# 프로세서, USER REGISTER ACCESS

일반적으로 FPGA 내부 프로세서를 사용할 경우 FPGA내에 설계하는 사용자 로직과 데이터를 주고받는 경우가 많습니다.

하지만 프로세서가 버스를 통해 user logic을 억세스하기 위해서는 버스 신호와 프로토콜을 이해해야 합니다. 정말 지루하고 따분하고 읽어도 읽어도 알아듣지 못할 말로 잔뜩 써 있는 것이 데이터 시트입니다.

XPS로 프로세서를 설계하면 user logic을 억세스하는 IP를 따로 제공하지 않기 때문에 설계자가 직접 만들어야 합니다.

뭐 프로토콜도 잘 알고 있고 프로세서의 동작상황도 잘 알고 있으면 못 만들 것도 없지만 생소한 프로그램에 낯선 프로세서와 처음 보는 프로토콜에 골탕 좀 먹으면 영 몹쓸 프로세서로 낙인 찍히기 쉽습니다.

그래서 이번 장에서는 user logic을 억세스하기 위한 제가 만든 IP를 사용하는 방법에 대해 살펴 보겠습니다.

## 내용 이해하기

일반적으로 임베디드 시스템을 설계하면 일단 프로세서를 기반으로 여러 주변장치들을 버스에 연결하고 동작을 상태를 점검하게 됩니다.

그러면 하나의 FPGA에는 하나의 임베디드 시스템을 구현하게 되는 거지요.



하지만 FPGA 내부에 임베디드 시스템을 하나만 구현하는 경우는 거의 없습니다. 다른 유저로직이 연결 됩니다.

이때 문제는 이 유저 로직과 임베디드 시스템간에 데이터를 교환해야 하는데 이 부분이 조금 애매 합니다.



일반적으로 FPGA 설계자라고 한다면 임베디드 시스템과 유저 로직간에 데이터를 주고 받을 때 필요한 신호 또는 프로토콜은 다음과 같이 생각하는 경우가 많습니다.



왜 자일링스에서 이런 기본적인 IP를 만들지 않는지는 모르겠어요.

뭐 여러가지 이유가 있겠지만 가장 큰 이유는 저 같은 사람 고생하라고 하는 것이 아닐까 합니다.

결국에 만들려고 하는 IP는 다음 그림과 같이 필요한 신호들을 유저 로직에서 볼 수 있도록 해주는 것 입니다.



데이터를 주고 받기 위한 신호들은 정리 했지만 이것을 어떻게 만드느냐?

그리고 실제 읽고 쓰는 타이밍은 어떻게 되는가에 대해서 차례대로 알아보겠습니다.

먼저 신호의 이름을 분석해 보겠습니다.

먼저 데이터 버스는 입력과 출력이 따로 정의된 것을 알 수 있습니다. 마스터가 데이터를 출력할 때는 data\_to\_user 포트를 통해 데이터를 출력하고 데이터 입력은 data\_cs[3:0] 따라 data\_from\_user0 ~ data\_from\_user3 포트를 통해서 입력된다는 것으로 추정할 수 있습니다.

어드레스는 모두 10비트 이므로 1024개의 번지를 지정할 수 있습니다.

User\_cs가 4비트이고 어드레스가 10비트이고 데이터가 32비트 이므로 4 \* 1024 개의 32비트 워드를 디코딩할 수 있다는 얘기 입니다.

자 이런 것을 만들기 위해서 어떻게 해야 하는가?

막상 신호 자체는 간단해 보이지만 만들려고 하면 머리가 멍~~~ 해지는 것이 사실 입니다.







처음부터 뎀비면 마이 다칩니다.

먼저 자일링스에서 제공하는 기본적인 몇가지 기능을 알아 보도록 하겠습니다.

사실 이 글을 읽고 있는 분들은 마이크로블레이즈가 어떤 버스에서 어떤 프로토콜을 통해 데이터를 주고 받는지 알지 못할 겁니다.

그러니까 한 1000 여 페이지 정도 되는 데이터 쉬트를 읽어 보고 버스와 프로토콜을 이해해 본인 스스로 HDL 코딩하라고 하면 대부분 포기하게 되죠.

그래서 자일링스는 유저 로직을 억세스할 수 있는 기본 플랫폼을 제시해 줍니다.



자 이제 그 기본 플랫폼을 만들어 봅시다.

이 부분은 비디오 클립으로 보는 것이 좋을 것 같습니다.

설치된 폴더를 보니까 아래와 같습니다.

$ISE\_INSTALL$\ISE\_DS\edk\_user\_repository\MyProcessorIPLib\pcores

폴더 이름은 axi\_user\_logic\_4096x32\_4cs\_10clk\_v1\_00\_a 입니다.

axi\_user\_logic\_4096x32\_4cs\_10clk\_v1\_00\_a 폴더 밑에는 3개의 폴더가 기본적으로 있습니다.

Data

Dev

Hdl

여기서 저는 먼저 hdl 폴더에 먼저 관심을 가져 봅니다.

Hdl 폴더 밑, VHDL 폴더 밑에 보면 axi\_user\_logic\_4096x32\_4cs\_10clk.vhd 파일과 user\_logic.vhd 파일 2개가 있습니다.

먼저 user\_logic.vhd를 보면 다음과 같습니다.

굳이 이 코드를 설명하려고 덤비는 이유는 다음과 같습니다. 먼저 버스 트랜잭션, 즉 버스를 통해 데이터를 주고 받을 때 언제 시작해서 언제 끝냐느냐를 알아야 하는데 … 이것 땜시 전체 소스를 가져다 붙였습니다.

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

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

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

4 --

5 -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6 -- \*\* Copyright (c) 1995-2011 Xilinx, Inc. All rights reserved. \*\*

7 -- \*\* \*\*

8 -- \*\* Xilinx, Inc. \*\*

9 -- \*\* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" \*\*

10 -- \*\* AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND \*\*

11 -- \*\* SOLUTIONS FOR XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, \*\*

12 -- \*\* OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, \*\*

13 -- \*\* APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION \*\*

14 -- \*\* THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, \*\*

15 -- \*\* AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE \*\*

16 -- \*\* FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY \*\*

17 -- \*\* WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE \*\*

18 -- \*\* IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR \*\*

19 -- \*\* REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF \*\*

20 -- \*\* INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS \*\*

21 -- \*\* FOR A PARTICULAR PURPOSE. \*\*

22 -- \*\* \*\*

23 -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24 --

25 ------------------------------------------------------------------------------

26 -- Filename: user\_logic.vhd

27 -- Version: 1.00.a

28 -- Description: User logic.

29 -- Date: Sat Dec 24 17:14:44 2011 (by Create and Import Peripheral Wizard)

30 -- VHDL Standard: VHDL'93

31 ------------------------------------------------------------------------------

32 -- Naming Conventions:

33 -- active low signals: "\*\_n"

34 -- clock signals: "clk", "clk\_div#", "clk\_#x"

35 -- reset signals: "rst", "rst\_n"

36 -- generics: "C\_\*"

37 -- user defined types: "\*\_TYPE"

38 -- state machine next state: "\*\_ns"

39 -- state machine current state: "\*\_cs"

40 -- combinatorial signals: "\*\_com"

41 -- pipelined or register delay signals: "\*\_d#"

42 -- counter signals: "\*cnt\*"

43 -- clock enable signals: "\*\_ce"

44 -- internal version of output port: "\*\_i"

45 -- device pins: "\*\_pin"

46 -- ports: "- Names begin with Uppercase"

47 -- processes: "\*\_PROCESS"

48 -- component instantiations: "<ENTITY\_>I\_<#|FUNC>"

49 ------------------------------------------------------------------------------

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

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

65 -- Entity section

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

67 -- Definition of Generics:

68 -- C\_SLV\_AWIDTH -- Slave interface address bus width

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

70 -- C\_NUM\_MEM -- Number of memory spaces

71 --

72 -- Definition of Ports:

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

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

75 -- Bus2IP\_Addr -- Bus to IP address bus

76 -- Bus2IP\_CS -- Bus to IP chip select for user logic memory selection

77 -- Bus2IP\_RNW -- Bus to IP read/not write

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

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

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

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

82 -- Bus2IP\_Burst -- Bus to IP burst-mode qualifier

83 -- Bus2IP\_BurstLength -- Bus to IP burst length

84 -- Bus2IP\_RdReq -- Bus to IP read request

85 -- Bus2IP\_WrReq -- Bus to IP write request

86 -- IP2Bus\_AddrAck -- IP to Bus address acknowledgement

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

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

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

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

91 -- Type\_of\_xfer -- Transfer Type

92 ------------------------------------------------------------------------------

93

94 entity user\_logic is

95 generic

96 (

97 -- ADD USER GENERICS BELOW THIS LINE ---------------

98 --USER generics added here

99 -- ADD USER GENERICS ABOVE THIS LINE ---------------

100

101 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

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

104 C\_SLV\_DWIDTH : integer := 32;

105 C\_NUM\_MEM : integer := 4

106 -- DO NOT EDIT ABOVE THIS LINE ---------------------

107 );

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;

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

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 ------------------------------------------

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

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

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);

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;

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

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

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;

전체 entity의 이름은 user logic이고요108라인부터 보면 포트를 정의했는데 111라인을 보면 “user port는 여기에다 정의해라” 라는 친절한 주석문이 있습니다.

116라인부터 134라인까지는 아마도 프로세서로에서 사용하는 버스의 신호라고 예상할 수 있습니다. 슬슬 읽어보면 대충 감을 때릴 수 있는데 조금 햇갈리는 것은 burst라든지 burst length 라는 것이 여기는 설명하지 않을 테니까 넘어가고요.

AddrAck, RaAck, WrAck같은 신호를 보니 음… 버스가 정해진 클럭에 트랜잭션이 끝나는 것이 아니고 슬레이브 쪽에서 응답을 해줘야 하나보다… 이렇게 추측할 수 있습니다.

신호이름들의 특성을 보면 Bus 🡺 IP, IP 🡺 Bus로 왔다 갔다 하는 방향이 정확힐 설정되어 있습니다. 다른 말로 입출력 포트가 아니고 입력이나 출력으로 단방향으로 정해졌다는 얘기 입니다.

당연한 얘기겠죠. FPGA 내부에서는 BUFT 가 없으니까요.



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;

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 );



 그래서 FPGA 내부에서는 IO라는 포트를 잘 사용하지 않고요 실제 FPGA의 IO와 만나는 부분에서만 사용하는 것이 좋습니다.

자 이제 entity 내부는 어떻게 구현되어 있는지 살펴보죠.

다음 코드는 메모리를 구현하는 VHDL 코드 입니다.

먼저 157, 158 라인에서 메모리 타입을 선언하고 8비트짜리 256개를 모두 4개 선언했네요. 그리고 나서 212라인부터 246 라인까지 BRAM을 인~퍼~런~싱~ 했습니다.

인퍼런싱이라는 것은 코드를 보면 내가 BRAM을 쓰지는 않았지만 합성툴이 가만보고 “아~~ 이놈의 레지스터 집합의 동작 특성을 보니 메모리구만… 그러면 BRAM으로 합성해야지”라고 자동으로 판단해서 BRAM을 사용하도록 해주는 코딩 스타일 입니다.

조금 더 자세히 애기하자면 합성툴은 레지스터의 크기와 동작특성을 보고 그냥 레지스터로 합성할지, Distributed Ram으로 합성할지 BRAM으로 합성할지 자동적으로 판단 합니다.

여기서는 입력과 출력이 모두 클럭에 동기가 되고 레지스터의 크기가 비교적 크다고 판단했기 때문에 합성툴은 자동으로 BRAM으로 합니다.

 그렇게 할 수도 있고 아닐 수도 있지만 인퍼런싱의 장점은 FPGA 패밀리에 영향을 받지 않는 다는 것과 심지어는 다른 벤더의 제품을 써도 (~~이런거 알려주면 안돼는데…쩝) 합성을 해준다는거… 이게 중요합니다.

즉 프로젝트가 바꿔도 디바이스 벤더가 바꿔도 (제발 그러지는 말아 주세요 흑흑) 같은 코드를 쓸 수 있다는 겁니다.

실제로 프로젝트할 때마다 바꿔보세요… 내가 왜 이 개고생인가 라는 생각이 저절로 납니다.

코드자체에서 어려운 점은 없어요.

어려우면 VHDL 공부부터 다시 하세요.

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);

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

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);

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

자 다음은 2가지를 한꺼번에 보겠습니다. 먼저 249라인을 보면 MUX가 하나 구현되어 있습니다.

CASE ~ WHEN 보면 이런 회로도가 당연히 떠 올라야 해요

안떠오르면

VHDL 공부부터 다시 하세요.

간단하네요 mem\_select에 따라서 mem\_ip2bus\_data에 연결되는 포트가 4개중 하나로 결정되네요.

두번째가 중요 합니다.

이렇게 mux에서 선택된 최정 출력은 다시 265라인에서 IP2Bus\_data로 연결되는데 그 시점이 언제냐 하면 mem\_read\_ack가 ‘1일 때 입니다. 그렇지 않으면 ‘0’ 입니다.

 FPGA 내부에는 ‘Z’ 신호를 만들어주는 BUFT가 없어요. 그러니까 FPGA 내부에는 OR 버스를 쓴다고 생각할 수 있습니다. OR 연산에서는 0은 DON’T CARE이니까… 국산 말로 “개 무 시” 라고 할 수 있습니다.

그러면 언제 mem\_read\_ack를 ‘1’ 만드느냐는 조금 있다가 설명하고 나머지 268, 269, 270 라인의 IP2Bus\_\*Ack 신호 삼총사는 일단 슬레이브에서 만드는 것을 확인할 수 있습니다.

 어떻게 만들까 이 삼총사도 조금있다가 설명 합니다.

271라인의 IP2Bus\_Error은 그냥 ‘0’으로 묶어 둡니다.

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

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

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

자 이제 이 ACK 삼총사를 설명하려고 합니다.

먼저 188라인에서 두개의 신호는 서로 연결되었고요 189라인을 보면 4개의 RdCE가 OR연산을 하네요. 4개중 어느 하나라도 ‘1이면 mem\_read\_enable은 ‘1’이 되죠.

192라인에서는 Bus2IP\_Addr중 일부분을 잘라내서 mem\_address라고 연결했습니다. 그런데 여기서 (9 downto 2) 라는 표현 중에 앞에 9는 어드레스 라인의 최대 크기를 정하는데 사용한다고 하지만 왜 ‘2’부터 어드레스라인을 연결했을까?

 이것은 데이터 버스의 크기가 32비트 이기 때문에 어드레스가 하나 증가할 때만 4바이트씩 증가한다라는 것을 강력히 암시하는 것 입니다.

아닙니다. 122라인을 보면 Bus2IP\_BE 라는 포트가 있습니다. 이 포트는 4개의 바이트중 어느 바이트를 통해 데이터를 주고 받는지를 알려 줍니다.

190라인과 191 라인을 보면 뭔가 좀 다르다는 느낌이 확 ~~ 다가 옵니다.

먼저 191라인을 보면 mem\_write\_ack는 4개의 Bus2IP\_WrCE 중 하나만 ‘1’ 이 되더라도 mem\_write\_ack가 ‘1’ 됩니다.

버스 마스터 입장에서는 쓰기 응답 (write ack)이 바로 돌아오기 때문에 아닌말로 한 클럭 만에 버스 트랜잭션을 마치게 됩니다.

그런데 190 라인의 mem\_read\_ack를 보면 조금 다르지요.

머리속에서 회로를 빨리 그려보세요.

일단 mem\_read\_enable을 보면 4개의 Bus2IP\_RdCE중 어느 하나라도 ‘1’ 이면 mem\_read\_enable도 ‘1’ 이 됩니다.

그런데 mem\_read\_enable 신호를 그대로 ack로 연결한 것이 아니라 라인 197번부터 208까지 보면 2개의 FF으로 연결한 것을 알 수 있습니다. 그리고 라인 190에서 NOT 연산과 AND 연산을 해서 mem\_read\_ack로 연결한 것을 확인할 수 있습니다.

그 이유는 앞에서 언급한 256개의 8비트 레지스터 블록 4개는 모두 BRAM으로 합성될 것을 알고 있는 것 입니다. BRAM에 데이터를 쓸 떄는 레이턴시가 없지만 읽을 때는 레이턴시가 1클럭 있어요.

그래서 버스 트랜잭션을 좀 길게 늘어뜨려야 하는데 그 방법이 바로 ack를 원하는 클럭 만큼 딜레이를 시켜서 넘겨 주는 것 입니다.

그렇게 하려면 한 클럭 딜레이 시키려고 하니까 FF을 사용해야 하고 한 펄스만을 만들려고 하니까 적당히 NOT 연산과 AND 연산을 하게 됩니다.

머리속에서 회로도 그려보시거나 한번 타이밍도를 그려보세요. 참고로 모두 ACTIVE HIGH 입니다.

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;

이렇게 함으로써 USER\_LOGIC.VHD를 분석을 했습니다.

다음은 AXI\_USER\_LOGIC\_.VHD 차례 입니다.

 이 코드는 너무 길고 관심을 가지고 살펴야 하는 부분이 적어서 일부 코드만 설명하도록 합니다.

라인 138에는 entity 이름이 정해져 있고요 라인 167부터 포트들이 정의 되어 있습니다. 라인 170을 보면 user port를 정하는 블록이 남겨져 있습니다. 나중에 이 부분을 수정하도록 하겠습니다.

138 entity axi\_user\_logic\_4096x32\_4cs\_10clk is

139 generic

140 (

141 -- ADD USER GENERICS BELOW THIS LINE ---------------

142 --USER generics added here

143 -- ADD USER GENERICS ABOVE THIS LINE ---------------

144

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

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

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

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

149 C\_S\_AXI\_ID\_WIDTH : integer := 4;

150 C\_RDATA\_FIFO\_DEPTH : integer := 0;

151 C\_INCLUDE\_TIMEOUT\_CNT : integer := 1;

152 C\_TIMEOUT\_CNTR\_VAL : integer := 8;

153 C\_ALIGN\_BE\_RDADDR : integer := 0;

154 C\_S\_AXI\_SUPPORTS\_WRITE : integer := 1;

155 C\_S\_AXI\_SUPPORTS\_READ : integer := 1;

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

157 C\_S\_AXI\_MEM0\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

158 C\_S\_AXI\_MEM0\_HIGHADDR : std\_logic\_vector := X"00000000";

159 C\_S\_AXI\_MEM1\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

160 C\_S\_AXI\_MEM1\_HIGHADDR : std\_logic\_vector := X"00000000";

161 C\_S\_AXI\_MEM2\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

162 C\_S\_AXI\_MEM2\_HIGHADDR : std\_logic\_vector := X"00000000";

163 C\_S\_AXI\_MEM3\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

164 C\_S\_AXI\_MEM3\_HIGHADDR : std\_logic\_vector := X"00000000"

165 -- DO NOT EDIT ABOVE THIS LINE ---------------------

166 );

167 port

168 (

169 -- ADD USER PORTS BELOW THIS LINE ------------------

170 --USER ports added here

171 -- ADD USER PORTS ABOVE THIS LINE ------------------

172

173 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

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

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

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

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

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

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

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

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

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

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

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

라인 383부터 420을 보면 앞에서 설명해드린 user\_logic을 port map하는 과정이 보입니다.

기억하실지 모르겠네요 Bus2IPxxxx, IP2Busxxxx라는 신호들 말입니다.

기억 안나는게 당연해요. 기억하는게 오히려 이상한 겁니다.

여러분 라인 383을 보면 조금 느낌이 다르다 ~~~ 라는 기분이 들지 않습니까?

Verilog와 VHDL을 비교해보면 서로 장단점이 있는데 그중 하나가 component를 선언하는냐 마느냐 입니다. 당근 VHDL을 선언하고 verilog는 특별히 하지는 않습니다.

VHDL에서도 component 선언 없이 port map을 할 수 있는데 그것이 바로 라인 383과 같이 하는 겁니다.

이렇게 선언해 주면 합성툴은 현재 폴터에서 user\_logic이라는 것이 있다고 가정을 하고 합성을 해주죠.

나중에 포트가 틀리면 어떻게 하죠?

 당연히 에러가 납니다.

하여간 나중에 user logic쪽의 포트가 수정된다면 당연히 라인 397근처에다 어떻게 달라졌는지 알려줘야 합니다.

383 USER\_LOGIC\_I : entity axi\_user\_logic\_4096x32\_4cs\_10clk\_v1\_00\_a.user\_logic

384 generic map

385 (

386 -- MAP USER GENERICS BELOW THIS LINE ---------------

387 --USER generics mapped here

388 -- MAP USER GENERICS ABOVE THIS LINE ---------------

389

390 C\_SLV\_AWIDTH => USER\_SLV\_AWIDTH,

391 C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH,

392 C\_NUM\_MEM => USER\_NUM\_MEM

393 )

394 port map

395 (

396 -- MAP USER PORTS BELOW THIS LINE ------------------

397 --USER ports mapped here

398 -- MAP USER PORTS ABOVE THIS LINE ------------------

399

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

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

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

403 Bus2IP\_CS => ipif\_Bus2IP\_CS(USER\_NUM\_MEM-1 downto 0),

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

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

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

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

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

409 Bus2IP\_Burst => ipif\_Bus2IP\_Burst,

410 Bus2IP\_BurstLength => user\_Bus2IP\_BurstLength,

411 Bus2IP\_RdReq => ipif\_Bus2IP\_RdReq,

412 Bus2IP\_WrReq => ipif\_Bus2IP\_WrReq,

413 IP2Bus\_AddrAck => user\_IP2Bus\_AddrAck,

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

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

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

417 IP2Bus\_Error => user\_IP2Bus\_Error,

418 Type\_of\_xfer => user\_Type\_of\_xfer

419 );

420

자 이렇게 해서 HDL 부분에 대한 설명을 마쳤습니다.

자 지금 부터는 실제 이 코드를 뜯어 고치도록 하겠습니다.

우리는 아래 그림이 필요 합니다.

USER\_LOGIC.VHD에는 이런 포트가 없어요.

만들어 줘야 합니다.



다음 리스트는 수정된 user \_logic.vhd 코드 입니다. 무엇이 달라졌는지 차레대로 설명 하겠습니다.

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

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

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

4 --

5 -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6 -- \*\* Copyright (c) 1995-2011 Xilinx, Inc. All rights reserved. \*\*

7 -- \*\* \*\*

8 -- \*\* Xilinx, Inc. \*\*

9 -- \*\* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" \*\*

10 -- \*\* AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND \*\*

11 -- \*\* SOLUTIONS FOR XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, \*\*

12 -- \*\* OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, \*\*

13 -- \*\* APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION \*\*

14 -- \*\* THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT, \*\*

15 -- \*\* AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE \*\*

16 -- \*\* FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY \*\*

17 -- \*\* WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE \*\*

18 -- \*\* IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR \*\*

19 -- \*\* REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF \*\*

20 -- \*\* INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS \*\*

21 -- \*\* FOR A PARTICULAR PURPOSE. \*\*

22 -- \*\* \*\*

23 -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24 --

25 ------------------------------------------------------------------------------

26 -- Filename: user\_logic.vhd

27 -- Version: 1.00.a

28 -- Description: User logic.

29 -- Date: Sun Dec 18 15:11:13 2011 (by Create and Import Peripheral Wizard)

30 -- VHDL Standard: VHDL'93

31 ------------------------------------------------------------------------------

32 -- Naming Conventions:

33 -- active low signals: "\*\_n"

34 -- clock signals: "clk", "clk\_div#", "clk\_#x"

35 -- reset signals: "rst", "rst\_n"

36 -- generics: "C\_\*"

37 -- user defined types: "\*\_TYPE"

38 -- state machine next state: "\*\_ns"

39 -- state machine current state: "\*\_cs"

40 -- combinatorial signals: "\*\_com"

41 -- pipelined or register delay signals: "\*\_d#"

42 -- counter signals: "\*cnt\*"

43 -- clock enable signals: "\*\_ce"

44 -- internal version of output port: "\*\_i"

45 -- device pins: "\*\_pin"

46 -- ports: "- Names begin with Uppercase"

47 -- processes: "\*\_PROCESS"

48 -- component instantiations: "<ENTITY\_>I\_<#|FUNC>"

49 ------------------------------------------------------------------------------

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

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

65 -- Entity section

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

67 -- Definition of Generics:

68 -- C\_SLV\_AWIDTH -- Slave interface address bus width

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

70 -- C\_NUM\_MEM -- Number of memory spaces

71 --

72 -- Definition of Ports:

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

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

75 -- Bus2IP\_Addr -- Bus to IP address bus

76 -- Bus2IP\_CS -- Bus to IP chip select for user logic memory selection

77 -- Bus2IP\_RNW -- Bus to IP read/not write

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

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

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

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

82 -- Bus2IP\_Burst -- Bus to IP burst-mode qualifier

83 -- Bus2IP\_BurstLength -- Bus to IP burst length

84 -- Bus2IP\_RdReq -- Bus to IP read request

85 -- Bus2IP\_WrReq -- Bus to IP write request

86 -- IP2Bus\_AddrAck -- IP to Bus address acknowledgement

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

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

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

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

91 -- Type\_of\_xfer -- Transfer Type

92 ------------------------------------------------------------------------------

93

94 entity user\_logic is

95 generic

96 (

97 -- ADD USER GENERICS BELOW THIS LINE ---------------

98 --USER generics added here

99 -- ADD USER GENERICS ABOVE THIS LINE ---------------

100

101 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

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

104 C\_SLV\_DWIDTH : integer := 32;

105 C\_NUM\_MEM : integer := 4

106 -- DO NOT EDIT ABOVE THIS LINE ---------------------

107 );

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(9 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

124 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

126 Bus2IP\_Clk : in std\_logic;

127 Bus2IP\_Resetn : in std\_logic;

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

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

130 Bus2IP\_RNW : in std\_logic;

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

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

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

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

135 Bus2IP\_Burst : in std\_logic;

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

137 Bus2IP\_RdReq : in std\_logic;

138 Bus2IP\_WrReq : in std\_logic;

139 IP2Bus\_AddrAck : out std\_logic;

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

141 IP2Bus\_RdAck : out std\_logic;

142 IP2Bus\_WrAck : out std\_logic;

143 IP2Bus\_Error : out std\_logic;

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

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

163 signal mem\_write\_enable : std\_logic;

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

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

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

167 signal mem\_write\_ack\_dly1 : std\_logic;

168 signal mem\_write\_ack\_dly2 : std\_logic;

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

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

171

172 begin

173 mem\_select <= Bus2IP\_CS;

174 user\_cs <= mem\_select;

175 ----

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

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

178 ----

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

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

181 ----

182 mem\_write\_ack <= mem\_write\_ack\_dly1 and (not mem\_write\_ack\_dly2);

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

184 ----

185 user\_clk <= Bus2IP\_Clk;

186 ----

187 user\_add <= Bus2IP\_Addr(11 downto 2);

188 ----

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

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

191

192

193 ACK\_PROC : process( Bus2IP\_Clk ) is

194 begin

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

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

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

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

199 mem\_write\_ack\_dly1 <= '0';

200 mem\_write\_ack\_dly2 <= '0';

201 else

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

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

204 mem\_write\_ack\_dly1 <= mem\_write\_enable;

205 mem\_write\_ack\_dly2 <= mem\_write\_ack\_dly1;

206 end if;

207 end if;

208 end process ACK\_PROC;

209

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

211 begin

212 case mem\_select is

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

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

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

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

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

218 end case;

219 end process MEM\_IP2BUS\_DATA\_PROC;

220

221 ------------------------------------------

222 -- Example code to drive IP to Bus signals

223 ------------------------------------------

224 IP2Bus\_Data <= mem\_ip2bus\_data when mem\_read\_ack = '1' else (others => '0');

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

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

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

228 IP2Bus\_Error <= '0';

229

230 end IMP;

먼저 entity 부분을 보면 라인 112부터 121까지 새로운 포트가 추가되었습니다. 새로추가된 포트들은 user\_rd, user\_wr, user\_cs … 등등등

entity user\_logic is

95 generic

96 (

97 -- ADD USER GENERICS BELOW THIS LINE ---------------

98 --USER generics added here

99 -- ADD USER GENERICS ABOVE THIS LINE ---------------

100

101 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

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

104 C\_SLV\_DWIDTH : integer := 32;

105 C\_NUM\_MEM : integer := 4

106 -- DO NOT EDIT ABOVE THIS LINE ---------------------

107 );

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(9 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 ------------------

먼저 user\_cs[3:0]을 만드는 과정을 보면 다음과 같습니다.

라인 105를 보면 현재 이 IP는 기본적으로 4개의 chip select를 가지고 있다는 것을 알 수 있습니다. Bus2IP\_CS를 meme\_select와 user\_cs로 연결 합니다. Mem\_select 신호는 라인 210부터 219까지 설명한 것과 같이 data-from\_user0 부터 3까지 4개의 32비트 입력중 하나를 선택해 mem\_ip2bus\_data 라는 신호에 연결 합니다.

94 entity user\_logic is

95 generic

96 (

97 -- ADD USER GENERICS BELOW THIS LINE ---------------

98 --USER generics added here

99 -- ADD USER GENERICS ABOVE THIS LINE ---------------

100

101 -- DO NOT EDIT BELOW THIS LINE ---------------------

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

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

104 C\_SLV\_DWIDTH : integer := 32;

105 C\_NUM\_MEM : integer := 4

106 -- DO NOT EDIT ABOVE THIS LINE ---------------------

107 );

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

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

153

154 end entity user\_logic;

155

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

157 -- Architecture section

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

159

160 architecture IMP of user\_logic is

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

171

172 begin

173 mem\_select <= Bus2IP\_CS;

174 user\_cs <= mem\_select;

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

211 begin

212 case mem\_select is

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

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

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

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

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

218 end case;

219 end process MEM\_IP2BUS\_DATA\_PROC;

자 이제 좀더 중요한 ack를 어떻게 할 것인가에 대해서 고민할 차례 입니다. 먼저 라인 175부터 180까지 각각 meme\_read\_enable과 mem-write\_enable 신호를 만듭니다. 이 신호는 그대로 user\_rd, user\_wr로 연결 됩니다.

175 ----

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

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

178 ----

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

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

181 ----

다음으로 ack 신호를 만들기 위해서 read/write에 대해 각각 2개씩 FF을 설계한후 mem\_read\_enable과 mem\_write\_enable 신호를 연결하면 2클럭 딜레이가 생기는데 이때 앞에서 라인 182, 183에서 하나의 펄스를 만들어 주고 그것을 그대로 mem\_write\_ack, mem\_read\_ack로 연결 합니다.

182 mem\_write\_ack <= mem\_write\_ack\_dly1 and (not mem\_write\_ack\_dly2);

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

193 ACK\_PROC : process( Bus2IP\_Clk ) is

194 begin

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

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

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

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

199 mem\_write\_ack\_dly1 <= '0';

200 mem\_write\_ack\_dly2 <= '0';

201 else

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

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

204 mem\_write\_ack\_dly1 <= mem\_write\_enable;

205 mem\_write\_ack\_dly2 <= mem\_write\_ack\_dly1;

206 end if;

207 end if;

208 end process ACK\_PROC;

이 ip에 의해서 억세스 되는 로직은 버스에서 사용하는 클럭을 그대로 사용하기 때문에 버스 클럭을 user\_clk으로 연결해 주고 user\_add로 Bus2IP\_Addr 포트를 적당히 잘라서 연결해 줍니다.

적당히 라는 말은 좀 애매하고 32비트 단위로 access하겠다고 하면 [#:2] 이런 식이 되어야 겠죠.

185 user\_clk <= Bus2IP\_Clk;

186 ----

187 user\_add <= Bus2IP\_Addr(11 downto 2);

 당연히 아니죠, user\_logic.vhd의 입출력 포트가 수정되었니 그것을 port map 하고 있는 axi\_user\_logicxxxx.vhd도 수정해야 합니다.

수정된 부분만 꺼내서 보면 다음과 같습니다.

먼저 라인 170부터 180라인을 보면 앞에서 추가된 여러 포트가 그대로 복사 됐습니다.

138 entity axi\_user\_logic\_4096x32\_4cs is

139 generic

140 (

141 -- ADD USER GENERICS BELOW THIS LINE ---------------

142 --USER generics added here

143 -- ADD USER GENERICS ABOVE THIS LINE ---------------

144

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

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

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

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

149 C\_S\_AXI\_ID\_WIDTH : integer := 4;

150 C\_RDATA\_FIFO\_DEPTH : integer := 0;

151 C\_INCLUDE\_TIMEOUT\_CNT : integer := 1;

152 C\_TIMEOUT\_CNTR\_VAL : integer := 8;

153 C\_ALIGN\_BE\_RDADDR : integer := 0;

154 C\_S\_AXI\_SUPPORTS\_WRITE : integer := 1;

155 C\_S\_AXI\_SUPPORTS\_READ : integer := 1;

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

157 C\_S\_AXI\_MEM0\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

158 C\_S\_AXI\_MEM0\_HIGHADDR : std\_logic\_vector := X"00000000";

159 C\_S\_AXI\_MEM1\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

160 C\_S\_AXI\_MEM1\_HIGHADDR : std\_logic\_vector := X"00000000";

161 C\_S\_AXI\_MEM2\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

162 C\_S\_AXI\_MEM2\_HIGHADDR : std\_logic\_vector := X"00000000";

163 C\_S\_AXI\_MEM3\_BASEADDR : std\_logic\_vector := X"FFFFFFFF";

164 C\_S\_AXI\_MEM3\_HIGHADDR : std\_logic\_vector := X"00000000"

165 -- DO NOT EDIT ABOVE THIS LINE ---------------------

166 );

167 port

168 (

169 -- ADD USER PORTS BELOW THIS LINE ------------------

170 --USER ports added here

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

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

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

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

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

176 user\_add : out std\_logic\_vector(9 downto 0);

177 user\_rd : out std\_logic;

178 user\_wr : out std\_logic;

179 user\_clk : out std\_logic;

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

181 -- ADD USER PORTS ABOVE THIS LINE ------------------

다음으로 포트맵 된 곳을 보겠습니다. 라인 408부터 417까지 보면 같은 신호 이름으로 그대로 포트 맵 되어 있습니다. User\_logic에 있는 port를 그대로 자신의 포트와 연결 했습니다.

390 ------------------------------------------

391 -- instantiate User Logic

392 ------------------------------------------

393 USER\_LOGIC\_I : entity axi\_user\_logic\_4096x32\_4cs\_v1\_00\_a.user\_logic

394 generic map

395 (

396 -- MAP USER GENERICS BELOW THIS LINE ---------------

397 --USER generics mapped here

398 -- MAP USER GENERICS ABOVE THIS LINE ---------------

399

400 C\_SLV\_AWIDTH => USER\_SLV\_AWIDTH,

401 C\_SLV\_DWIDTH => USER\_SLV\_DWIDTH,

402 C\_NUM\_MEM => USER\_NUM\_MEM

403 )

404 port map

405 (

406 -- MAP USER PORTS BELOW THIS LINE ------------------

407 --USER ports mapped here

408 data\_from\_user0 => data\_from\_user0,

409 data\_from\_user1 => data\_from\_user1,

410 data\_from\_user2 => data\_from\_user2,

411 data\_from\_user3 => data\_from\_user3,

412 data\_to\_user => data\_to\_user,

413 user\_add => user\_add,

414 user\_rd => user\_rd,

415 user\_wr => user\_wr,

416 user\_clk => user\_clk,

417 user\_cs => user\_cs,

418 -- MAP USER PORTS ABOVE THIS LINE ------------------

419

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

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

아닙니다. 아직 하나가 남았습니다.  
사실 이렇게 포트가 변경되면 포트가 변경된 것을 알려줘야 합니다.

어디에 알려줘야 하느냐 하면 mpd 파일 입니다. 다음 리스트는 수정된 mpd 파일인데 라인 43부터 52라인 까지 보면 수정된 포트들을 볼 수 있습니다.

이 부분은 누가 편집해야 하느냐 ? 그건 IP를 수정하는사람의 몫 입니다.

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

2 ##

3 ## Name : axi\_user\_logic\_4096x32\_4cs

4 ## Desc : Microprocessor Peripheral Description

5 ## : Automatically generated by PsfUtility

6 ##

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

8

9 BEGIN axi\_user\_logic\_4096x32\_4cs

10

11 ## Peripheral Options

12 OPTION IPTYPE = PERIPHERAL

13 OPTION IMP\_NETLIST = TRUE

14 OPTION HDL = MIXED

15 OPTION IP\_GROUP = MICROBLAZE:USER

16

17

18 ## Bus Interfaces

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

20

21 ## Generics for VHDL or Parameters for Verilog

22 PARAMETER C\_S\_AXI\_DATA\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI

23 PARAMETER C\_S\_AXI\_ADDR\_WIDTH = 32, DT = INTEGER, BUS = S\_AXI

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

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

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

27 PARAMETER C\_TIMEOUT\_CNTR\_VAL = 8, DT = INTEGER

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

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

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

31 PARAMETER C\_FAMILY = virtex6, DT = STRING

32 PARAMETER C\_S\_AXI\_MEM0\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM0\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI, ADDR\_TYPE = MEMORY

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

34 PARAMETER C\_S\_AXI\_MEM1\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM1\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI, ADDR\_TYPE = MEMORY

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

36 PARAMETER C\_S\_AXI\_MEM2\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM2\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI, ADDR\_TYPE = MEMORY

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

38 PARAMETER C\_S\_AXI\_MEM3\_BASEADDR = 0xffffffff, DT = std\_logic\_vector, PAIR = C\_S\_AXI\_MEM3\_HIGHADDR, ADDRESS = BASE, BUS = S\_AXI, ADDR\_TYPE = MEMORY

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

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

41

42 ## Ports

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

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

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

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

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

48 PORT user\_add = "", DIR = O, VEC = [9:0]

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

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

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

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

53 PORT S\_AXI\_ACLK = ACLK, DIR = I, SIGIS = CLK, BUS = S\_AXI

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

72 PORT S\_AXI\_AWID = AWID, DIR = I, VEC = [(C\_S\_AXI\_ID\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

73 PORT S\_AXI\_AWLEN = AWLEN, DIR = I, VEC = [7:0], BUS = S\_AXI

74 PORT S\_AXI\_AWSIZE = AWSIZE, DIR = I, VEC = [2:0], BUS = S\_AXI

75 PORT S\_AXI\_AWBURST = AWBURST, DIR = I, VEC = [1:0], BUS = S\_AXI

76 PORT S\_AXI\_AWLOCK = AWLOCK, DIR = I, BUS = S\_AXI

77 PORT S\_AXI\_AWCACHE = AWCACHE, DIR = I, VEC = [3:0], BUS = S\_AXI

78 PORT S\_AXI\_AWPROT = AWPROT, DIR = I, VEC = [2:0], BUS = S\_AXI

79 PORT S\_AXI\_WLAST = WLAST, DIR = I, BUS = S\_AXI

80 PORT S\_AXI\_BID = BID, DIR = O, VEC = [(C\_S\_AXI\_ID\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

81 PORT S\_AXI\_ARID = ARID, DIR = I, VEC = [(C\_S\_AXI\_ID\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

82 PORT S\_AXI\_ARLEN = ARLEN, DIR = I, VEC = [7:0], BUS = S\_AXI

83 PORT S\_AXI\_ARSIZE = ARSIZE, DIR = I, VEC = [2:0], BUS = S\_AXI

84 PORT S\_AXI\_ARBURST = ARBURST, DIR = I, VEC = [1:0], BUS = S\_AXI

85 PORT S\_AXI\_ARLOCK = ARLOCK, DIR = I, BUS = S\_AXI

86 PORT S\_AXI\_ARCACHE = ARCACHE, DIR = I, VEC = [3:0], BUS = S\_AXI

87 PORT S\_AXI\_ARPROT = ARPROT, DIR = I, VEC = [2:0], BUS = S\_AXI

88 PORT S\_AXI\_RID = RID, DIR = O, VEC = [(C\_S\_AXI\_ID\_WIDTH-1):0], ENDIAN = LITTLE, BUS = S\_AXI

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

90

91 END

자 이렇게 파일 3개를 수정 했습니다.

다음은 이 IP를 system.mhs 파일에는 어떻게 적용되는 예를 들어 설명하게습니다.

다음 리스트는 axi\_user\_logicxxx을 사용한 system.mhs 파일의 일부 입니다. 라인 46부터 55까지 external port로 axi\_user\_logicxxx 에서 나오는 포트가 그대로 1:1 로 선언 되어 있습니다.

라인 261을 보면 axi\_user\_logic\_xxx에 대한 선언이 있고 INSTANCE 이름으로 axi\_user\_logicxxxx\_0으로 정했습니다.

라인 264부터 271까지 4개의 CS에 의해서 선택되는 어드레스 공간을 정의했고 라인 274부터 283까지는 MPD 파일에 정의된 포트를 그대로 external port까지 연결 했습니다.

라인 272는 현재 axi\_user\_logicxxx 이 연결된 버스의 INSTANCE 이름 입니다.

사실 버스라고 할 때 이 버스는 단순히 하드웨어적인 신호의 연결 분 뿐 아니라 마스터와 슬레이브 간에 데이터를 교환할 수 있는 약속, 즉 프로토콜도 정의되어 있습니다. 이 프로토콜을 구현하기 위해서는 버스 자체도 설계하는 것이 쉽지 않습니다.

FPGA는 여러 개의 마이크로블레이즈를 합성할 수 있는데 각각 다른 마스터에 다른 버스를 연결할 수 있기 때문에 버스도 instance 이름으로 구분 합니다. 라인 297부터 303까지 보면 axi\_interconnect라는 IP를 사용했고 INSTANCE 이름으로 axi4lite\_0로 정했습니다. 라인 300을 보면 매우 중요한 옵션이 하나 있는데 axi의 내부 프로토콜을 어떻게 정의할 것인지를 결정 합니다.

예를 들어 C\_INTERCONNECT\_CONNECTIVITY\_MODE 값이 1 데이터를 최대한 빠르게 전송할 수 있도록 로직을 합성하고 값이 0이면 최소의 로직을 사용할 수 있도록 합성 해 줍니다.

이 버스에 연결되는 슬레이브들의 성질을 잘 살펴보고 나서 연결되는 방식을 결정해야 겠습니다.

Axi\_user\_logic을 데이터 주고 받는 양은 매우 적습니다. 그래서 여기서는 C\_INTERCONNECT\_CONNECTIVITY\_MODE = 0 인 axi4lite\_0에 연결한 것 입니다.

반면에 라인 305부터 310까지 보면 여기는 C\_INTERCONNECT\_CONNECTIVITY\_MODE 값이 1 입니다.

그렇지요. 이 파라미터에 대해서는 1이다 0이다 라고 값을 정하지 않았습니다. 각 IP는 각 파라미터 마다 기본 값이 있습니다. 디폴트 값이죠.

이 디폴트값을 그대로 사용할 때는 굳이 system.mhs 파일에 언급하지 않아도 됩니다. 즉 아무 것도 없기에 ‘1’ 이구나 라는 것을 알 수 있습니다.

mpd 파일을 보면 알 수 있습니다.

46 PORT user\_clk = axi\_user\_logic\_4096x32\_4cs\_0\_user\_clk, DIR = O

47 PORT data\_from\_user0 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user0, DIR = I, VEC = [31:0]

48 PORT user\_add = axi\_user\_logic\_4096x32\_4cs\_0\_user\_add, DIR = O, VEC = [9:0]

49 PORT data\_from\_user2 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user2, DIR = I, VEC = [31:0]

50 PORT data\_from\_user3 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user3, DIR = I, VEC = [31:0]

51 PORT data\_from\_user1 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user1, DIR = I, VEC = [31:0]

52 PORT data\_to\_user = axi\_user\_logic\_4096x32\_4cs\_0\_data\_to\_user, DIR = O, VEC = [31:0]

53 PORT user\_rd = axi\_user\_logic\_4096x32\_4cs\_0\_user\_rd, DIR = O

54 PORT user\_wr = axi\_user\_logic\_4096x32\_4cs\_0\_user\_wr, DIR = O

55 PORT user\_cs = axi\_user\_logic\_4096x32\_4cs\_0\_user\_cs, DIR = O, VEC = [3:0]

56

57

260

261 BEGIN axi\_user\_logic\_4096x32\_4cs

262 PARAMETER INSTANCE = axi\_user\_logic\_4096x32\_4cs\_0

263 PARAMETER HW\_VER = 1.00.a

264 PARAMETER C\_S\_AXI\_MEM0\_BASEADDR = 0x20000000

265 PARAMETER C\_S\_AXI\_MEM0\_HIGHADDR = 0x20000FFF

266 PARAMETER C\_S\_AXI\_MEM1\_BASEADDR = 0x21000000

267 PARAMETER C\_S\_AXI\_MEM1\_HIGHADDR = 0x21000FFF

268 PARAMETER C\_S\_AXI\_MEM2\_BASEADDR = 0x22000000

269 PARAMETER C\_S\_AXI\_MEM2\_HIGHADDR = 0x22000FFF

270 PARAMETER C\_S\_AXI\_MEM3\_BASEADDR = 0x23000000

271 PARAMETER C\_S\_AXI\_MEM3\_HIGHADDR = 0x23000FFF

272 BUS\_INTERFACE S\_AXI = axi4lite\_0

273 PORT S\_AXI\_ACLK = clk\_75\_0000MHzPLL0

274 PORT user\_clk = axi\_user\_logic\_4096x32\_4cs\_0\_user\_clk

275 PORT data\_from\_user0 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user0

276 PORT user\_add = axi\_user\_logic\_4096x32\_4cs\_0\_user\_add

277 PORT data\_from\_user2 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user2

278 PORT data\_from\_user3 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user3

279 PORT data\_from\_user1 = axi\_user\_logic\_4096x32\_4cs\_0\_data\_from\_user1

280 PORT data\_to\_user = axi\_user\_logic\_4096x32\_4cs\_0\_data\_to\_user

281 PORT user\_rd = axi\_user\_logic\_4096x32\_4cs\_0\_user\_rd

282 PORT user\_wr = axi\_user\_logic\_4096x32\_4cs\_0\_user\_wr

283 PORT user\_cs = axi\_user\_logic\_4096x32\_4cs\_0\_user\_cs

284 END

285

297 BEGIN axi\_interconnect

298 PARAMETER INSTANCE = axi4lite\_0

299 PARAMETER HW\_VER = 1.04.a

300 PARAMETER C\_INTERCONNECT\_CONNECTIVITY\_MODE = 0

301 PORT INTERCONNECT\_ARESETN = proc\_sys\_reset\_0\_Interconnect\_aresetn

302 PORT INTERCONNECT\_ACLK = clk\_75\_0000MHzPLL0

303 END

304

305 BEGIN axi\_interconnect

306 PARAMETER INSTANCE = axi4\_0

307 PARAMETER HW\_VER = 1.04.a

308 PORT interconnect\_aclk = clk\_75\_0000MHzPLL0

309 PORT INTERCONNECT\_ARESETN = proc\_sys\_reset\_0\_Interconnect\_aresetn

310 END

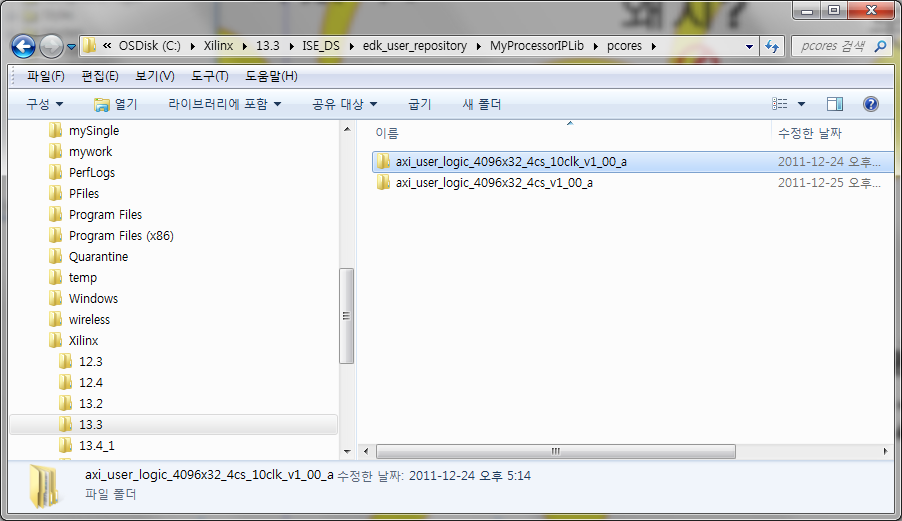
311

## Axi\_user\_logicxxx이 저장된 폴더를 찾아라

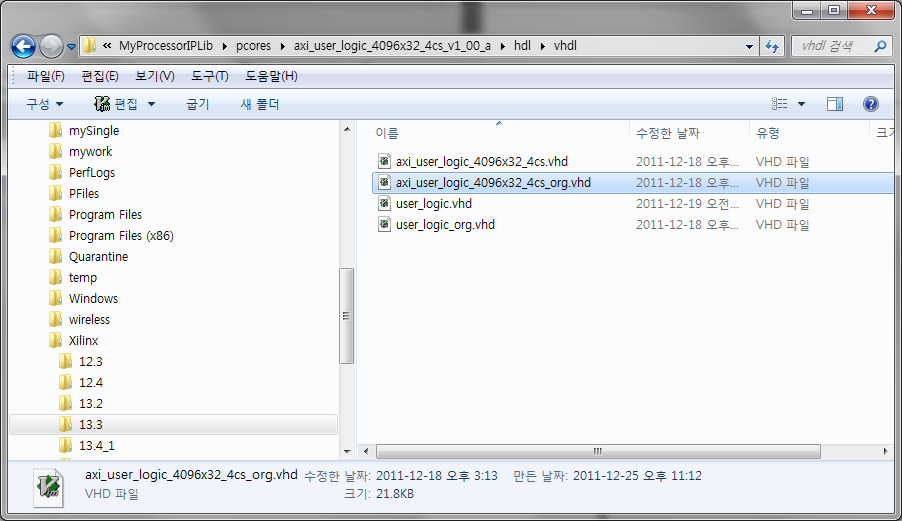
먼저 C:\Xilinx\13.3\ISE\_DS\edk\_user\_repository\MyProcessorIPLib\pcores폴더를 살펴 보겠습니다.

EDK에서 제공하지 않는 IP를 사용자가 만들었을 경우 다른 모든 XPS 프로젝트에서 사용할 수 있도록 하기 위해서는 이 폴더에 IP를 저장해야 합니다.

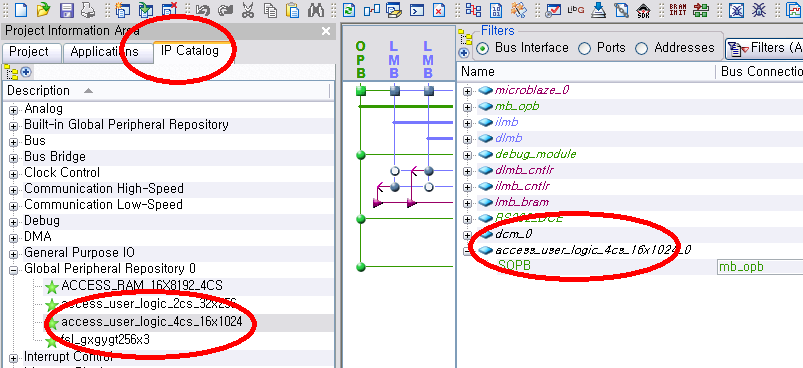
이 폴더에 axi\_user\_logic\_4096x32\_4cs\_v1\_00\_a가 있습니다.



이 IP는 access\_user\_logic\_4cs\_16x1024.vhd와 user\_logic.vhd로 구성 되어 있습니다.

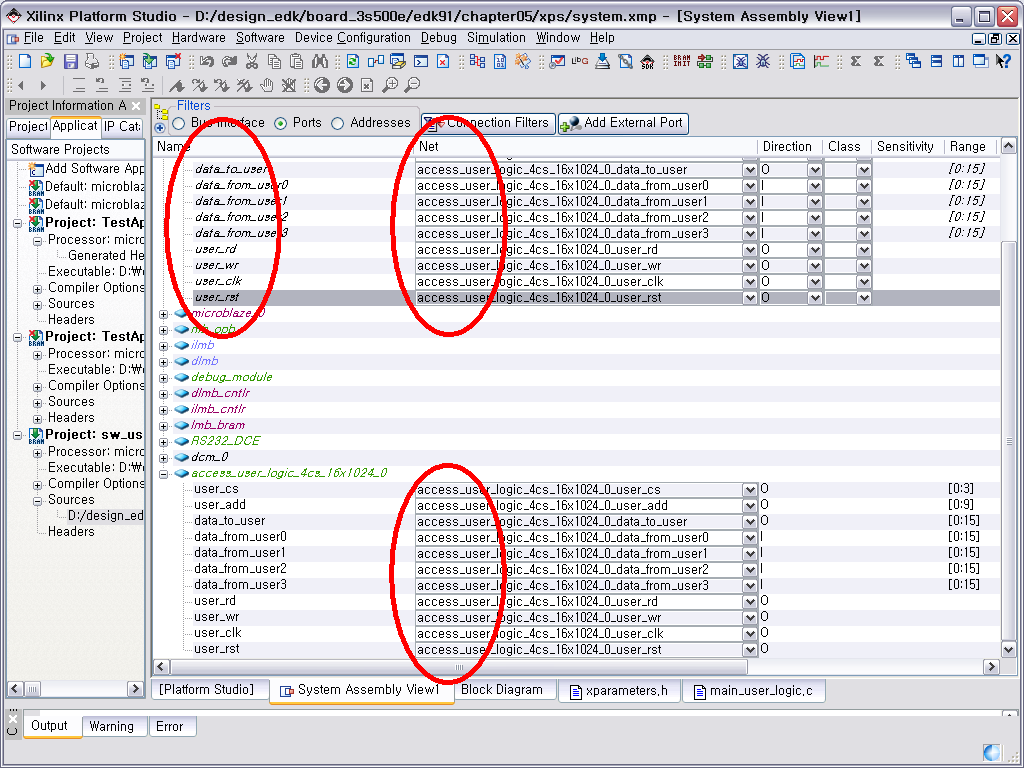


IP Catalog에서 access\_user\_logic을 drag & drop을 통해 가져와 버스에 연결 합니다.



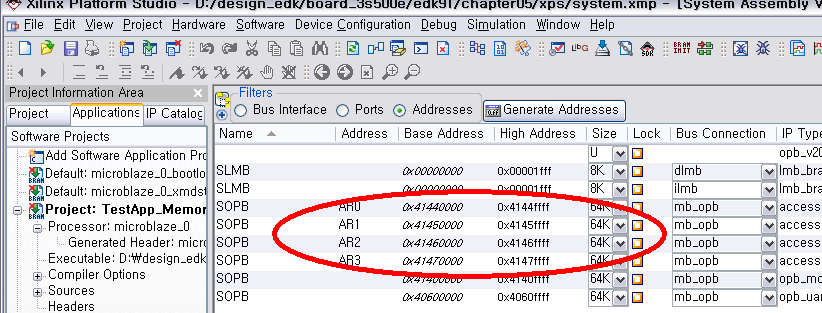
#### External port 연결

Access\_logic에 할당된 여러 local port를 external port에 연결 합니다.



#### 어드레스 할당

User\_logic은 모두 4개의 CS가 있습니다. 각각 64Kbyte 씩 어드레스를 할당 합니다.



### access\_user\_loigc\_4cs\_16x1024 mhs 파일

이 IP는 4개의 chip select를 가지고 있으며 각각의 chip select는 모두 16비트 데이터를 1024 개 디코딩 할 수 있는 어드레스를 가지고 있습니다.

BEGIN access\_user\_logic\_4cs\_16x1024

PARAMETER INSTANCE = access\_user\_logic\_4cs\_16x1024\_0

PARAMETER HW\_VER = 1.00.a

PARAMETER C\_AR0\_BASEADDR = 0x41440000

PARAMETER C\_AR0\_HIGHADDR = 0x4144ffff

PARAMETER C\_AR1\_BASEADDR = 0x41450000

PARAMETER C\_AR1\_HIGHADDR = 0x4145ffff

PARAMETER C\_AR2\_BASEADDR = 0x41460000

PARAMETER C\_AR2\_HIGHADDR = 0x4146ffff

PARAMETER C\_AR3\_BASEADDR = 0x41470000

PARAMETER C\_AR3\_HIGHADDR = 0x4147ffff

END

이 IP는 OPB 버스에 주변장치로 연결되도록 설계했습니다.

BEGIN access\_user\_logic\_4cs\_16x1024

PARAMETER INSTANCE = access\_user\_logic\_4cs\_16x1024\_0

PARAMETER HW\_VER = 1.00.a

BUS\_INTERFACE SOPB = mb\_opb

END

이중 user\_cs는 chip select, user\_add는 디코딩을 위한 어드레스 버스, user\_rd, user\_wr 읽기와 쓰기를 결정합니다.

Data\_from\_user0 ~ 3는 user\_cs에 의해 선택된 로직의 데이터를 프로세서로 전달하는 데이터 버스이며 data\_to\_user는 프로세서가 user logic으로 데이터를 전달하기 위한 데이터 버스로 사용됩니다.

User\_clk은 버스에서 사용하는 클럭을 사용하며 user\_rst도 버스 리셋을 그대로 사용합니다.

다음 리스트는 access\_user\_loigc을 사용하기 위한 mhs 파일입니다.

PORT user\_cs = access\_user\_logic\_4cs\_16x1024\_0\_user\_cs, DIR = O, VEC = [0:3]

PORT user\_add = access\_user\_logic\_4cs\_16x1024\_0\_user\_add, DIR = O, VEC = [0:9]

PORT data\_to\_user = access\_user\_logic\_4cs\_16x1024\_0\_data\_to\_user, DIR = O, VEC = [0:15]

PORT data\_from\_user0 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user0, DIR = I, VEC = [0:15]

PORT data\_from\_user1 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user1, DIR = I, VEC = [0:15]

PORT data\_from\_user2 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user2, DIR = I, VEC = [0:15]

PORT data\_from\_user3 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user3, DIR = I, VEC = [0:15]

PORT user\_rd = access\_user\_logic\_4cs\_16x1024\_0\_user\_rd, DIR = O

PORT user\_wr = access\_user\_logic\_4cs\_16x1024\_0\_user\_wr, DIR = O

PORT user\_clk = access\_user\_logic\_4cs\_16x1024\_0\_user\_clk, DIR = O

PORT user\_rst = access\_user\_logic\_4cs\_16x1024\_0\_user\_rst, DIR = O

BEGIN access\_user\_logic\_4cs\_16x1024

PARAMETER INSTANCE = access\_user\_logic\_4cs\_16x1024\_0

PARAMETER HW\_VER = 1.00.a

PARAMETER C\_AR0\_BASEADDR = 0x41440000

PARAMETER C\_AR0\_HIGHADDR = 0x4144ffff

PARAMETER C\_AR1\_BASEADDR = 0x41450000

PARAMETER C\_AR1\_HIGHADDR = 0x4145ffff

PARAMETER C\_AR2\_BASEADDR = 0x41460000

PARAMETER C\_AR2\_HIGHADDR = 0x4146ffff

PARAMETER C\_AR3\_BASEADDR = 0x41470000

PARAMETER C\_AR3\_HIGHADDR = 0x4147ffff

BUS\_INTERFACE SOPB = mb\_opb

PORT user\_cs = access\_user\_logic\_4cs\_16x1024\_0\_user\_cs

PORT user\_add = access\_user\_logic\_4cs\_16x1024\_0\_user\_add

PORT data\_to\_user = access\_user\_logic\_4cs\_16x1024\_0\_data\_to\_user

PORT data\_from\_user0 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user0

PORT data\_from\_user1 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user1

PORT data\_from\_user2 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user2

PORT data\_from\_user3 = access\_user\_logic\_4cs\_16x1024\_0\_data\_from\_user3

PORT user\_rd = access\_user\_logic\_4cs\_16x1024\_0\_user\_rd

PORT user\_wr = access\_user\_logic\_4cs\_16x1024\_0\_user\_wr

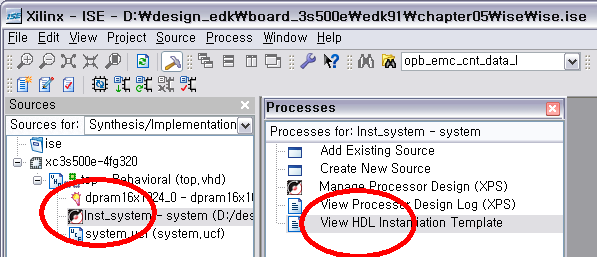
PORT user\_clk = access\_user\_logic\_4cs\_16x1024\_0\_user\_clk

PORT user\_rst = access\_user\_logic\_4cs\_16x1024\_0\_user\_rst

END

## ISE로 XPS 프로젝트 가져오기

다음 그림은 ISE에서 XPS 프로젝트 파일인 xmp 파일을 소스로 등록 한 화면 입니다. 프로세서 블록의 Instance Template를 보기 위해서는 XPS 프로젝트를 선택 한 후 “View HDL …”을 선택 합니다.



### top.vhd

#### system component 선언

top.vhd는 system이라는 프로세서 블록을 포함 합니다. 앞에서 만든 Template에서 component 선언 부분을 참고 하도록 합니다.

COMPONENT system

PORT(

fpga\_0\_RS232\_DCE\_RX\_pin : IN std\_logic;

sys\_clk\_pin : IN std\_logic;

sys\_rst\_pin : IN std\_logic;

data\_from\_user0 : IN std\_logic\_vector(0 to 15);

data\_from\_user1 : IN std\_logic\_vector(0 to 15);

data\_from\_user2 : IN std\_logic\_vector(0 to 15);

data\_from\_user3 : IN std\_logic\_vector(0 to 15);

fpga\_0\_RS232\_DCE\_TX\_pin : OUT std\_logic;

user\_cs : OUT std\_logic\_vector(0 to 3);

user\_add : OUT std\_logic\_vector(0 to 9);

data\_to\_user : OUT std\_logic\_vector(0 to 15);

user\_rd : OUT std\_logic;

user\_wr : OUT std\_logic;

user\_clk : OUT std\_logic;

user\_rst : OUT std\_logic

);

END COMPONENT

#### 메모리 component 선언

Coregen을 통해 16x1024의 dual port 메모리를 만듭니다. 다음 리스트는 dual port memory의 component 입니다.

component dpram16x1024

port (

addra: IN std\_logic\_VECTOR(9 downto 0);

addrb: IN std\_logic\_VECTOR(9 downto 0);

clka: IN std\_logic;

clkb: IN std\_logic;

dina: IN std\_logic\_VECTOR(15 downto 0);

dinb: IN std\_logic\_VECTOR(15 downto 0);

douta: OUT std\_logic\_VECTOR(15 downto 0);

doutb: OUT std\_logic\_VECTOR(15 downto 0);

ena: IN std\_logic;

enb: IN std\_logic;

wea: IN std\_logic;

web: IN std\_logic);

end component;

#### read process

user\_cs, user\_rd, user\_add는 모두 active high입니다. 다음은 레지스터를 읽기 위한 VHDL코드입니다.

제대로 읽히는 것을 확인하기 위해 “03ab”, “2321”, “abcd”와 같은 상수 값을 넣었습니다. 해당 어드레스에서 원하는 상수 값이 출력되는지 확인해야 합니다.

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_rd = '1' then

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

case user\_add is

when x"000" => data\_from\_user0 <= user00;

when x"001" => data\_from\_user0 <= user01;

when x"002" => data\_from\_user0 <= user02;

when x"003" => data\_from\_user0 <= x"03ab";

when others => null;

end case;

elsif user\_cs(1) = '1' then

case user\_add is

when x"000" => data\_from\_user1 <= user10 + 1;

when x"001" => data\_from\_user1 <= user11 + 1;

when x"002" => data\_from\_user1 <= user12 + 1;

when x"003" => data\_from\_user1 <= x"2321";

when others => null;

end case;

elsif user\_cs(2) = '1' then

case user\_add is

when x"000" => data\_from\_user2 <= user20 - 1;

when x"001" => data\_from\_user2 <= user21 - 1;

when x"002" => data\_from\_user2 <= user22 - 1;

when x"003" => data\_from\_user2 <= x"abcd";

when others => null;

end case;

end if;

end if;

end if;

end process;

#### write process

user\_wr는 active high입니다.

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_wr = '1' then

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

case user\_add is

when x"000" => user00 <= data\_to\_user;

when x"001" => user01 <= data\_to\_user;

when x"002" => user02 <= data\_to\_user;

when others => null;

end case;

elsif user\_cs(1) = '1' then

case user\_add is

when x"000" => user10 <= data\_to\_user;

when x"001" => user11 <= data\_to\_user;

when x"002" => user12 <= data\_to\_user;

when others => null;

end case;

elsif user\_cs(2) = '1' then

case user\_add is

when x"000" => user20 <= data\_to\_user;

when x"001" => user21 <= data\_to\_user;

when x"002" => user22 <= data\_to\_user;

when others => null;

end case;

end if;

end if;

end if;

end process;

#### dual port memory access

레지스터 외에도 메모리를 억세스하는 VHDL 코드입니다.

FPGA 내부에 있는 메모리는 모두 2개의 억세스 포트가 있는데 그 중 한 포트는 프로세서에서 억세스하고 나머지 한 포트는 사용자가 연결해서 사용합니다.

여기서는 모두 ground 처리 했습니다.

dpram16x1024\_0: dpram16x1024

port map (

addra => user\_add\_temp,

clka => user\_clk,

dina => data\_to\_user,

douta => data\_from\_user3,

ena => user\_cs(3),

wea => user\_wr,

------------------------------

clkb => gnd0,

addrb => gnd10,

dinb => gnd16,

doutb => open,

enb => gnd0,

web => gnd0);

#### user\_add , user\_add\_temp

프로세서는 모두 1024개 디코딩 하기 때문에 10개의 어드레스 포트가 필요합니다. 이것을 강제로 12비트로 할당하면 VHDL 코드가 간단해 질 수 있습니다.

user\_add <= "00" & user\_add\_temp;

user\_add 신호는 모두 12비트가 할당되어 있기 때문에 다음과 같이 x”000”, x”001”과 같이 16진수로 어드레스를 표현 할 수 있습니다.

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_rd = '1' then

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

case user\_add is

when x"000" => data\_from\_user0 <= user00;

when x"001" => data\_from\_user0 <= user01;

when x"002" => data\_from\_user0 <= user02;

when x"003" => data\_from\_user0 <= x"03ab";

when others => null;

다음은 access\_user\_logic\_4cs\_16x1024 라는 IP와 실제 레지스터와 메모리를 억세스하기 위한 VHDL 코드입니다.

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 18:34:22 02/28/2008

-- Design Name:

-- Module Name: top - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity top is

port (

fpga\_0\_RS232\_DCE\_RX\_pin : IN std\_logic;

fpga\_0\_RS232\_DCE\_TX\_pin : OUT std\_logic;

sys\_clk\_pin : IN std\_logic;

sys\_rst\_pin : IN std\_logic);

end top;

architecture Behavioral of top is

COMPONENT system

PORT(

fpga\_0\_RS232\_DCE\_RX\_pin : IN std\_logic;

sys\_clk\_pin : IN std\_logic;

sys\_rst\_pin : IN std\_logic;

data\_from\_user0 : IN std\_logic\_vector(0 to 15);

data\_from\_user1 : IN std\_logic\_vector(0 to 15);

data\_from\_user2 : IN std\_logic\_vector(0 to 15);

data\_from\_user3 : IN std\_logic\_vector(0 to 15);

fpga\_0\_RS232\_DCE\_TX\_pin : OUT std\_logic;

user\_cs : OUT std\_logic\_vector(0 to 3);

user\_add : OUT std\_logic\_vector(0 to 9);

data\_to\_user : OUT std\_logic\_vector(0 to 15);

user\_rd : OUT std\_logic;

user\_wr : OUT std\_logic;

user\_clk : OUT std\_logic;

user\_rst : OUT std\_logic

);

END COMPONENT;

component dpram16x1024

port (

addra: IN std\_logic\_VECTOR(9 downto 0);

addrb: IN std\_logic\_VECTOR(9 downto 0);

clka: IN std\_logic;

clkb: IN std\_logic;

dina: IN std\_logic\_VECTOR(15 downto 0);

dinb: IN std\_logic\_VECTOR(15 downto 0);

douta: OUT std\_logic\_VECTOR(15 downto 0);

doutb: OUT std\_logic\_VECTOR(15 downto 0);

ena: IN std\_logic;

enb: IN std\_logic;

wea: IN std\_logic;

web: IN std\_logic);

end component;

signal user00, user01, user02 : std\_logic\_vector(0 to 15) := x"0000";

signal user10, user11, user12 : std\_logic\_vector(0 to 15) := x"0000";

signal user20, user21, user22 : std\_logic\_vector(0 to 15) := x"0000";

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

signal user\_add\_temp : std\_logic\_vector(0 to 9);

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

signal user\_clk, user\_wr, user\_rd : std\_logic;

signal data\_to\_user, data\_from\_user0, data\_from\_user1, data\_from\_user2, data\_from\_user3 : std\_logic\_vector(0 to 15);

signal gnd0 : std\_logic;

signal gnd16 : std\_logic\_vector(0 to 15);

signal gnd10 : std\_logic\_vector(0 to 9);

begin

gnd0 <= '0'; gnd16 <= x"0000"; gnd10 <= "0000000000";

user\_add <= "00" & user\_add\_temp;

dpram16x1024\_0: dpram16x1024

port map (

addra => user\_add\_temp,

clka => user\_clk,

dina => data\_to\_user,

douta => data\_from\_user3,

ena => user\_cs(3),

wea => user\_wr,

------------------------------

clkb => gnd0,

addrb => gnd10,

dinb => gnd16,

doutb => open,

enb => gnd0,

web => gnd0);

Inst\_system: system PORT MAP(

fpga\_0\_RS232\_DCE\_RX\_pin => fpga\_0\_RS232\_DCE\_RX\_pin ,

fpga\_0\_RS232\_DCE\_TX\_pin => fpga\_0\_RS232\_DCE\_TX\_pin ,

sys\_clk\_pin => sys\_clk\_pin ,

sys\_rst\_pin => sys\_rst\_pin ,

user\_cs => user\_cs ,

user\_add => user\_add\_temp ,

data\_to\_user => data\_to\_user ,

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

data\_from\_user1 => data\_from\_user1 ,

data\_from\_user2 => data\_from\_user2 ,

data\_from\_user3 => data\_from\_user3 ,

user\_rd => user\_rd ,

user\_wr => user\_wr ,

user\_clk => user\_clk ,

user\_rst => open

);

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_rd = '1' then

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

case user\_add is

when x"000" => data\_from\_user0 <= user00;

when x"001" => data\_from\_user0 <= user01;

when x"002" => data\_from\_user0 <= user02;

when x"003" => data\_from\_user0 <= x"03ab";

when others => null;

end case;

elsif user\_cs(1) = '1' then

case user\_add is

when x"000" => data\_from\_user1 <= user10 + 1;

when x"001" => data\_from\_user1 <= user11 + 1;

when x"002" => data\_from\_user1 <= user12 + 1;

when x"003" => data\_from\_user1 <= x"2321";

when others => null;

end case;

elsif user\_cs(2) = '1' then

case user\_add is

when x"000" => data\_from\_user2 <= user20 - 1;

when x"001" => data\_from\_user2 <= user21 - 1;

when x"002" => data\_from\_user2 <= user22 - 1;

when x"003" => data\_from\_user2 <= x"abcd";

when others => null;

end case;

end if;

end if;

end if;

end process;

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_wr = '1' then

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

case user\_add is

when x"000" => user00 <= data\_to\_user;

when x"001" => user01 <= data\_to\_user;

when x"002" => user02 <= data\_to\_user;

when others => null;

end case;

elsif user\_cs(1) = '1' then

case user\_add is

when x"000" => user10 <= data\_to\_user;

when x"001" => user11 <= data\_to\_user;

when x"002" => user12 <= data\_to\_user;

when others => null;

end case;

elsif user\_cs(2) = '1' then

case user\_add is

when x"000" => user20 <= data\_to\_user;

when x"001" => user21 <= data\_to\_user;

when x"002" => user22 <= data\_to\_user;

when others => null;

end case;

end if;

end if;

end if;

end process;

end Behavioral;

### 소프트웨어 코드

#### 헤더파일

access\_user\_logic에 관한 매크로는 xparameters.h에 저장되어 있습니다.

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR 0x41440000

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_HIGHADDR 0x4144FFFF

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR1\_BASEADDR 0x41450000

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR1\_HIGHADDR 0x4145FFFF

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR2\_BASEADDR 0x41460000

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR2\_HIGHADDR 0x4146FFFF

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR3\_BASEADDR 0x41470000

#define XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR3\_HIGHADDR 0x4147FFFF

#### main\_user\_logic.c

16비트 어드레스 포인터를 선언합니다.

Xuint16 \*add;

만일 습관적으로 Xuint32 \*add로 선언하면 전혀 엉뚱한 결과를 가져오게 됩니다.

Add 변수를 초기화 합니다.

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR;

차례대로 ‘0’, ‘1’, ‘2’를 write 합니다.

for(i = 0; i < 3; i++) {

\*add++ = i;

}

다시 어드레스를 초기화 합니다.

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR;

데이터를 읽어 터미널에 출력합니다.

for(i = 0; i < 4; i++) {

xil\_printf("\n\rread data = %x", \*add++);

}

다음은 완성된 main\_user\_logic.c 입니다.

#include "xparameters.h"

#include "xbasic\_types.h"

void main()

{

Xuint16 \*add;

Xuint32 i;

xil\_printf("\n\r%s",\_\_TIME\_\_);

xil\_printf("\n\rXPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR");

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR;

for(i = 0; i < 3; i++) {

\*add++ = i;

}

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR0\_BASEADDR;

for(i = 0; i < 4; i++) {

xil\_printf("\n\rread data = %x", \*add++);

}

xil\_printf("\n\rXPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR1\_BASEADDR");

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR1\_BASEADDR;

for(i = 0; i < 3; i++) {

\*add++ = i;

}

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR1\_BASEADDR;

for(i = 0; i < 4; i++) {

xil\_printf("\n\rread data = %x", \*add++);

}

xil\_printf("\n\rXPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR2\_BASEADDR");

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR2\_BASEADDR;

for(i = 0; i < 3; i++) {

\*add++ = i;

}

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR2\_BASEADDR;

for(i = 0; i < 4; i++) {

xil\_printf("\n\rread data = %x", \*add++);

}

xil\_printf("\n\rXPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR3\_BASEADDR memory");

xil\_printf("\n\rpress anykey to start init & read back");

inbyte();

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR3\_BASEADDR;

for(i = 0; i < 1024; i++) {

\*add++ = i;

}

add = (Xuint16 \*)XPAR\_ACCESS\_USER\_LOGIC\_4CS\_16X1024\_0\_AR3\_BASEADDR;

for(i = 0; i < 1024; i++) {

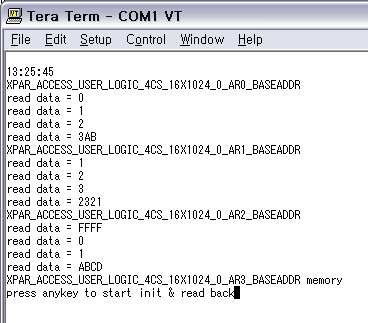
xil\_printf("\n\radd =%d, data = %x", i, \*add++);

}

}

### 실행결과

다음은 터미널에 출력된 화면입니다.



앞에서 살펴보았던 read process를 다시 살펴보겠습니다.

process(user\_clk)

begin

if user\_clk'event and user\_clk = '1' then

if user\_rd = '1' then

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

case user\_add is

when x"000" => data\_from\_user0 <= user00;

when x"001" => data\_from\_user0 <= user01;

when x"002" => data\_from\_user0 <= user02;

when x"003" => data\_from\_user0 <= x"03ab";

when others => null;

end case;

elsif user\_cs(1) = '1' then

case user\_add is

when x"000" => data\_from\_user1 <= user10 + 1;

when x"001" => data\_from\_user1 <= user11 + 1;

when x"002" => data\_from\_user1 <= user12 + 1;

when x"003" => data\_from\_user1 <= x"2321";

when others => null;

end case;

elsif user\_cs(2) = '1' then

case user\_add is

when x"000" => data\_from\_user2 <= user20 - 1;

when x"001" => data\_from\_user2 <= user21 - 1;

when x"002" => data\_from\_user2 <= user22 - 1;

when x"003" => data\_from\_user2 <= x"abcd";

when others => null;

end case;

end if;

end if;

end if;

end process;

화면상에 표시된 3AB, 2321, ABCD와 VHDL 코드를 비교해보면 read 동작이 제대로 되고 있음을 알 수 있습니다.

첫 번째 read는 write 된 값을 그대로 읽었고 두 번째 read는 write된 값보다 하나씩 더해서 읽었습니다. 마지막은 write 된 것보다 하나 뺀 값을 읽어서 처음에 FFFF가 출력되었습니다.

## User\_logic.vhd 분석

Access\_user\_logic\_4cs\_16x1024 정도만 있으면 웬만한 user logic 인터페이스는 할 수 있습니다. 하지만 프로세서가 이 ip를 access할 때 읽기, 쓰기 타이밍이 얼마나 되는지 알아 볼 필요가 있습니다.

이런 타이밍을 수정하기 위해서는 access\_user\_logic\_4cs\_16x1024.vhd와 같이 있는 user\_logic.vhd를 이해하고 수정할 필요가 있습니다.

### User\_logic.vhd / entity 분석

#### local port

access\_user\_logic\_4cs\_16x1024는 4개의 CS가 있습니다.

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

1024개의 디코딩 영역을 만들기 위한 어드레스 버스가 있습니다.

user\_add : out std\_logic\_vector(0 to 9);

프로세서가 데이터를 기록하기 위한 데이터 버스가 있습니다.

data\_to\_user : out std\_logic\_vector(0 to 15);

유저 로직으로부터 데이터를 읽기 위한 데이터 버스가 있습니다.

data\_from\_user0 : in std\_logic\_vector(0 to 15);

data\_from\_user1 : in std\_logic\_vector(0 to 15);

data\_from\_user2 : in std\_logic\_vector(0 to 15);

data\_from\_user3 : in std\_logic\_vector(0 to 15);

그 외 RD, WR, CLK, RST 신호가 있습니다.

user\_rd : out std\_logic;

user\_wr : out std\_logic;

user\_clk : out std\_logic;

user\_rst : out std\_logic;

다음 리스트는 user\_logic의 entity를 보여주는 VHDL 코드 입니다.

entity user\_logic is

port

(

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

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

user\_add : out std\_logic\_vector(0 to 9);

data\_to\_user : out std\_logic\_vector(0 to 15);

data\_from\_user0 : in std\_logic\_vector(0 to 15);

data\_from\_user1 : in std\_logic\_vector(0 to 15);

data\_from\_user2 : in std\_logic\_vector(0 to 15);

data\_from\_user3 : in std\_logic\_vector(0 to 15);

user\_rd : out std\_logic;

user\_wr : out std\_logic;

user\_clk : out std\_logic;

user\_rst : out std\_logic;

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

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

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

Bus2IP\_Clk : in std\_logic;

Bus2IP\_Reset : in std\_logic;

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

Bus2IP\_RNW : in std\_logic;

IP2Bus\_Ack : out std\_logic;

IP2Bus\_Retry : out std\_logic;

IP2Bus\_Error : out std\_logic;

IP2Bus\_ToutSup : out std\_logic;

Bus2IP\_ArData : in std\_logic\_vector(0 to C\_MAX\_AR\_DWIDTH-1);

Bus2IP\_ArBE : in std\_logic\_vector(0 to C\_MAX\_AR\_DWIDTH/8-1);

Bus2IP\_ArCS : in std\_logic\_vector(0 to C\_NUM\_ADDR\_RNG-1);

IP2Bus\_ArData : out std\_logic\_vector(0 to C\_MAX\_AR\_DWIDTH-1)

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

);

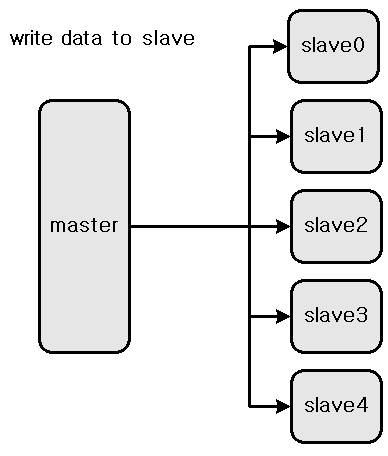
end entity user\_logic;

### User\_logic.vhd / read data bus, write data bus 분석

앞에 entity에 있는 포트에 대해 설명할 때 읽기 위한 데이터 버스와 쓰기 위한 데이터 버스가 따로 준비된 것을 확인할 수 있습니다.

일반적으로 프로세서는 데이터 버스가 하나로 읽기와 쓰기를 합니다. 프로세서가 쓰기 동작을 할 경우는 FPGA내에 데이터 버스를 처리하는 것은 문제가 없습니다.

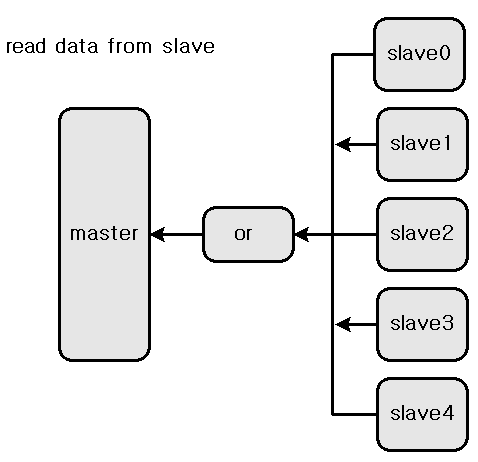
다음 그림은 하나의 프로세서가 여러 주변장치에 쓰기 데이터 버스를 연결한 모습을 보여 줍니다.



하지만 읽기 동작을 할 경우에는 프로세서에 의해 선택된 주변장치에서 나온 데이터가 선택되지 않은 주변장치에서 나오는 데이터 버스의 영향을 받으면 안됩니다. 그래서 일반적으로 사용하는 것이 tri-state 버스인데 프로세서에 의해 선택되지 않은 주변장치는 자신의 데이터 버스를 tri-state 상태로 만들어 버립니다.

문제는 여기서 발생 합니다.

요즘 사용되는 모든 FPGA 내부에는 tri-state 버스가 없습니다. 그래서 나온 대안이 바로 OR 버스 입니다.



OR 버스에서 don’t care 상태는 ‘0’ 입니다. 선택되지 않는 주변장치의 모든 읽기 전용 데이터 버스의 상태는 ‘0’으로 하면 프로세서가 데이터를 안전하게 읽을 수 있습니다.

다음 리스트는 OR 버스를 사용했을 때 전용 데이터 버스를 처리하기 위한 VHDL 코드 입니다. 선택되지 않았을 때 IP2Bus\_ArData 가 ‘0’ 이 입력 된 것을 확인할 수 있습니다.

-- implement Block RAM read mux

IP2BUS\_ARDATA\_PROC : process(user\_select, data\_from\_user0, data\_from\_user1, data\_from\_user2, data\_from\_user3)

begin

case user\_select is

when "1000" => IP2Bus\_ArData <= data\_from\_user0;

when "0100" => IP2Bus\_ArData <= data\_from\_user1;

when "0010" => IP2Bus\_ArData <= data\_from\_user2;

when "0001" => IP2Bus\_ArData <= data\_from\_user3;

when others => IP2Bus\_ArData <= (others => '0');

end case;

end process IP2BUS\_ARDATA\_PROC;

### User\_logic.vhd / acknowledge 신호

프로세서는 항상 데이터를 읽고, 쓰기 위한 마스터로 동작 합니다. 그렇기 때문에 프로세서는 읽고, 쓰기 위한 버스를 발생을 시키기는 하지만 한번 발생시킨 버스 사이클을 언제 끝내야 하는지는 어드레스에 의해 선택된 주변장치에 의해 결정 됩니다.

다음 신호들은 버스 사이클을 끝내기 위한 신호들을 설명하고 버스 사이클을 늘리기 위한 방법에 대해 설명합니다.

참고로 모든 버스 신호들은 active high 입니다.

#### user\_rd\_s, user\_wr\_s

프로세서가 access\_user\_logic\_4cs\_16x1024에 할당된 4개의 어드레스 범위에 읽기 쓰기 할 경우 user\_rd\_s, user\_wr\_s 신호는 ‘1’이 됩니다.

user\_rd\_s <= (Bus2IP\_ArCS(0) or Bus2IP\_ArCS(1) or Bus2IP\_ArCS(2) or Bus2IP\_ArCS(3)) and Bus2IP\_RNW;

user\_wr\_s <= (Bus2IP\_ArCS(0) or Bus2IP\_ArCS(1) or Bus2IP\_ArCS(2) or Bus2IP\_ArCS(3)) and not(Bus2IP\_RNW);

#### user\_rd\_ack, user\_wr\_ack

user\_rd\_s와 user\_wr\_s는 버스 클럭에 의해 한 클럭 딜레이 된 후 user\_rd\_ack, user\_wr\_ack 로 연결 됩니다.

#### IP2Bus\_Ack

User\_rd\_ack와 user\_wr\_ack를 or 연산해 BUS로 되돌려 주면 버스 사이클이 끝나게 됩니다.

다음 리스트는 프로세서의 버스의 읽기, 쓰기 동작을 끝내기 위해 acknowledge 신호를 만들기 위한 VHDL 코드입니다.

user\_rd\_s <= (Bus2IP\_ArCS(0) or Bus2IP\_ArCS(1) or Bus2IP\_ArCS(2) or Bus2IP\_ArCS(3)) and Bus2IP\_RNW;

user\_wr\_s <= (Bus2IP\_ArCS(0) or Bus2IP\_ArCS(1) or Bus2IP\_ArCS(2) or Bus2IP\_ArCS(3)) and not(Bus2IP\_RNW);

user\_rd\_ack <= user\_rd\_dly1;

user\_wr\_ack <= user\_wr\_dly1;

user\_rd\_wr\_ack : process( Bus2IP\_Clk ) is

begin

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

if ( Bus2IP\_Reset = '1' ) then

user\_rd\_dly1 <= '0';

user\_wr\_dly1 <= '0';

else

user\_rd\_dly1 <= user\_rd\_s;

user\_wr\_dly1 <= user\_wr\_s;

end if;

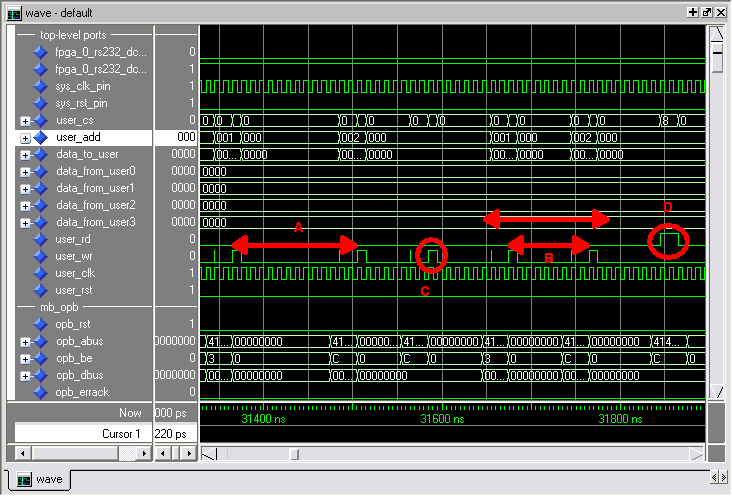
end if;

end process user\_rd\_wr\_ack;

IP2Bus\_Ack <= user\_rd\_ack or user\_wr\_ack;

다음은 버스 사이클을 2클럭 동안 유지시킨 것을 보여주는 modelsim wave 파형 입니다.

주의할 것은 읽기 동작은 2 클럭 동안 유지하지만 쓰기 동작은 1 클럭 동안만 유지 합니다. 그 이유에 대해서는 나중에 설명 됩니다.



### 그 밖의 신호들

다음 3개의 신호는 모두 ground 처리 합니다.

IP2Bus\_Error <= '0';

IP2Bus\_Retry <= '0';

IP2Bus\_ToutSup <= '0';

### 버스 사이클 늘리기

버스 사이클을 늘리기 위해서는 ack 신호를 늦게 돌려 주면 쉽게 해결 됩니다.

다음 리스트는 읽기, 쓰기 사이클이 시작되었을 때 만들어진 user\_rd\_s, user\_wr\_s 신호를 6개의 DFF를 거쳐서 돌려주고 있습니다.

user\_rd\_wr\_ack : process( Bus2IP\_Clk ) is

begin

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

if ( Bus2IP\_Reset = '1' ) then

user\_rd\_dly1 <= '0';

user\_wr\_dly1 <= '0';

rd\_d0 <= '0';

rd\_d1 <= '0';

rd\_d2 <= '0';

rd\_d3 <= '0';

rd\_d4 <= '0';

wr\_d0 <= '0';

wr\_d1 <= '0';

wr\_d2 <= '0';

wr\_d3 <= '0';

wr\_d4 <= '0';

else

rd\_d0 <= user\_rd\_s;

rd\_d1 <= rd\_d0;

rd\_d2 <= rd\_d1;

rd\_d3 <= rd\_d2;

rd\_d4 <= rd\_d3;

user\_rd\_dly1 <= rd\_d4;

---------------------------------------

wr\_d0 <= user\_wr\_s;

wr\_d1 <= wr\_d0;

wr\_d2 <= wr\_d1;

wr\_d3 <= wr\_d2;

wr\_d4 <= wr\_d3;

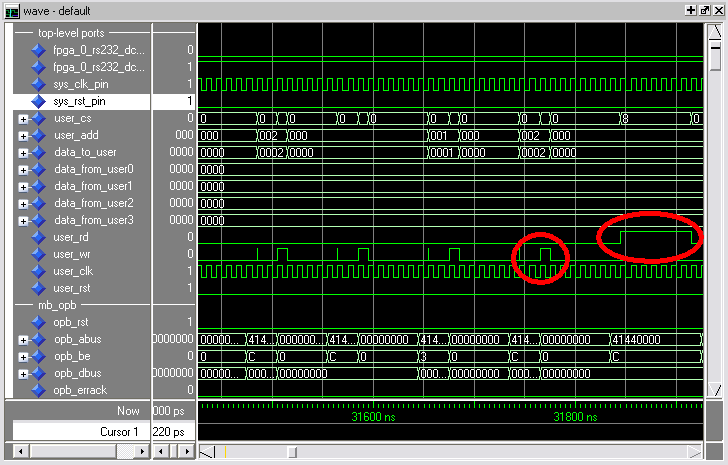
user\_wr\_dly1 <= wr\_d4;

end if;

end if;

end process user\_rd\_wr\_ack

다음 그림은 읽기 사이클이 6클럭 동안 지속되는 것을 보여주는 modelsim wave 파형 입니다.



### 쓰기 동작과 읽기 동작 사이클을 똑같이 하기

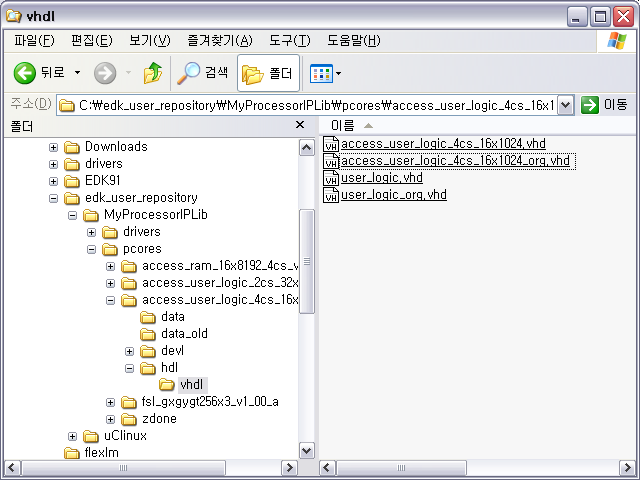
#### Posted Write Transactions

쓰기 동작 사이클이 acknowledge 신호에 의해 영향을 받지 않고 1 사이클 만에 끝나는 이유는 OPB 버스가 Posted Write Transactions 이라는 쓰기 동작을 지원하기 때문 입니다.

Posted Write transaction 모드를 사용하면 프로세서는 쓰고자 하는 데이터를 써 버리고 맙니다. 주변장치가 데이터를 제대로 읽었다고 가정하는 모드이기 때문에 프로세서 입장에서는 효율이 높은 모드라 할 수 있습니다.

이 모드를 사용하지 않으려면 access\_user\_logic\_4cs\_16x1024.vhd에서 수정해야 하는 부분이 있습니다.

다음 그림은 해당 코드가 있는 폴더를 보여 줍니다.



#### access\_user\_logic\_4cs\_16x1024.vhd / ENABLE\_POSTED\_WRITE

ENABLE\_POSTED\_WRITE 가 ‘0’ Posted Write 모드를 사용하고 ‘1’ 이면 Posted Write 모드를 사용하지 않습니다.

다음 리스트는 Post Write 모드를 사용하지 않도록 수정한 VHDL 코드 입니다.

signal ENABLE\_POSTED\_WRITE : std\_logic\_vector(0 to ARD\_ID\_ARRAY'length-1) := (others => '1'); -- enable posted write behavior

다음 그림은 Post Write 모드를 사용하지 않을 때 쓰기 사이클도 읽기 사이클과 마찬가지고 6클럭 동안 유지되는 것을 보여주는 modelsim wave 파형 입니다.

