를 읽을 때까지 계속 기다립니다.

732 when CMD\_RUN =>

733 if ( Bus2IP\_Mst\_CmdAck = '1' and Bus2IP\_Mst\_Cmplt = '0' ) then

734 mst\_cmd\_sm\_state <= CMD\_WAIT\_FOR\_DATA;

735 cnt\_st <= x"3";

736 elsif ( Bus2IP\_Mst\_Cmplt = '1' ) then

737 mst\_cmd\_sm\_state <= CMD\_DONE;

738 cnt\_st <= x"4";

739 if ( Bus2IP\_Mst\_Cmd\_Timeout = '1' ) then

740 mst\_cmd\_sm\_set\_error <= '1';

741 mst\_cmd\_sm\_set\_timeout <= '1';

742 elsif ( Bus2IP\_Mst\_Error = '1' ) then

743 -- AXI4LITE data transfer error

744 mst\_cmd\_sm\_set\_error <= '1';

745 end if;

746 else

747 mst\_cmd\_sm\_state <= CMD\_RUN;

748 cnt\_st <= x"2";

749 mst\_cmd\_sm\_rd\_req <= mst\_cntl\_rd\_req;

750 mst\_cmd\_sm\_wr\_req <= mst\_cntl\_wr\_req;

751 mst\_cmd\_sm\_ip2bus\_addr <= mst\_ip2bus\_addr;

752 mst\_cmd\_sm\_ip2bus\_be <= mst\_ip2bus\_be(15 downto 16-C\_MST\_NATIVE\_DATA\_WIDTH/8 );

753 mst\_cmd\_sm\_xfer\_type <= mst\_cntl\_burst;

754 mst\_cmd\_sm\_xfer\_length <= mst\_xfer\_length;

755 mst\_cmd\_sm\_bus\_lock <= mst\_cntl\_bus\_lock;

756 end if;

마지막으로 읽어야 하는 버스트 갯수를 비교해 봐서 다 읽지 못했면 다시 CMD\_RUN 상태로 바꾸고 모두 읽은 상태라면 CMD\_IDLE 상태로 바꾸게 됩니다.

773 when CMD\_DONE =>

774 if(cnt\_burst = c\_cnt\_burst - 1) then

775 if(cnt\_repeat = c\_cnt\_repeat - 1) then

776 cnt\_repeat <= (others => '0');

777 cnt\_repeat <= (others => '0');

778 mst\_cmd\_sm\_clr\_go <= '1';

779 mst\_cmd\_sm\_set\_done <= '1';

780 mst\_cmd\_sm\_busy <= '0';

781 mst\_cmd\_sm\_state <= FINISH;

782 cnt\_st <= x"5";

783 else

만약 개발자가 c\_cnt\_repeat 가 0xffff이면 무한대로 반복한다고 판단 합니다.

784 cnt\_burst <= (others => '0');

785 if(c\_cnt\_repeat /= x"ffff") then

786 cnt\_repeat <= cnt\_repeat + 1;

787 end if;

788 mst\_cmd\_sm\_set\_done <= '1';

789 mst\_cmd\_sm\_busy <= '0';

790 mst\_cmd\_sm\_state <= CMD\_IDLE;

791 end if;

792

메모리 컨트롤러로 부터 전달 받은 메모리 데이터를 FIFO에 쓰기 위한 조건을 라인 996에 언급하였습니다.

996 mst\_fifo\_valid\_write\_xfer <= not(Bus2IP\_MstRd\_src\_rdy\_n) and mst\_llrd\_sm\_dst\_rdy and not(rd\_fifo\_full\_s);

997 mst\_fifo\_valid\_read\_xfer <= not(Bus2IP\_MstWr\_dst\_rdy\_n) and mst\_llwr\_sm\_src\_rdy and not(wr\_fifo\_empty\_s);

998 Bus2IP\_Reset <= not (Bus2IP\_Resetn);

999

라인 1102부터 1114까지 async fifo에 대한 조건을 보여주고 있습니다.

1102rd\_fifo : afifo\_128to32

1103 PORT MAP (

1104 rst => fifo\_rst,

1105 wr\_clk => Bus2IP\_Clk,

1106 rd\_clk => rd\_fifo\_clk,

1107 din => Bus2IP\_MstRd\_d,

1108 wr\_en => mst\_fifo\_valid\_write\_xfer,

1109 rd\_en => rd\_fifo\_rd\_en,

1110 dout => rd\_fifo\_data,

1111 full => rd\_fifo\_full\_s,

1112 empty => rd\_fifo\_empty\_s,

1113 rd\_data\_count => rd\_fifo\_wr\_cnt\_s

1114 );

1115

라인 1121~1136까지는 일단 읽기 시작하면 fifo는 empty가 발생하지 않아야 합니다. 이렇게 되기 위해서는 메모리 콘트롤러 충분히 빠르게 데이터를 FIFO에 써 넣어 주어야 합니다.

하지만 써주는 마스터와 읽어가는 마스터간에 속도 차이가 나면 이 부분을 디버깅 하기 위해 로직을 추가 하였습니다.

1121

1122 process(rd\_fifo\_clk)

1123 begin

1124 if rd\_fifo\_clk = '1' and rd\_fifo\_clk = '1' then

1125 if npi\_rd\_ready\_s = '1' then

1126 if rd\_fifo\_empty\_s = '1' then

1127 cnt\_rd\_fifo\_empty\_s <= cnt\_rd\_fifo\_empty\_s + 1;

1128 end if;

1129 else

1130 if cnt\_clear\_d0 = '1' then

1131 cnt\_rd\_fifo\_empty\_s <= (others => '0');

1132 end if;

1133 end if;

1134 cnt\_clear\_d0 <= cnt\_clear;

1135 end if;

1136 end process;

1137

다음 그림과 같이 1번은 프로세서가 현재 메모리에 데이터를 쓰는 과정이고 2번은 npi\_128 메모리로 부터 데이터를 버스트 방식으로 읽는 파형을 보여 줍니다.

다음 리스트는 npi\_128를 싸고 있는 top 모듈 입니다.

156 entity npi\_128 is

157 generic

158 (

159 -- ADD USER GENERICS BELOW THIS LINE ---------------

160 --USER generics added here

161 -- ADD USER GENERICS ABOVE THIS LINE ---------------

162

163 -- DO NOT EDIT BELOW THIS LINE ---------------------

164 -- Bus protocol parameters, do not add to or delete

165 C\_S\_AXI\_DATA\_WIDTH : integer := 32;

166 C\_S\_AXI\_ADDR\_WIDTH : integer := 32;

167 C\_S\_AXI\_MIN\_SIZE : std\_logic\_vector := X"000001FF";

168 C\_USE\_WSTRB : integer := 0;

169 C\_DPHASE\_TIMEOUT : integer := 8;

170 C\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

171 C\_HIGHADDR : std\_logic\_vector := X"00000000";

172 C\_FAMILY : string := "virtex6";

173 C\_NUM\_REG : integer := 8;

174 C\_NUM\_MEM : integer := 1;

175 C\_SLV\_AWIDTH : integer := 32;

176 C\_SLV\_DWIDTH : integer := 32;

177 C\_M\_AXI\_ADDR\_WIDTH : integer := 32;

현재 phy 모드가 4:1 이기 때문에 128로 설정해야 합니다.

178 C\_M\_AXI\_DATA\_WIDTH : integer := 128;

179 C\_MAX\_BURST\_LEN : integer := 256;

180 C\_NATIVE\_DATA\_WIDTH : integer := 128;

181 C\_LENGTH\_WIDTH : integer := 20;

182 C\_ADDR\_PIPE\_DEPTH : integer := 1

183 -- DO NOT EDIT ABOVE THIS LINE ---------------------

184 );

185 port

186 (

187 -- ADD USER PORTS BELOW THIS LINE ------------------

188 --USER ports added here

npi\_128가 필요로 하는 포트를 정의하였습니다.

189 rd\_fifo\_clk : in std\_logic;

190 rd\_fifo\_rd\_en : in std\_logic;

191 rd\_fifo\_data : out std\_logic\_vector(31 downto 0);

192 rd\_fifo\_empty : out std\_logic;

193 rd\_fifo\_full : out std\_logic;

194 npi\_rd\_ready : out std\_logic;

195

196 -- ADD USER PORTS ABOVE THIS LINE ------------------

197

198 -- DO NOT EDIT BELOW THIS LINE ---------------------

나머지는 AXI 포트 입니다.

199 -- Bus protocol ports, do not add to or delete

200 S\_AXI\_ACLK : in std\_logic;

201 S\_AXI\_ARESETN : in std\_logic;

202 S\_AXI\_AWADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);

203 S\_AXI\_AWVALID : in std\_logic;

204 S\_AXI\_WDATA : in std\_logic\_vector(C\_S\_AXI\_DATA\_WIDTH-1 downto 0);

205 S\_AXI\_WSTRB : in std\_logic\_vector((C\_S\_AXI\_DATA\_WIDTH/8)-1 downto 0);

206 S\_AXI\_WVALID : in std\_logic;

207 S\_AXI\_BREADY : in std\_logic;

208 S\_AXI\_ARADDR : in std\_logic\_vector(C\_S\_AXI\_ADDR\_WIDTH-1 downto 0);

다음 리스트는 현재 IP top모듈과 user\_logic을 인터페이스 하는 부분 입니다.

라인 566을 보면 IP top 모듈에서 정해진 값이 user\_logic까지 전달 됩니다.

556 ------------------------------------------

557 -- instantiate User Logic

558 ------------------------------------------

559 USER\_LOGIC\_I : entity npi\_128\_v1\_00\_a.user\_logic

560 generic map

561 (

562 -- MAP USER GENERICS BELOW THIS LINE ---------------

563 --USER generics mapped here

564 -- MAP USER GENERICS ABOVE THIS LINE ---------------

565

566 C\_MST\_NATIVE\_DATA\_WIDTH => USER\_MST\_NATIVE\_DATA\_WIDTH,

567 C\_LENGTH\_WIDTH => USER\_LENGTH\_WIDTH,

568 C\_MST\_AWIDTH => USER\_MST\_AWIDTH,

569 C\_NUM\_REG => USER\_NUM\_REG,

570 C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH

571 )

572 port map

573 (

574 -- MAP USER PORTS BELOW THIS LINE ------------------

575 --USER ports mapped here

576 rd\_fifo\_clk => rd\_fifo\_clk,

577 rd\_fifo\_rd\_en => rd\_fifo\_rd\_en,

578 rd\_fifo\_data => rd\_fifo\_data,

579 rd\_fifo\_empty => rd\_fifo\_empty,

580 rd\_fifo\_full => rd\_fifo\_full,

581 npi\_rd\_ready => npi\_rd\_ready,

582

583 -- MAP USER PORTS ABOVE THIS LINE ------------------

584

585 Bus2IP\_Clk => ipif\_Bus2IP\_Clk,

586 Bus2IP\_Resetn => rst\_Bus2IP\_Reset\_tmp,

587 Bus2IP\_Data => ipif\_Bus2IP\_Data,

588 Bus2IP\_BE => ipif\_Bus2IP\_BE,

589 Bus2IP\_RdCE => user\_Bus2IP\_RdCE,

590 Bus2IP\_WrCE => user\_Bus2IP\_WrCE,

591 IP2Bus\_Data => user\_IP2Bus\_Data,

npi\_128 로직을 만들기 위해서는 aync fifo를 설계해야 하는데 이 부분은 주로 coregen을 통해서 구현 합니다.

다음 리스트는 npi\_128 구현하는데 필요한 네트리스트가 무엇이진 알려 줍니다.

##############################################################################

## Filename: D:\project\mch\_test\pcores/mch\_master\_demo\_v1\_00\_a/data/mch\_master\_demo\_v2\_1\_0.bbd

## Description: Black Box Definition

## Date: Thu Jun 15 15:23:49 2006 (by Create and Import Peripheral Wizard)

##############################################################################

Files

################################################################################

afifo\_128to32.ng

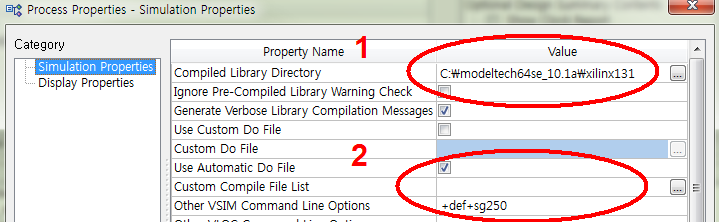
MPD 파일에서는 현재 npi\_128가 네트리스트를 포함하고 있다는 것을 알려주기 위해 다음과 같은 문법을 사용 합니다.

OPTION STYLE = MIX

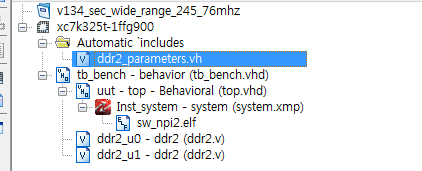
## npi\_128 시뮬레이션 설명

### 시뮬레이션 환경 설정

XPS에서 정의한 임베디드 시스템을 시뮬레이션 하기 위해서는 다음과 같이 시뮬레이션 환경설정이 되어 있어야 합니다. 1번은 현재 모델심용 라이브러리의 위치를 지정하고 2번은 현재 사용하는 DDR2 메모리에 대한 시뮬레이션 정보 입니다. 여기서는 250nsec 이니까 400mhz에서 동작하는 시뮬레이션 모델을 사용하다는 것을 의미 합니다.



이 +def+sg250은 다음 그림과 같이 ddr2\_parameters.VH 라는 파일에 정의되어 있는 여러 타이밍 정보를 공급하게 됩니다.



### 시뮬레이션 결과

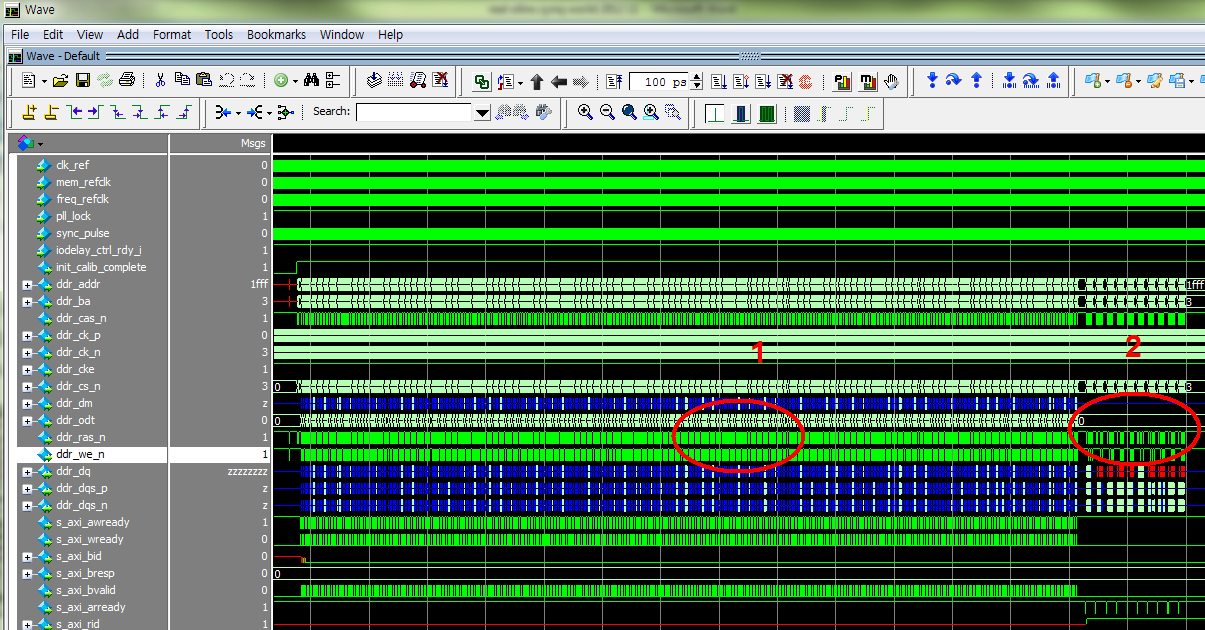
다음 그림과 같이 프로세서가 적당히 npi\_128 내부에 레지터스를 억세스 하고나면 시뮬레이션을 할 수 있습니다.

그림 1은 프로세서가 현재 메모리에 특정한 패턴을 기록하는 부분이고 그림 2는 npi\_128가 메모리로 부터 데이터를 읽기 시작하는 부분 입니다.

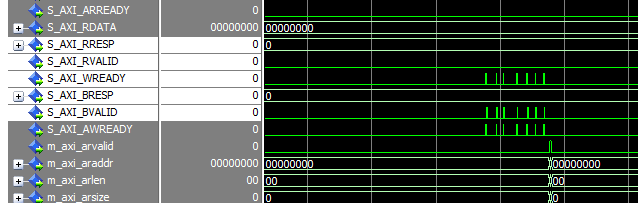
실제로 프로세서가 이렇게 오랫동안 메모리를 억세스 하더라도 실제로 억세스하는 데이터 량은 npi\_128에 비해 턱없이 부족 합니다.

뒤에서 설명하지만 1번동안 프로세서가 하는 일은 0x200개의 데이터를 메모리에 차례대로 쓰는 것이고 2번은 npi\_128가 데이터를 읽는 부분인데 이 짧은 구간에만 벌써 10번씩 0x800개의 바이트를 읽었습니다.

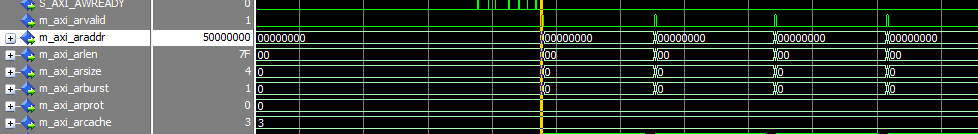
이런면에 프로세서는 메모리를 효율적으로 사용하기에는 (적어도 프로세서 내부에 데이터를 버스트방식으로 전달하는 특별한 블록이 없다면) 적당한 마스터가 아닙니다.



다음 그림과 같이 프로세서는 npi\_128를 동작시키기 위해서 npi\_128 내부에 있는 레지터스를 억세스 해야 합니다.



프로세서가 레지터스 억세스가 끝이 나면 npi\_128가 데이터를 읽기 시작 합니다. 처음 시작할 때는 0x5000\_0000 입니다. 두번째를 살펴보면 0x5000\_0800 입니다. 프로세서가 지시한데로 로직은 잘 동작하고 있습니다.



## npi\_128 디바이스 드라이버

다음 리스트는 npi\_128를 억세스하기 위한 소프트웨어 코드 입니다. 일단 시뮬레이션을 할 때는 어디까지 시뮬레이션이 진행되었는 알기가 쉽지 않기 때문에 간단히 GPIO를 통해 값을 써 넣어 줌으로써 진행 사항을 파악하도록 하겠습니다.

라인 4,8,10,12에서는 현재 진행사항을 알기 위한 GPIO 억세스 코드 입니다.

라인 5,6번은 메모리 콘트롤러 메모리를 켈리브레이션 하는 동안 기다리는 부분 입니다.

메모리 켈리브레이션이라는 것은 고속으로 동작하는 메모리가 안전하게 데이터를 주고 받을 수 있도록 하기 위해서 Valid Window를 찾는 동작이라고 할 수 있습니다.

이 동작은 메모리에 따라 조금 다르기 때문에 프로세서가 안전하게 메모리를 억세스하기 위해서는 InitDone이라는 신호를 모니터링 하는 것이 좋습니다.

현재 설계에서는 initDone을 GPIO를 통해서 모니터링 하고 있습니다.

1 int main\_rd()

2 {

3 volatile int i;

4 \*(int \*)XPAR\_AXI\_GPIO\_1\_BASEADDR = 0x1;

5 while(1)

6 if((\*(int \*)XPAR\_AXI\_GPIO\_0\_BASEADDR) & 0x1)

7 break;

8 \*(int \*)XPAR\_AXI\_GPIO\_1\_BASEADDR = 0x2;

라인 9번에서는 메모리를 초기화는 작업을 하는 곳 입니다. 이전 그림에서 1번에 해당한다고 할 수 있습니다.

9 test\_init();

10 \*(int \*)XPAR\_AXI\_GPIO\_1\_BASEADDR = 0x3;

라인 11은 npi에 데이터를 읽기위한 명령을 내리는 부분 입니다. 이 부분을 제대로 이해하기 위해서는 앞에서 언급한 user\_logic의 레지터스 설계 부분과 디바이스 드라이버를 같이 생각해야 합니다.

11 axi\_npi\_rd(XPAR\_NPI\_128\_0\_BASEADDR, XPAR\_AXI\_7SDDR\_0\_S\_AXI\_BASEADDR, 0x5, 0x2);

12 \*(int \*)XPAR\_AXI\_GPIO\_1\_BASEADDR = 0x4;

13

14 for(i = 0; i < 400; i++);

Empty counter를 계산한 결과를 클리어 시키고 싶을 때 사용 합니다.

15 \*(int \*) (XPAR\_NPI\_128\_0\_BASEADDR + 0x1c) = 87654325;

16 \*(int \*) (XPAR\_NPI\_128\_0\_BASEADDR + 0x1c) = 87654320;

17 }

18

}

## npi\_128 system.mhs 설명

다음 리스트는 system.mhs에 관련된 일부 소스 코드 입니다.

라인 19부터 39까지는 시스템에서 필요로 하는 입출력 포트에 대한 정의 입니다. 여기서는 주로 메모리 인터페이스와 npi\_128의 FIFO 인터페이스가 정의되어 있습니다.

19 PORT ddr\_cke = axi\_7series\_ddrx\_0\_ddr\_cke, DIR = O, VEC = [0:0]

20 PORT ddr\_dqs\_p = ddr\_dqs\_p, DIR = IO, VEC = [3:0]

21 PORT ddr\_dqs\_n = ddr\_dqs\_n, DIR = IO, VEC = [3:0]

22 PORT ddr\_dq = ddr\_dq, DIR = IO, VEC = [31:0]

23 PORT ddr\_dm = axi\_7series\_ddrx\_0\_ddr\_dm, DIR = O, VEC = [3:0]

24 PORT ddr\_cas\_n = axi\_7series\_ddrx\_0\_ddr\_cas\_n, DIR = O

25 PORT ddr\_ba = axi\_7series\_ddrx\_0\_ddr\_ba, DIR = O, VEC = [1:0]

26 PORT ddr\_ck\_n = axi\_7series\_ddrx\_0\_ddr\_ck\_n, DIR = O, VEC = [1:0], SIGIS = CLK

27 PORT ddr\_cs\_n = axi\_7series\_ddrx\_0\_ddr\_cs\_n, DIR = O, VEC = [1:0]

28 PORT ddr\_ck\_p = axi\_7series\_ddrx\_0\_ddr\_ck\_p, DIR = O, VEC = [1:0], SIGIS = CLK

29 PORT ddr\_odt = axi\_7series\_ddrx\_0\_ddr\_odt, DIR = O, VEC = [1:0]

30 PORT ddr\_ras\_n = axi\_7series\_ddrx\_0\_ddr\_ras\_n, DIR = O

31 PORT ddr\_addr = axi\_7series\_ddrx\_0\_ddr\_addr, DIR = O, VEC = [12:0]

32 PORT ddr\_we\_n = axi\_7series\_ddrx\_0\_ddr\_we\_n, DIR = O

33 PORT npi\_rd\_fifo\_clk = npi\_128\_0\_rd\_fifo\_clk, DIR = I

34 PORT npi\_rd\_fifo\_rd\_en = npi\_128\_0\_rd\_fifo\_rd\_en, DIR = I

35 PORT npi\_rd\_fifo\_data = npi\_128\_0\_rd\_fifo\_data, DIR = O, VEC = [31:0]

36 PORT npi\_rd\_ready = npi\_128\_0\_npi\_rd\_ready, DIR = O

37 PORT npi\_rd\_fifo\_full = npi\_128\_0\_rd\_fifo\_full, DIR = O

38 PORT npi\_rd\_fifo\_empty = npi\_128\_0\_rd\_fifo\_empty, DIR = O

39 PORT led = axi\_gpio\_1\_GPIO\_IO\_O, DIR = O, VEC = [7:0]

라인 200부터 256까지는 메모리 콘트롤러에 대한 정의 입니다.

200 BEGIN axi\_7series\_ddrx

201 PARAMETER INSTANCE = axi\_7series\_ddrx\_0

202 PARAMETER HW\_VER = 1.03.a

203 PARAMETER C\_MEM\_PARTNO = CUSTOM

204 PARAMETER C\_CK\_WIDTH = 2

205 PARAMETER C\_DM\_WIDTH = 4

206 PARAMETER C\_DQ\_WIDTH = 32

207 PARAMETER C\_DQS\_WIDTH = 4

208 PARAMETER C\_DRAM\_TYPE = DDR2

209 PARAMETER C\_NCS\_PER\_RANK = 2

210 PARAMETER C\_RTT\_NOM = 75

현재 메모리의 마스터로서는 npi\_128과 프로세서의 캐시 콘트롤러 입니다.

211 PARAMETER C\_INTERCONNECT\_S\_AXI\_MASTERS = npi\_128\_0.M\_AXI & microblaze\_0.M\_AXI\_DC

외부에는 아래 파트번호에 해당하는 메모리가 연결되어 있습니다.

212 PARAMETER C\_MEM\_BASEPARTNO = MT47H128M16XX-25E

메모리 파트에 해당하는 파라미터를 정의했습니다.

213 PARAMETER C\_BANK\_WIDTH = 2

214 PARAMETER C\_TCKE = 7500

215 PARAMETER C\_TFAW = 45000

216 PARAMETER C\_TPRDI = 1000000

217 PARAMETER C\_TRAS = 40000

218 PARAMETER C\_TRCD = 15000

219 PARAMETER C\_TREFI = 7800000

220 PARAMETER C\_TRFC = 197500

221 PARAMETER C\_TRP = 12500

222 PARAMETER C\_TRRD = 10000

223 PARAMETER C\_TRTP = 7500

224 PARAMETER C\_TWTR = 7500

225 PARAMETER C\_TZQI = 128000000

226 PARAMETER C\_TZQCS = 64

227 # PARAMETER C\_SIM\_BYPASS\_INIT\_CAL = SIM\_INIT\_CAL\_FULL

228 PARAMETER C\_S\_AXI\_BASEADDR = 0x50000000

229 PARAMETER C\_S\_AXI\_HIGHADDR = 0x5fffffff

이 부분이 128로 된 이유는 현재 PHY가 4:1 모드로 정해져 있고 외부 메모리가 32비트인 것을 가정했기 때문 입니다.

230 PARAMETER C\_S\_AXI\_DATA\_WIDTH = 128

231 # PARAMETER C\_SIM\_BYPASS\_INIT\_CAL = SKIP # , results in modelsim stop, but it is fastest way open a wecase

232 # PARAMETER C\_SIM\_BYPASS\_INIT\_CAL = OFF, results in too slow

233 BUS\_INTERFACE S\_AXI = axi4\_0

234 PORT freq\_refclk = clk\_400\_0000MHz1PLLE0\_nobuf

235 PORT mem\_refclk = clk\_400\_0000MHzPLLE0\_nobuf

236 PORT sync\_pulse = clk\_25\_0000MHz10PLLE0\_nobuf

237 PORT clk = clk\_100\_0000MHzPLLE0

238 PORT clk\_ref = clk\_200\_0000MHzPLLE0

239 PORT ddr\_cke = axi\_7series\_ddrx\_0\_ddr\_cke

240 PORT ddr\_dqs\_p = ddr\_dqs\_p

241 PORT ddr\_dm = axi\_7series\_ddrx\_0\_ddr\_dm

242 PORT ddr\_parity = axi\_7series\_ddrx\_0\_ddr\_parity

243 PORT ddr\_dqs\_n = ddr\_dqs\_n

244 PORT ddr\_cas\_n = axi\_7series\_ddrx\_0\_ddr\_cas\_n

245 PORT ddr\_ba = axi\_7series\_ddrx\_0\_ddr\_ba

246 PORT ddr\_ck\_n = axi\_7series\_ddrx\_0\_ddr\_ck\_n

247 PORT ddr\_dq = ddr\_dq

248 PORT ddr\_cs\_n = axi\_7series\_ddrx\_0\_ddr\_cs\_n

249 PORT ddr\_ck\_p = axi\_7series\_ddrx\_0\_ddr\_ck\_p

250 PORT ddr\_odt = axi\_7series\_ddrx\_0\_ddr\_odt

251 PORT ddr\_ras\_n = axi\_7series\_ddrx\_0\_ddr\_ras\_n

252 PORT ddr\_addr = axi\_7series\_ddrx\_0\_ddr\_addr

253 PORT ddr\_we\_n = axi\_7series\_ddrx\_0\_ddr\_we\_n

254 PORT pll\_lock = proc\_sys\_reset\_0\_Dcm\_locked

255 PORT init\_calib\_complete = axi\_7series\_ddrx\_0\_init\_calib\_complete

256 END

257

다음은 npi\_128에 대한 정의 입니다.

265 BEGIN npi\_128

266 PARAMETER INSTANCE = npi\_128\_0

267 PARAMETER HW\_VER = 1.00.a

268 PARAMETER C\_BASEADDR = 0x71000000

269 PARAMETER C\_HIGHADDR = 0x7100FFFF

270 PARAMETER C\_INTERCONNECT\_M\_AXI\_READ\_FIFO\_DEPTH = 512

271 BUS\_INTERFACE M\_AXI = axi4\_0

272 BUS\_INTERFACE S\_AXI = axi4lite\_0

273 PORT S\_AXI\_ACLK = clk\_100\_0000MHzPLLE0

274 PORT m\_axi\_aclk = clk\_100\_0000MHzPLLE0

275 PORT rd\_fifo\_clk = npi\_128\_0\_rd\_fifo\_clk

276 PORT rd\_fifo\_rd\_en = npi\_128\_0\_rd\_fifo\_rd\_en

277 PORT rd\_fifo\_data = npi\_128\_0\_rd\_fifo\_data

278 PORT npi\_rd\_ready = npi\_128\_0\_npi\_rd\_ready

279 PORT rd\_fifo\_full = npi\_128\_0\_rd\_fifo\_full

280 PORT rd\_fifo\_empty = npi\_128\_0\_rd\_fifo\_empty

281 END

282

메모리 콘트롤러가 켈리브레이션을 마쳤는지 검사하기 위한 GPIO입니다.

283 BEGIN axi\_gpio

284 PARAMETER INSTANCE = axi\_gpio\_0

285 PARAMETER HW\_VER = 1.01.b

286 PARAMETER C\_GPIO\_WIDTH = 1

287 PARAMETER C\_ALL\_INPUTS = 1

288 PARAMETER C\_BASEADDR = 0x40000000

289 PARAMETER C\_HIGHADDR = 0x4000ffff

290 BUS\_INTERFACE S\_AXI = axi4lite\_0

291 PORT S\_AXI\_ACLK = clk\_100\_0000MHzPLLE0

292 PORT GPIO\_IO\_I = axi\_7series\_ddrx\_0\_init\_calib\_complete

293 END

294

시뮬레이션 상태를 알기 위해 LED GPIO를 하나 설계 했습니다.

295 BEGIN axi\_gpio

296 PARAMETER INSTANCE = axi\_gpio\_1

297 PARAMETER HW\_VER = 1.01.b

298 PARAMETER C\_GPIO\_WIDTH = 8

299 PARAMETER C\_BASEADDR = 0x40040000

300 PARAMETER C\_HIGHADDR = 0x4004ffff

301 BUS\_INTERFACE S\_AXI = axi4lite\_0

302 PORT S\_AXI\_ACLK = clk\_100\_0000MHzPLLE0

303 PORT GPIO\_IO\_O = axi\_gpio\_1\_GPIO\_IO\_O

304 END

305

## system\_top.vhd 설명

다음 리스트는 임베디드 시스템을 sub module로 하는 top 파일의 일부 코드 입니다.

라인 32~54는 FPGA와 연결되는 외부 핀에 대한 정의 입니다. 메모리 인터럽트와 UART, RESET 및 클럭 정보등이 있습니다.

31

32 entity top is

33 PORT(

34 RS232\_Uart\_1\_sin : IN std\_logic;

35 clk245\_76mhz : IN std\_logic;

36 RESET : IN std\_logic;

37 CLK\_P : IN std\_logic;

38 CLK\_N : IN std\_logic;

39 ddr\_dqs\_p : INOUT std\_logic\_vector(3 downto 0);

40 ddr\_dqs\_n : INOUT std\_logic\_vector(3 downto 0);

41 ddr\_dq : INOUT std\_logic\_vector(31 downto 0);

42 RS232\_Uart\_1\_sout : OUT std\_logic;

43 led : OUT std\_logic\_vector(7 downto 0);

44 ddr\_cke : OUT std\_logic\_vector(0 to 0);

45 ddr\_dm : OUT std\_logic\_vector(3 downto 0);

46 ddr\_cas\_n : OUT std\_logic;

47 ddr\_ba : OUT std\_logic\_vector(1 downto 0);

48 ddr\_ck\_n : OUT std\_logic\_vector(1 downto 0);

49 ddr\_cs\_n : OUT std\_logic\_vector(1 downto 0);

50 ddr\_ck\_p : OUT std\_logic\_vector(1 downto 0);

51 ddr\_odt : OUT std\_logic\_vector(1 downto 0);

52 ddr\_ras\_n : OUT std\_logic;

53 ddr\_addr : OUT std\_logic\_vector(12 downto 0);

54 ddr\_we\_n : OUT std\_logic

55 );

56 end top;

라인 100부터 127까지는 포트 맵을 정했으며 라인 129에서는 npi\_128로 부터 데이터를 읽기 위한 조건을 언급하였습니다.

57

58 architecture Behavioral of top is

59

99 begin

100 Inst\_system: system PORT MAP(

101 RS232\_Uart\_1\_sout =>RS232\_Uart\_1\_sout ,

102 RS232\_Uart\_1\_sin =>RS232\_Uart\_1\_sin ,

103 RESET =>RESET ,

104 CLK\_P =>CLK\_P ,

105 CLK\_N =>CLK\_N ,

106 led => led,

107 ddr\_cke =>ddr\_cke ,

108

124 npi\_rd\_ready =>npi\_rd\_ready,

125 npi\_rd\_fifo\_full =>npi\_rd\_fifo\_full,

126 npi\_rd\_fifo\_empty =>npi\_rd\_fifo\_empty

127 );

128

129 npi\_rd\_fifo\_rd\_en <= not npi\_rd\_fifo\_empty when npi\_rd\_ready = '1' else '0';

130

131 end Behavioral;

132

다음 리스트는 시뮬레이션을 위한 테스트 벤치 일부 코드 입니다.

라인 35부터 66까지는 포트에 대한 정의를 보여 주고 있습니다.

34

35 ENTITY tb\_bench IS

36 END tb\_bench;

37

38 ARCHITECTURE behavior OF tb\_bench IS

39

40 -- Component Declaration for the Unit Under Test (UUT)

41

42 COMPONENT top

43 PORT(

44 RS232\_Uart\_1\_sin : IN std\_logic;

45 RESET : IN std\_logic;

46 CLK\_P : IN std\_logic;

47 CLK\_N : IN std\_logic;

48 clk245\_76mhz : IN std\_logic;

49 led : OUT std\_logic\_vector(7 downto 0);

50 ddr\_dqs\_p : INOUT std\_logic\_vector(3 downto 0);

51 ddr\_dqs\_n : INOUT std\_logic\_vector(3 downto 0);

52 ddr\_dq : INOUT std\_logic\_vector(31 downto 0);

53 RS232\_Uart\_1\_sout : OUT std\_logic;

54 ddr\_cke : OUT std\_logic\_vector(0 to 0);

55 ddr\_dm : OUT std\_logic\_vector(3 downto 0);

56 ddr\_cas\_n : OUT std\_logic;

57 ddr\_ba : OUT std\_logic\_vector(1 downto 0);

58 ddr\_ck\_n : OUT std\_logic\_vector(1 downto 0);

59 ddr\_cs\_n : OUT std\_logic\_vector(1 downto 0);

60 ddr\_ck\_p : OUT std\_logic\_vector(1 downto 0);

61 ddr\_odt : OUT std\_logic\_vector(1 downto 0);

62 ddr\_ras\_n : OUT std\_logic;

63 ddr\_addr : OUT std\_logic\_vector(12 downto 0);

64 ddr\_we\_n : OUT std\_logic

65 );

66 END COMPONENT;

라인 71부터 88까지는 현재 사용 중인 메모리에 대한 시뮬레이션 모델을 정의하였습니다.

67

68 -- No clocks detected in port list. Replace <clock> below with

69 -- appropriate port name

70 COMPONENT ddr2

71 --COMPONENT ddr2\_model

72 PORT(

73 ck : IN std\_logic;

74 ck\_n : IN std\_logic;

75 cke : IN std\_logic;

76 cs\_n : IN std\_logic;

77 ras\_n : IN std\_logic;

78 cas\_n : IN std\_logic;

79 we\_n : IN std\_logic;

80 dm\_rdqs : INOUT std\_logic\_vector(1 downto 0);

81 ba : IN std\_logic\_vector(1 downto 0);

82 addr : IN std\_logic\_vector(12 downto 0);

83 dq : INOUT std\_logic\_vector(15 downto 0);

84 dqs : INOUT std\_logic\_vector(1 downto 0);

85 dqs\_n : INOUT std\_logic\_vector(1 downto 0);

86 rdqs\_n: OUT std\_logic\_vector(1 downto 0);

87 odt : in std\_logic);

88 END COMPONENT;

89

111 BEGIN

112 -- fpga\_0\_clk\_1\_sys\_clk\_pin <= not fpga\_0\_clk\_1\_sys\_clk\_pin after 4 ns; --- 125mhz

113 -- fpga\_0\_clk\_1\_sys\_clk\_pin <= not fpga\_0\_clk\_1\_sys\_clk\_pin after 20 ns; --- 25mhz

114

115 CLK\_P <= not CLK\_P after 5.0 ns;

116 CLK\_N <= not CLK\_P;

117 RESET <= '1' after 500 ns;

라인 118에서는 npi\_rd\_fifo\_clk에 공급될 클럭을 정의하고 있습니다. 현재는 245.76Mhz로 동작을 시키고 있습니다.

118 clk245\_76mhz <= not clk245\_76mhz after 2.034 ns;

119 --clk245\_76mhz <= not clk245\_76mhz after 1.034 ns;

라인 120부터 179까지 메모리 모델과 시스템간의 연결 상태를 보여 줍니다.

120

121 -- Instantiate the Unit Under Test (UUT)

122 uut: top PORT MAP (

123 RS232\_Uart\_1\_sin => RS232\_Uart\_1\_sin,

124 clk245\_76mhz => clk245\_76mhz,

125 RESET => RESET,

126 CLK\_P => CLK\_P,

127 CLK\_N => CLK\_N,

128 ddr\_dqs\_p => ddr\_dqs\_p,

129 ddr\_dqs\_n => ddr\_dqs\_n,

130 ddr\_dq => ddr\_dq,

131 led => led,

132 RS232\_Uart\_1\_sout => RS232\_Uart\_1\_sout,

133 ddr\_cke => ddr\_cke,

134 ddr\_dm => ddr\_dm,

135 ddr\_cas\_n => ddr\_cas\_n,

136 ddr\_ba => ddr\_ba,

137 ddr\_ck\_n => ddr\_ck\_n,

138 ddr\_ck\_p => ddr\_ck\_p,

139 ddr\_cs\_n => ddr\_cs\_n,

140 ddr\_odt => ddr\_odt,

141 ddr\_ras\_n => ddr\_ras\_n,

142 ddr\_addr => ddr\_addr,

143 ddr\_we\_n => ddr\_we\_n

144 );

현재 외부에는 16비트 메모리 2개가 준비되어 있습니다. 특별히 160라인의 rdqs\_n은 open 처리 하였습니다.

145

146 ddr2\_u0 : ddr2 port map (

147 ck => ddr\_ck\_p(0),

148 ck\_n => ddr\_ck\_n(0),

149 cke => ddr\_cke(0),

150 cs\_n => ddr\_cs\_n(0),

151 ras\_n => ddr\_ras\_n,

152 cas\_n => ddr\_cas\_n,

153 we\_n => ddr\_we\_n,

154 ba => ddr\_ba,

155 addr => ddr\_addr,

156 dq => ddr\_dq(15 downto 0),

157 dqs => ddr\_dqs\_p(1 downto 0),

158 dqs\_n => ddr\_dqs\_n(1 downto 0),

159 dm\_rdqs => ddr\_dm(1 downto 0),

160 rdqs\_n => open,

161 odt => ddr\_odt(0));

162

163 ddr2\_u1 : ddr2 port map (

164 ck => ddr\_ck\_p(1),

165 ck\_n => ddr\_ck\_n(1),

166 cke => ddr\_cke(0),

167 cs\_n => ddr\_cs\_n(1),

168 ras\_n => ddr\_ras\_n,

169 cas\_n => ddr\_cas\_n,

170 we\_n => ddr\_we\_n,

171 ba => ddr\_ba,

172 addr => ddr\_addr,

173 dq => ddr\_dq(31 downto 16),

174 dqs => ddr\_dqs\_p(3 downto 2),

175 dqs\_n =>ddr\_dqs\_n(3 downto 2),

176 dm\_rdqs => ddr\_dm (3 downto 2),

177 rdqs\_n => open,

178 odt => ddr\_odt(1));

1. D;

# Custome Axi Master, NPI\_V4\_10\_a

## Npi\_v4\_10\_a 설명

### User\_logic.vhd

새로 추가된 npi\_v4\_10\_a의 user\_logic.vhd에 대한 리스트 입니다.

>>>>>>>

52 ------------------------------------------------------------------------------

53 -- npi\_v4\_10\_a is for fumate

54 -- tx logic would start when tx end end

55 -- rx logic is same with npi\_v4\_00\_a

56 -- do not edit below this line --------------------

57 library ieee;

58 use ieee.std\_logic\_1164.all;

59 use ieee.std\_logic\_arith.all;

60 use ieee.std\_logic\_unsigned.all;

61

라인 62~64는 async fifo를 사용할 때 inferrencing 방식을 사용하기 위한 라이브러리를 등록 합니다.

62 library proc\_common\_v3\_00\_a;

63 use proc\_common\_v3\_00\_a.all;

64 use proc\_common\_v3\_00\_a.family.all;

65

66 -- synopsys translate\_off

67 library xilinxcorelib;

68 library unisim;

69 library simprim;

70 -- synopsys translate\_on

71

72

>>>>>>>>>

129 entity user\_logic is

130 generic

131 (

132 -- add user generics below this line ---------------

133 --user generics added here

라인 134에서 npi에서 사용할 fifo의 크기를 정 합니다.

134 c\_fifo\_size : integer := 1024;

135 -- add user generics above this line ---------------

136

137 c\_family : string := "virtex6";

138 -- do not edit below this line ---------------------

139 -- bus protocol parameters, do not add to or delete

라인 140~144에서 중요한 파라미터는 c\_length\_width입니다. 일반적으로 이 값은 12비트 입니다. 하지만 12비트만 가지고는 최대 지정할 수 있는 비트수가 2^12이기 때문에 이 값을 20비트로 증가시켜야 합니다.

현재 이 파라미터는 상위모듈에서 전달되기 때문에 현재 12라는 값이 보이더라도 그 값을 그대로 믿으면 안됩니다. 반드시 상위 모듈에서 어떤 값으로 전달 되는지 살펴봐야 합니다.

140 c\_mst\_native\_data\_width : integer := 32;

141 c\_length\_width : integer := 12;

142 c\_mst\_awidth : integer := 32;

143 c\_num\_reg : integer := 12;

144 c\_slv\_dwidth : integer := 32

145 -- do not edit above this line ---------------------

146 );

147 port

148 (

npi에서 사용하는 fifo 인터페이스 신호를 정의 합니다.

라인 151~165를 보면 fifo 인터페이스를 그대로 사용하고 있다는 것을 확인할 수 있습니다.

한가지 다른 점은 라인 156, 157, 165, 165 에 있는 ready, interrupt 포트 입니다.

Ready는 현재 NPI 내부의 state machine이 동작하는 데 필요한 자료를 모두 프로세서에게 전달 받은면 assert되고 interrupt는 npi가 각각 rd/wr 동작을 마치면 인터럽트 요청을 합니다.

149 -- add user ports below this line ------------------

150 --user ports added here

151 rd\_fifo\_clk : in std\_logic;

152 rd\_fifo\_rd\_en : in std\_logic;

153 rd\_fifo\_data : out std\_logic\_vector(31 downto 0);

154 rd\_fifo\_empty : out std\_logic;

155 rd\_fifo\_full : out std\_logic;

156 rd\_fifo\_ready : out std\_logic;

157 rd\_fifo\_interrupt : out std\_logic;

158 ----------------------------------------------------

159 wr\_fifo\_clk : in std\_logic;

160 wr\_fifo\_wr\_en : in std\_logic;

161 wr\_fifo\_data : in std\_logic\_vector(31 downto 0);

162 wr\_fifo\_empty : out std\_logic;

163 wr\_fifo\_full : out std\_logic;

164 wr\_fifo\_ready : out std\_logic;

165 wr\_fifo\_interrupt : out std\_logic;

166

167 -- add user ports above this line ------------------

168

라인 169~211까지는 master로직에 필요한 IPIC신호들을 보여 줍니다.

169 -- do not edit below this line ---------------------

170 -- bus protocol ports, do not add to or delete

171 bus2ip\_clk : in std\_logic;

172 bus2ip\_resetn : in std\_logic;

173 bus2ip\_data : in std\_logic\_vector(c\_slv\_dwidth-1 downto 0);

174 bus2ip\_be : in std\_logic\_vector(c\_slv\_dwidth/8-1 downto 0);

175 bus2ip\_rdce : in std\_logic\_vector(c\_num\_reg-1 downto 0);

176 bus2ip\_wrce : in std\_logic\_vector(c\_num\_reg-1 downto 0);

177 ip2bus\_data : out std\_logic\_vector(c\_slv\_dwidth-1 downto 0);

178 ip2bus\_rdack : out std\_logic;

179 ip2bus\_wrack : out std\_logic;

180 ip2bus\_error : out std\_logic;

라인 181~209까지는 주로 마스터 로직에서 필요로 하는 포트에 대한 정의 입니다. 가능하면 라인별로 설명하도록 하겠습니다.

마스터 IP가 인터커넥터에게 Rd Req 하는 포트 입니다. Ack 신호가 되돌아 올 때까지 신호를 유지 합니다.

181 ip2bus\_mstrd\_req : out std\_logic;

182 ip2bus\_mstwr\_req : out std\_logic;

마스터 IP가 억세스 하고하는 어드레스 값을 전달하는 포트 입니다.

183 ip2bus\_mst\_addr : out std\_logic\_vector(c\_mst\_awidth-1 downto 0);

데이터 버스 중 어느 바이트가 의미 있는 알려주는 포트 입니다.

184 ip2bus\_mst\_be : out std\_logic\_vector((c\_mst\_native\_data\_width/8)-1 downto 0);

마스터가 억세스하고자 하는 바이트 수를 나타내는 포트 입니다.

185 ip2bus\_mst\_length : out std\_logic\_vector(c\_length\_width-1 downto 0);

마스터가 억세스 할 때 burst 방식인지 아닌지 알려주는 포트 입니다.

186 ip2bus\_mst\_type : out std\_logic;

187 ip2bus\_mst\_lock : out std\_logic;

188 ip2bus\_mst\_reset : out std\_logic;

마스터 IP가 Rd/Wr 요청을 하면 인터커넥터가 응답하는 포트 입니다. 만약 잘못된 방법 (예를 들어 어드레스가 잘못되거나 length가 0인 경우)으로 요청이 와도 이 포트를 통해 인터커넥터는 응답을 하지만 라인 191에 있는 error 를 같이 assert 시켜줍니다.

따라서 마스터 IP는 ack 뿐 아니라 error도 같이 모니터링해야 하는데 실제 state machine에서는 이런 부분들이 반영되어 있습니다.

189 bus2ip\_mst\_cmdack : in std\_logic;

마스터 IP의 억세스 요청에 따른 데이터 전송이 끝나면 assert됩니다.

190 bus2ip\_mst\_cmplt : in std\_logic;

191 bus2ip\_mst\_error : in std\_logic;

192 bus2ip\_mst\_rearbitrate : in std\_logic;

193 bus2ip\_mst\_cmd\_timeout : in std\_logic;

마스터 IP가 Rd 할 때 데이터가 들어오는 포트 입니다.

194 bus2ip\_mstrd\_d : in std\_logic\_vector(c\_mst\_native\_data\_width-1 downto 0);

32비트 데이터 폭을 가지고 있는 경우, 14바이트만 읽으려고 한다면 3번째까지는 데이터 폭을 다 쓰지만 마지막에는 2바이트만 읽어야 합니다. 이때 남아있는 바이트수가 얼마인지 알려주는 포트 입니다.

195 bus2ip\_mstrd\_rem : in std\_logic\_vector((c\_mst\_native\_data\_width)/8-1 downto 0);

마스터의 RD 요청에 따라 인터커넥터가 데이터를 전달하기 시작할 때와 끝날 때 한번씩 assert 되는 포트 입니다.

196 bus2ip\_mstrd\_sof\_n : in std\_logic;

197 bus2ip\_mstrd\_eof\_n : in std\_logic;

마스터의 RD 요청에 인터커넥터 상에 데이터가 준비되어 있으면 assert 됩니다.

198 bus2ip\_mstrd\_src\_rdy\_n : in std\_logic;

Burst 전송을 강제로 종료 된 것을 알려주는 포트 입니다.

199 bus2ip\_mstrd\_src\_dsc\_n : in std\_logic;

200 ip2bus\_mstrd\_dst\_rdy\_n : out std\_logic;

201 ip2bus\_mstrd\_dst\_dsc\_n : out std\_logic;

라인 202부터는 마스터 IP 쓰기 요청에 수행하기 위해 필요한 포트를 보여 줍니다.

202 ip2bus\_mstwr\_d : out std\_logic\_vector(c\_mst\_native\_data\_width-1 downto 0);

203 ip2bus\_mstwr\_rem : out std\_logic\_vector((c\_mst\_native\_data\_width)/8-1 downto 0);

204 ip2bus\_mstwr\_src\_rdy\_n : out std\_logic;

205 ip2bus\_mstwr\_src\_dsc\_n : out std\_logic;

206 ip2bus\_mstwr\_sof\_n : out std\_logic;

207 ip2bus\_mstwr\_eof\_n : out std\_logic;

208 bus2ip\_mstwr\_dst\_rdy\_n : in std\_logic;

209 bus2ip\_mstwr\_dst\_dsc\_n : in std\_logic

210 -- do not edit above this line ---------------------

211 );

212

213 attribute max\_fanout : string;

214 attribute sigis : string;

215

216 attribute sigis of bus2ip\_clk : signal is "clk";

217 attribute sigis of bus2ip\_resetn : signal is "rst";

218 attribute sigis of ip2bus\_mst\_reset: signal is "rst";

219

220 end entity user\_logic;

221

222 ------------------------------------------------------------------------------

223 -- architecture section

224 ------------------------------------------------------------------------------

225

226 architecture imp of user\_logic is

227

>>>>>>>>

238 signal slv\_reg7 : std\_logic\_vector(c\_slv\_dwidth-1 downto 0);

239 signal slv\_reg\_write\_sel : std\_logic\_vector(7 downto 0);

240 signal slv\_reg\_read\_sel : std\_logic\_vector(7 downto 0);

241 signal slv\_ip2bus\_data : std\_logic\_vector(c\_slv\_dwidth-1 downto 0);

242 signal slv\_read\_ack : std\_logic;

243 signal slv\_write\_ack : std\_logic;

244

245 ------------------------------------------

246 -- signals for user logic master model example

247 ------------------------------------------

마스터에서 필요한 여러 신호들을 정의 합니다.

248 -- signals for master model control/status registers write/read

249 signal mst\_ip2bus\_data : std\_logic\_vector(c\_slv\_dwidth-1 downto 0);

250 signal mst\_reg\_write\_req : std\_logic;

251 signal mst\_reg\_read\_req : std\_logic;

252 signal mst\_reg\_write\_sel : std\_logic\_vector(3 downto 0);

253 signal mst\_reg\_read\_sel : std\_logic\_vector(3 downto 0);

254 signal mst\_write\_ack : std\_logic;

255 signal mst\_read\_ack : std\_logic;

256 -- signals for master model control/status registers

257 type byte\_reg\_type is array(0 to 15) of std\_logic\_vector(7 downto 0);

라인 258은 마스터 로직에서 읽기 또는 쓰기 동작 요청 신호 입니다.

258 signal mst\_reg : byte\_reg\_type;

259 signal mst\_byte\_we : std\_logic\_vector(15 downto 0);

260 signal mst\_cntl\_rd\_req : std\_logic;

261 signal mst\_cntl\_wr\_req : std\_logic;

262 signal mst\_cntl\_bus\_lock : std\_logic;

263 signal mst\_cntl\_burst : std\_logic;

라인 264은 마스터가 억세스하고자 하는 어드레스 값을 정합니다.

264 signal mst\_ip2bus\_addr : std\_logic\_vector(c\_mst\_awidth-1 downto 0);

라인 265는 마스터가 억세스하려는 데이터 크기 입니다.

265 signal mst\_xfer\_length : std\_logic\_vector(c\_length\_width-1 downto 0);

266 signal mst\_xfer\_reg\_len : std\_logic\_vector(19 downto 0);

267 signal mst\_xfer\_reg\_len\_rd : std\_logic\_vector(19 downto 0);

268 signal mst\_xfer\_reg\_len\_wr : std\_logic\_vector(19 downto 0);

269 signal mst\_ip2bus\_be : std\_logic\_vector(15 downto 0);

라인 270은 내부 state machine이 동작하고 있다는 것을 알려 줍니다.

270 signal mst\_go : std\_logic;

271 -- signals for master model command interface state machine

272 type cmd\_cntl\_sm\_type is (cmd\_idle, cmd\_run, cmd\_wait\_for\_data, finish, cmd\_done);

273 signal mst\_cmd\_sm\_state : cmd\_cntl\_sm\_type;

274 signal mst\_cmd\_sm\_set\_done : std\_logic;

275 signal mst\_cmd\_sm\_set\_error : std\_logic;

276 signal mst\_cmd\_sm\_set\_timeout : std\_logic;

277 signal mst\_cmd\_sm\_busy : std\_logic;

278 signal mst\_cmd\_sm\_clr\_go : std\_logic;

279 signal mst\_cmd\_sm\_rd\_req : std\_logic;

280 signal mst\_cmd\_sm\_wr\_req : std\_logic;

281 signal mst\_cmd\_sm\_reset : std\_logic;

282 signal mst\_cmd\_sm\_bus\_lock : std\_logic;

283 signal mst\_cmd\_sm\_ip2bus\_addr : std\_logic\_vector(c\_mst\_awidth-1 downto 0);

284 signal mst\_cmd\_sm\_ip2bus\_be : std\_logic\_vector(c\_mst\_native\_data\_width/8-1 downto 0);

285 signal mst\_cmd\_sm\_xfer\_type : std\_logic;

286 signal mst\_cmd\_sm\_xfer\_length : std\_logic\_vector(c\_length\_width-1 downto 0);

287 signal mst\_cmd\_sm\_start\_rd\_llink : std\_logic;

288 signal mst\_cmd\_sm\_start\_wr\_llink : std\_logic;

289 -- signals for master model read locallink interface state machine

290 type rd\_llink\_sm\_type is (llrd\_idle, llrd\_go);

291 signal mst\_llrd\_sm\_state : rd\_llink\_sm\_type;

292 signal mst\_llrd\_sm\_dst\_rdy : std\_logic;

293 -- signals for master model write locallink interface state machine

294 type wr\_llink\_sm\_type is (llwr\_idle, llwr\_sngl\_init, llwr\_sngl, llwr\_brst\_init, llwr\_brst, llwr\_brst\_last\_beat);

295 signal mst\_llwr\_sm\_state : wr\_llink\_sm\_type;

296 signal mst\_llwr\_sm\_src\_rdy : std\_logic;

297 signal mst\_llwr\_sm\_sof : std\_logic;

298 signal mst\_llwr\_sm\_eof : std\_logic;

299 signal mst\_llwr\_byte\_cnt : integer;

300 signal mst\_fifo\_valid\_write\_xfer : std\_logic;

301 signal mst\_fifo\_valid\_read\_xfer : std\_logic;

302 signal bus2ip\_reset : std\_logic;

303 attribute sigis of bus2ip\_reset : signal is "rst";

라인 304~320은 프로세서가 NPI의 동작상태를 제어하거나 모니터링할 때 필요한 신호들을 정의 합니다.

304 ------------------------------------------------------------------------------------------------------------------------------------------------------

305 ---- below register is used for user function,

306 ------------------------------------------------------------------------------------------------------------------------------------------------------

307 signal c\_cnt\_burst, cnt\_burst : std\_logic\_vector(11 downto 0) := x"000"; -- save burst count

308 signal c\_cnt\_repeat, cnt\_repeat: std\_logic\_vector(15 downto 0) := x"0000"; -- save how many repeat burst count, 0xffff means unlimit operation

309 signal cnt\_rx, cnt\_tx : std\_logic\_vector(3 downto 0) := x"0"; -- check whether rx, tx clock is operating

310 signal cnt\_rx\_bus, cnt\_tx\_bus : std\_logic\_vector(3 downto 0) := x"0"; -- check whether rx, tx clock is operating

311 signal mst\_sm\_sts : std\_logic\_vector(3 downto 0); -- to debug, uP can read master status machine activity

312 signal rd\_fifo\_full\_s, rd\_fifo\_empty\_s, wr\_fifo\_full\_s, wr\_fifo\_empty\_s : std\_logic; -- show fifo status

313 signal sm\_reset\_s : std\_logic; -- for uP to reset master status machine operation

314 signal fifo\_rst\_s : std\_logic; -- for uP to reset fifo, uP need to monitor fifo's full port until it goes low, fifo needs a time to reset its memory, full port is used to reset

315

316 signal rd\_fifo\_ready\_s, wr\_fifo\_ready\_s : std\_logic; -- show rd, wr state machine is ready to operate

317 signal wr\_fifo\_wr\_cnt\_s, rd\_fifo\_wr\_cnt\_s : std\_logic\_vector(1 downto 0); -- 00 means less than 25%, 01 means 25~50%, 10 menas 50~75 %, 11 menas more than 75%

318 signal npi\_ready\_sts : std\_logic\_vector(1 downto 0); -- save npi status

319 signal fifo\_sts : std\_logic\_vector(3 downto 0); -- save fifo status

320 signal mst\_cmd\_sts : std\_logic\_vector(3 downto 0); -- save mst error status

321 ------------------------------------------------------------------------------------------------------------------------------------------------------

322 ------------------------------------------------------------------------------------------------------------------------------------------------------

323

324 ----------------------------------------------------------------------------------------------------

325 -- npi\_v4\_10\_a module

326 ----------------------------------------------------------------------------------------------------

327 type st\_tx\_type is (tx0, tx1, tx2, tx3, tx4, tx5);

328 signal st\_tx : st\_tx\_type;

329 signal used\_for\_wr : std\_logic := '1';

330 signal fumate\_start\_tx, fumate\_start\_tx\_bus\_clk : std\_logic;

331 signal fumate\_tx\_data\_cnt, fumate\_tx\_data\_const, fumate\_tx\_data\_const\_bus\_clk : std\_logic\_vector(15 downto 0);

332 signal fumate\_base\_add, fumate\_base\_add\_bus\_clk : std\_logic\_vector(1 downto 0) := "00";

333 signal fumate\_tx\_int\_cnt, fumate\_wr\_en\_count : std\_logic\_vector(15 downto 0) := x"0000";

334

335 --- debug

336 signal ip2bus\_mstwr\_d\_fifo : std\_logic\_vector(31 downto 0);

337 signal ip2bus\_mstwr\_d\_s : std\_logic\_vector(7 downto 0) := (others => '0');

338

339 --- int debug

340 signal wr\_fifo\_interrupt\_s, rd\_fifo\_interrupt\_s : std\_logic;

341 signal cnt\_int : std\_logic\_vector(11 downto 0);

342 type st\_int\_type is (st\_0, st\_1, st\_2);

343 signal st\_int : st\_int\_type;

344 ----------------------------------------------------------------------------------------------------

345 ----------------------------------------------------------------------------------------------------

346 begin

347

348 --user logic implementation added here

349

350 ------------------------------------------

351 -- example code to read/write user logic slave model s/w accessible registers

352 --

353 -- note:

354 -- the example code presented here is to show you one way of reading/writing

355 -- software accessible registers implemented in the user logic slave model.

356 -- each bit of the bus2ip\_wrce/bus2ip\_rdce signals is configured to correspond

357 -- to one software accessible register by the top level template. for example,

358 -- if you have four 32 bit software accessible registers in the user logic,

359 -- you are basically operating on the following memory mapped registers:

360 --

361 -- bus2ip\_wrce/bus2ip\_rdce memory mapped register

362 -- "1000" c\_baseaddr + 0x0

363 -- "0100" c\_baseaddr + 0x4

364 -- "0010" c\_baseaddr + 0x8

365 -- "0001" c\_baseaddr + 0xc

366 --

367 ------------------------------------------

368 slv\_reg\_write\_sel <= bus2ip\_wrce(7 downto 0);

369 slv\_reg\_read\_sel <= bus2ip\_rdce(7 downto 0);

370 slv\_write\_ack <= bus2ip\_wrce(0) or bus2ip\_wrce(1) or bus2ip\_wrce(2) or bus2ip\_wrce(3) or bus2ip\_wrce(4) or bus2ip\_wrce(5) or bus2ip\_wrce(6) or bus2ip\_wrce(7);

371 slv\_read\_ack <= bus2ip\_rdce(0) or bus2ip\_rdce(1) or bus2ip\_rdce(2) or bus2ip\_rdce(3) or bus2ip\_rdce(4) or bus2ip\_rdce(5) or bus2ip\_rdce(6) or bus2ip\_rdce(7);

372

373 fifo\_sts <= wr\_fifo\_full\_s & wr\_fifo\_empty\_s & rd\_fifo\_full\_s & rd\_fifo\_empty\_s;

374 npi\_ready\_sts <= wr\_fifo\_ready\_s & rd\_fifo\_ready\_s;

375 mst\_cmd\_sts <= mst\_cmd\_sm\_set\_done & mst\_cmd\_sm\_set\_error & mst\_cmd\_sm\_set\_timeout & mst\_go;

376

377

NPI가 동작중인 상태들을 저장하는 레지스터 블록을 설계한 부분 입니다. 이 부분은 나중에 프로세서가 필요할 때 디바이스 드라이버를 통해 읽어 볼 수 있습니다.

378 slave\_reg\_read\_proc : process(slv\_reg\_read\_sel, cnt\_rx\_bus, cnt\_tx\_bus, mst\_ip2bus\_addr, mst\_sm\_sts, mst\_cmd\_sts, npi\_ready\_sts, fifo\_sts)

379 constant npi\_ver : std\_logic\_vector(9 downto 0) := "00" & x"41"; -- "00" & x"40" -- org npi, "00" & x"41" -- for fumate

380 begin

381 case slv\_reg\_read\_sel is

382 when "10000000" => slv\_ip2bus\_data <= npi\_ver & cnt\_rx\_bus & cnt\_tx\_bus & npi\_ready\_sts & mst\_sm\_sts & mst\_cmd\_sts & fifo\_sts;

383 when "01000000" => slv\_ip2bus\_data <= mst\_ip2bus\_addr;

384 when "00100000" => slv\_ip2bus\_data <= mst\_cntl\_burst & mst\_cntl\_bus\_lock & mst\_cntl\_wr\_req & mst\_cntl\_rd\_req & wr\_fifo\_wr\_cnt\_s & rd\_fifo\_wr\_cnt\_s & c\_cnt\_burst & cnt\_burst;

385 when "00010000" => slv\_ip2bus\_data <= c\_cnt\_repeat & cnt\_repeat;

386 when "00001000" => slv\_ip2bus\_data <= fumate\_wr\_en\_count & fumate\_tx\_int\_cnt;

387 when "00000100" => slv\_ip2bus\_data <= x"0000" & fumate\_tx\_data\_const\_bus\_clk;

388 when "00000010" => slv\_ip2bus\_data <= x"000" & mst\_cmd\_sm\_xfer\_length;

389 when "00000001" => slv\_ip2bus\_data <= x"000" & mst\_xfer\_reg\_len\_wr;

390 when others => slv\_ip2bus\_data <= (others => '0');

391 end case;

392 end process slave\_reg\_read\_proc;

Rx,Tx 클럭이 제대로 연결되었는지 알려주는 카운터 입니다.

393

394 process(rd\_fifo\_clk)

395 begin

396 if rd\_fifo\_clk'event and rd\_fifo\_clk = '1' then

397 cnt\_rx <= cnt\_rx + 1;

398 end if;

399 end process;

400

401

402 process(wr\_fifo\_clk)

403 begin

404 if wr\_fifo\_clk'event and wr\_fifo\_clk = '1' then

405 cnt\_tx <= cnt\_tx + 1;

406 end if;

407 end process;

408

카운터 값을 버스 클럭에 다시 싱크를 맞추어서 읽는 로직 입니다.

409 process(bus2ip\_clk)

410 begin

411 if rising\_edge(bus2ip\_clk) then

412 cnt\_rx\_bus <= cnt\_rx;

413 cnt\_tx\_bus <= cnt\_tx;

414

415 end if;

416 end process;

417

418

라인419~474는 프로세서가 state machine을 동작시키기 위해 어떤 레지스터를 어떻게 제어해야 하는지 보여 줍니다.

디바이스 드라이버는 이 해설을 기초로 해서 만들어져 있습니다.

419 -----------------------------------------

420 -- example code to demonstrate user logic master model functionality

421 --

422 -- note:

423 -- the example code presented here is to show you one way of stimulating

424 -- the axi4lite master interface under user control. it is provided for

425 -- demonstration purposes only and allows the user to exercise the axi4lite

426 -- master interface during test and evaluation of the template.

427 -- this user logic master model contains a 16-byte flattened register and

428 -- the user is required to initialize the value to desire and then write to

429 -- the model's 'go' port to initiate the user logic master operation.

430 --

431 -- control register (c\_baseaddr + offset + 0x0):

432 -- bit 0 - rd (read request control)

433 -- bit 1 - wr (write request control)

434 -- bit 2 - bl (bus lock control)

435 -- bit 3 - brst (burst assertion control)

436 -- bit 4-7 - spare (spare control bits)

437 -- status register (c\_baseaddr + offset + 0x1):

438 -- bit 0 - done (transfer done status)

439 -- bit 1 - busy (user logic master is busy)

440 -- bit 2 - error (user logic master request got error response)

441 -- bit 3 - tmout (user logic master request is timeout)

442 -- bit 2-7 - spare (spare status bits)

443 -- addrress register (c\_baseaddr + offset + 0x4):

444 -- bit 0-31 - target address (this 32-bit value is used to populate the

445 -- ip2bus\_mst\_addr(0:31) address bus during a read or write

446 -- user logic master operation)

447 -- byte enable register (c\_baseaddr + offset + 0x8):

448 -- bit 0-15 - master be (this 16-bit value is used to populate the

449 -- ip2bus\_mst\_be byte enable bus during a read or write user

450 -- logic master operation for single data beat transfer)

451 -- length register (c\_baseaddr + offset + 0xc):

452 -- bit 0-3 - reserved

453 -- bit 4-15 - transfer length (this 12-bit value is used to populate the

454 -- ip2bus\_mst\_length(0:11) transfer length bus which specifies

455 -- the number of bytes (1 to 4096) to transfer during user logic

456 -- master read or write fixed length burst operations)

457 -- go register (c\_baseaddr + offset + 0xf):

458 -- bit 0-7 - go port (write to this byte address initiates the user

459 -- logic master transfer, data key value of 0x0a must be used)

460 --

461 -- note: offset may be different depending on your address space configuration,

462 -- by default it's either 0x0 or 0x100. refer to ipif address range array

463 -- for actual value.

464 --

465 -- here's an example procedure in your software application to initiate a 4-byte

466 -- write operation (single data beat) of this master model:

467 -- 1. write 0x40 to the control register

468 -- 2. write the target address to the address register

469 -- 3. write valid byte lane value to the be register

470 -- - note: this value must be aligned with ip2bus address

471 -- 4. write 0x0004 to the length register

472 -- 5. write 0x0a to the go register, this will start the master write operation

473 --

474 ------------------------------------------

475 mst\_reg\_write\_req <= bus2ip\_wrce(8) or bus2ip\_wrce(9) or bus2ip\_wrce(10) or bus2ip\_wrce(11);

476 mst\_reg\_read\_req <= bus2ip\_rdce(8) or bus2ip\_rdce(9) or bus2ip\_rdce(10) or bus2ip\_rdce(11);

477 mst\_reg\_write\_sel <= bus2ip\_wrce(11 downto 8);

478 mst\_reg\_read\_sel <= bus2ip\_rdce(11 downto 8);

479 mst\_write\_ack <= mst\_reg\_write\_req;

480 mst\_read\_ack <= mst\_reg\_read\_req;

481

콘트롤러 레지스터의 각 비트를 state machine에서 필요로 하는 블록과 연결합니다. 여기서 레지스터의 각 비트를 억세스하는 코드는 참조할만 합니다.

482 -- rip control bits from master model registers

483 ------------ offset 0

484 mst\_cntl\_rd\_req <= mst\_reg(0)(0);

485 mst\_cntl\_wr\_req <= mst\_reg(0)(1);

486 mst\_cntl\_bus\_lock <= mst\_reg(0)(2);

487 mst\_cntl\_burst <= mst\_reg(0)(3);

488 ------------ offset 2

489 c\_cnt\_burst <= mst\_reg(3)(3 downto 0) & mst\_reg(2);

490 ------------ offset 4

491 -- mst\_ip2bus\_addr <= mst\_reg(7) & mst\_reg(6) & mst\_reg(5) & mst\_reg(4);

492 -- mst\_ip2bus\_addr <= mst\_reg(7)(7 downto 1) & cnt\_burst & x"00";

493 --- max pattern 307200

494 --- 307200/128 == 2400 == 0x960

495 --- every burst move 128byte ==> 0x80 ==> 0b1000\_0000

496 --- 24 + (12 + 1) + 7

497 -- npi\_v4\_00\_a mst\_ip2bus\_addr <= mst\_reg(7) & mst\_reg(6)(7 downto 3) & cnt\_burst & mst\_reg(5)(6 downto 0);

Npi를 rd로 사용할 때와 wr로 사용할 때 어드레스를 지정하는 방법이 차이가 있습니다.

498 mst\_ip2bus\_addr <= mst\_reg(7) & mst\_reg(6) & fumate\_base\_add\_bus\_clk & "00" & x"000";

499 used\_for\_wr <= mst\_reg(8)(0);

500 --mst\_ip2bus\_be <= mst\_reg(9) & mst\_reg(8);

501 mst\_ip2bus\_be <= x"ffff";

502 c\_cnt\_repeat <= mst\_reg(11) & mst\_reg(10);

503 mst\_xfer\_reg\_len\_rd <= x"00080";

504 mst\_xfer\_reg\_len\_wr <= "00" & fumate\_tx\_data\_const\_bus\_clk & "00";

505 mst\_xfer\_length <= mst\_xfer\_reg\_len\_wr(c\_length\_width - 1 downto 0 ) when used\_for\_wr = '1' else mst\_xfer\_reg\_len\_rd(c\_length\_width - 1 downto 0 );

506

507 -- implement byte write enable for each byte slice of the master model registers

508 master\_reg\_byte\_wr\_en : process( bus2ip\_be, mst\_reg\_write\_req, mst\_reg\_write\_sel ) is

509 constant be\_width : integer := c\_slv\_dwidth/8;

510 begin

511 for byte\_index in 0 to 15 loop

512 mst\_byte\_we(byte\_index) <= mst\_reg\_write\_req and

513 mst\_reg\_write\_sel(3 - (byte\_index/be\_width) ) and

514 bus2ip\_be(byte\_index- ((byte\_index/be\_width)\*be\_width));

515 ----------------------------------------------------------------------------------------------------

516 -- mst\_byte\_we(0) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 0/4) and bus2ip\_be(0 - ((0/4)\*4);

517 -- mst\_byte\_we(1) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 1/4) and bus2ip\_be(1 - ((1/4)\*4);

518 -- mst\_byte\_we(2) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 2/4) and bus2ip\_be(2 - ((2/4)\*4);

519 -- mst\_byte\_we(3) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 3/4) and bus2ip\_be(3 - ((3/4)\*4);

520 -- mst\_byte\_we(4) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 4/4) and bus2ip\_be(4 - ((4/4)\*4);

521 -- mst\_byte\_we(5) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 5/4) and bus2ip\_be(5 - ((5/4)\*4);

522 -- mst\_byte\_we(7) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 7/4) and bus2ip\_be(7 - ((7/4)\*4);

523 -- mst\_byte\_we(11) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 11/4) and bus2ip\_be(11 - ((11/4)\*4);

524 -- mst\_byte\_we(15) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3 - 15/4) and bus2ip\_be(15 - ((15/4)\*4);

525 --

526 -- mst\_byte\_we(0) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3) and bus2ip\_be(0);

527 -- mst\_byte\_we(1) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3) and bus2ip\_be(1);

528 -- mst\_byte\_we(2) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3) and bus2ip\_be(2);

529 -- mst\_byte\_we(3) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(3) and bus2ip\_be(3);

530 -- mst\_byte\_we(4) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(2) and bus2ip\_be(0);

531 -- mst\_byte\_we(5) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(2) and bus2ip\_be(1);

532 -- mst\_byte\_we(6) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(2) and bus2ip\_be(2);

533 -- mst\_byte\_we(7) <= mst\_reg\_write\_req and mst\_reg\_write\_sel(2) and bus2ip\_be(3);

534 ----------------------------------------------------------------------------------------------------

535 -- mst\_reg\_write\_sel(3) ==> master offset (0x100) + 0x0

536 -- mst\_reg\_write\_sel(2) ==> master offset (0x100) + 0x4

537 -- mst\_reg\_write\_sel(1) ==> master offset (0x100) + 0x8

538 -- mst\_reg\_write\_sel(0) ==> master offset (0x100) + 0xc

539 ----------------------------------------------------------------------------------------------------

540

541 end loop;

542

543 end process master\_reg\_byte\_wr\_en;

544

545 -- implement master model registers

546 master\_reg\_write\_proc : process( bus2ip\_clk ) is

547 constant be\_width : integer := c\_slv\_dwidth/8;

548 begin

549

550 if( bus2ip\_clk'event and bus2ip\_clk = '1' ) then

551 if( bus2ip\_resetn = '0' ) then

552 mst\_reg(0 to 14) <= (others => "00000000");

553 else

554 -- control register (byte 0)

555 if( mst\_byte\_we(0) = '1' ) then

556 mst\_reg(0) <= bus2ip\_data(7 downto 0);

557 -- mst\_reg(0)(0) used for rd\_req

558 -- mst\_reg(0)(1) used for wr\_req

559 -- mst\_reg(0)(2) used for bus\_lock

560 -- mst\_reg(0)(3) used for control\_burst

561 end if;

562 -- status register (byte 1)

563 mst\_reg(1)(1) <= mst\_cmd\_sm\_busy;

564 if( mst\_byte\_we(1) = '1' ) then

565 -- allows a clear of the 'done'/'error'/'timeout'

566 --mst\_reg(1)(0) <= bus2ip\_data((1-(1/be\_width)\*be\_width)\*8);

567 --mst\_reg(1)(2) <= bus2ip\_data((1-(1/be\_width)\*be\_width)\*8+2);

568 --mst\_reg(1)(3) <= bus2ip\_data((1-(1/be\_width)\*be\_width)\*8+3);

569

570 mst\_reg(1)(0) <= bus2ip\_data(8);

571 mst\_reg(1)(2) <= bus2ip\_data(10);

572 mst\_reg(1)(3) <= bus2ip\_data(11);

573 else

574 -- 'done'/'error'/'timeout' from master control state machine

575 mst\_reg(1)(0) <= mst\_cmd\_sm\_set\_done or mst\_reg(1)(0);

576 mst\_reg(1)(2) <= mst\_cmd\_sm\_set\_error or mst\_reg(1)(2);

577 mst\_reg(1)(3) <= mst\_cmd\_sm\_set\_timeout or mst\_reg(1)(3);

578 mst\_reg(1)(4) <= mst\_go;

579 end if;

580 -- byte 2 and 3 are reserved

581 -- address register (byte 4 to 7)

582 -- be register (byte 8 to 9)

583 -- length register (byte 12 to 13)

584 -- byte 10, 11 and 14 are reserved

585 for byte\_index in 2 to 14 loop

586 if( mst\_byte\_we(byte\_index) = '1' ) then

587 mst\_reg(byte\_index) <= bus2ip\_data(

588 (byte\_index-(byte\_index/be\_width)\*be\_width)\*8+7 downto

589 (byte\_index-(byte\_index/be\_width)\*be\_width)\*8);

590 end if;

591 end loop;

592 end if;

593 end if;

594 end process master\_reg\_write\_proc;

595

596 -- implement master model write only 'go' port

597 master\_write\_go\_port : process( bus2ip\_clk ) is

598 constant go\_data\_key : std\_logic\_vector(7 downto 0) := x"0a";

599 constant go\_byte\_lane : integer := 15;

600 constant be\_width : integer := c\_slv\_dwidth/8;

601 begin

602 if( bus2ip\_clk'event and bus2ip\_clk = '1' ) then

603 if( bus2ip\_resetn = '0' or mst\_cmd\_sm\_clr\_go = '1' or sm\_reset\_s = '1' ) then

604 mst\_go <= '0';

605 --elsif( mst\_cmd\_sm\_busy = '0' and mst\_byte\_we(go\_byte\_lane) = '1' and bus2ip\_data((go\_byte\_lane-(go\_byte\_lane/be\_width)\*be\_width)\*8+7 downto (go\_byte\_lane-(go\_byte\_lane/be\_width)\*be\_width)\*8)= go\_data\_key ) then

프로세서가 NPI에게 실행 명령을 해석하는 부분 입니다. 현재 NPI가 정지상태이고 마스터 억세스 레지스터 16번째 (15번)에 쓰는 동작을 하고 데이터 버스에 실린 데이터 값중 상위 8비트가 go\_data\_key와 같은 경우 mst\_go 레지스터 assert 됩니다.

606 elsif( mst\_cmd\_sm\_busy = '0' and mst\_byte\_we(15) = '1' and bus2ip\_data(31 downto 24) = go\_data\_key ) then

607 mst\_go <= '1';

608 else

609 null;

610 end if;

611 end if;

612 end process master\_write\_go\_port;

613

614 -- implement master model register read mux

615 master\_reg\_read\_proc : process( mst\_reg\_read\_sel, mst\_reg ) is

616 constant be\_width : integer := c\_slv\_dwidth/8;

617 begin

618 case mst\_reg\_read\_sel is

619 when "1000" =>

620 for byte\_index in 0 to be\_width-1 loop

621 mst\_ip2bus\_data(byte\_index\*8+7 downto byte\_index\*8) <= mst\_reg(byte\_index);

622 end loop;

623 when "0100" =>

624 for byte\_index in 0 to be\_width-1 loop

625 mst\_ip2bus\_data(byte\_index\*8+7 downto byte\_index\*8) <= mst\_reg(be\_width+byte\_index);

626 end loop;

627 when "0010" =>

628 for byte\_index in 0 to be\_width-1 loop

629 mst\_ip2bus\_data(byte\_index\*8+7 downto byte\_index\*8) <= mst\_reg(be\_width\*2+byte\_index);

630 end loop;

631 when "0001" =>

632 for byte\_index in 0 to be\_width-1 loop

633 if( byte\_index = be\_width-1 ) then

634 -- go port is not readable

635 mst\_ip2bus\_data(byte\_index\*8+7 downto byte\_index\*8) <= (others => '0');

636 else

637 mst\_ip2bus\_data(byte\_index\*8+7 downto byte\_index\*8) <= mst\_reg(be\_width\*3+byte\_index);

638 end if;

639 end loop;

640 when others =>

641 mst\_ip2bus\_data <= (others => '0');

642 end case;

643

644 end process master\_reg\_read\_proc;

645

646 -- user logic master command interface assignments

647 ip2bus\_mstrd\_req <= mst\_cmd\_sm\_rd\_req;

648 ip2bus\_mstwr\_req <= mst\_cmd\_sm\_wr\_req;

649 ip2bus\_mst\_addr <= mst\_cmd\_sm\_ip2bus\_addr;

650 ip2bus\_mst\_be <= mst\_cmd\_sm\_ip2bus\_be;

651 ip2bus\_mst\_type <= mst\_cmd\_sm\_xfer\_type;

652 ip2bus\_mst\_length <= mst\_cmd\_sm\_xfer\_length;

653 ip2bus\_mst\_lock <= mst\_cmd\_sm\_bus\_lock;

654 ip2bus\_mst\_reset <= mst\_cmd\_sm\_reset;

655

프로세서의 명령을 해석하는 state mahince 입니다.

656 --implement master command interface state machine

657 master\_cmd\_sm\_proc : process( bus2ip\_clk ) is

658 begin

659

660 if( bus2ip\_clk'event and bus2ip\_clk = '1' ) then

661 if( bus2ip\_resetn = '0' or sm\_reset\_s = '1') then

662 -- reset condition

663 mst\_cmd\_sm\_clr\_go <= '0';

664 mst\_cmd\_sm\_rd\_req <= '0';

665 mst\_cmd\_sm\_wr\_req <= '0';

666 mst\_cmd\_sm\_bus\_lock <= '0';

667 mst\_cmd\_sm\_reset <= '0';

668 mst\_cmd\_sm\_ip2bus\_addr <= (others => '0');

669 mst\_cmd\_sm\_ip2bus\_be <= (others => '0');

670 mst\_cmd\_sm\_xfer\_type <= '0';

671 mst\_cmd\_sm\_xfer\_length <= (others => '0');

672 mst\_cmd\_sm\_set\_done <= '0';

673 mst\_cmd\_sm\_set\_error <= '0';

674 mst\_cmd\_sm\_set\_timeout <= '0';

675 mst\_cmd\_sm\_busy <= '0';

676 mst\_cmd\_sm\_start\_rd\_llink <= '0';

677 mst\_cmd\_sm\_start\_wr\_llink <= '0';

678 cnt\_burst <= (others => '0');

679 cnt\_repeat <= (others => '0');

680 fumate\_tx\_int\_cnt <= (others => '0');

681 rd\_fifo\_interrupt\_s <= '0';

682 wr\_fifo\_interrupt\_s <= '0';

683 mst\_sm\_sts<= x"1";

684 mst\_cmd\_sm\_state <= cmd\_idle;

685 else

686

687 -- default condition

688 mst\_cmd\_sm\_clr\_go <= '0';

689 mst\_cmd\_sm\_rd\_req <= '0';

690 mst\_cmd\_sm\_wr\_req <= '0';

691 mst\_cmd\_sm\_bus\_lock <= '0';

692 mst\_cmd\_sm\_reset <= '0';

693 mst\_cmd\_sm\_ip2bus\_addr <= (others => '0');

694 mst\_cmd\_sm\_ip2bus\_be <= (others => '0');

695 mst\_cmd\_sm\_xfer\_type <= '0';

696 mst\_cmd\_sm\_xfer\_length <= (others => '0');

697 mst\_cmd\_sm\_set\_done <= '0';

698 mst\_cmd\_sm\_set\_error <= '0';

699 mst\_cmd\_sm\_set\_timeout <= '0';

700 mst\_cmd\_sm\_busy <= '1';

701 mst\_cmd\_sm\_start\_rd\_llink <= '0';

702 mst\_cmd\_sm\_start\_wr\_llink <= '0';

703 rd\_fifo\_interrupt\_s <= '0';

704 wr\_fifo\_interrupt\_s <= '0';

705 -- state transition

706 case mst\_cmd\_sm\_state is

현재 명령을 기다리는 상태이지만 rd 또는 wr 명령이 내려오면 해당하는 state machine이 동작하도록 플레그를 셋팅 합니다.

707 when cmd\_idle =>

708 if( mst\_go = '1') then

709 if( mst\_cntl\_rd\_req = '1' ) then

710 if rd\_fifo\_wr\_cnt\_s /= "11" then -- wait until fifo has 25% data

711 mst\_cmd\_sm\_start\_rd\_llink <= '1';

712 mst\_sm\_sts<= x"2";

713 mst\_cmd\_sm\_state <= cmd\_run;

714 end if;

715 elsif( mst\_cntl\_wr\_req = '1' ) then

716 --- npi\_v4\_00\_a if wr\_fifo\_wr\_cnt\_s /= "00" then -- wait until fifo has 25% data

717 if fumate\_start\_tx\_bus\_clk = '1' then

718 mst\_cmd\_sm\_start\_wr\_llink <= '1';

719 mst\_sm\_sts<= x"2";

720 mst\_cmd\_sm\_state <= cmd\_run;

721 end if;

722 end if;

723 else

724 mst\_sm\_sts<= x"1";

725 mst\_cmd\_sm\_state <= cmd\_idle;

726 mst\_cmd\_sm\_busy <= '0';

727 end if;

728 when cmd\_run =>

729 if( bus2ip\_mst\_cmdack = '1' and bus2ip\_mst\_cmplt = '0' ) then

730 mst\_sm\_sts<= x"3";

731 mst\_cmd\_sm\_state <= cmd\_wait\_for\_data;

732 elsif( bus2ip\_mst\_cmplt = '1' ) then

733 mst\_sm\_sts<= x"4";

734 mst\_cmd\_sm\_state <= cmd\_done;

735 if( bus2ip\_mst\_cmd\_timeout = '1' ) then

736 mst\_cmd\_sm\_set\_error <= '1';

737 mst\_cmd\_sm\_set\_timeout <= '1';

738 elsif( bus2ip\_mst\_error = '1' ) then

739 -- axi4lite data transfer error

740 mst\_cmd\_sm\_set\_error <= '1';

741 end if;

742 else

743 mst\_sm\_sts<= x"2";

744 mst\_cmd\_sm\_state <= cmd\_run;

745 mst\_cmd\_sm\_rd\_req <= mst\_cntl\_rd\_req;

746 mst\_cmd\_sm\_wr\_req <= mst\_cntl\_wr\_req;

747 mst\_cmd\_sm\_ip2bus\_addr <= mst\_ip2bus\_addr;

748 mst\_cmd\_sm\_ip2bus\_be <= mst\_ip2bus\_be(15 downto 16-c\_mst\_native\_data\_width/8 );

749 mst\_cmd\_sm\_xfer\_type <= mst\_cntl\_burst;

750 mst\_cmd\_sm\_xfer\_length <= mst\_xfer\_length;

751 mst\_cmd\_sm\_bus\_lock <= mst\_cntl\_bus\_lock;

752 end if;

753 when cmd\_wait\_for\_data =>

754 if( bus2ip\_mst\_cmplt = '1' ) then

755 mst\_sm\_sts <= x"4";

756 mst\_cmd\_sm\_state <= cmd\_done;

757 if( bus2ip\_mst\_cmd\_timeout = '1' ) then

758 -- axi4lite address phase timeout

759 mst\_cmd\_sm\_set\_error <= '1';

760 mst\_cmd\_sm\_set\_timeout <= '1';

761 elsif( bus2ip\_mst\_error = '1' ) then

762 -- axi4lite data transfer error

763 mst\_cmd\_sm\_set\_error <= '1';

764 end if;

765 else

766 mst\_sm\_sts<= x"3";

767 mst\_cmd\_sm\_state <= cmd\_wait\_for\_data;

768 end if;

769 when cmd\_done =>

770 if(cnt\_burst = c\_cnt\_burst - 1) then

771 cnt\_burst <= (others => '0');

772 if(c\_cnt\_repeat /= x"ffff") then

773 if(cnt\_repeat = c\_cnt\_repeat - 1) then

774 if mst\_cntl\_rd\_req = '1' then

775 rd\_fifo\_interrupt\_s <= '1';

776 end if;

777 mst\_cmd\_sm\_clr\_go <= '1';

778 mst\_cmd\_sm\_set\_done <= '1';

779 mst\_cmd\_sm\_busy <= '0';

780 mst\_sm\_sts <= x"5";

781 mst\_cmd\_sm\_state <= finish;

782 else

783 fumate\_tx\_int\_cnt <= fumate\_tx\_int\_cnt + 1;

784 cnt\_repeat <= cnt\_repeat + 1;

785 wr\_fifo\_interrupt\_s <= '1';

786 mst\_cmd\_sm\_state <= cmd\_idle;

787 end if;

788 else

789 fumate\_tx\_int\_cnt <= fumate\_tx\_int\_cnt + 1;

790 wr\_fifo\_interrupt\_s <= '1';

791 mst\_sm\_sts<= x"1";

792 mst\_cmd\_sm\_state <= cmd\_idle;

793 end if;

794 else

795 cnt\_burst <= cnt\_burst + 1;

796 mst\_sm\_sts<= x"1";

797 mst\_cmd\_sm\_set\_done <= '1';

798 mst\_cmd\_sm\_busy <= '0';

799 mst\_sm\_sts<= x"1";

800 mst\_cmd\_sm\_state <= cmd\_idle;

801 end if;

802 when finish=>

803 if mst\_go = '0' then

804 mst\_sm\_sts <= x"1";

805 mst\_cmd\_sm\_state <= cmd\_idle;

806 end if;

807 when others =>

808 mst\_cmd\_sm\_busy <= '0';

809 mst\_sm\_sts <= x"1";

810 mst\_cmd\_sm\_state <= cmd\_idle;

811 end case;

812 end if;

813 end if;

814 end process master\_cmd\_sm\_proc;

815

RD Fifo 블록에 일정한 데이터가 차면 플레그가 세팅되고 데이터가 모두 전송되고 나면 인터럽트가 발생하도록 되어 있습니다.

내부에는 카운터를 둬서 몇 번 인터럽트가 발생했는지 저장 합니다.

816

817 process(bus2ip\_clk)

818 begin

819 if rising\_edge(bus2ip\_clk) then

820 if bus2ip\_resetn = '0' then

821 wr\_fifo\_interrupt <= '0';

822 st\_int <= st\_0;

823 else

824 case st\_int is

825 when st\_0 =>

826 if wr\_fifo\_interrupt\_s = '1' then

827 wr\_fifo\_interrupt <= '1';

828 cnt\_int <= (others => '0');

829 st\_int <= st\_1;

830 elsif rd\_fifo\_interrupt\_s = '1' then

831 rd\_fifo\_interrupt <= '1';

832 cnt\_int <= (others => '0');

833 st\_int <= st\_2;

834 end if;

835 when st\_1 =>

836 if cnt\_int = x"1a4" then

837 wr\_fifo\_interrupt <= '0';

838 st\_int <= st\_0;

839 else

840 cnt\_int <= cnt\_int + 1;

841 end if;

842 when st\_2 =>

843 if cnt\_int = x"1a4" then

844 rd\_fifo\_interrupt <= '0';

845 st\_int <= st\_0;

846 else

847 cnt\_int <= cnt\_int + 1;

848 end if;

849 when others => null;

850 end case;

851 end if;

852 end if;

853 end process;

854

855

856

857

858

859 -- user logic master read locallink interface assignments

860 ip2bus\_mstrd\_dst\_rdy\_n <= not(mst\_llrd\_sm\_dst\_rdy);

861 ip2bus\_mstrd\_dst\_dsc\_n <= '1'; -- do not throttle data

862

863 -- implement a simple state machine to enable the

864 -- read locallink interface to transfer data

865 llink\_rd\_sm\_process : process( bus2ip\_clk ) is

866 begin

867

868 if( bus2ip\_clk'event and bus2ip\_clk = '1' ) then

869 if( bus2ip\_resetn = '0' ) then

870 mst\_llrd\_sm\_state <= llrd\_idle;

871 mst\_llrd\_sm\_dst\_rdy <= '0';

872

873 else

874 mst\_llrd\_sm\_state <= llrd\_idle;

875 mst\_llrd\_sm\_dst\_rdy <= '0';

876 case mst\_llrd\_sm\_state is

877 when llrd\_idle =>

878 if( mst\_cmd\_sm\_start\_rd\_llink = '1') then

879 mst\_llrd\_sm\_state <= llrd\_go;

880 else

881 mst\_llrd\_sm\_state <= llrd\_idle;

882 end if;

883 when llrd\_go =>

884 if( mst\_llrd\_sm\_dst\_rdy = '1' and bus2ip\_mstrd\_src\_rdy\_n = '0' and bus2ip\_mstrd\_eof\_n = '0' ) then

885 mst\_llrd\_sm\_state <= llrd\_idle;

886 else

887 mst\_llrd\_sm\_state <= llrd\_go;

888 mst\_llrd\_sm\_dst\_rdy <= '1';

889 end if;

890 when others =>

891 mst\_llrd\_sm\_state <= llrd\_idle;

892 end case;

893 end if;

894 else

895 null;

896 end if;

897 end process llink\_rd\_sm\_process;

898

Rx\_fifo를 정의한 inferrencing 코드 입니다.

이렇게 inferrencing코드를 사용하는 이유는 다음과 같습니다.

일반적으로Fifo는 coregen을 가지고 만들지만 이때는 FPGA 패밀리의 의존성이 있습니다.

공용코드를 만들고 싶어도 이 fifo 때문에 매번 새롭게 코드를 고쳐야 하는 부담이 생기게 됩니다.

라인 899~938까지 정의된 코드는 패밀리의 의존성이 없고 여러가지 파라미터들이 있어서 쉽게 원하는 값을 만들 수 있습니다.

899 rx\_fifo : entity proc\_common\_v3\_00\_a.async\_fifo\_fg

900 generic map(

901 c\_allow\_2n\_depth => 1, -- new paramter to leverage fifo gen 2\*\*n depth

902 c\_family => c\_family, -- new for fifo gen

903 c\_data\_width => 32,

904 c\_enable\_rlocs => 0, -- not supported in fg

905 c\_fifo\_depth => c\_fifo\_size,

906 c\_has\_almost\_empty => 0,

907 c\_has\_almost\_full => 0,

908 c\_has\_rd\_ack => 0,

909 c\_has\_rd\_count => 1,

910 c\_has\_rd\_err => 0,

911 c\_has\_wr\_ack => 0,

912 c\_has\_wr\_count => 1,

913 c\_has\_wr\_err => 0,

914 c\_rd\_ack\_low => 0,

915 c\_rd\_count\_width => 2,

916 c\_rd\_err\_low => 0,

917 c\_use\_blockmem => 1, -- 0 = distributed ram, 1 = bram

918 c\_wr\_ack\_low => 0,

919 c\_wr\_count\_width => 2,

920 c\_wr\_err\_low => 0 )

921 port map(

922 din => bus2ip\_mstrd\_d,

923 wr\_en => mst\_fifo\_valid\_write\_xfer,

924 wr\_clk => bus2ip\_clk,

925 rd\_clk => rd\_fifo\_clk,

926 rd\_en => rd\_fifo\_rd\_en,

927 ainit => fifo\_rst\_s,

928 dout => rd\_fifo\_data,

929 full => rd\_fifo\_full\_s,

930 empty => rd\_fifo\_empty\_s,

931 almost\_full => open,

932 almost\_empty => open,

933 wr\_count => rd\_fifo\_wr\_cnt\_s,

934 rd\_count => open,

935 rd\_ack => open,

936 rd\_err => open,

937 wr\_ack => open,

938 wr\_err => open);

939

940 -- user logic master write locallink interface assignments

941 ip2bus\_mstwr\_src\_rdy\_n <= not(mst\_llwr\_sm\_src\_rdy);

942 ip2bus\_mstwr\_src\_dsc\_n <= '1'; -- do not throttle data

943 ip2bus\_mstwr\_rem <= (others => '0');

944 ip2bus\_mstwr\_sof\_n <= not(mst\_llwr\_sm\_sof);

945 ip2bus\_mstwr\_eof\_n <= not(mst\_llwr\_sm\_eof);

946

947 -- implement a simple state machine to enable the

948 -- write locallink interface to transfer data

949

950 llink\_wr\_sm\_proc : process( bus2ip\_clk ) is

951 constant bytes\_per\_beat : integer := c\_mst\_native\_data\_width/8;

952 begin

953

954 if( bus2ip\_clk'event and bus2ip\_clk = '1' ) then

955 if( bus2ip\_resetn = '0' ) then

956

957 -- reset condition

958 mst\_llwr\_sm\_state <= llwr\_idle;

959 mst\_llwr\_sm\_src\_rdy <= '0';

960 mst\_llwr\_sm\_sof <= '0';

961 mst\_llwr\_sm\_eof <= '0';

962 mst\_llwr\_byte\_cnt <= 0;

963

964 else

965

966 -- default condition

967 mst\_llwr\_sm\_state <= llwr\_idle;

968 mst\_llwr\_sm\_src\_rdy <= '0';

969 mst\_llwr\_sm\_sof <= '0';

970 mst\_llwr\_sm\_eof <= '0';

971 mst\_llwr\_byte\_cnt <= 0;

972 -- state transition

973 case mst\_llwr\_sm\_state is

974 when llwr\_idle =>

975 if( mst\_cmd\_sm\_start\_wr\_llink = '1' and mst\_cntl\_burst = '0' ) then

976 mst\_llwr\_sm\_state <= llwr\_sngl\_init;

977 elsif( mst\_cmd\_sm\_start\_wr\_llink = '1' and mst\_cntl\_burst = '1' ) then

978 mst\_llwr\_sm\_state <= llwr\_brst\_init;

979 else

980 mst\_llwr\_sm\_state <= llwr\_idle;

981 end if;

982

983 when llwr\_sngl\_init =>

984 mst\_llwr\_sm\_state <= llwr\_sngl;

985 mst\_llwr\_sm\_src\_rdy <= '1';

986 mst\_llwr\_sm\_sof <= '1';

987 mst\_llwr\_sm\_eof <= '1';

988

989 when llwr\_sngl =>

990 -- destination discontinue write

991 if( bus2ip\_mstwr\_dst\_dsc\_n = '0' and bus2ip\_mstwr\_dst\_rdy\_n = '0' ) then

992 mst\_llwr\_sm\_state <= llwr\_idle;

993 mst\_llwr\_sm\_src\_rdy <= '0';

994 mst\_llwr\_sm\_eof <= '0';

995 -- single data beat transfer complete

996 elsif( mst\_fifo\_valid\_read\_xfer = '1' ) then

997 mst\_llwr\_sm\_state <= llwr\_idle;

998 mst\_llwr\_sm\_src\_rdy <= '0';

999 mst\_llwr\_sm\_sof <= '0';

1000 mst\_llwr\_sm\_eof <= '0';

1001 -- wait on destination

1002 else

1003 mst\_llwr\_sm\_state <= llwr\_sngl;

1004 mst\_llwr\_sm\_src\_rdy <= '1';

1005 mst\_llwr\_sm\_sof <= '1';

1006 mst\_llwr\_sm\_eof <= '1';

1007 end if;

1008

1009 when llwr\_brst\_init =>

1010 mst\_llwr\_sm\_state <= llwr\_brst;

1011 mst\_llwr\_sm\_src\_rdy <= '1';

1012 mst\_llwr\_sm\_sof <= '1';

1013 mst\_llwr\_byte\_cnt <= conv\_integer(mst\_xfer\_length);

1014 when llwr\_brst =>

1015 if( mst\_fifo\_valid\_read\_xfer = '1' ) then

1016 mst\_llwr\_sm\_sof <= '0';

1017 else

1018 mst\_llwr\_sm\_sof <= mst\_llwr\_sm\_sof;

1019 end if;

1020 -- destination discontinue write

1021 if( bus2ip\_mstwr\_dst\_dsc\_n = '0' and bus2ip\_mstwr\_dst\_rdy\_n = '0' ) then

1022 mst\_llwr\_sm\_state <= llwr\_idle;

1023 mst\_llwr\_sm\_src\_rdy <= '1';

1024 mst\_llwr\_sm\_eof <= '1';

1025 -- last data beat write

1026 elsif( mst\_fifo\_valid\_read\_xfer = '1' and (mst\_llwr\_byte\_cnt-bytes\_per\_beat) <= bytes\_per\_beat ) then

1027 mst\_llwr\_sm\_state <= llwr\_brst\_last\_beat;

1028 mst\_llwr\_sm\_src\_rdy <= '1';

1029 mst\_llwr\_sm\_eof <= '1';

1030 -- wait on destination

1031 else

1032 mst\_llwr\_sm\_state <= llwr\_brst;

1033 mst\_llwr\_sm\_src\_rdy <= '1';

1034 -- decrement write transfer counter if it's a valid write

1035 if( mst\_fifo\_valid\_read\_xfer = '1' ) then

1036 mst\_llwr\_byte\_cnt <= mst\_llwr\_byte\_cnt - bytes\_per\_beat;

1037 else

1038 mst\_llwr\_byte\_cnt <= mst\_llwr\_byte\_cnt;

1039 end if;

1040 end if;

1041 when llwr\_brst\_last\_beat =>

1042 -- destination discontinue write

1043 if( bus2ip\_mstwr\_dst\_dsc\_n = '0' and bus2ip\_mstwr\_dst\_rdy\_n = '0' ) then

1044 mst\_llwr\_sm\_state <= llwr\_idle;

1045 mst\_llwr\_sm\_src\_rdy <= '0';

1046 -- last data beat done

1047 elsif( mst\_fifo\_valid\_read\_xfer = '1' ) then

1048 mst\_llwr\_sm\_state <= llwr\_idle;

1049 mst\_llwr\_sm\_src\_rdy <= '0';

1050 -- wait on destination

1051 else

1052 mst\_llwr\_sm\_state <= llwr\_brst\_last\_beat;

1053 mst\_llwr\_sm\_src\_rdy <= '1';

1054 mst\_llwr\_sm\_eof <= '1';

1055 end if;

1056 when others =>

1057 mst\_llwr\_sm\_state <= llwr\_idle;

1058 end case;

1059

1060 end if;

1061 else

1062 null;

1063 end if;

1064

1065 end process llink\_wr\_sm\_proc;

1066

메모리로 write할 때 필요로 하는 fifo를 설계 합니다.

1067 tx\_fifo : entity proc\_common\_v3\_00\_a.async\_fifo\_fg

1068 generic map(

1069 c\_allow\_2n\_depth => 1, -- new paramter to leverage fifo gen 2\*\*n depth

1070 c\_family => c\_family, -- new for fifo gen

1071 c\_data\_width => 32,

1072 c\_enable\_rlocs => 0, -- not supported in fg

1073 c\_fifo\_depth => c\_fifo\_size,

1074 c\_has\_almost\_empty => 0,

1075 c\_has\_almost\_full => 0,

1076 c\_has\_rd\_ack => 0,

1077 c\_has\_rd\_count => 1,

1078 c\_has\_rd\_err => 0,

1079 c\_has\_wr\_ack => 0,

1080 c\_has\_wr\_count => 1,

1081 c\_has\_wr\_err => 0,

1082 c\_rd\_ack\_low => 0,

1083 c\_rd\_count\_width => 2,

1084 c\_rd\_err\_low => 0,

1085 c\_use\_blockmem => 1, -- 0 = distributed ram, 1 = bram

1086 c\_wr\_ack\_low => 0,

1087 c\_wr\_count\_width => 2,

1088 c\_wr\_err\_low => 0)

1089 port map(

1090 din => wr\_fifo\_data,

1091 wr\_en => wr\_fifo\_wr\_en,

1092 wr\_clk => wr\_fifo\_clk,

1093 rd\_en => mst\_fifo\_valid\_read\_xfer,

1094 rd\_clk => bus2ip\_clk,

1095 ainit => fifo\_rst\_s,

1096 --dout => ip2bus\_mstwr\_d\_fifo,

1097 dout => ip2bus\_mstwr\_d,

1098 full => wr\_fifo\_full\_s,

1099 empty => wr\_fifo\_empty\_s,

1100 almost\_full => open,

1101 almost\_empty => open,

1102 wr\_count => wr\_fifo\_wr\_cnt\_s,

1103 rd\_count => open,

1104 rd\_ack => open,

1105 rd\_err => open,

1106 wr\_ack => open,

1107 wr\_err => open);

1108

1109 process(bus2ip\_clk)

1110 begin

1111 if rising\_edge(bus2ip\_clk) then

1112 ip2bus\_mstwr\_d\_s <= ip2bus\_mstwr\_d\_s + 1;

1113 end if;

1114 end process;

1115

1116 -- ip2bus\_mstwr\_d <= ip2bus\_mstwr\_d\_fifo(31 downto 8) & ip2bus\_mstwr\_d\_s(7 downto 0);

1117

1118

1119 mst\_fifo\_valid\_write\_xfer <= not(bus2ip\_mstrd\_src\_rdy\_n) and mst\_llrd\_sm\_dst\_rdy and not(rd\_fifo\_full\_s);

1120 mst\_fifo\_valid\_read\_xfer <= not(bus2ip\_mstwr\_dst\_rdy\_n) and mst\_llwr\_sm\_src\_rdy and not(wr\_fifo\_empty\_s);

1121 bus2ip\_reset <= not (bus2ip\_resetn);

1122

라인 1123, 1124에서 User logic에서 현재 npi의 rd/wr 쪽 이 제대로 준비되어 있는 알 수 있는 플레그를 셋팅 합니다.

1123 rd\_fifo\_ready\_s <= mst\_go and mst\_cntl\_rd\_req;

1124 wr\_fifo\_ready\_s <= mst\_go and mst\_cntl\_wr\_req;

1125

1126 rd\_fifo\_ready <= rd\_fifo\_ready\_s;

1127 wr\_fifo\_ready <= wr\_fifo\_ready\_s;

1128

1129

1130 rd\_fifo\_full <= rd\_fifo\_full\_s;

1131 rd\_fifo\_empty <= rd\_fifo\_empty\_s;

1132 wr\_fifo\_full <= wr\_fifo\_full\_s;

1133 wr\_fifo\_empty <= wr\_fifo\_empty\_s;

1134 ------------------------------------------

1135 -- example code to drive ip to bus signals

1136 ------------------------------------------

1137 ip2bus\_data <= slv\_ip2bus\_data when slv\_read\_ack = '1' else

1138 mst\_ip2bus\_data when mst\_read\_ack = '1' else

1139 (others => '0');

1140

1141 -- implement slave model software accessible register(s)

1142 slave\_reg\_write\_proc : process(bus2ip\_clk) is

1143 begin

1144

1145 if bus2ip\_clk'event and bus2ip\_clk = '1' then

1146 if bus2ip\_resetn = '0' then

1147 slv\_reg7 <= (others => '0');

1148 else

1149 case slv\_reg\_write\_sel is

1150 when "00000001" =>

1151 for byte\_index in 0 to (c\_slv\_dwidth/8)-1 loop

1152 if (bus2ip\_be(byte\_index) = '1') then

1153 slv\_reg7(byte\_index\*8+7 downto byte\_index\*8) <= bus2ip\_data(byte\_index\*8+7 downto byte\_index\*8);

1154 end if;

1155 end loop;

1156 when others => null;

1157 end case;

1158 end if;

1159 end if;

1160 end process slave\_reg\_write\_proc;

1161

Fifo를 리셋하는 부분과 state machine을 리셋 하는 부분 입니다. 프로세서가 특정한 값을 레지스터에 쓰면 동작하도록 만들었습다.

1162 fifo\_rst\_s <= '1' when slv\_reg7(3 downto 0) = x"1" else bus2ip\_reset;

1163 sm\_reset\_s <= '1' when slv\_reg7(3 downto 0) = x"4" else '0';

1164

1165 ip2bus\_wrack <= slv\_write\_ack or mst\_write\_ack;

1166 ip2bus\_rdack <= slv\_read\_ack or mst\_read\_ack;

1167 ip2bus\_error <= '0';

라인 1168~1213은 rx\_dv가 deassert 될 때까지 데이터 카운터를 저장한 후 wr master로직이 동작할 수 있도록 동기화 시켜 줍니다.

1168 ----------------------------------------------------------------------------------------------------

1169 -- npi\_v4\_10\_a

1170 ----------------------------------------------------------------------------------------------------

1171 process(wr\_fifo\_clk)

1172 begin

1173 if rising\_edge(wr\_fifo\_clk) then

1174 if (bus2ip\_reset = '1' or sm\_reset\_s = '1') then

1175 st\_tx <= tx0;

1176 fumate\_base\_add <= (others => '0');

1177 fumate\_wr\_en\_count <= (others => '0');

1178 fumate\_tx\_data\_cnt <= (others => '0');

1179 fumate\_start\_tx <= '0';

1180 else

1181 case st\_tx is

1182 when tx0 =>

1183 if wr\_fifo\_wr\_en = '1' then

1184 fumate\_tx\_data\_cnt <= x"0002";

1185 st\_tx <= tx1;

1186 else

1187 fumate\_start\_tx <= '0';

1188 end if;

1189 when tx1 =>

1190 if wr\_fifo\_wr\_en = '0' then

1191 fumate\_start\_tx <= '1';

1192 fumate\_wr\_en\_count <= fumate\_wr\_en\_count + 1;

1193 fumate\_base\_add <= fumate\_base\_add + 1;

1194 fumate\_tx\_data\_const <= fumate\_tx\_data\_cnt;

1195 st\_tx <= tx2;

1196 else

1197 fumate\_tx\_data\_cnt <= fumate\_tx\_data\_cnt + 1;

1198 end if;

1199 when tx2 =>

1200 st\_tx <= tx3;

1201 when tx3 =>

1202 st\_tx <= tx4;

1203 when tx4 =>

1204 fumate\_start\_tx <= '0';

1205 st\_tx <= tx5;

1206 when tx5 =>

1207 st\_tx <= tx0;

1208 when others =>

1209 end case;

1210 end if;

1211 end if;

1212 end process;

1213

라인 1214 ~ 1223에는 user clk에 동기된 여러 신호를 버스 클럭에 다시 동기 시키는 부분 입니다.

1214 ------- resync

1215 process(bus2ip\_clk)

1216 begin

1217 if rising\_edge(bus2ip\_clk) then

1218 fumate\_tx\_data\_const\_bus\_clk <= fumate\_tx\_data\_const;

1219 fumate\_base\_add\_bus\_clk <= fumate\_base\_add;

1220 fumate\_start\_tx\_bus\_clk <= fumate\_start\_tx;

1221 end if;

1222 end process;

1223 end IMP;

1224

1225

## 디바이스 드라이버 설명

1 /\*

2 \* npi.c

3 \*

4 \* Created on: Jan 15, 2013

5 \* Author: hyukk

6 \*/

7

함수의 원형을 정의 합니다.

8 #include "npi.h"

9

현재 NPI 상태를 report 합니다.

10 int axi\_npi\_status(int npi\_base\_add)

11 {

12 unsigned int \*add, data;

13 unsigned int npi\_ver, cnt\_rx\_bus, cnt\_tx\_bus, npi\_ready\_sts, mst\_sm\_sts, mst\_cmd\_sts, fifo\_sts;

14 unsigned int mst\_ip2bus\_addr, c\_cnt\_burst, cnt\_burst, c\_cnt\_repeat, cnt\_repeat;

15 unsigned int fumate\_wr\_en\_count,fumate\_tx\_int\_cnt;

16 add = (unsigned int \*) npi\_base\_add;

17

18 data = \*add++;

19 fifo\_sts = data & 0xf; data = data >> 4; // // offset 0x0

20 mst\_cmd\_sts = data & 0xf; data = data >> 4;

21 mst\_sm\_sts = data & 0xf; data = data >> 4;

22 npi\_ready\_sts = data & 0x3; data = data >> 2;

23 cnt\_tx\_bus = data & 0xf; data = data >> 4;

24 cnt\_rx\_bus = data & 0xf; data = data >> 4;

25 npi\_ver = data & 0x3ff; data = data >> 10;

26

27 data = \*add++;

28 mst\_ip2bus\_addr = data; // // offset 0x4

29

30 data = \*add++;

31 cnt\_burst = data & 0xfff; data = data >> 12; // // offset 0x8

32 c\_cnt\_burst = data & 0xfff; data = data >> 12;

33

34 data = \*add++;

35 cnt\_repeat = data & 0xffff; data = data >> 16; // // offset 0xc

36 c\_cnt\_repeat = data & 0xffff; data = data >> 16;

37

38 data = \*add++;

39 fumate\_tx\_int\_cnt = data & 0xffff; data = data >> 16; // offset 0x10

40 fumate\_wr\_en\_count = data & 0xffff; data = data >> 16;

41

42 xil\_printf("\n\r %30s 0x%04x", "npi ver", npi\_ver);

43 if(npi\_ver == 0x41) {

44 xil\_printf("\n\r %30s", "0x41 ver is used for fumate");

45 }

46 switch (mst\_sm\_sts) {

47 case 1: xil\_printf("\n\r %30s", "npi is CMD\_IDLE"); break;

48 case 2: xil\_printf("\n\r %30s", "npi is CMD\_RUN"); break;

49 case 3: xil\_printf("\n\r %30s", "npi is CMD\_WAIT\_FOR\_DATA"); break;

50 case 4: xil\_printf("\n\r %30s", "npi is CMD\_DONE"); break;

51 case 5: xil\_printf("\n\r %30s", "npi is FINISH"); break;

52 default: xil\_printf("\n\r %30s", "unknown"); break;

53 }

54

55 xil\_printf("\n\r %30s 0x%x", "rd\_fifo\_empty is", fifo\_sts & 0x1); fifo\_sts = fifo\_sts >> 1;

56 xil\_printf("\n\r %30s 0x%x", "rd\_fifo\_full is", fifo\_sts & 0x1); fifo\_sts = fifo\_sts >> 1;

57 xil\_printf("\n\r %30s 0x%x", "wr\_fifo\_empty is", fifo\_sts & 0x1); fifo\_sts = fifo\_sts >> 1;

58 xil\_printf("\n\r %30s 0x%x", "wr\_fifo\_full is", fifo\_sts & 0x1); fifo\_sts = fifo\_sts >> 1;

59

60 xil\_printf("\n\r %30s 0x%x", "rd\_fifo\_ready\_s is", npi\_ready\_sts & 0x1); npi\_ready\_sts = npi\_ready\_sts >> 1;

61 xil\_printf("\n\r %30s 0x%x", "wr\_fifo\_ready\_s is", npi\_ready\_sts & 0x1); npi\_ready\_sts = npi\_ready\_sts >> 1;

62

63 xil\_printf("\n\r %30s 0x%x", "mst\_go is", mst\_cmd\_sts & 0x1); mst\_cmd\_sts = mst\_cmd\_sts >> 1;

64 xil\_printf("\n\r %30s 0x%x", "mst\_cmd\_sm\_set\_timeout is", mst\_cmd\_sts & 0x1); mst\_cmd\_sts = mst\_cmd\_sts >> 1;

65 xil\_printf("\n\r %30s 0x%x", "mst\_cmd\_sm\_set\_error is", mst\_cmd\_sts & 0x1); mst\_cmd\_sts = mst\_cmd\_sts >> 1;

66 xil\_printf("\n\r %30s 0x%x", "mst\_cmd\_sm\_set\_done is", mst\_cmd\_sts & 0x1); mst\_cmd\_sts = mst\_cmd\_sts >> 1;

67

68 xil\_printf("\n\r %30s 0x%x", "current access add is", mst\_ip2bus\_addr);

69 xil\_printf("\n\r %30s 0x%x", "cnt\_burst is", cnt\_burst);

70 xil\_printf("\n\r %30s 0x%x", "c\_cnt\_burst is", c\_cnt\_burst);

71 xil\_printf("\n\r %30s 0x%x", "cnt\_repeat is", cnt\_repeat);

72 xil\_printf("\n\r %30s 0x%x", "c\_cnt\_repeat is", c\_cnt\_repeat);

73 xil\_printf("\n\r %30s 0x%x", "cnt\_rx is", cnt\_rx\_bus);

74 xil\_printf("\n\r %30s 0x%x", "cnt\_tx is", cnt\_tx\_bus);

75

76 if(npi\_ver == 0x41) {

77 xil\_printf("\n\r %30s 0x%x", "fumate\_tx\_int\_cnt is", fumate\_tx\_int\_cnt);

78 xil\_printf("\n\r %30s 0x%x", "fumate\_wr\_en\_cnt is", fumate\_wr\_en\_count);

79 }

80 }

81

현재 NPI 버전 정보를 보여 줍니다.

82 int axi\_npi\_ver(int npi\_base\_add)

83 {

84 int \*add;

85

86 add = (int \*) npi\_base\_add;

87 return ((\*add >> 22) & 0x3ff);

88 }

89

Npi를 읽기용으로 쓸 것인지 쓰기용으로 쓸 것인지 선택 합니다.

90 void set\_axi\_npi\_as\_wr(int npi\_base\_add)

91 {

92 if(axi\_npi\_ver(npi\_base\_add) == 0x41)

93 \*(char \*) (npi\_base\_add + 0x100 + 0x8) = 0x1;

94

95 }

96

97 void set\_axi\_npi\_as\_rd(int npi\_base\_add)

98 {

99 if(axi\_npi\_ver(npi\_base\_add) == 0x41)

100 \*(char \*) (npi\_base\_add + 0x100 + 0x8) = 0x0;

101 }

102

NPI를 읽기 모드로 사용할 경우 사용 하는 함수 입니다.

103 void axi\_npi\_rd(int npi\_base\_add, int src, int cnt\_burst, int cnt\_repeat)

104 {

105 #define GO\_CMD 0x0a

106 \*(char \*) (npi\_base\_add + 0x100 + 0x0) = 0x9;

107 \*(short int \*) (npi\_base\_add + 0x100 + 0x2) = cnt\_burst;

108 \*(short int \*) (npi\_base\_add + 0x100 + 0x6) = (src >> 16) & 0xffff;

109 \*(short int \*) (npi\_base\_add + 0x100 + 0xa) = cnt\_repeat;

110 \*(volatile char \*) (npi\_base\_add + 0x100 + 0xf) = (GO\_CMD);

111 \*(volatile char \*) (npi\_base\_add + 0x100 + 0xf) = (0x0b);

112 }

113

NPI를 쓰기 용도로 사용할 경우 사용하는 함수 입니다.

114 void axi\_npi\_wr(int npi\_base\_add, int src, int cnt\_burst, int cnt\_repeat)

115 {

116 #define GO\_CMD 0x0a

117 \*(char \*) (npi\_base\_add + 0x100 + 0x0) = 0xa;

118 \*(short int \*) (npi\_base\_add + 0x100 + 0x2) = cnt\_burst;

119 \*(short int \*) (npi\_base\_add + 0x100 + 0x6) = (src >> 16) & 0xffff;;

120 \*(short int \*) (npi\_base\_add + 0x100 + 0xa) = cnt\_repeat;

121 \*(volatile char \*) (npi\_base\_add + 0x100 + 0xf) = (GO\_CMD);

122 \*(volatile char \*) (npi\_base\_add + 0x100 + 0xf) = (0x0b);

123 }

124

NPI가 현재 동작중인지 확인하는 함수 입니다.

125 int is\_axi\_npi\_busy(int npi\_base\_add)

126 {

127 int \*add;

128 add = (unsigned int \*) npi\_base\_add;

129

130 return ((\*add >> 4) & 0x01);

131 }

132

NPI 내부의 여러 state machine과 fifo를 reset하는 함수 입니다.

133 void axi\_npi\_reset(int npi\_base\_add)

134 {

135 #define SM\_RESET\_VALUE 0x4

136

137 \*(volatile int \*) (npi\_base\_add + 0x1c) = SM\_RESET\_VALUE;

138 \*(volatile int \*) (npi\_base\_add + 0x1c) = 0x0;

139 }

140

141 void axi\_npi\_fifo\_reset(int npi\_base\_add)

142 {

143 #define FIFO\_RESET\_VALUE 0x1

144

145 \*(volatile int \*) (npi\_base\_add + 0x1c) = FIFO\_RESET\_VALUE;

146 \*(volatile int \*) (npi\_base\_add + 0x1c) = 0x0;

147 }

148

149

# Custome Axi Master, NPI\_V4\_00\_a

# PL (Programmable Logic)에서 인터럽트 요청

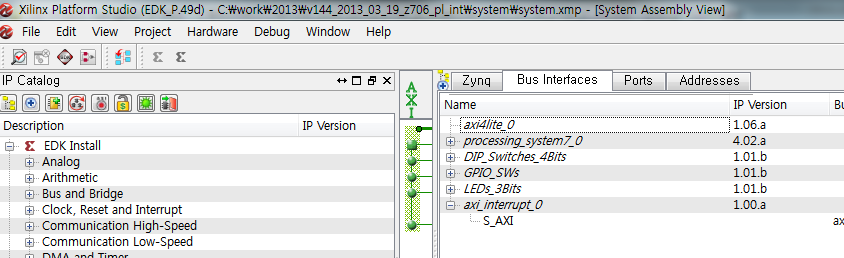
당연한 얘기 이지만 징크의 PS 에 있는 모든 주변장치는 이미 인터럽트 콘트롤러와 연결되어 있기 때문에 인터럽트 모드를 사용하는 것과 하지 않는 것은 전적으로 소프트웨어에 달려 있습니다. 하지만 개발자가 직접 만든 User Logic에서 인터럽트 요청을 할 때도 많이 있습니다.

이번 장에서는 PL에서 프로세서에게 인터럽트 요청하는 User Logic을 어떻게 만드는지 알아보도록 하겠습니다.

## 시스템 구성

### XPS/Bus interface

현재는 AXI4Lite 버전에 3개의 GPIO와 1개의 axi\_interrupt가 연결되어 있습니다.



### System.mhs

다음 리스트는 Z706보드에서 인터럽트 관련 IP를 추가한 mhs 파일 입니다.

# ##############################################################################

# Created by Base System Builder Wizard for Xilinx EDK 14.4 Build EDK\_P.49d

# Tue Mar 19 14:21:49 2013

# Target Board: xilinx.com zc706 Rev B

# Family: zynq

# Device: xc7z045

# Package: ffg900

# Speed Grade: -1

# ##############################################################################

PARAMETER VERSION = 2.1.0

GPIO 3개를 사용 합니다.

PORT LEDs\_3Bits\_TRI\_O = LEDs\_3Bits\_TRI\_O, DIR = O, VEC = [2:0]

PORT GPIO\_SWs\_TRI\_IO = GPIO\_SWs\_TRI\_IO, DIR = IO, VEC = [2:0]

PORT DIP\_Switches\_TRI\_I = DIP\_Switches\_TRI\_I, DIR = I, VEC = [3:0]

ZYNQ의 MIO를 사용 합니다.

PORT processing\_system7\_0\_MIO = processing\_system7\_0\_MIO, DIR = IO, VEC = [53:0]

DDR3 를 사용 합니다.

PORT processing\_system7\_0\_PS\_SRSTB = processing\_system7\_0\_PS\_SRSTB, DIR = I

PORT processing\_system7\_0\_PS\_CLK = processing\_system7\_0\_PS\_CLK, DIR = I, SIGIS = CLK

PORT processing\_system7\_0\_PS\_PORB = processing\_system7\_0\_PS\_PORB, DIR = I

PORT processing\_system7\_0\_DDR\_Clk = processing\_system7\_0\_DDR\_Clk, DIR = IO, SIGIS = CLK

>>>>

PORT processing\_system7\_0\_DDR\_VRN = processing\_system7\_0\_DDR\_VRN, DIR = IO

PORT processing\_system7\_0\_DDR\_VRP = processing\_system7\_0\_DDR\_VRP, DIR = IO

PL에 AXI 인터커넥터를 연결 합니다. 현재는 AXI4Lite를 정의 했습니다.

BEGIN axi\_interconnect

PARAMETER INSTANCE = axi4lite\_0

PARAMETER HW\_VER = 1.06.a

PARAMETER C\_INTERCONNECT\_CONNECTIVITY\_MODE = 0

PORT interconnect\_aclk = processing\_system7\_0\_FCLK\_CLK0

PORT INTERCONNECT\_ARESETN = processing\_system7\_0\_FCLK\_RESET0\_N\_0

END

3개의 GPIO정의 입니다. 모두 AXI4Lite에 연결 됩니다.

BEGIN axi\_gpio

PARAMETER INSTANCE = LEDs\_3Bits

PARAMETER HW\_VER = 1.01.b

PARAMETER C\_GPIO\_WIDTH = 3

PARAMETER C\_ALL\_INPUTS = 0

PARAMETER C\_INTERRUPT\_PRESENT = 0

PARAMETER C\_IS\_DUAL = 0

PARAMETER C\_BASEADDR = 0x41200000

PARAMETER C\_HIGHADDR = 0x4120ffff

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = processing\_system7\_0\_FCLK\_CLK0

PORT GPIO\_IO\_O = LEDs\_3Bits\_TRI\_O

END

BEGIN axi\_gpio

PARAMETER INSTANCE = GPIO\_SWs

PARAMETER HW\_VER = 1.01.b

PARAMETER C\_GPIO\_WIDTH = 3

PARAMETER C\_ALL\_INPUTS = 1

PARAMETER C\_INTERRUPT\_PRESENT = 1

PARAMETER C\_IS\_DUAL = 0

PARAMETER C\_BASEADDR = 0x41220000

PARAMETER C\_HIGHADDR = 0x4122ffff

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = processing\_system7\_0\_FCLK\_CLK0

PORT GPIO\_IO = GPIO\_SWs\_TRI\_IO

PORT IP2INTC\_Irpt = GPIO\_SWs\_IP2INTC\_Irpt

END

BEGIN axi\_gpio

PARAMETER INSTANCE = DIP\_Switches\_4Bits

PARAMETER HW\_VER = 1.01.b

PARAMETER C\_GPIO\_WIDTH = 4

PARAMETER C\_ALL\_INPUTS = 1

PARAMETER C\_INTERRUPT\_PRESENT = 0

PARAMETER C\_IS\_DUAL = 0

PARAMETER C\_BASEADDR = 0x41240000

PARAMETER C\_HIGHADDR = 0x4124ffff

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = processing\_system7\_0\_FCLK\_CLK0

PORT GPIO\_IO\_I = DIP\_Switches\_TRI\_I

END

PS에 있는 여러 주변장치 중 UART1 만 사용하도록 정했습니다.

BEGIN processing\_system7

PARAMETER INSTANCE = processing\_system7\_0

PARAMETER HW\_VER = 4.02.a

PARAMETER C\_DDR\_RAM\_HIGHADDR = 0x3FFFFFFF

PARAMETER C\_USE\_M\_AXI\_GP0 = 1

PARAMETER C\_S\_AXI\_HP0\_HIGHADDR = 0x1FFFFFFF

PARAMETER C\_EN\_EMIO\_CAN0 = 0

PARAMETER C\_EN\_EMIO\_CAN1 = 0

>>>

PARAMETER C\_EN\_EMIO\_UART1 = 0

PARAMETER C\_EN\_EMIO\_MODEM\_UART0 = 0

PARAMETER C\_EN\_EMIO\_MODEM\_UART1 = 0

PARAMETER C\_EN\_EMIO\_WDT = 0

PARAMETER C\_EMIO\_GPIO\_WIDTH = 64

PARAMETER C\_EN\_QSPI = 0

>>>

PARAMETER C\_EN\_UART0 = 0

PARAMETER C\_EN\_UART1 = 1

PARAMETER C\_EN\_MODEM\_UART0 = 0

PARAMETER C\_EN\_MODEM\_UART1 = 0

PARAMETER C\_EN\_USB0 = 0

PARAMETER C\_EN\_USB1 = 0

PARAMETER C\_EN\_WDT = 0

PARAMETER C\_EN\_DDR = 1

PARAMETER C\_EN\_GPIO = 0

PARAMETER C\_FCLK\_CLK0\_FREQ = 50000000

PARAMETER C\_FCLK\_CLK1\_FREQ = 50000000

PARAMETER C\_FCLK\_CLK2\_FREQ = 50000000

PARAMETER C\_FCLK\_CLK3\_FREQ = 50000000

PARAMETER C\_EN\_EMIO\_GPIO = 0

BUS\_INTERFACE M\_AXI\_GP0 = axi4lite\_0

PORT MIO = processing\_system7\_0\_MIO

PORT PS\_SRSTB = processing\_system7\_0\_PS\_SRSTB

PORT PS\_CLK = processing\_system7\_0\_PS\_CLK

>>>

PORT DDR\_VRP = processing\_system7\_0\_DDR\_VRP

PORT FCLK\_CLK0 = processing\_system7\_0\_FCLK\_CLK0

PORT FCLK\_RESET0\_N = processing\_system7\_0\_FCLK\_RESET0\_N\_0

PORT M\_AXI\_GP0\_ACLK = processing\_system7\_0\_FCLK\_CLK0

프로세서 블록에는 모두 16개의 인터럽트를 사용할 수 있습니다. 그중에 2개의 인터럽트소스를 사용하는데 하나는 axi\_interrupt IP가 소스이고 나머지 하나는 GPIO를 통해서 입력을 받도록 합니다.

이때는 그냥 숫자 값으로 써주면 되는데 숫자와 mhs 관계는 왼쪽부터 91, 90, 89, … 이렇게 감소 합니다.

PORT IRQ\_F2P = DIP\_Switches\_TRI\_I[0] & axi\_interrupt\_0\_IP2INTC\_Irpt

따라서 DIP Switch 입력은 91번 나머지는 90번 인터럽트 번호와 연결 됩니다. 참고하세요

PORT IRQ\_F2P = DIP\_Switches\_TRI\_I[0] & axi\_interrupt\_0\_IP2INTC\_Irpt

END

완성된 axi\_interrupt IP 입니다.

BEGIN axi\_interrupt

PARAMETER INSTANCE = axi\_interrupt\_0

PARAMETER HW\_VER = 1.00.a

PARAMETER C\_BASEADDR = 0x74200000

PARAMETER C\_HIGHADDR = 0x7420ffff

PARAMETER C\_INTERRUPT\_PRESENT = 1

BUS\_INTERFACE S\_AXI = axi4lite\_0

PORT S\_AXI\_ACLK = processing\_system7\_0\_FCLK\_CLK0

인터럽트 관련 핀이 추가되어 있습니다.

PORT IP2INTC\_Irpt = axi\_interrupt\_0\_IP2INTC\_Irpt

END

### Axi\_interrupt/User\_logic.vhd

인터럽트를 발생시키기 위한 user logic에 대한 리스트 입니다.

>>>>

85

86 entity user\_logic is

87 generic

88 (

89 -- ADD USER GENERICS BELOW THIS LINE ---------------

90 --USER generics added here

91 -- ADD USER GENERICS ABOVE THIS LINE ---------------

92

93 -- DO NOT EDIT BELOW THIS LINE ---------------------

94 -- Bus protocol parameters, do not add to or delete

95 C\_SLV\_DWIDTH : integer := 32;

96 C\_NUM\_REG : integer := 4;

97 C\_NUM\_INTR : integer := 2

98 -- DO NOT EDIT ABOVE THIS LINE ---------------------

99 );

100 port

101 (

102 -- ADD USER PORTS BELOW THIS LINE ------------------

103 --USER ports added here

104 -- ADD USER PORTS ABOVE THIS LINE ------------------

105

106 -- DO NOT EDIT BELOW THIS LINE ---------------------

107 -- Bus protocol ports, do not add to or delete

108 Bus2IP\_Clk : in std\_logic;

109 Bus2IP\_Reset : in std\_logic;

110 Bus2IP\_Data : in std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

111 Bus2IP\_BE : in std\_logic\_vector(0 to C\_SLV\_DWIDTH/8-1);

112 Bus2IP\_RdCE : in std\_logic\_vector(0 to C\_NUM\_REG-1);

113 Bus2IP\_WrCE : in std\_logic\_vector(0 to C\_NUM\_REG-1);

114 IP2Bus\_Data : out std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

115 IP2Bus\_RdAck : out std\_logic;

116 IP2Bus\_WrAck : out std\_logic;

117 IP2Bus\_Error : out std\_logic;

인터럽트 관련 포트가 정의 되어 있습니다.

118 IP2Bus\_IntrEvent : out std\_logic\_vector(0 to C\_NUM\_INTR-1)

119 -- DO NOT EDIT ABOVE THIS LINE ---------------------

120 );

121

122 attribute MAX\_FANOUT : string;

123 attribute SIGIS : string;

124

125 attribute SIGIS of Bus2IP\_Clk : signal is "CLK";

126 attribute SIGIS of Bus2IP\_Reset : signal is "RST";

127

128 end entity user\_logic;

129

130 ------------------------------------------------------------------------------

131 -- Architecture section

132 ------------------------------------------------------------------------------

133

134 architecture IMP of user\_logic is

135

136 --USER signal declarations added here, as needed for user logic

137

138 ------------------------------------------

139 -- Signals for user logic slave model s/w accessible register example

140 ------------------------------------------

141 signal slv\_reg0 : std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

142 signal slv\_reg1 : std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

143 signal slv\_reg2 : std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

144 signal slv\_reg3 : std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

145 signal slv\_reg\_write\_sel : std\_logic\_vector(0 to 3);

146 signal slv\_reg\_read\_sel : std\_logic\_vector(0 to 3);

147 signal slv\_ip2bus\_data : std\_logic\_vector(0 to C\_SLV\_DWIDTH-1);

148 signal slv\_read\_ack : std\_logic;

149 signal slv\_write\_ack : std\_logic;

150

151 ------------------------------------------

152 -- Signals for user logic interrupt example

153 ------------------------------------------

154 signal intr\_counter : std\_logic\_vector(0 to C\_NUM\_INTR-1);

155

156 begin

157

>>>>>>

인터럽트를 만들기 위한 카운터를 만들고 특정 조건일 때 인터럽트가 발생하도록 합니다.

240 ------------------------------------------

241 -- Example code to generate user logic interrupts

242 --

243 -- Note:

244 -- The example code presented here is to show you one way of generating

245 -- interrupts from the user logic. This code snippet infers a counter

246 -- and generate the interrupts whenever the counter rollover (the counter

247 -- will rollover ~21 sec @50Mhz).

248 ------------------------------------------

249 INTR\_PROC : process( Bus2IP\_Clk ) is

250 constant COUNT\_SIZE : integer := 30;

251 constant ALL\_ONES : std\_logic\_vector(0 to COUNT\_SIZE-1) := (others => '1');

252 variable counter : std\_logic\_vector(0 to COUNT\_SIZE-1);

253 begin

254

255 if ( Bus2IP\_Clk'event and Bus2IP\_Clk = '1' ) then

256 if ( Bus2IP\_Reset = '1' ) then

257 counter := (others => '0');

258 intr\_counter <= (others => '0');

259 else

260 counter := counter + 1;

261 if ( counter = ALL\_ONES ) then

262 intr\_counter <= (others => '1');

263 else

264 intr\_counter <= (others => '0');

265 end if;

266 end if;

267 end if;

268

269 end process INTR\_PROC;

270

271 IP2Bus\_IntrEvent <= intr\_counter;

272

273 ------------------------------------------

274 -- Example code to drive IP to Bus signals

275 ------------------------------------------

276 IP2Bus\_Data <= slv\_ip2bus\_data when slv\_read\_ack = '1' else

277 (others => '0');

278

279 IP2Bus\_WrAck <= slv\_write\_ack;

280 IP2Bus\_RdAck <= slv\_read\_ack;

281 IP2Bus\_Error <= '0';

282

283 end IMP;

### Axi\_interrupt/MPD file

다음 리스트는 axi\_interrupt를 구현한 MPD 파일 입니다.

1 ###################################################################

2 ##

3 ## Name : axi\_interrupt

4 ## Desc : Microprocessor Peripheral Description

5 ## : Automatically generated by PsfUtility

6 ##

7 ###################################################################

8

9 BEGIN axi\_interrupt

10

11 ## Peripheral Options

12 OPTION IPTYPE = PERIPHERAL

13 OPTION IMP\_NETLIST = TRUE

14 OPTION HDL = VHDL

15 OPTION IP\_GROUP = MICROBLAZE:USER

16 OPTION DESC = AXI\_INTERRUPT

17 OPTION ARCH\_SUPPORT\_MAP = (others=DEVELOPMENT)

18

19

20 ## Bus Interfaces

21 BUS\_INTERFACE BUS = S\_AXI, BUS\_STD = AXI, BUS\_TYPE = SLAVE

22

23 ## Generics for VHDL or Parameters for Verilog

24 PARAMETER C\_S\_AXI\_DATA\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

25 PARAMETER C\_S\_AXI\_ADDR\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI, ASSIGNMENT = CONSTANT

26 PARAMETER C\_S\_AXI\_MIN\_SIZE = 0x000001ff, DT = std\_logic\_vector, BUS = S\_AXI

27 PARAMETER C\_USE\_WSTRB = 0, DT = INTEGER

28 PARAMETER C\_DPHASE\_TIMEOUT = 8, DT = INTEGER

29 PARAMETER C\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, MIN\_SIZE = 0x200, PAIR = C\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI

30 PARAMETER C\_HIGHADDR = 0x00000000, DT = std\_logic\_vector, PAIR = C\_BASEADDR, ADDRESS = HIGH, BUS = S\_AXI

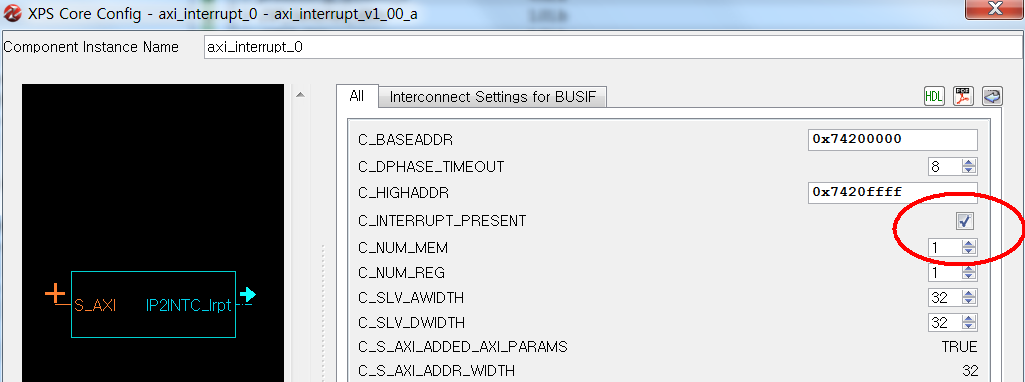
31 PARAMETER C\_FAMILY = virtex6, DT = STRING

32 PARAMETER C\_NUM\_REG = 1, DT = INTEGER

33 PARAMETER C\_NUM\_MEM = 1, DT = INTEGER

라인 34에 있는 파라미터를 사용하면 IP를 구성할 때 인터럽트를 사용할 것인지를 결정할 수 있습니다. 현재 기본 값은 0이므로 포트는 나타나지 않지만 체크 박스는 나타나 있습니다.

다음 그림은 XPS에서 인터럽트를 활성화 시킨 것 입니다.



34 PARAMETER C\_INTERRUPT\_PRESENT = 0, DT = INTEGER, RANGE = (0,1)

35 PARAMETER C\_SLV\_AWIDTH = 32, DT = INTEGER

36 PARAMETER C\_SLV\_DWIDTH = 32, DT = INTEGER

37 PARAMETER C\_S\_AXI\_PROTOCOL = AXI4LITE, TYPE = NON\_HDL, ASSIGNMENT = CONSTANT, DT = STRING, BUS = S\_AXI

38

39 ## Ports

40 PORT S\_AXI\_ACLK = "", DIR = I, SIGIS = CLK, BUS = S\_AXI

41 PORT S\_AXI\_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS = S\_AXI

42 PORT S\_AXI\_AWADDR = AWADDR, DIR = I, VEC = [(C\_S\_AXI\_ADDR\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

43 PORT S\_AXI\_AWVALID = AWVALID, DIR = I, BUS = S\_AXI

44 PORT S\_AXI\_WDATA = WDATA, DIR = I, VEC = [(C\_S\_AXI\_DATA\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

45 PORT S\_AXI\_WSTRB = WSTRB, DIR = I, VEC = [((C\_S\_AXI\_DATA\_WIDTH/8)-1):0], ENDIAN = LITTLE, BUS = S\_AXI

46 PORT S\_AXI\_WVALID = WVALID, DIR = I, BUS = S\_AXI

47 PORT S\_AXI\_BREADY = BREADY, DIR = I, BUS = S\_AXI

48 PORT S\_AXI\_ARADDR = ARADDR, DIR = I, VEC = [(C\_S\_AXI\_ADDR\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

49 PORT S\_AXI\_ARVALID = ARVALID, DIR = I, BUS = S\_AXI

50 PORT S\_AXI\_RREADY = RREADY, DIR = I, BUS = S\_AXI

51 PORT S\_AXI\_ARREADY = ARREADY, DIR = O, BUS = S\_AXI

52 PORT S\_AXI\_RDATA = RDATA, DIR = O, VEC = [(C\_S\_AXI\_DATA\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

53 PORT S\_AXI\_RRESP = RRESP, DIR = O, VEC = [1:0], BUS = S\_AXI

54 PORT S\_AXI\_RVALID = RVALID, DIR = O, BUS = S\_AXI

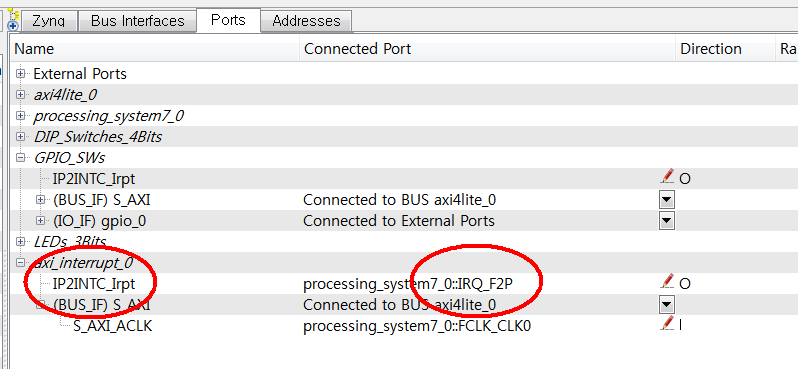
55 PORT S\_AXI\_WREADY = WREADY, DIR = O, BUS = S\_AXI

56 PORT S\_AXI\_BRESP = BRESP, DIR = O, VEC = [1:0], BUS = S\_AXI

57 PORT S\_AXI\_BVALID = BVALID, DIR = O, BUS = S\_AXI

58 PORT S\_AXI\_AWREADY = AWREADY, DIR = O, BUS = S\_AXI

인터럽트를 추가하기 위해서 포트를 설정하고 인터럽트가 활성화 되었을 때 인터럽트 핀이 나타나도록 하였습니다.



59 PORT IP2INTC\_Irpt = "", DIR = O, SIGIS = INTERRUPT, SENSITIVITY = LEVEL\_HIGH, INTERRUPT\_PRIORITY = MEDIUM, ISVALID = (C\_INTERRUPT\_PRESENT == 1)

60

61 END

### C source code

1 #include <stdio.h>

2 #include "xil\_io.h"

3 #include "xil\_types.h"

4 #include "axi\_interrupt.h"

5 #include "xparameters.h"

6 #include "xil\_io.h"

7 #include "xil\_exception.h"

8 #include "xscugic.h"

9

10 #define SLCR\_CAN\_RST\_ADDR 0xF8000220

11 #define SLCR\_CAN\_RST\_VALUE 0xF

12 #define SLCR\_WDT\_CLK\_SEL (XPS\_SYS\_CTRL\_BASEADDR + 0x304)

13 #define SLCR\_LOCK\_ADDR (XPS\_SYS\_CTRL\_BASEADDR + 0x4)

14 #define SLCR\_UNLOCK\_ADDR (XPS\_SYS\_CTRL\_BASEADDR + 0x8)

15 #define SLCR\_FPGA\_RST\_CTRL\_ADDR (XPS\_SYS\_CTRL\_BASEADDR + 0x240)

16 #define SLCR\_LVL\_SHFTR\_EN\_ADDR (XPS\_SYS\_CTRL\_BASEADDR + 0x900)

17 #define SLCR\_LVL\_SHFTR\_EN\_VALUE 0xF

18 #define SLCR\_LOCK\_KEY\_VALUE 0x767B

19 #define SLCR\_UNLOCK\_KEY\_VALUE 0xDF0D

20 #define SLCR\_MIO\_LOOPBACK (XPS\_SYS\_CTRL\_BASEADDR + 0x804)

21 #define READ\_WRITE\_MUL\_FACTOR 0x10

22

23

24

징크에서 인터럽트를 사용하기 위한 기본적인 규칙이 있습니다. 먼저 라인 26,27처럼 구조체를 선언 합니다.

25

26 XScuGic InterruptController; /\* Instance of the Interrupt Controller \*/

27 static XScuGic\_Config \*GicConfig;/\* The configuration parameters of the controller \*/

28

라인 29~41까지 인터럽트 핸들러를 작성 합니다.

29 void AXI\_INTERRUPT\_Intr\_DefaultHandler91(void \* baseaddr\_p)

30 {

31 Xuint32 baseaddr;

32 Xuint32 IntrStatus;

33 Xuint32 IpStatus;

34 baseaddr = (Xuint32) baseaddr\_p;

35

36 {

37 xil\_printf("User logic interrupt 91! \n\r");

38 IpStatus = AXI\_INTERRUPT\_mReadReg(baseaddr, AXI\_INTERRUPT\_INTR\_IPISR\_OFFSET);

39 AXI\_INTERRUPT\_mWriteReg(baseaddr, AXI\_INTERRUPT\_INTR\_IPISR\_OFFSET, IpStatus);

40 }

41 }

42

라인 43~62까지 인터럽트 구조체를 초기화 합니다.

43 int ScuGicInterrupt\_Init()

44 {

45 int Status;

46 /\*

47 \* Initialize the interrupt controller driver so that it is ready to

48 \* use.

49 \* \*/

50 Xil\_ExceptionInit();

51

52 GicConfig = XScuGic\_LookupConfig(XPAR\_PS7\_SCUGIC\_0\_DEVICE\_ID);

53 if (NULL == GicConfig) {

54 return XST\_FAILURE;

55 }

56

57 Status = XScuGic\_CfgInitialize(&InterruptController, GicConfig,

58 GicConfig->CpuBaseAddress);

59

60 if (Status != XST\_SUCCESS) {

61 return XST\_FAILURE;

62 }

63

64 /\*

65 \* Setup the Interrupt System

66 \* \*/

67

라인 68~81까지 기본 인터럽트 벡터 테이블을 초기화 합니다.

68 /\*

69 \* Connect the interrupt controller interrupt handler to the hardware

70 \* interrupt handling logic in the ARM processor.

71 \*/

72 Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_IRQ\_INT,

73 (Xil\_ExceptionHandler) XScuGic\_InterruptHandler,

74 (void \*) &InterruptController);

75

76

77 /\*

78 \* Connect a device driver handler that will be called when an

79 \* interrupt for the device occurs, the device driver handler performs

80 \* the specific interrupt processing for the device

81 \*/

90에 인터럽트 소스와 91번 인터럽트 소스에 대해서 인터럽트 핸들러와 연결 합니다.

82 Status = XScuGic\_Connect(&InterruptController,90,

83 (Xil\_ExceptionHandler)AXI\_INTERRUPT\_Intr\_DefaultHandler,

84 (void \*) &InterruptController);

85

86 Status = XScuGic\_Connect(&InterruptController,91,

87 (Xil\_ExceptionHandler)AXI\_INTERRUPT\_Intr\_DefaultHandler91,

88 (void \*) &InterruptController);

89

각 인터럽트 소스의 인터럽트 요청을 활성화 합니다.

90 XScuGic\_Enable(&InterruptController, 90);

91 XScuGic\_Enable(&InterruptController, 91);

92

93 AXI\_INTERRUPT\_EnableInterrupt(XPAR\_AXI\_INTERRUPT\_0\_BASEADDR);

94

95 /\*

96 \* Enable interrupts in the ARM

97 \*/

98 Xil\_ExceptionEnable();

99

라인 100 ~ 109 인터럽트 소스가 rising edge일 때 인터럽트 요청을 받을 수 있도록 합니다.

100 //Only used for edge sensitive Interrupts

101 XScuGic\_SetPriorityTriggerType(&InterruptController, 90,

102 0xa0, 3);

103

104 XScuGic\_SetPriorityTriggerType(&InterruptController, 91,

105 0xa0, 3);

106

107 if (Status != XST\_SUCCESS) {

108 return XST\_FAILURE;

109 }

110

111 }

112

113

114

115 int main()

116 {

117

118 int xstatus;

119 int write\_loop\_index;

120 int read\_loop\_index;

121 int TEST\_AXI\_USER\_NUM\_REG = 4;

122

123 /\* SLCR unlock \*/

124 Xil\_Out32(SLCR\_UNLOCK\_ADDR, SLCR\_UNLOCK\_KEY\_VALUE);

125

126 /\* SLCR Enabling Level shifting \*/

127 Xil\_Out32(SLCR\_LVL\_SHFTR\_EN\_ADDR, SLCR\_LVL\_SHFTR\_EN\_VALUE);

128 /\* SLCR clearing PL Reset \*/

129 Xil\_Out32(SLCR\_FPGA\_RST\_CTRL\_ADDR, 0x0);

130 Xil\_Out32(SLCR\_FPGA\_RST\_CTRL\_ADDR, 0xf);

131 Xil\_Out32(SLCR\_FPGA\_RST\_CTRL\_ADDR, 0x0);

132

133 /\* SLCR lock \*/

134 Xil\_Out32(SLCR\_LOCK\_ADDR, SLCR\_LOCK\_KEY\_VALUE);

135

136 xil\_printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

137 xil\_printf("\* User Peripheral Self Test\n\r");

138 xil\_printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\r");

139

140 /\*

141 \* Write to user logic slave module register(s) and read back

142 \*/

143 xil\_printf("User logic slave module test...\n\r");

144

145 for (write\_loop\_index = 0 ; write\_loop\_index < TEST\_AXI\_USER\_NUM\_REG; write\_loop\_index++)

146

147 AXI\_INTERRUPT\_mWriteSlaveReg0 (XPAR\_AXI\_INTERRUPT\_0\_BASEADDR, write\_loop\_index\*4, (write\_loop\_index+1)\*READ\_WRITE\_MUL\_FACTOR);

148 for (read\_loop\_index = 0 ; read\_loop\_index < TEST\_AXI\_USER\_NUM\_REG; read\_loop\_index++)

149 if ( AXI\_INTERRUPT\_mReadSlaveReg0 (XPAR\_AXI\_INTERRUPT\_0\_BASEADDR, read\_loop\_index\*4) != (read\_loop\_index+1)\*READ\_WRITE\_MUL\_FACTOR)

150 {

151 xil\_printf (" - slave register write/read failed\n\r");

152 }

153 else

154 {

155 xil\_printf(" - slave register write/read passed\n\r");

156 }

157

158 /\*

159 \* Reset the device to get it back to the default state

160 \*/

161

162 xil\_printf("Soft reset test...\n\r");

163 AXI\_INTERRUPT\_mReset(XPAR\_AXI\_INTERRUPT\_0\_BASEADDR);

164 xil\_printf(" - write 0x0000000A to software reset register\n\r");

165 /\* Read the registers at the base address to ensure that this is indeed working \*/

166 if ( (AXI\_INTERRUPT\_mReadSlaveReg0 (XPAR\_AXI\_INTERRUPT\_0\_BASEADDR, 0)) != 0x0){

167 xil\_printf(" - soft reset failed\n\n\r");

168 }

169 else{

170 xil\_printf(" - soft reset passed\n\n\r");

171 }

172

173 /\*

174 \* Interrupt Test

175 \*/

176

177

178

179

180 xstatus = ScuGicInterrupt\_Init();

181

182 if (xstatus != XST\_SUCCESS) {

183 return XST\_FAILURE;

184 }

185

186 //Wait For interrupt;

187

188 print("Wait for the interrupt to trigger (~15 seconds) \r\n");

189 print("########################################\r\n");

190 print(" \r\n");

191

15초 마다 메시지가 출력 되는지 기다려 봅시다.

192 while(1)

193 {

194 }

195

196 return 0;

197 }

198

199

200

201

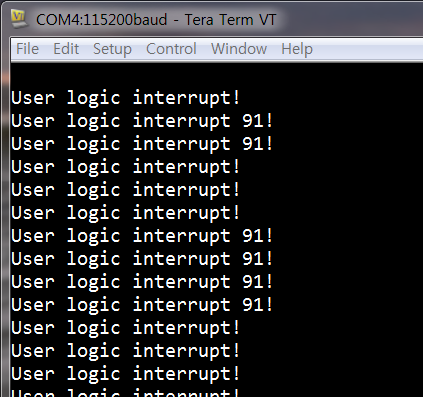
202

203

204

### TeraTerm을 이용한 확인

화면에 2종류의 인터럽트 소스를 확인 할 수 있습니다. 먼저 axi\_interrupt에 의해서는 정기적으로 인터럽트가 발생하고 91이라고 하는 것은 스위치가 움직일 때 마다 인터럽트가 발생한다는 것을 보여주고 있습니다.



## Bug at 14.4

EDK14.4에는 약간의 버그가 있습니다.

아래 라인 101을 보면 90이라는 숫자가 있는데 사실 xparameters.h에서 이 값은 적당한 매크로 값으로 바꿔져야 합니다.

100 //Only used for edge sensitive Interrupts

101 XScuGic\_SetPriorityTriggerType(&InterruptController, 90,

102 0xa0, 3);

103

104 XScuGic\_SetPriorityTriggerType(&InterruptController, 91,

105 0xa0, 3);

그런데 system.mhs에 있는 external port 에 있는 신호를 인터럽트 소스로 등록 할 때는 툴에서 제대로 매크로를 만들어 내지 못합니다.