

Verilog FPGA Program 3

DDR Controller

(Target Board : Arty A7-35T)

아이힐

Version 2.7

목차

| | | |
|-------|---------------------------|----|
| 1 | 개요 | 5 |
| 2 | HW 구성 | 7 |
| 2.1 | USB-JTAG 를 이용하는 방법 | 8 |
| 2.2 | JTAG-HS2(or HS3)을 이용하는 방법 | 9 |
| 3 | DDR Controller IP 생성 | 11 |
| 3.1 | 프로젝트 생성 | 11 |
| 3.2 | Memory IP 생성 | 16 |
| 3.3 | Memory IP 구조 | 29 |
| 3.4 | User Interface Block | 31 |
| 3.5 | User Interface 신호 | 32 |
| 3.5.1 | Address/Command 관련 신호 | 32 |
| 3.5.2 | Write 관련 신호 | 32 |
| 3.5.3 | Read 관련 신호 | 33 |
| 3.5.4 | 기타 신호 | 33 |
| 3.6 | User Interface Timing | 34 |
| 3.6.1 | Address/Command Timing | 34 |
| 3.6.2 | Write Timing | 34 |
| 3.6.3 | Read Timing | 35 |
| 4 | Simulation | 36 |
| 4.1 | Simulation 환경 설정 | 36 |
| 4.2 | Simulation | 40 |
| 4.3 | Simulation 결과 확인 | 43 |
| 4.3.1 | init_calib_complete | 43 |
| 4.3.2 | reset, clock | 43 |
| 4.3.3 | Write Timing 분석 | 44 |
| 4.3.4 | Read Timing 분석 | 45 |
| 5 | User Interface Logic 구현 | 47 |
| 5.1 | 개요 | 48 |
| 5.1.1 | Write Timing 정의 | 49 |
| 5.1.2 | Read Timing 정의 | 50 |
| 5.1.3 | 코드 구조 | 51 |
| 5.2 | Write Module 구현 | 53 |
| 5.2.1 | mig7_write8 모듈 구현 | 53 |
| 5.2.2 | mig7_write8 모듈 simulation | 56 |
| 5.2.3 | mig7_write 모듈 구현 | 60 |

| | |
|---|-----|
| 5.2.4 mig7_write 모듈 simulation..... | 63 |
| 5.3 Read Module 구현 | 64 |
| 5.3.1 mig7_read8 모듈 구현 | 64 |
| 5.3.2 mig7_read8 모듈 simulation..... | 67 |
| 5.3.3 mig7_read 모듈 구현 | 68 |
| 5.3.4 mig7_read 모듈 simulation..... | 70 |
| 5.4 User Interface 모듈 구현 (ddr_test 모듈)..... | 72 |
| 5.4.1 Test Scenario | 72 |
| 5.4.2 Write Sequence | 72 |
| 5.4.3 Read Sequence | 73 |
| 5.4.4 ddr_test 모듈 구현..... | 73 |
| 5.4.5 ddr_test 모듈 simulation..... | 75 |
| 5.5 mig_top Module 구현..... | 77 |
| 5.6 Top Module 구현..... | 77 |
| 5.7 Top Module Simulation..... | 78 |
| 5.8 Bitstream 생성 | 83 |
| 5.9 Bitstream 다운로드 & 확인 | 87 |
| 6 DDR3 Memory Access 속도..... | 91 |
| 6.1 mig7_write8 수정 | 93 |
| 6.2 mig7_read8 수정 | 94 |
| 6.3 mig7_write_top 수정..... | 95 |
| 6.4 mig7_read_top 수정..... | 96 |
| 6.5 ddr_test 수정..... | 97 |
| 6.6 define.v 수정 | 98 |
| 6.7 MIG7 Memory Interface IP 수정 | 99 |
| 6.8 clk_gen 수정..... | 100 |
| 6.9 Arty35Top.v 수정 | 100 |
| 6.10 Bitstream 생성 및 결과 확인 | 101 |
| 7 Frame Buffer 구현 | 104 |
| 7.1 데이터 준비 | 104 |
| 7.2 Image Decoder | 106 |
| 7.2.1 코드 구현 | 106 |
| 7.2.2 Simulation | 108 |
| 7.3 Frame Buffer 개요..... | 110 |
| 7.4 frame_write..... | 112 |
| 7.4.1 frame_write 모듈 구현 | 114 |
| 7.4.2 frame_write 모듈 simulation | 120 |

| | |
|--|-----|
| 7.5 frame_read..... | 122 |
| 7.5.1 frame_read 모듈 구현 | 122 |
| 7.5.2 frame_read 모듈 simulation..... | 128 |
| 7.6 영상 데이터를 이용한 Frame Buffer 확인 | 129 |
| 7.6.1 image_dec.v..... | 129 |
| 7.6.2 mig_top_frame.v..... | 129 |
| 7.6.3 Arty35Top_frame.v | 130 |
| 7.6.4 tb_arty35Top_frame.v..... | 131 |
| 7.6.5 simulation..... | 132 |
| 7.6.6 이미지 확인하기 | 135 |
| 8 32Bits Interface 구현 | 136 |
| 8.1 프로젝트 생성..... | 136 |
| 8.2 Memory IP 생성 | 137 |
| 8.3 Simulation 을 통한 생성된 IP 동작 이해..... | 146 |
| 8.3.1 clock 신호..... | 151 |
| 8.3.2 init_calib_complete..... | 152 |
| 8.3.3 Memory read / write | 153 |
| 8.4 User Interface 구현 | 155 |
| 8.4.1 mig32_read8.v mig32_write8.v 수정..... | 156 |
| 8.4.2 mig32_read_top, mig32_write_top 수정 | 157 |
| 8.4.3 ddr32_test 수정..... | 159 |
| 8.5 mig32_top 모듈 simulation..... | 161 |
| 8.6 결론 | 167 |
| 9 Spartan6 DDR Controller 구현 | 168 |
| 9.1 프로젝트 생성..... | 168 |
| 9.2 IP 생성 | 171 |
| 9.3 mcb review | 185 |
| 9.4 simulation 을 통한 동작 확인..... | 192 |
| 9.4.1 calib_done | 194 |
| 9.4.2 c3_clk0, c3_RST0 | 194 |
| 9.4.3 write timing | 195 |
| 9.4.4 read timing | 197 |
| 9.5 코드 구현 | 199 |
| 9.5.1 mcb_write_128x16 | 199 |
| 9.5.2 simulation mcb_write_128x16 | 202 |
| 9.5.3 mcb_write | 207 |
| 9.5.4 simulation mcb_write | 210 |

| | |
|--|-----|
| 9.5.5 mcb_read_128x16 | 214 |
| 9.5.6 simulation mcb_read_128x16 | 217 |
| 9.5.7 mcb_read..... | 221 |
| 9.5.8 simulation mcb_read..... | 224 |
| 9.6 전 영역 read/write 구현 | 227 |
| 9.6.1 mcb_test..... | 227 |
| 9.6.2 simulation mcb_test..... | 232 |
| 9.6.3 tb_mcb_top..... | 238 |
| 9.7 결론 | 250 |
| 10 DDR4 Controller..... | 251 |
| 10.1 프로젝트 생성 | 251 |
| 10.2 IP 생성 | 252 |
| 10.3 simulation..... | 255 |
| 10.4 코드 구현 | 261 |
| 10.5 Address 관련..... | 267 |
| 10.6 xdc 생성 | 269 |
| 10.7 결과 확인 | 270 |
| 10.8 결론 | 272 |
| 11 참고 자료 | 273 |
| 12 Revision Histroy | 274 |

1. 개요

FPGA에는 많은 기능들이 IP 형태로 제공되고 있습니다. 그 중에서 영상데이터를 처리하는데 주로 사용되는 DDR Controller에 대해서 설명합니다. DDR Controller를 구현하는 방법은 크게 2가지가 있습니다. 데이터시트를 보고 코드로 그대로 구현하는 방법과 Xilinx 사에서 제공하는 IP를 사용하여 구현하는 방법입니다. 데이터시트를 보고 구현하는 방법은 경력이 오래된 분들이 주로 사용하는 방법입니다. (예전에 저자도 SDRAM Controller를 직접 구현해서 사용한 적이 있습니다. 다행히 문제없이 잘 동작하였고, ASIC을 만들어 제품으로 사용되었습니다). 그러나 시스템이 복잡해지면 일일이 코드로 구현하는 일이 만만치가 않습니다. 그래서 자신이 필요한 특정한 부분만 구현해서 사용하는 경우가 많은데, 그럴 경우 프로젝트가 바뀔 때마다 매번 거기에 맞게 구현해 주어야 하는 어려움이 있습니다. 반면에 Xilinx 사에서 제공하는 IP는 매우 잘 만들어진 범용으로 사용가능한 IP입니다. 한번 사용법을 익혀 두면 언제든 크게 변경하지 않아도 여러 프로젝트에 사용할 수 있습니다. 그러나 처음에 사용법을 익히는 과정이 만만치가 않습니다. IP를 어떻게 만들어야 하는지, IP 생성시 선택해야 하는 옵션들은 무엇을 선택해야 하는지, 또한 어떻게 사용해야 하는지를 이해하는 것이 쉽지 않습니다. Xilinx 사에서 많은 문서와 샘플코드를 제공하고 있지만, 너무 많은 문서와 코드들이 있어서, 어떤 문서를 봐야 하는지(어떤 문서는 수백 페이지가 넘는 문서도 많이 있습니다), 어떤 코드를 봐야 하는지 매우 어려운 문제입니다. 저자가 DDR Controller를 구현하면서 이러한 많은 어려움을 경험하였습니다. 뭔가 필요한 부분만 정리된 하나의 문서가 있으면 좋지 않을까 하는 생각을 많이 하였습니다. 처음부터 복잡한 용어와 복잡한 과정을 다 이해하며 개발하기에는 너무나 어려움이 많습니다. 처음에는 필요한 부분만 이해하고 개발을 진행해서 DDR을 read / write 해 보고, 점점 더 이해도를 높여가는 것이 효율적입니다. 먼저 전체적인 구조를 이해하고, 나중에 세부적인 내용들을 이해하는 것이 효과적입니다. 본서는 DDR Controller를 처음 접하는 개발자들도 쉽게 접근할 수 있도록 구성되어 있습니다.

본서에서는 DDR Controller IP를 생성하는 과정과 생성된 IP가 어떻게 동작하는지 Simulation을 통하여 동작과정을 이해하고, 생성된 IP를 이용하여 범용적으로 사용할 수 있는 DDR Controller를 설계합니다. 마지막으로 영상데이터 처리를 위한 Frame 버퍼를 구현하는 과정을 설명합니다.

본서는 v2.0에서 내용을 전체적으로 다시 구성하였습니다. v1.0에서 부족하게 생각되었던 부분들을 보완하고 마지막장에 프레임 버퍼를 구현하는 내용을 추가하였습니다. 본서는 오랫동안 ISP(Image Signal Processing)개발을 진행했던 저자의 경험을 바탕으로 구성되었습니다. 본서의 내용들은 실무에 바로 적용할 수 있는 코드들입니다. 내용이 다소 어려운 부분들이 포함될 수 있지만, 이해가 되지 않는 부분들은 2-3번 정도 정독을 하시고, 코드들을 Simulation 하면서 동작내용을 이해한다면 DDR Controller를 이해하고 구현하는데 부족함이 없을 것으로 기대합니다.

본서는 Digilent 사에서 판매하는 Arty A7 개발보드에서 실습을 진행합니다. FPGA를 다루는데 있어서 HW에 적용하고 그 결과를 확인하는 것은 매우 중요합니다. Simulation으로 구현하는 것은 한계가 있습니다. 반드시 HW에서 검증을 해야 합니다. (본서의 마지막 장에서 다루는 프레임 버퍼에 대한 내용은 적당한 HW를 찾지 못해서 Simulation으로만 검증을 진행하는 아쉬움이 있습니다. 추후 적당한 보드를 찾아서 HW 검증까지 진행할 수 있길 기대합니다) 본서를 구매하시는 분들에게는 본서에서 설명된 모든 소스를 무료로 제공하여 드립니다. 본서에서 소개되는 모든 소스코드들은 실무에 바로 적용할 수 있는 코드들로 구성되어 있습니다. 본서를 통하여서 DDR Controller를 구현하고 사용하는데 많은 도움이 되시길 바랍니다.

본문의 구성은 다음과 같습니다.

2장에서는 실습에 사용되는 Arty A7 보드에 대해서 설명합니다.

3장에서는 DDR Controller IP를 생성하는 방법과 User Interface Logic에 대해서 설명합니다.

4장에서는 생성된 DDR Controller IP의 Simulation을 통하여 동작 특성을 이해합니다. 이는 범용으로 사용가능한 DDR Controller를 구현하는데 매우 중요합니다.

5장에서는 범용으로 사용가능한 DDR Controller를 구현하고, 이를 이용하여 DDR Memory의 전 영역을 Read/Write 하는 것을 구현합니다. 또한 Bitstream을 생성해서 보드에서 결과를 확인합니다.

6장은 Memory Access 속도에 관한 내용을 살펴보고, 최대 속도를 구현하기 위한 내용을 다루게 됩니다.

7장에서는 5장에서 구현한 범용 DDR Controller를 이용하여 영상 데이터 처리를 위한 Frame Buffer를 구현합니다.

8장은 DDR3 2개를 사용하여 32bits Interface를 구현하는 내용입니다.

9장은 Spartan6에서 DDR Memory Controller를 구현합니다. 툴은 ISE 14.7을 사용합니다.

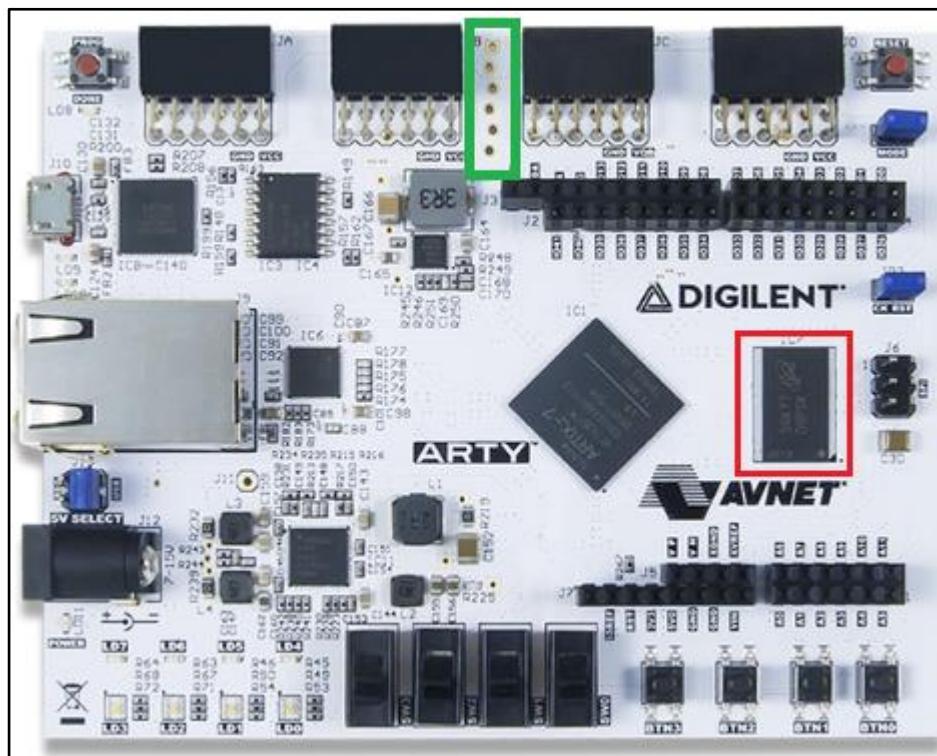
10장은 참고 자료입니다.



2. HW 구성

본서에서 실습에 사용하는 보드는 Digilent 사의 Arty A7-35T 개발 보드입니다. Arty A7-35T에는 Micron 사의 DDR3 (2Gb, MT41K128M16JT-125K)가 포함되어 있습니다. 본서에서는 이 DDR3를 Access(Read/Write) 하는 범용으로 사용 가능한 Controller를 구현합니다. [그림 2-1]은 Arty A7-35T 보드입니다. 보드 오른쪽의 사각형으로 표시된 부분이 DDR3 입니다. 보드의 상단 중앙에 표시된 부분은 FPGA Configuration 파일을 다운로드하기 위하여 JTAG-HS2를 연결하는 J8 커넥터입니다.

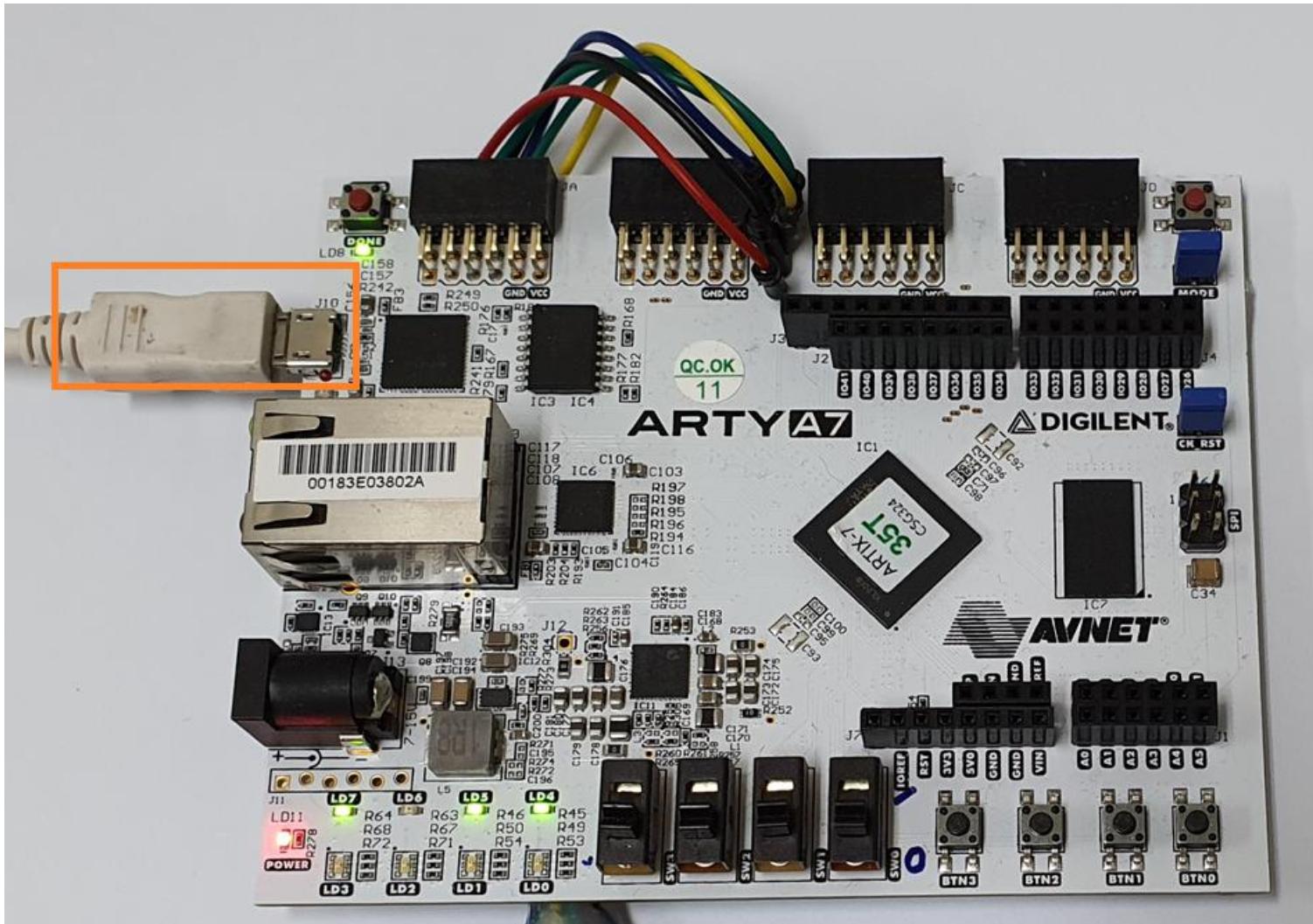
[그림 2-1] Arty A7-35T



Bitstream을 다운로드 하는 방법은 2가지가 있습니다.

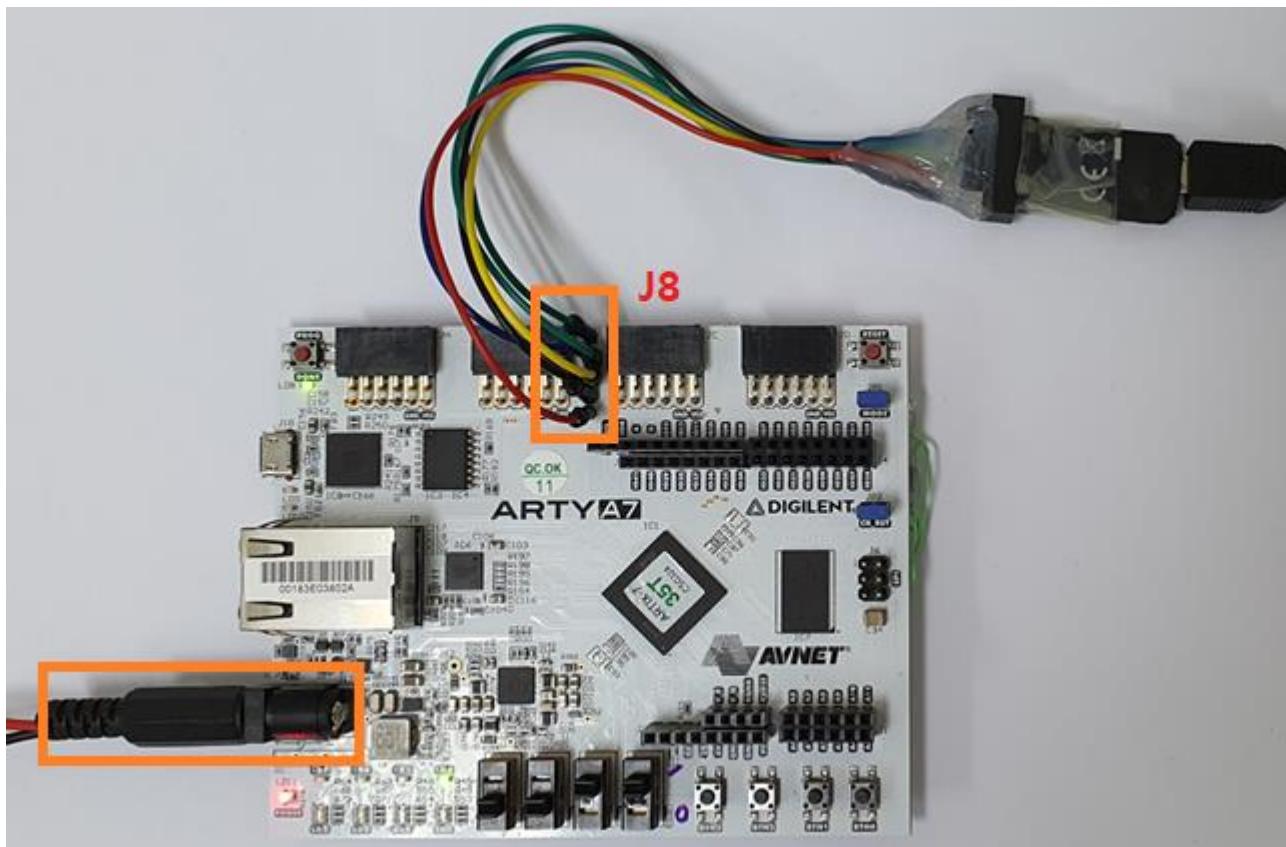
2.1 USB-JTAG를 이용하는 방법

Arty A7-35T 보드에는 USB-JTAG Programming circuit이 포함되어 있습니다. 아래 그림의 J10 USB 포트를 통하여 전원을 공급하고 프로그램(Bitstream)을 다운로드할 수 있습니다. 이때 외부 전원(J13 DC 잭)을 연결하지 않아야 합니다. 프로그램 다운로드는 “5.9 Bitstream 다운로드 & 확인”에 자세히 설명되어 있습니다.



2.2 JTAG-HS2(or HS3)을 이용하는 방법

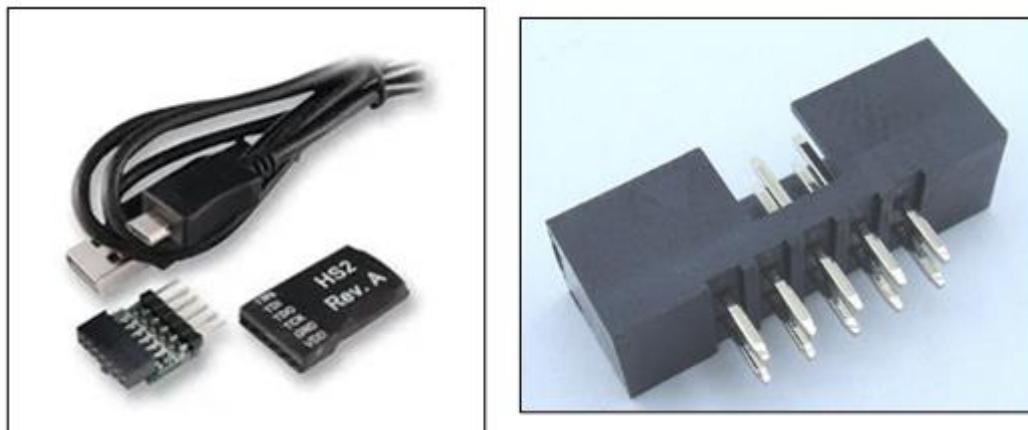
J8 커넥터에 JTAG-HS3을 연결하고, 전원은 J13(DC잭)을 통하여 7V ~ 15V DC를 인가합니다. 외부 전원을 사용하기 때문에 J10 USB 포트를 연결하지 않아야 합니다. (J10 USB 포트를 연결하면, JTAG-HS3 과 충돌이 발생해서 다운로드가 되지 않습니다) 보드(J8)와 JTAG-HS3 핀 맵은 [표 2-1]을 참고하시길 바랍니다. 프로그램 다운로드는 “5.9 Bitstream 다운로드 & 확인”에 자세히 설명되어 있습니다.



[표 2-1] Arty A7과 JTAG-HS3 (BH200-14S) 연결 핀 맵

| Arty A7 J8 | JTAG-HS3 (BH200-14S) | Signal |
|------------|----------------------|--------|
| 1 | 4 | TMS |
| 2 | 10 | TDI |
| 3 | 8 | TDO |
| 4 | 6 | TCK |
| 5 | 1,3,5,7,9,11,13 | GND |
| 6 | 2 | 3.3V |

[그림 2-2] JTAG-HS3(HS2), B200-14S



IHL

3. DDR Controller IP 생성

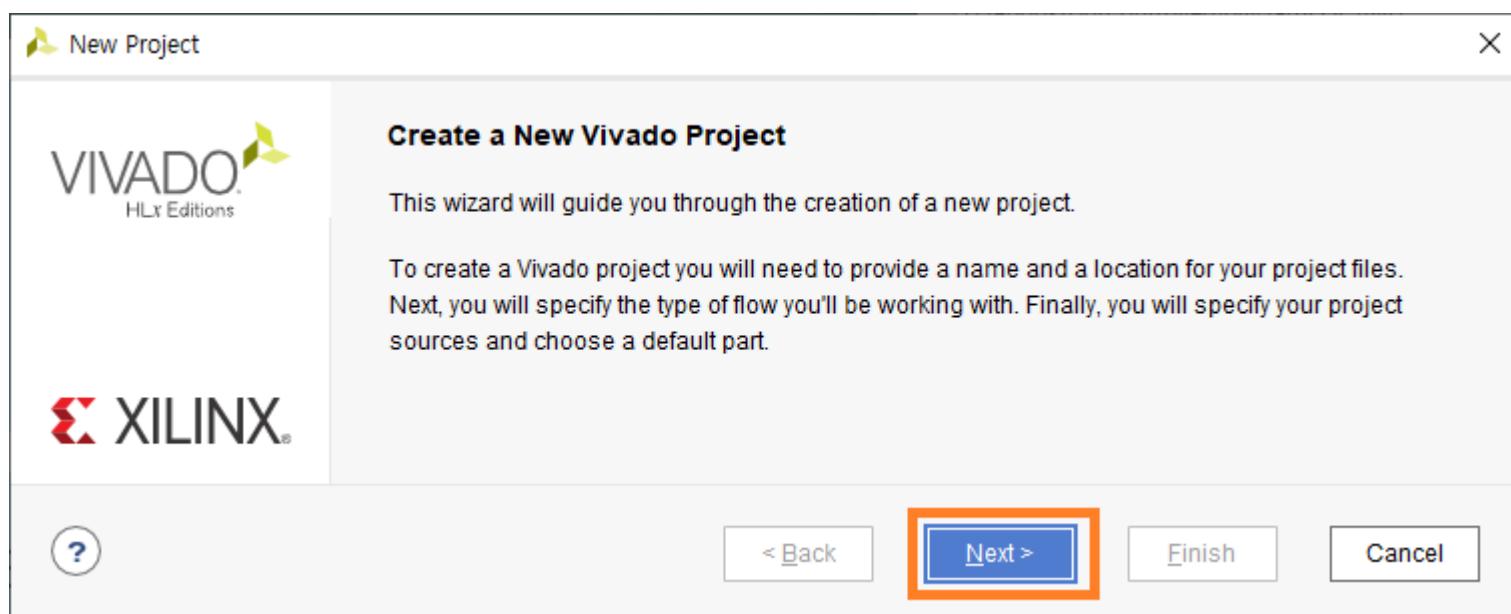
본서에서는 vivado 2018.3을 사용하여 진행합니다. (2018.3 이후 버전에서도 큰 차이 없이 구현 가능합니다). 본 장에서는 프로젝트를 생성하고 DDR Controller IP를 생성하는 과정을 설명합니다.

3.1 프로젝트 생성

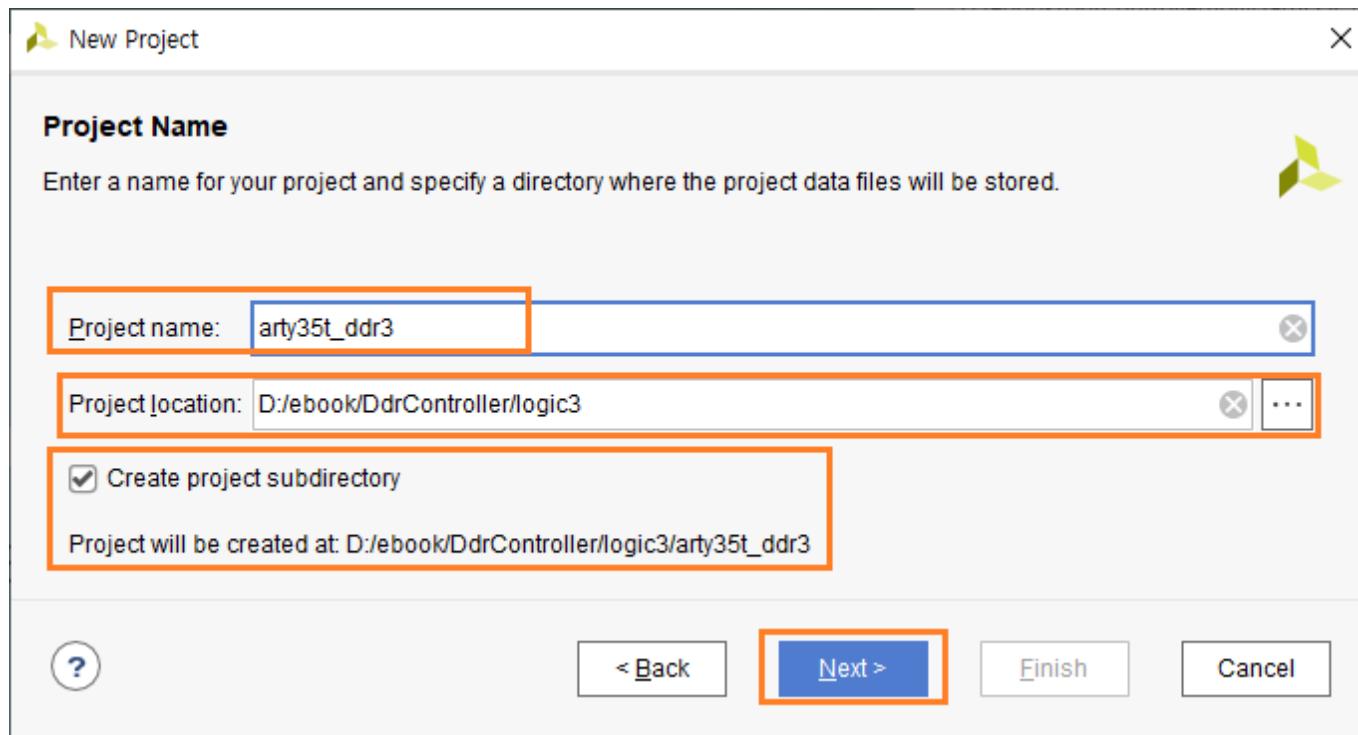
vivado 2018.3을 실행하고 Create Project를 클릭해서 프로젝트를 생성합니다.



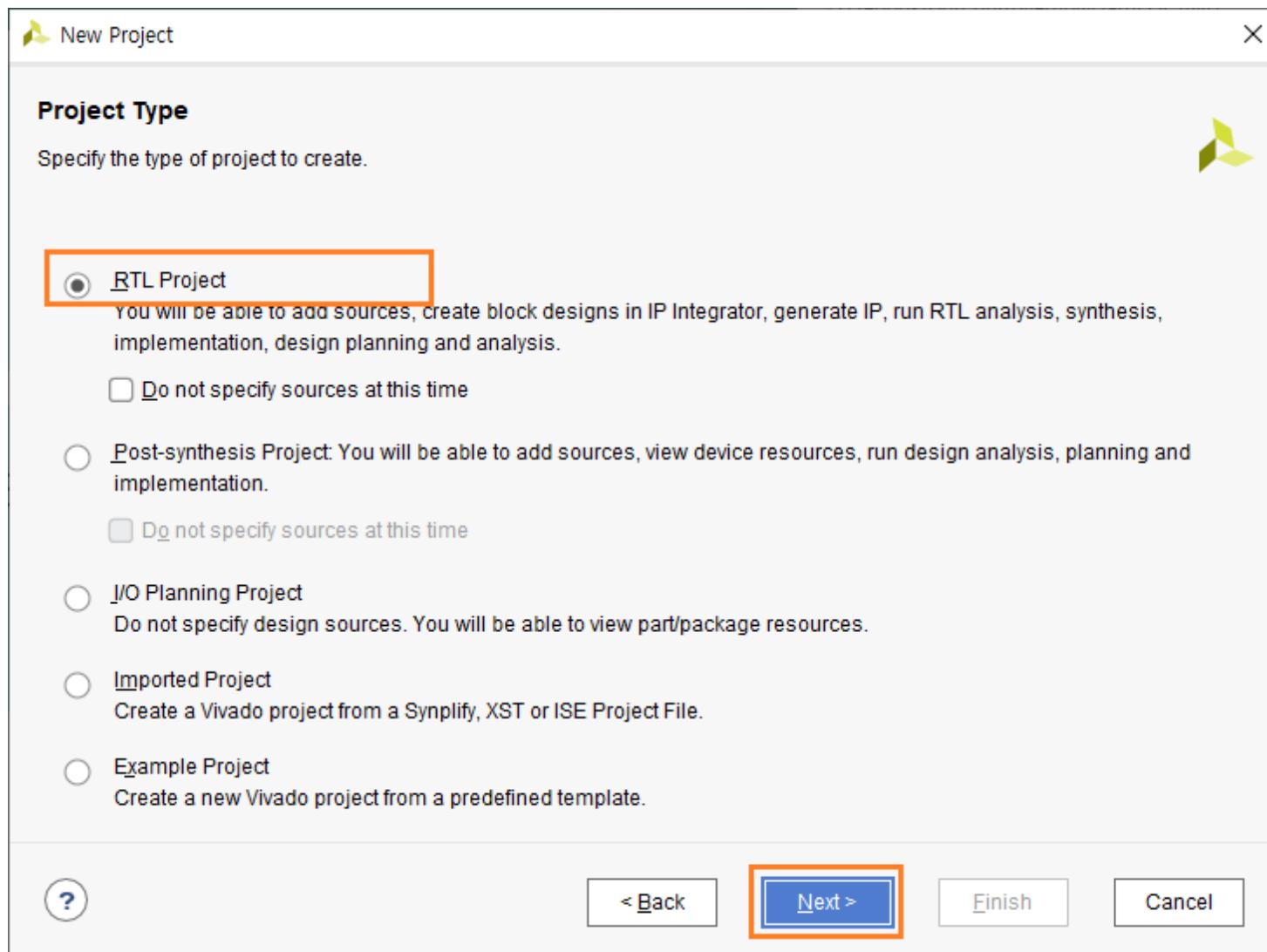
Next 버튼을 클릭합니다.



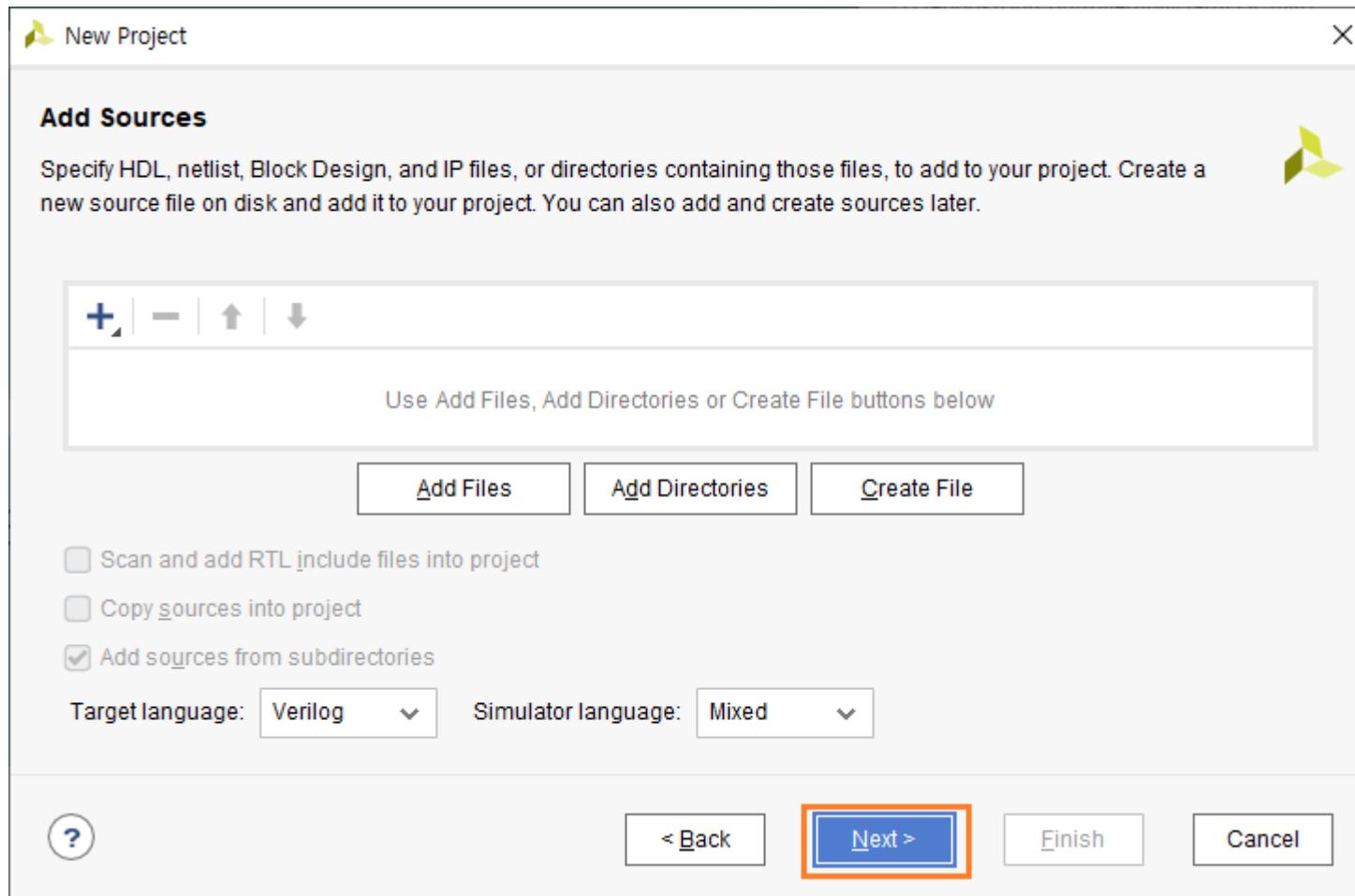
프로젝트 생성위치를 선택하고 프로젝트 이름을 입력합니다. “Create project subdirectory”항목을 Check 해서 새로운 프로젝트 폴더를 생성하도록 합니다. Next 버튼을 클릭합니다.



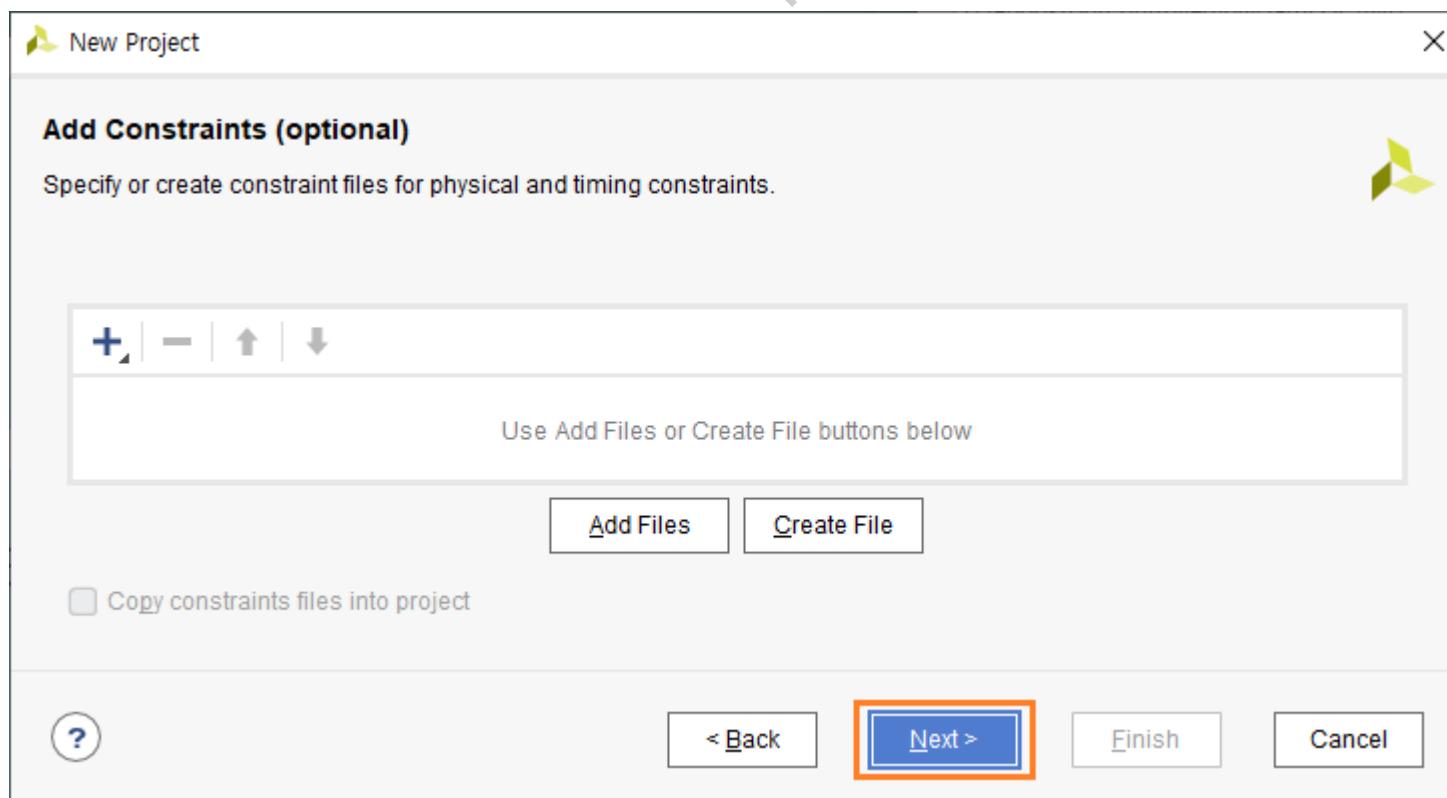
Verilog(RTL)로 코드를 구현하기 때문에 “RTL Project”를 선택하고 Next 버튼을 클릭합니다.



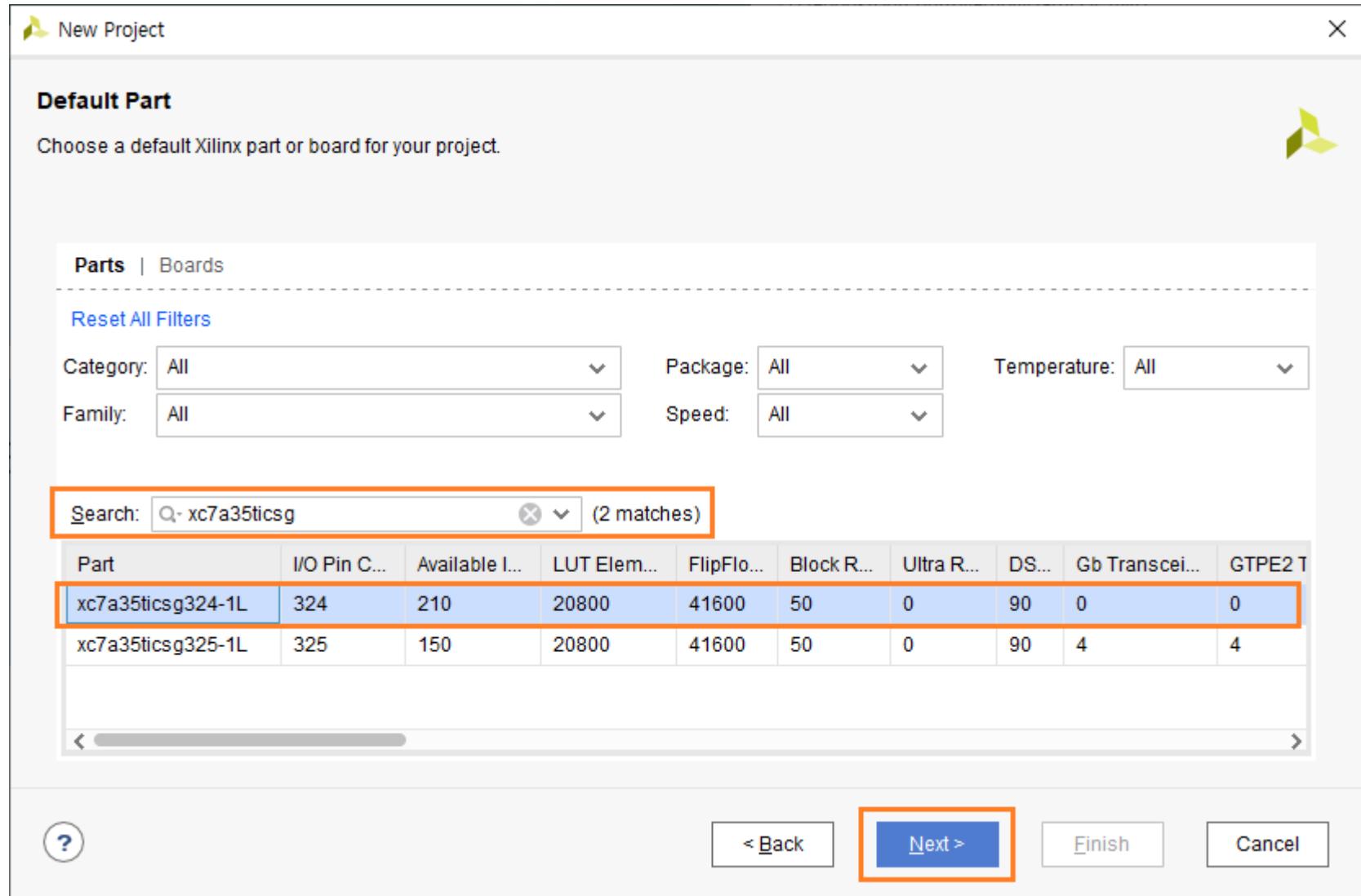
코드는 Memory Controller IP 생성 후 진행할 예정이기 때문에 소스 추가 없이 Next 버튼을 클릭합니다.



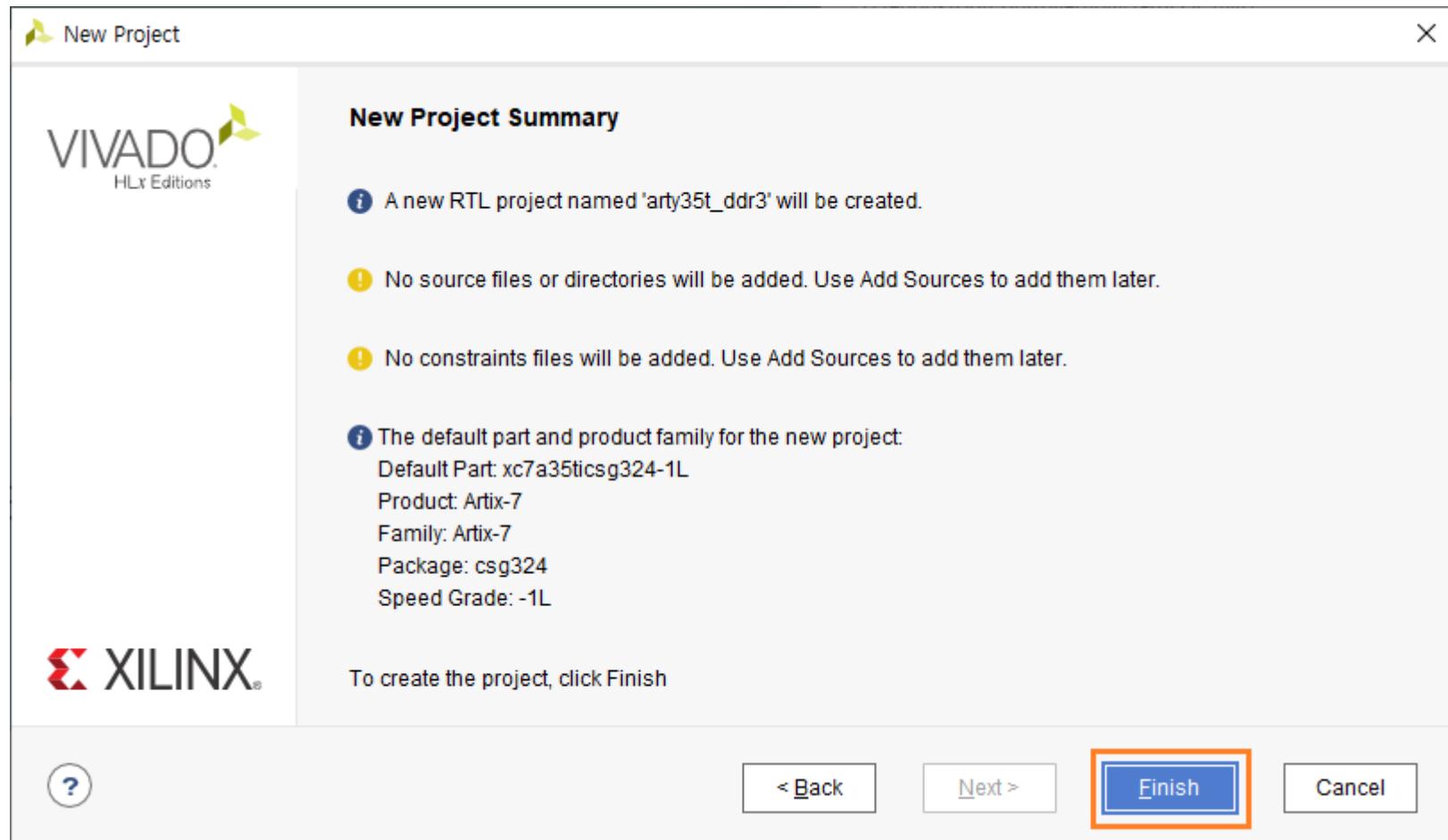
Constraints 파일도 IP 생성 후 추가합니다. Next 버튼을 클릭합니다.



Part를 선택합니다. “Search : “에 xc7a35ticsg를 입력하고 목록에서 “xc7a35ticsg324-1L”을 선택하고 Next 버튼을 클릭 합니다.

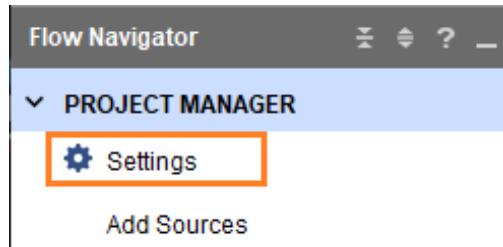


Finish 버튼을 클릭해서 프로젝트를 생성합니다.

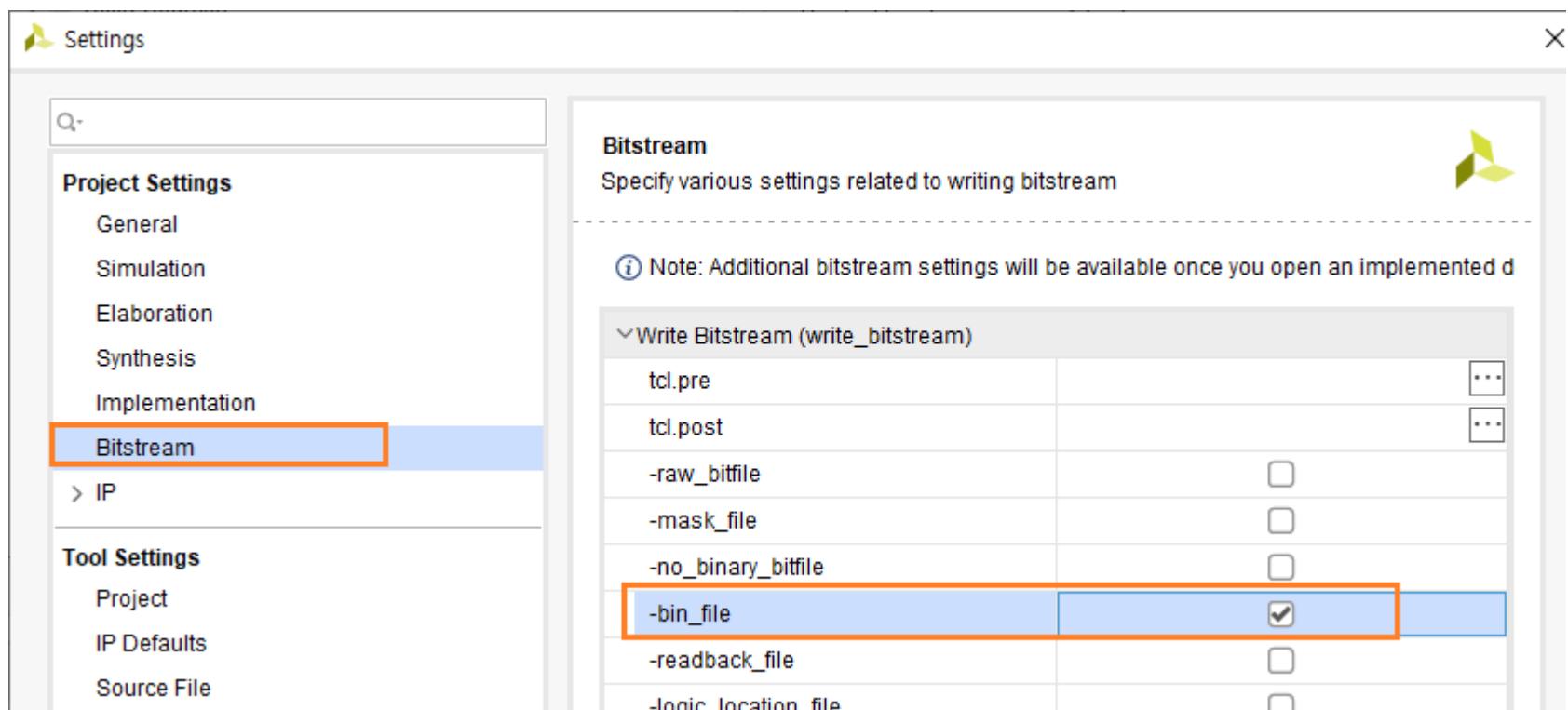


3.2 Memory IP 생성

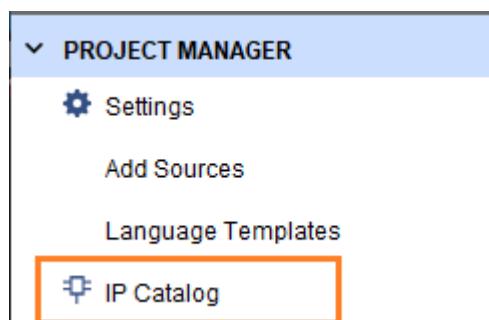
메모리 IP를 생성하기 전에 먼저 프로젝트 속성을 설정합니다. Bitstream 파일을 생성할 때, bin 파일을 생성하도록 설정합니다. Bitstream을 다운로드 할 때, bin 파일을 사용합니다. PROJECT MANAGER - Settings을 클릭합니다.



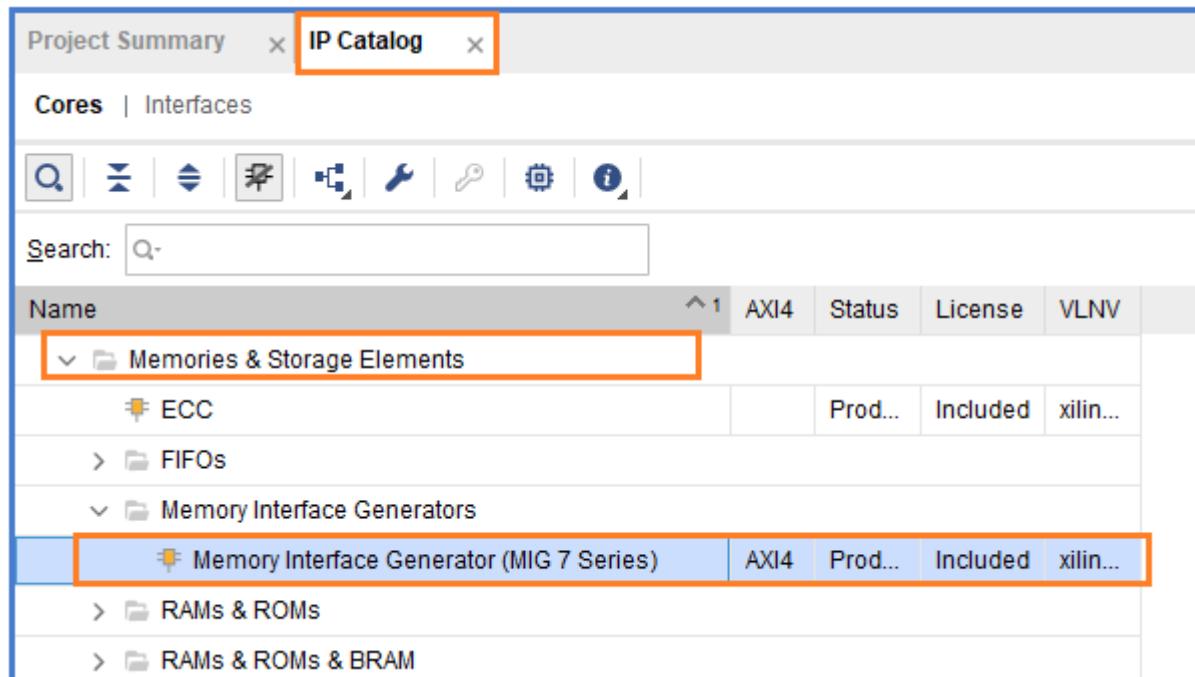
Project Settings - Bitstream 을 클릭하고 “-bin_file” 항목을 Check 후에 OK 버튼을 클릭합니다.



DDR Controller IP를 생성하기 위하여 PROJECT MANAGER - IP Catalog 를 클릭합니다.

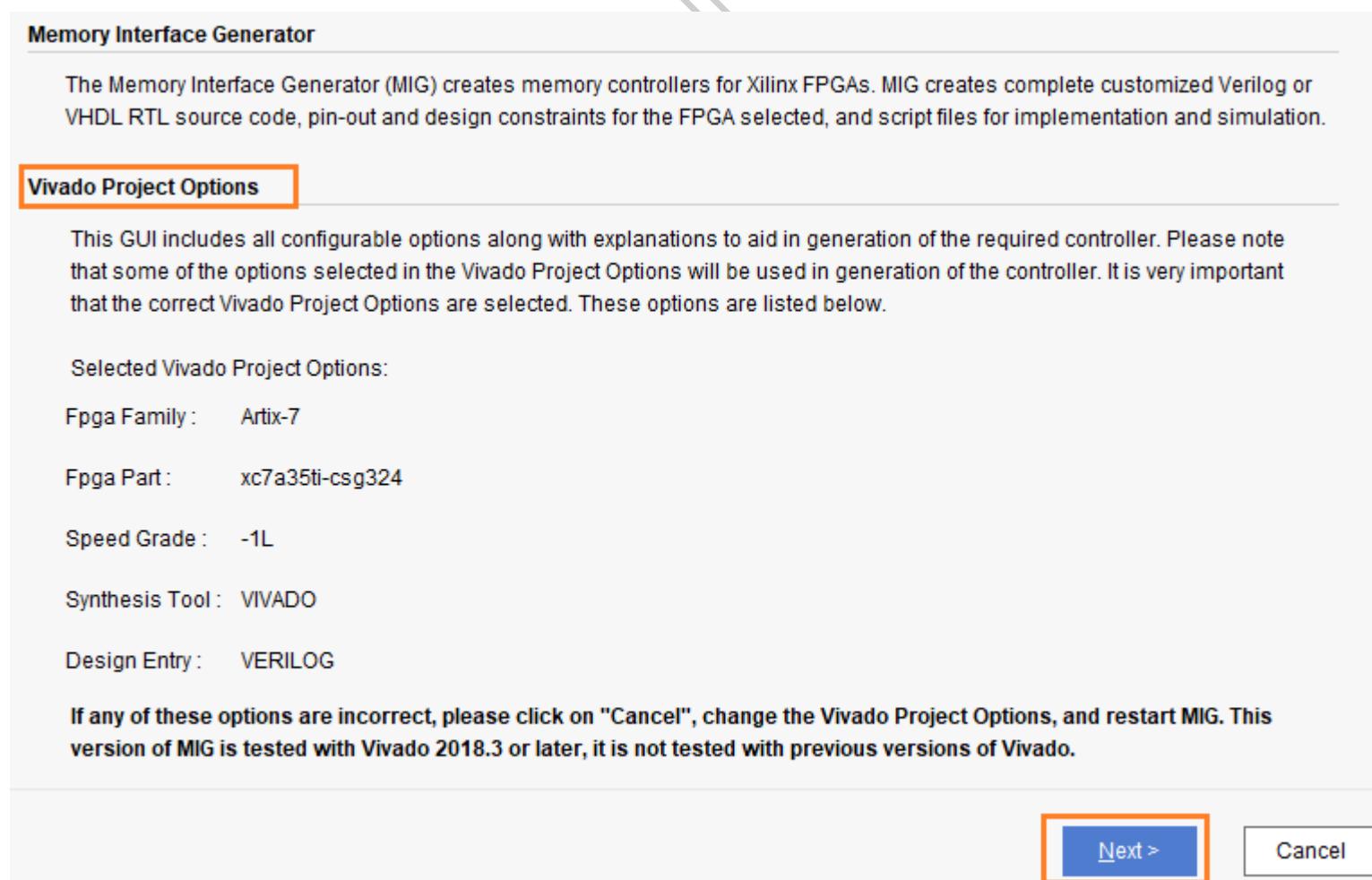


IP Catalog 윈도에서 Memories & Storage Elements - Memory Interface Generators - Memory Interface Generator (MIG 7 Series)을 더블 클릭해서 Memory Interface Generator를 실행합니다.



Memory Interface Generator는 여러가지 사용자 옵션을 선택해주면 거기에 따라서 Memory Interface(or Controller)를 생성해 주는 도구입니다. 사용자 옵션은 크게 HW적인 옵션(Memory 종류, Memory 파트 넘버, Clock Speed, 핀 관련 정보 등)과 SW적인 옵션 (Address 구조, Burst Type 등)으로 구성되어 있습니다. DDR Memory는 고속으로 동작하기 때문에 옵션을 변경할 때에는 매우 신중하게 생각해서 변경해야 합니다. 처음에는 대부분 기본 값을 사용하고 기능이 검증된 후에 값을 변경하는 것이 좋습니다. 본서에서도 최대한 기본적으로 제공하는 옵션들을 사용하고 몇몇 옵션들만 변경해서 사용하도록 하겠습니다. (이러한 옵션들은 미리 구현되고 검증된 사항들입니다.)

Project Options을 확인하고 Next 버튼을 클릭합니다.



Component Name을 입력하고 Next 버튼을 클릭합니다. Multi-Controller는 여러 개의 Memory를 사용하는 경우에 사용하는 개수를 지정합니다. 고속으로 영상데이터를 처리하기 위해서는 여러 개의 Memory를 사용하는 경우가 많습니다. 우리가 사용하는 Memory는 16bits 입니다. 만일 속도를 2배로 늘이고 싶으면 Memory를 2개를 사용하여 Data Width를 32bits로 사용할 수 있습니다. 또는 4개를 사용하여 64bits로 사용할 수 있습니다. 우리가 사용하는 Arty A7보드에는 메모리가 1개 장착되어 있으므로 1로 설정합니다. AXI4 Interface는 주로 MicroBlaze와 같은 프로세서에서 메모리를 사용할 때 선택합니다. 프로세서 없이 영상 데이터를 처리할 때에는 AXI4 Interface를 사용하지 않습니다.

MIG Output Options

Create Design
Select this option to generate a memory controller. Generating a memory controller will create RTL, XDC, implementation and simulation files.

Verify Pin Changes and Update Design
Selecting this feature verifies the modified XDC for a design already generated through MIG. This option will allow you to change the pin out and validate it instantly. It updates the input XDC file to be compatible with the current version of MIG. While updating the XDC it preserves the pin outs of the input XDC. This option will also generate the new design with the Component Name you selected in this page.

Component Name

Please specify the component name for the memory interface. The design directories will be generated under a directory with this name. Three directories will be created "example_design", "user_design" and "docs". The user_design will contain the generated memory interface. The example_design adds a simple example application connected to the generated memory interface.

Component Name 

Multi-Controller

Up to maximum of 8 controllers with a combination of DDR3 SDRAM, QDRII+ SRAM or RLDRAM II can be generated. The number of controllers that can be accommodated may be limited by the data width and the number of banks available in device. Refer user guide for more information

Number of controllers 

AXI4 Interface

Enables the AXI4 interface. AXI4 interface is supported only for DDR3 SDRAM and DDR2 SDRAM controllers with Verilog design entry.

AXI4 Interface

[!\[\]\(8ecebb56a627b2abdf02bcca57cc8de8_img.jpg\)< Back](#)

[!\[\]\(0e2d2c6dd57d59693160a7375112fcd8_img.jpg\)Next >](#)

[!\[\]\(4754b25d33ada2f1f68e511fdb6476a0_img.jpg\)Cancel](#)

Pin Compatible FPGAs 옵션은 Next 버튼을 클릭합니다.

Pin Compatible FPGAs

Pin Compatible FPGAs include all devices with the same package and speed grade as the target device. Different FPGA devices with the same package do not have the same bonded pins. By selecting Pin Compatible FPGAs, MIG will only select pins that are common between the target device and all selected devices. Use the default XDC in the part folder for the target part. If the target part is changed, use the appropriate XDC in the compatible_ucf folder. **If a Pin Compatible FPGA is not chosen now and later a different FPGA is used, the generated XDC may not work for the new device and a board spin may be required.** MIG only ensures that MIG generated pin out is compatible among the selected compatible FPGA devices. Unselected devices will not be considered for compatibility during the pin allocation process.

A blank list indicates that there are no compatible parts exist for the selected target part and this page can be skipped.

Note that different parts in the same package will have different internal package skew values. De-rate the minimum period appropriately in the Controller Options page when different parts in the same package are used. Consult the User Guide for more information.

Target FPGA: `xc7a35ti-csg324 -1L`

Pin Compatible FPGAs

- ✓ artix7
 - ✓ 7a
 - xc7a15ti-csg324
 - xc7a50ti-csg324
 - xc7a75ti-csg324
 - xc7a100ti-csg324

< Back **Next >** Cancel

Memory Selection에서는 DDR3 SDRAM을 선택하고 Next 버튼을 클릭합니다.

Memory Selection

Select the type of memory interface. Please refer to the User Guide for a detailed list of supported controllers for each FPGA family. The list below shows currently available interface(s) for the specific FPGA, speed grade and design entry chosen.

Select the controller type:

DDR3 SDRAM

DDR2 SDRAM

LPDDR2 SDRAM

< Back **Next >** Cancel

Controller Options 윈도입니다. 이번 옵션은 아주 중요한 사항들이 많이 포함되어 있습니다.

- ✓ Clock Period : 설정할 수 있는 값의 범위는 2500 ~ 3300ps 입니다. 기본값으로 3225ps(310.08Mhz)로 설정되어 있습니다. Arty A7보드에는 기본 Clock으로 100Mhz를 사용합니다. PLL을 사용하여 100Mhz를 분주해서 메모리의 Clock으로 사용해야 하는데 100Mhz로 310.08Mhz를 생성할 수 없습니다. (PLL이라는 것이 “출력주파수 = 입력주파수 * M/N”의 구조로 되어 있습니다. M,N의 값은 주로 0~15 or 0~31로 범위가 한정되어 있습니다.) 따라서 100Mhz로 생성가능한 주파수를 설정하도록 합니다. 3077ps (324.99Mhz -> 325Mhz로 설정)로 설정합니다.
- ✓ PHY to Controller Clock Ratio : Clock Period를 3077로 변경하면 4:1로 고정됩니다.
- ✓ Memory Part : Arty A7보드에 장착되어 있는 Memory에 제일 가까운 값을 선택합니다. 회로도에는 “MT41K128M16JT-125K”로 되어 있습니다. Memory List 중에 MT41K128M16XX-15E를 선택합니다.
- ✓ Memory Voltage : 회로도를 참조하시면 1.35V로 설정되어 있음을 알 수 있습니다. 1.35V를 선택합니다.

| DDR Voltage Configuration | | | |
|---------------------------|---------|--------|--------|
| R261 | R266 | DDRVCC | DDRVTT |
| Load | No Load | 1.5V | 0.75V |
| No Load | Load | 1.35V | 0.675V |

- ✓ Data Width : 16 bits를 선택합니다.
- ✓ Data Mask : Memory를 Access 할 때, 바이트 단위로 특정 바이트를 Access하지 않게 설정할 수 있습니다. 즉 해당 바이트의 Mask bit를 1로 설정하면 해당 바이트는 그대로 두고 나머지 바이트만 Access하게 됩니다. 기본값인 Enable을 사용합니다.
- ✓ Number of Bank Machines : 기본값 4를 사용합니다.
- ✓ ORDERING : Normal 을 사용합니다.

여기에 설명된 값들은 제가 여러 자료들을 찾아보고, 그 결과를 검증한 자료들로 구성된 값들입니다. Memory Controller는 설정 값에 매우 민감하게 영향을 받습니다. 따라서 먼저 검증된 값들을 사용하여 구현하고 확인한 후, 다른 설정 값들을 적용해 보시길 바랍니다. 처음부터 내 생각대로 설정하면 오류가 발생할 수 있고 디버깅하기가 매우 어렵습니다. 설정 값으로 인한 오류인지, 코드를 잘못 구현해서 발생한 오류인지를 알아내기가 쉽지 않습니다.

Options for Controller 0 - DDR3 SDRAM

Clock Period: Choose the clock period for the desired frequency. The allowed period range(2500 - 3300) is a function of the selected FPGA part and FPGA speed grade. Refer to the User Guide for more information.

3,077 ps 324.99 MHz

ⓘ To achieve optimum resource utilization, maintain default clock period given by the tool or a value greater than default clock period. Please contact Xilinx Technical Support for further information

PHY to Controller Clock Ratio: Select the PHY to Memory Controller clock ratio. The PHY operates at the Memory Clock Period chosen above. The controller operates at either 1/4 or 1/2 of the PHY rate. The selected Memory Clock Period will limit the choices.

4:1

Memory Type: Select the memory type. Type(s) marked with a warning symbol are not compatible with the frequency selection above.

Components

Memory Part: Select the memory part. Part(s) marked with a warning symbol are not compatible with the frequency selection above. Find an equivalent part or create a part using the "Create Custom Part" button if the part needed is not listed here. The "Create Custom Part" feature is not supported for RLDRAM II.

MT41K128M16XX-15E

Create Custom Part

Memory Voltage: Select the Voltage of the Memory part selected.

1.35V

Data Width: Select the Data Width. Parts marked with a warning symbol are not compatible with the frequency and memory part selected above.

16

ECC: MIG supports ECC for 72 bit data width configuration. To be able to select ECC, select a data width that has ECC supported.

Disabled

Data Mask: Enable or disable the generation of Data Mask (DM) pins using this check box. This option can be selectable only if the memory part selected has DM pins. Uncheck this box to not use data masks and save FPGA I/Os that are used for DM signals. ECC designs (DDR3 SDRAM, DDR2 SDRAM) will not use Data Mask.



Number of Bank Machines: This parameter defines the number of bank machines. A given bank machine manages a single DRAM bank at any given time.

4

Note: Setting a lower value will result in lower resource utilization, but may effect controller efficiency for certain traffic patterns.

ORDERING: Normal mode allows the memory controller to reorder commands to the memory to obtain the highest possible efficiency. Strict mode forces the controller to execute commands in the exact order received.

Normal

Memory Details: 2Gb, x16, row:14, col:10, bank:3, data bits per strobe:8, with data mask, single rank, 1.35V, 1.5V

< Back

Next >

Cancel

Next 버튼을 클릭합니다.

Memory Options 입니다.

- ✓ Input Clock Period : 100Mhz를 사용합니다. Memory IP에서 325Mhz를 생성하여 사용합니다.
- ✓ Output Driver Impedance Control : RZQ/6 을 사용합니다.
- ✓ RTT : RZQ/6을 사용합니다.

나머지 값들은 기본값을 사용하고 Next 버튼을 클릭합니다.

Memory Options C0 - DDR3 SDRAM

Input Clock Period: Select the period for the PLL input clock (CLKIN). MIG determines the allowable input clock periods based on the Memory Clock Period entered above and the clocking guidelines listed in the User Guide. The generated design will use the selected Input Clock and Memory Clock Periods to generate the required PLL parameters. If the required input clock period is not available, the Memory Clock Period must be modified.

10000 ps (100 MHz)

Choose the Memory Options for the memory device. Memory Option selections are restricted to those supported by the controller. Consult the memory vendor data sheet for more information.

Read Burst Type and Length
The burst type determines the data ordering within a burst. Consult the memory datasheet for more information. Burst length 8 is the only supported value.

Sequential

Output Driver Impedance Control
Programmable impedance for the output buffer.

RZQ/6

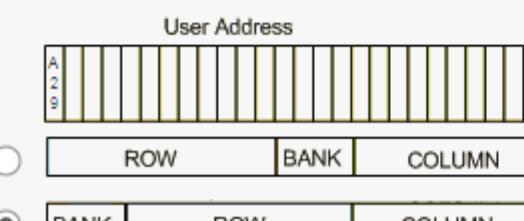
RTT (nominal) - On Die Termination (ODT)
Select the nominal value of ODT for the DQ, DQS/DQS# and DM signals on the component or DIMM interface. This must be set to RZQ/6 (40 ohms) for data rates at 1333 Mbps and above. In 2 slot DIMM configurations this value will be used for the unwritten slot during a write and will also be used for the unselected slot during a read. Use board level simulation to choose the optimum value.

RZQ/6

Controller Chip Select Pin
The Chip Select (CS#) pin can be tied low externally to save one pin in the address/command group when this selection is set to 'Disable'. Disable is only valid for single rank configurations.

Enable

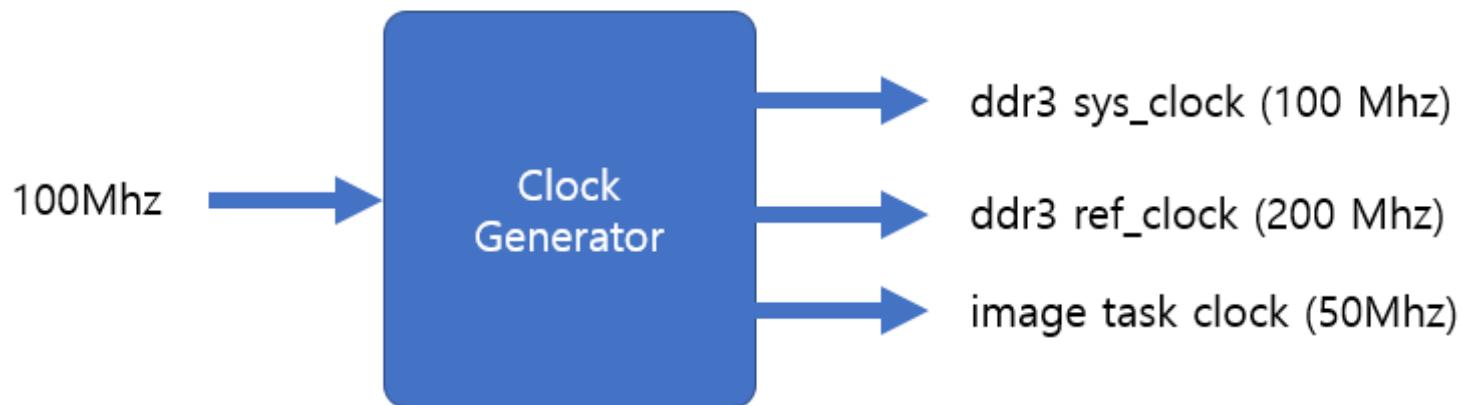
Memory Address Mapping Selection

User Address

 ROW | BANK | COLUMN
 BANK | ROW | COLUMN

< Back **Next >** Cancel

FPGA Options을 설정합니다.

- ✓ System Clock : 앞에서 설정한 Input Clock을 어떻게 받는지를 설정합니다. 외부 핀으로부터 바로 공급받는 방법 (Differential, Single-ended)과 내부(No Buffer)에서 공급받는 방법이 있습니다. 우리는 외부에서 100Mhz Main Clock을 받아 Clock Generator를 이용하여 내부에서 사용하는 여러가지 Clock을 생성하여 사용합니다. 따라서 내부에서 공급받기 때문에 No Buffer를 선택합니다.



- ✓ Reference Clock : System Clock 과 같이 내부에서 공급받기 때문에 No Buffer를 선택합니다. 200 Mhz를 사용합니다.
- ✓ Internal Vref : Arty a7 회로도를 보면 FPGA 핀 R5 (IO_L19N_T3_VREF_34)에 DDR3_ODT 신호가 연결되어 있습니다. 따라서 R5 핀을 Internal Vref 핀이 아니라 일반 IO 핀으로 사용하기 위하여 Check 합니다. 이 부분을 설정하지 않으면 나중에 핀 설정 부분에서 에러가 발생합니다.
- ✓ 나머지 값들을 기본값들을 사용합니다.

Next 버튼을 클릭합니다.

System Clock
Choose the desired input clock configuration. Design clock can be Differential or Single-Ended.
System Clock

Reference Clock
Choose the desired reference clock configuration. Reference clock can be Differential or Single-Ended.
Reference Clock

System Reset Polarity
Choose the desired System Reset Polarity.
System Reset Polarity

Debug Signals Control
This feature allows various debug signals present in the IP to be monitored on the ChipScope tool. The debug signals include status signals of various PHY calibration stages. Enabling this feature will connect all the debug signals to the ChipScope ILA and VIO cores in the example design top module. A part of each bus in the debug interface has been grounded so that users can replace the grounded signals with the required signals.
Debug Signals for Memory Controller

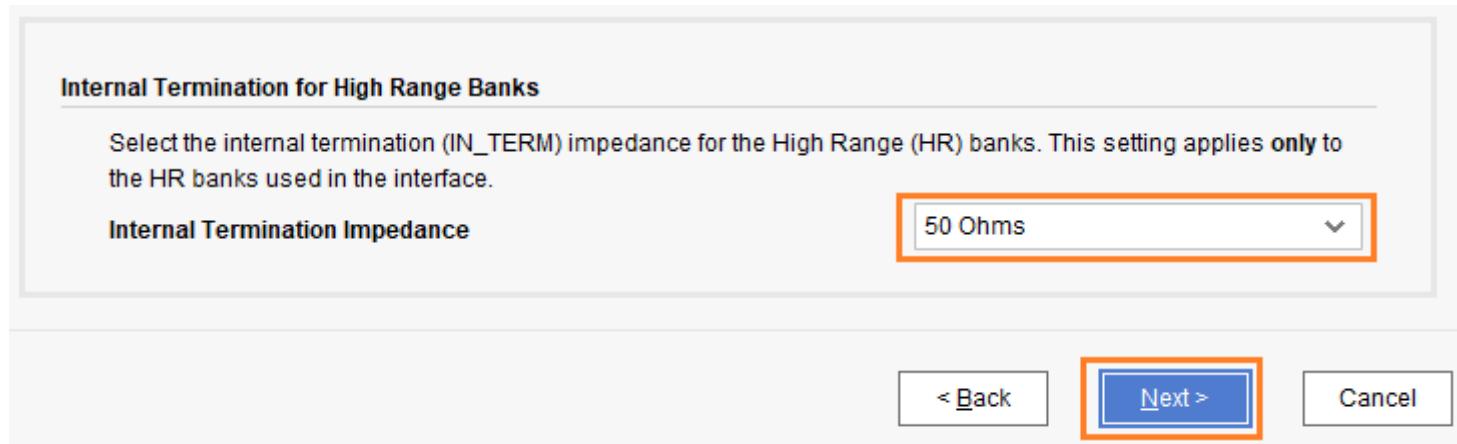
Sample Data Depth
This selects the value of Sample Data depth for Chipscope ILA used in Debug logic.
Sample Data Depth

Internal Vref
Internal Vref can be used to allow the use of the Vref pins as normal IO pins. This option can only be used at 800 Mbps and lower data rates. This can free 2 pins per bank where inputs are used. This setting has no effect on banks with only outputs.
Internal Vref

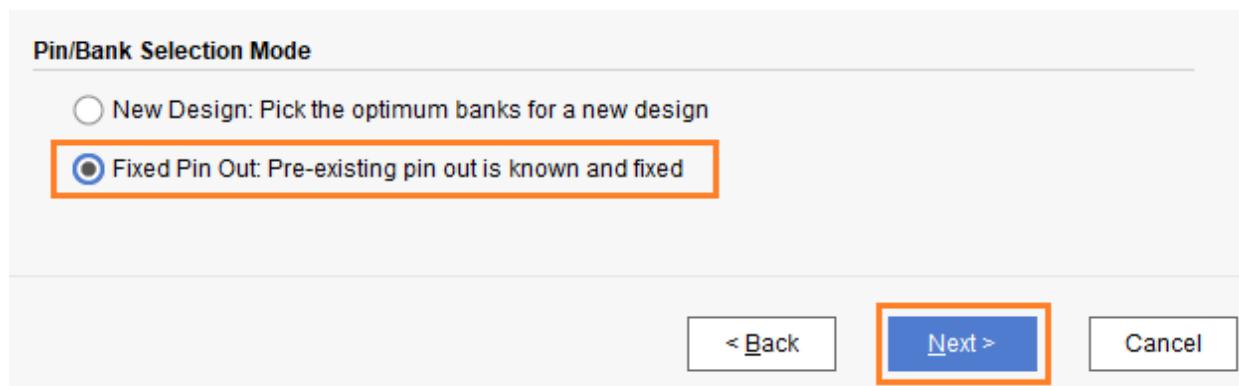
IO Power Reduction
Significantly reduces average IO power by automatically disabling DQ/DQS IBUFs and internal terminations during WRITES and periods of inactivity
IO Power Reduction

XADC Instantiation
The memory interface uses the temperature reading from the XADC block to perform temperature compensation and keep the read DQS centered in the data window. There is one XADC block per device. If the XADC is not currently used anywhere in the design, enable this option to have the block instantiated. If the XADC is already used, disable this MIG option. The user is then required to provide the temperature value to the top level 12-bit device_temp_i input port. Refer to Answer Record 51687 or the UG586 for detailed information.
XADC Instantiation

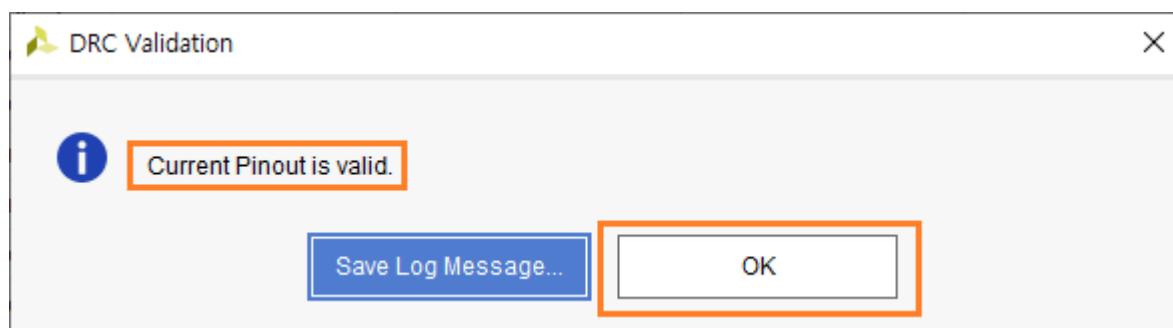
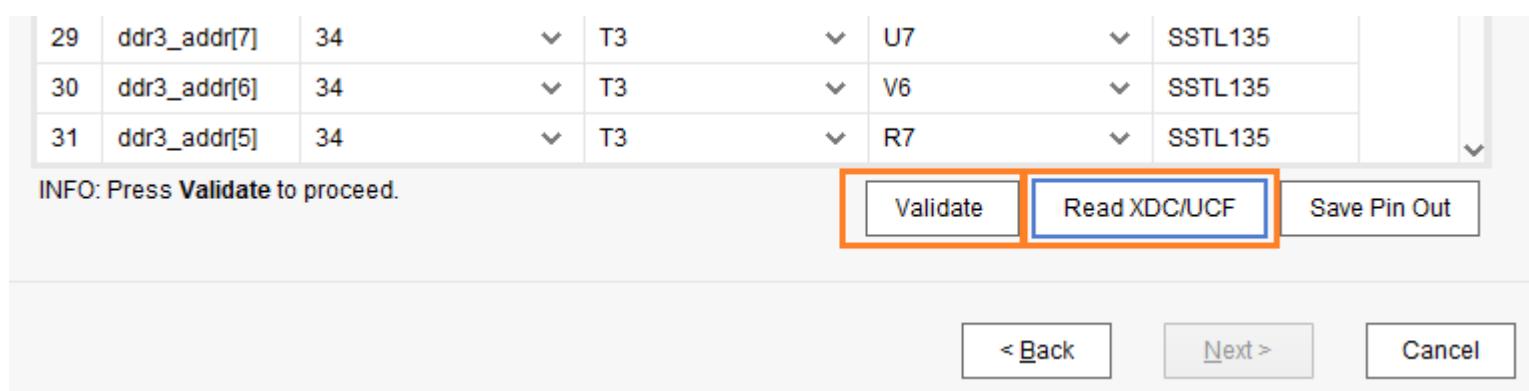
Internal Termination Impedance : 50 ohms (기본값)을 선택하고 Next 버튼을 클릭합니다.



Pin/Bank Selection Mode는 Manual로 각각의 핀들을 설정해야 합니다. “Fixed Pin Out:”을 선택하고 Next 버튼을 클릭합니다.



Pin Selection 원도입니다. DDR Memory의 각각의 핀들을 회로도를 보고 설정해 주어야 합니다. 미리 설정된 파일이 있으면 그 파일을 Read 해서 사용할 수 있습니다. 제공된 소스에서 “arty35t_ddr_ddr_controller.ucf” 파일을 불러옵니다. Read XDC/UCF 버튼을 클릭해서 해당 파일을 불러옵니다. 그러면 각각의 핀들이 할당된 것을 확인할 수 있습니다. “Validate” 버튼을 클릭해서 유효성을 확인한 후 에러가 없으면 Next 버튼이 활성화 됩니다. 에러가 없으면 Next 버튼을 클릭합니다.



Pin Selection For Controller 0 - DDR3 SDRAM

| | Signal Name | Bank Number | Byte Number | Pin Number | IO Standard |
|----|---------------|-------------|-------------|------------|--------------|
| 1 | ddr3_dq[0] | 34 | T0 | K5 | SSTL135 |
| 2 | ddr3_dq[1] | 34 | T0 | L3 | SSTL135 |
| 3 | ddr3_dq[2] | 34 | T0 | K3 | SSTL135 |
| 4 | ddr3_dq[3] | 34 | T0 | L6 | SSTL135 |
| 5 | ddr3_dq[4] | 34 | T0 | M3 | SSTL135 |
| 6 | ddr3_dq[5] | 34 | T0 | M1 | SSTL135 |
| 7 | ddr3_dq[6] | 34 | T0 | L4 | SSTL135 |
| 8 | ddr3_dq[7] | 34 | T0 | M2 | SSTL135 |
| 9 | ddr3_dq[8] | 34 | T1 | V4 | SSTL135 |
| 10 | ddr3_dq[9] | 34 | T1 | T5 | SSTL135 |
| 11 | ddr3_dq[10] | 34 | T1 | U4 | SSTL135 |
| 12 | ddr3_dq[11] | 34 | T1 | V5 | SSTL135 |
| 13 | ddr3_dq[12] | 34 | T1 | V1 | SSTL135 |
| 14 | ddr3_dq[13] | 34 | T1 | T3 | SSTL135 |
| 15 | ddr3_dq[14] | 34 | T1 | U3 | SSTL135 |
| 16 | ddr3_dq[15] | 34 | T1 | R3 | SSTL135 |
| 17 | ddr3_dm[0] | 34 | T0 | L1 | SSTL135 |
| 18 | ddr3_dm[1] | 34 | T1 | U1 | SSTL135 |
| 19 | ddr3_dqs_p[0] | 34 | T0 | N2 | DIFF_SSTL135 |
| 20 | ddr3_dqs_n[0] | 34 | T0 | N1 | DIFF_SSTL135 |
| 21 | ddr3_dqs_p[1] | 34 | T1 | U2 | DIFF_SSTL135 |
| 22 | ddr3_dqs_n[1] | 34 | T1 | V2 | DIFF_SSTL135 |
| 23 | ddr3_addr[13] | 34 | T3 | T8 | SSTL135 |
| 24 | ddr3_addr[12] | 34 | T3 | T6 | SSTL135 |
| 25 | ddr3_addr[11] | 34 | T3 | U6 | SSTL135 |
| 26 | ddr3_addr[10] | 34 | T3 | R6 | SSTL135 |
| 27 | ddr3_addr[0] | 34 | T3 | U7 | SSTL135 |

Validation successful. Press Next to proceed.

기타 핀들을 설정합니다. 회로도를 참조해서 설정합니다.

- ✓ sys_rst : Memory Controller Reset입니다. C2 (CK_RST) 핀에 할당합니다.
- ✓ init_calib_complete : Memory는 파워가 인가되면 초기 설정을 합니다. 초기 설정이 완료되면 이 핀을 High로 만들어 줍니다. 초기 설정이 완료되었는지를 확인하기 위하여 LED4(H5)에 연결합니다. 전원 인가 후 LED4가 On이 되면 정상적으로 Memory가 초기화가 되었다는 것을 의미합니다. 만일 LED4가 On이 되지 않으면 에러가 발생한 것입니다.
- ✓ tg_compare_error : 메모리 Block 중에 Traffic Generator Block의 오류가 발생했음을 알려줍니다. 이 핀을 LED3_R(K1)에 할당합니다. 만약 LED3_R이 ON이 되면 Traffic Generator Block에 오류가 발생함을 의미합니다.

Next 버튼을 클릭합니다.

System Signals Selection

Select the system pins below appropriately for the interface. Customization of these pins can also be made in the XDC after the design is generated. For more information see [UG586 Bank and Pin rules](#).

System Clock and Reference Clock pin selections will not be visible if the 'No Buffer' option was selected in the FPGA Options page.

System Signals

These signals may be connected internally to other logic or brought out to a pin.

- **sys_rst**: This input signal is used to reset the interface.
- **init_calib_complete**: This signal indicates that the interface has completed calibration and memory initialization and is ready for commands. LOC constraint will be generated in XDC for Example design only based on "Pin Number" selection below.
- **error**: This output signal indicates that the traffic generator in the Example Design has detected a data mismatch. This signal does not exist in the User Design.

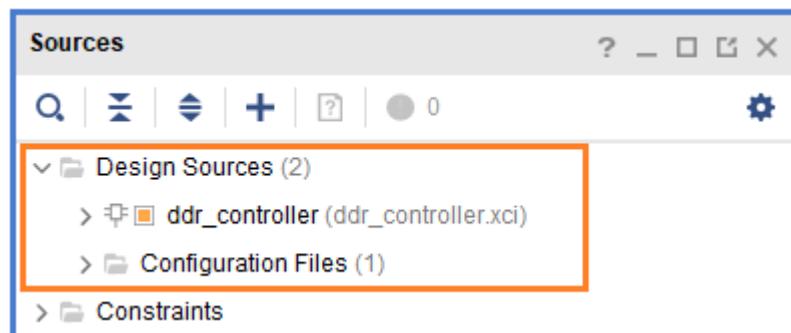
| Signal Name | Bank Number | Pin Number |
|---------------------|-------------|------------|
| sys_rst | 35 | C2 |
| init_calib_complete | 35 | H5 |
| tg_compare_error | 35 | K1 |

All pins must be constrained to specific locations in order to generate a bit file in the implementation phase (this is not required for simulation).

[< Back](#) Next > [Cancel](#)

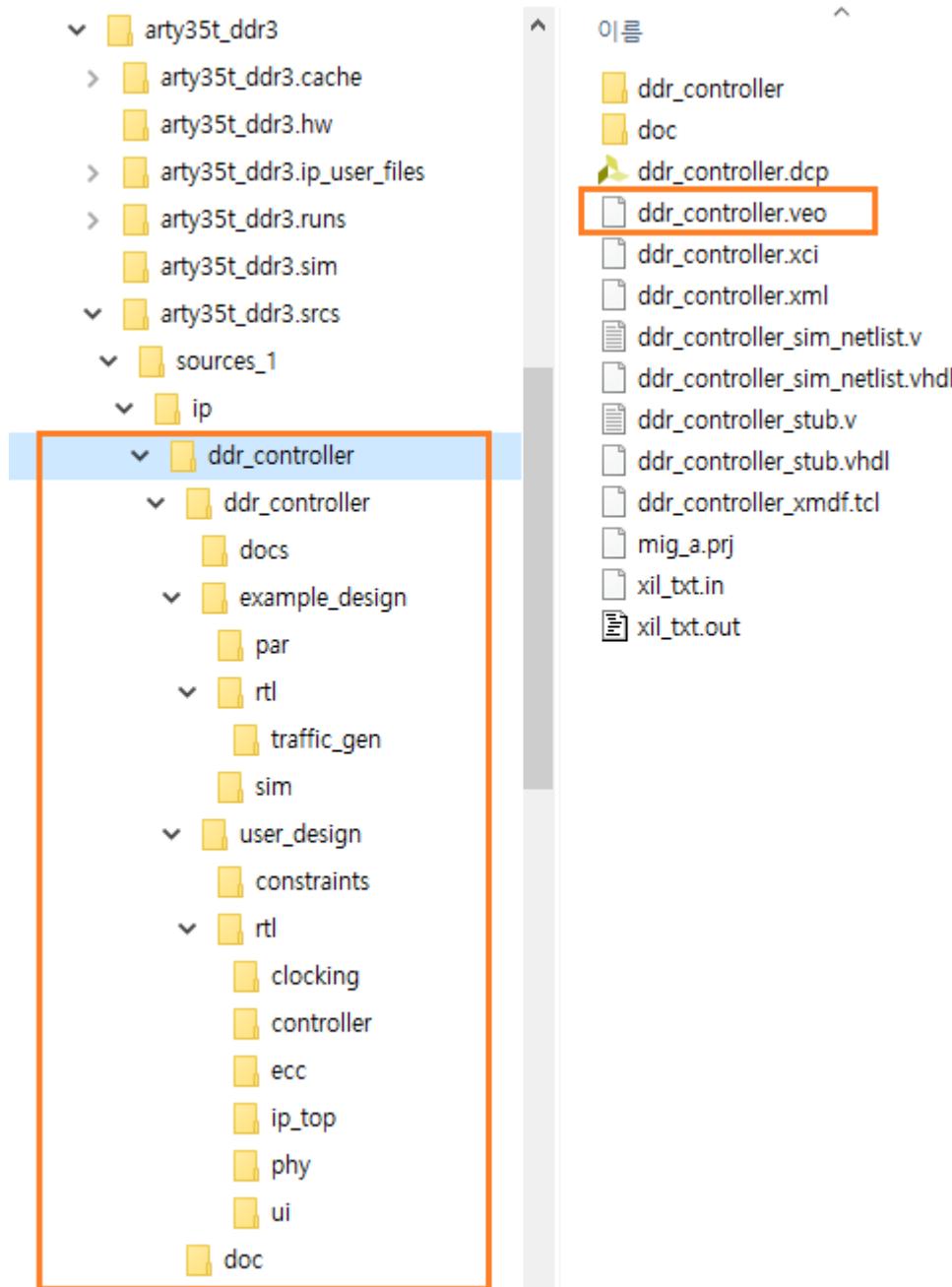
모든 설정이 완료되었습니다. Summary를 확인하고, License Agreement에 Accept 후에 몇번의 Next를 클릭하고 Generate를 클릭하면 Memory Controller IP가 생성됩니다.

Memory Controller가 생성되었습니다. Design Sources에 ddr_controller IP가 생성된 것을 확인할 수 있습니다.



3.3 Memory IP 구조

이번절에서는 생성된 Memory IP의 구조를 살펴보겠습니다. 아래 그림은 폴더 구조를 보여줍니다. 프로젝트 이름은 앞에서 설정한 “arty35_ddr3”입니다.

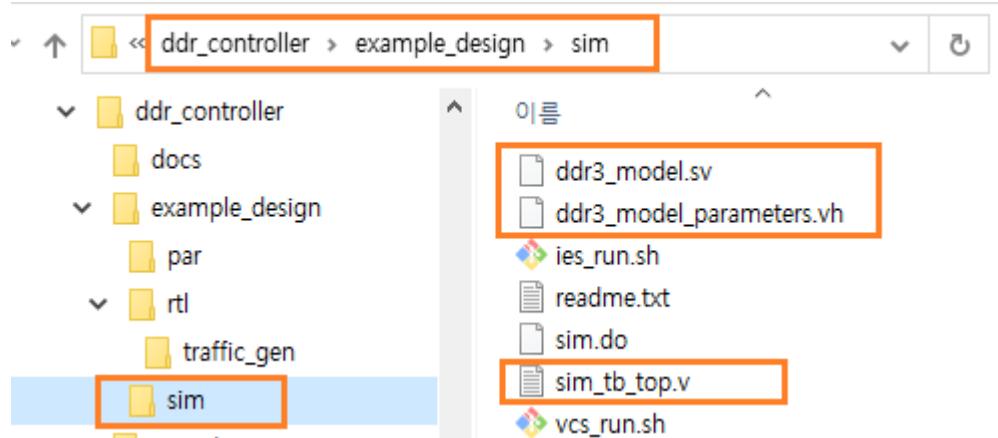


먼저 프로젝트 구조를 보면 크게 runs, sim, srcs 폴더로 구성되어 있습니다. runs 폴더는 Synthesis, Implementation, Bitstream 관련된 결과 파일들이 저장되고, sim 폴더는 simulation 관련 파일들이 저장되고, srcs 폴더는 소스 파일과 생성된 IP, Configuration 파일이 저장됩니다. 생성된 IP는 “arty35t_ddr3.srcs\source_1\ip\ddr_controller” 폴더에 저장되어 있습니다. ddr_controller 폴더에는 IP를 코드에서 Instant 하기 위한 Template Code를 제공합니다. ddr_controller.veo 파일이 verilog template code입니다.

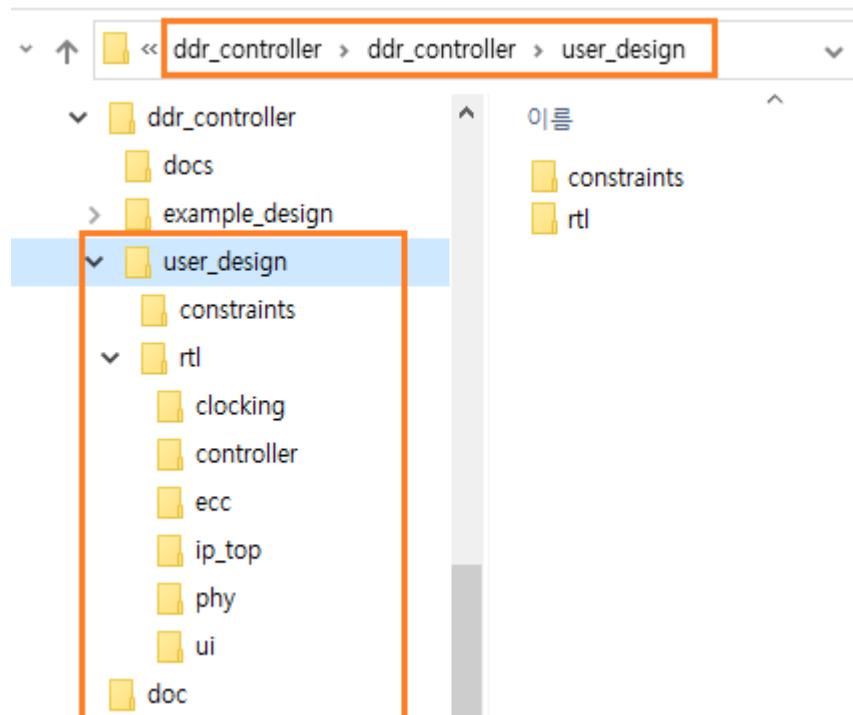
ddr_controller 폴더에 ddr_controller 폴더가 또 있습니다. 이 폴더 안에는 크게 example_design, user_design 폴더가 있습니다. 이 폴더들이 매우 중요한 폴더입니다.

- ✓ example_design : Memory IP의 동작을 확인하기 위한 Simulation 파일들이 포함되어 있습니다. Simulation Top Module은 “example_design\sim\sim_tb_top.v”입니다. 우리는 다음장에서 Memory IP의 동작을 확인하기 위하여 이 파일을 통해 simulation을 진행하도록 합니다. Memory IP를 simulation 하는 것은 매우 중요합니다. 사

실 메모리의 구조는 매우 복잡해서 그 내용을 다 이해하는 것은 쉽지 않습니다. 마찬가지로 Memory IP의 내용을 모두 이해하는 것도 매우 어려운 일입니다. 우리는 simulation을 통하여 Memory IP의 동작을 이해하고 우리가 필요한 부분을 코드로 구현할 것입니다. 이부분은 다음장에서 자세히 설명합니다. sim 폴더 안에는 메모리를 코드로 모델링한 파일들이 포함되어 있습니다. ddr3_model.sv 파일이 모델링한 파일이고, ddr3_model_parameters.vh 파일은 모델링과 관련된 파라미터들이 정의된 파일입니다. 이 파일들은 Memory IP를 생성할 때 설정된 Options에 따라 구성되었습니다.

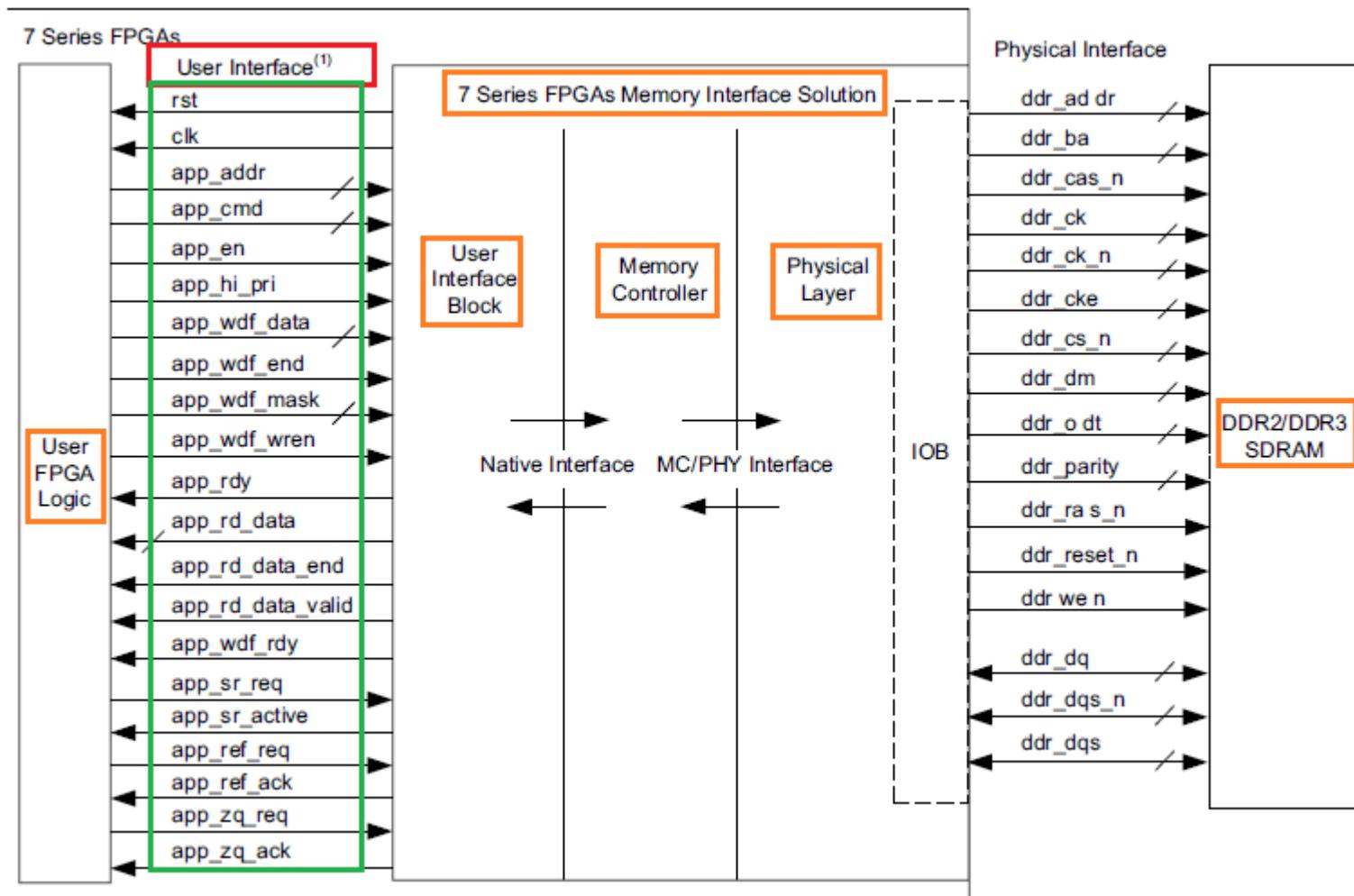


- ✓ user_design : Memory IP를 사용하기 위한 Source 코드들입니다. constraints 폴더에는 앞에서 설정한 핀 Assign 정보가 포함된 xdc 파일이 있습니다. 이 파일에는 DDR Memory와 관련된 핀의 정보만 포함되어 있습니다. 사용자가 그 외의 핀들을 추가하려면 별도의 xdc 파일을 생성해서 프로젝트 폴더에 추가해 주면 됩니다. 그 외에 많은 소스 코드들이 포함되어 있습니다. 우리는 이 코드들을 모두 분석해서 구현하지는 않습니다. Memory IP의 동작을 이해하고 필요한 부분을 사용할 뿐입니다. 예를 들어 FPGA내부의 Clock Generator나 내부 Memory를 사용할 때 내부의 소스코드들을 모두 분석하지는 않습니다. Clock Module이 어떻게 동작하고, 내부 Memory들이 어떻게 동작하는지를 알고 이를 이용하여 우리 목적에 맞게 코드를 구현합니다. Memory Controller도 마찬가지입니다. Simulation을 통하여 동작을 파악하고 우리가 필요한 부분을 구현해서 사용합니다.



3.4 User Interface Block

이번 절에서는 Memory IP를 사용하기 위한 기본적인 구성을 이해합니다. 세부적인 내용을 알 필요는 없지만 전체적인 구성과 동작에 대해서는 이해할 필요가 있습니다. 이번 절의 내용은 Xilinx사의 문서 “Zynq-7000 Soc and 7Series Devices Memory Interface Solutions v4.2” 내용 중에서 “Ch1. DDR3 and DDR2 SDRAM Memory Interface Solution”의 내용을 바탕으로 작성되었습니다. 본서를 통해 Memory IP를 이해하고자 하는 분들은 한번쯤 상기 문서를 보시는 것을 추천해 드립니다. 아래 그림은 DDR Controller System Block을 보여줍니다. (본 절에서 사용된 그림의 출처는 상기 문서입니다)



오른쪽의 “DDR2/DDR3 SDRAM”은 Physical Memory를 나타냅니다. 왼쪽의 “7 Series FPGAs”는 FPGA 내부를 나타냅니다. “7 Series FPGAs Memory Interface Solution”은 우리가 생성한 Memory IP를 나타내고 왼쪽의 “User FPGA Logic”은 우리가 구현해야 할 사용자 Interface를 나타냅니다. “7 Series FPGAs Memory Interface Solution”은 User Interface Block, Memory Controller, Physical Layer로 구성됩니다. Physical Layer는 실제 HW 메모리와 연결되는 부분이고, User Interface Block은 우리가 설계하는 User FPGA Logic과 연결되고, Memory Controller는 IP 내부에서 User Interface Block과 Physical Layer와 연결되어 있습니다.

따라서 우리가 관심있게 살펴봐야 할 부분은 User FPGA Logic과 연결되는 User Interface Block입니다. 나머지 Memory Controller와 Physical Layer는 IP 내부에서 동작하기 때문에 우리가 몰라도 됩니다. 우리는 User Interface Block의 동작을 중점적으로 연구할 것입니다. 특히 User Interface 신호들을 중점적으로 분석하고 이해할 것입니다. User Interface Block의 동작을 정확히 이해하고 있으면 거기에 맞게 User FPGA Logic을 구현할 수 있고 결과적으로 Physical Memory의 Interface를 구현하게 될 것입니다.

3.5 User Interface 신호

User Interface 신호들은 크게 Address/Command 관련 신호, Write 관련 신호, Read 관련 신호, 기타로 구성되어 있습니다. 신호들의 In/Out은 Memory IP(User Interface Block) 관점입니다.

3.5.1 Address/Command 관련 신호

- ✓ app_addr[27:0] (In) : Address입니다. 보드에 장착된 메모리의 용량은 총 2Gb입니다. 메모리의 데이터 버스는 16bits입니다. 따라서 Address 당 Data Width는 16bits입니다. 계산상으로 Address는 27bits면 됩니다. ($2^{27} \times 16 = 2\text{Gb}$) 그러나 생성된 Memory Controller의 Address는 28bits를 사용함에 주의해야 합니다. 또한 Memory Controller는 Address의 Space를 Flat하게 만들어 사용합니다. 즉 사용자는 0번지부터 0x7ff_ffff 까지, 메모리의 물리적 구조인 row, column, bank에 무관하게 사용할 수 있습니다. 사용자가 Address를 입력하면 내부적으로 row, column, bank를 알아서 설정해 줍니다. 이러한 부분이 FPGA에서 생성해주는 IP를 사용하는 장점중의 하나라고 생각됩니다.
- ✓ app_cmd[2:0] (In) : read시에는 1(001b), Write 시에는 0(000b)입니다.
- ✓ app_en (In) : app_addr, app_cmd 신호의 strobe 신호입니다. app_addr, app_cmd가 준비되고 app_en을 ‘H’로 주면 app_addr, app_cmd가 전달됩니다.
- ✓ app_rdy (Out) : User Interface Block이 커맨드를 받을 준비가 되었음을 알려줍니다. 만일 app_rdy 상태가 “L” 상태이면, app_addr, app_cmd, app_en 신호는 app_rdy가 “H”가 될 때까지 값을 유지해야 합니다.
- ✓ app_hi_pri (In) : not used.

3.5.2 Write 관련 신호

- ✓ app_wdf_data[127:0] (In) : 메모리에 Write하기 위한 data입니다. User Interface Block은 128bits 단위로 데이터를 Access합니다. 앞에서 메모리를 생성할 때, PHY to Controller Clock Ratio를 4:1로 설정하였기 때문입니다. 만약 이 값을 2:1로 설정하면 64bits 단위로 Access가 이루어집니다. 이 부분은 구현된 Memory IP의 특성이므로 사용자가 변경할 수 없는 부분입니다. 다음장의 Simulation에서 확인하겠지만 Address는 8의 배수 단위로 설정되고 데이터는 한번에 128bits를 단위로 Access가 이루어집니다.
- ✓ app_wdf_wren (In) : Write Strobe입니다.
- ✓ app_wdf_mask[7:0] (In) : 8 바이트(128bits) 중 Access하지 않는 바이트를 mask합니다.
- ✓ app_wdf_end (In) : 128bits 단위로 Access가 이루어 질 때에는 app_wdf_wren 신호와 동일한 신호를 전달합니다.
- ✓ app_wdf_rdy (Out) : User Interface Block이 데이터를 받을 준비가 되었음을 나타내는 신호입니다. User Logic은 app_wdf_rdy 신호가 “H”일 때 데이터를 Write해야 합니다.

3.5.3 Read 관련 신호

- ✓ app_rd_data[127:0] (Out) : 메모리에 Read Address와 Command를 전송하면 Memory IP는 Physical Memory에서 데이터를 읽어서 User Logic으로 데이터를 전달합니다. 28bits 단위로 Access 합니다.
- ✓ app_rd_data_valid (Out) : app_rd_data 가 유효한 값인지를 나타냅니다. User Logic은 이 값이 "H" 일 때에만 app_rd_data를 유효한 데이터로 처리해야 합니다.
- ✓ app_rd_data_end (Out) : 128bits로 Access 할 때에는 app_rd_data_valid 신호와 동일한 신호가 출력됩니다.

3.5.4 기타 신호

- ✓ app_ref_req (In) : not used, 1'b0 입력
- ✓ app_zq_req (I) : not used, 1'b0 입력
- ✓ app_sr_req (In) : not used, 1'b0 입력
- ✓ app_ref_ack (Out) : not used.
- ✓ app_zq_ack (Out) : not used.
- ✓ app_sr_active (Out) : not used.

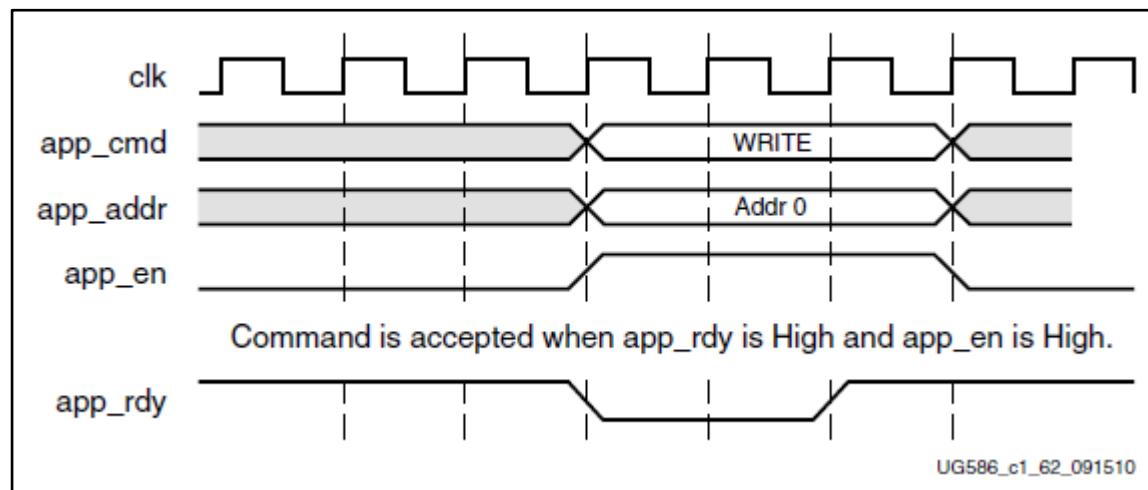
/HIL

3.6 User Interface Timing

이번 절에서는 User Interface Timing에 대해서 설명합니다.

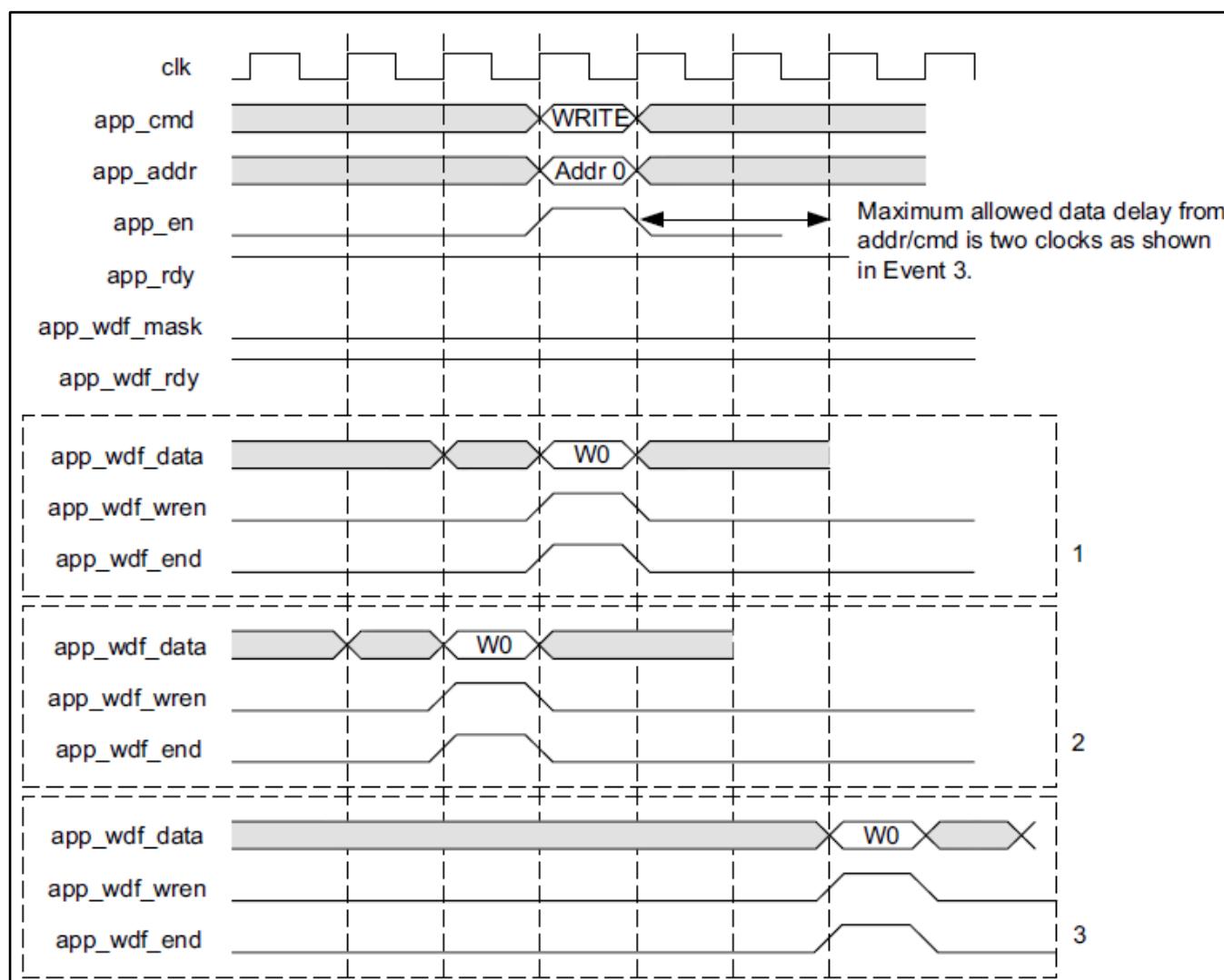
3.6.1 Address / Command Timing

아래는 Address / Command 대한 Timing을 나타냅니다. app_rdy 신호가 'L' 일 때에는 app_cmd, app_addr, app_en 신호를 app_rdy 신호가 'H'가 될 때까지 값을 계속 유지해야 합니다.



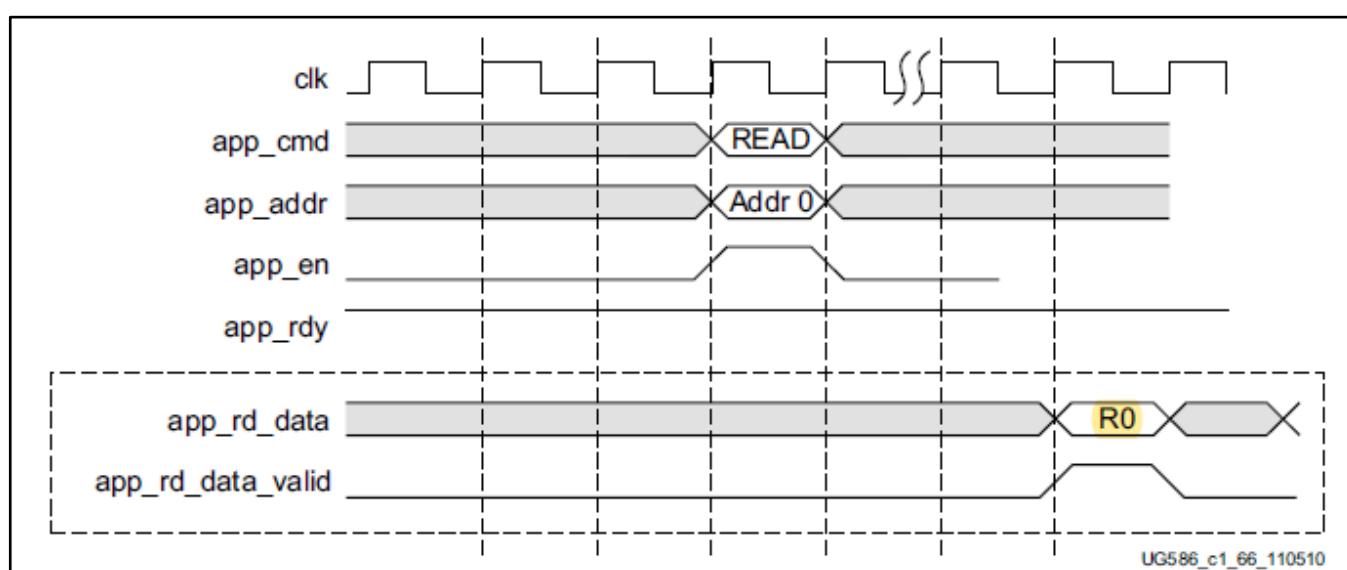
3.6.2 Write Timing

아래는 Write에 대한 Timing을 나타냅니다. 데이터를 Write하기 위해서는 커맨드의 한클락 전이나(2번), 커맨드와 동일 클락(1번), 커맨드후 2클락 이내(3번)에 데이터를 전송해야 합니다. 본서에서는 1번, 2번은 고려하지 않고 3번을 기준으로 코드를 구현합니다. Address/Command를 먼저 전송하고 뒤에 Data를 전송하는 구조를 사용합니다. 본서에서는 영상데이터를 처리하기 위한 Memory Controller를 구현하기 때문입니다. 영상데이터를 처리할 때에는 보통 라인 단위로 Read / Write 가 이루어 지며, Read / Write 를 번갈아서 사용하지 않습니다. 왜냐하면 Read / Write를 번갈아 사용하게 되면 Read / Write 간에 Switching Time이 필요하고 이로 인해 데이터 처리 속도가 떨어지기 때문입니다. 영상 데이터를 처리할 때에는 한라인을 모두 Write 한 후에 Read 하거나, 한라인을 모두 Read 한 후에 Write 하도록 구현합니다. 일부분은 코드를 구현하는 부분과 프레임 버퍼를 설계하는 부분에서 자세히 설명하도록 하겠습니다.



3.6.3 Read Timing

아래는 Read Timing입니다. Read 데이터는 커맨드 전송 후, app_rd_data_valid 가 ‘H’일 때 유효한 데이터가 전송됩니다. User Logic은 app_rd_data_valid가 “H”일 때 app_rd_data를 읽으면 됩니다.

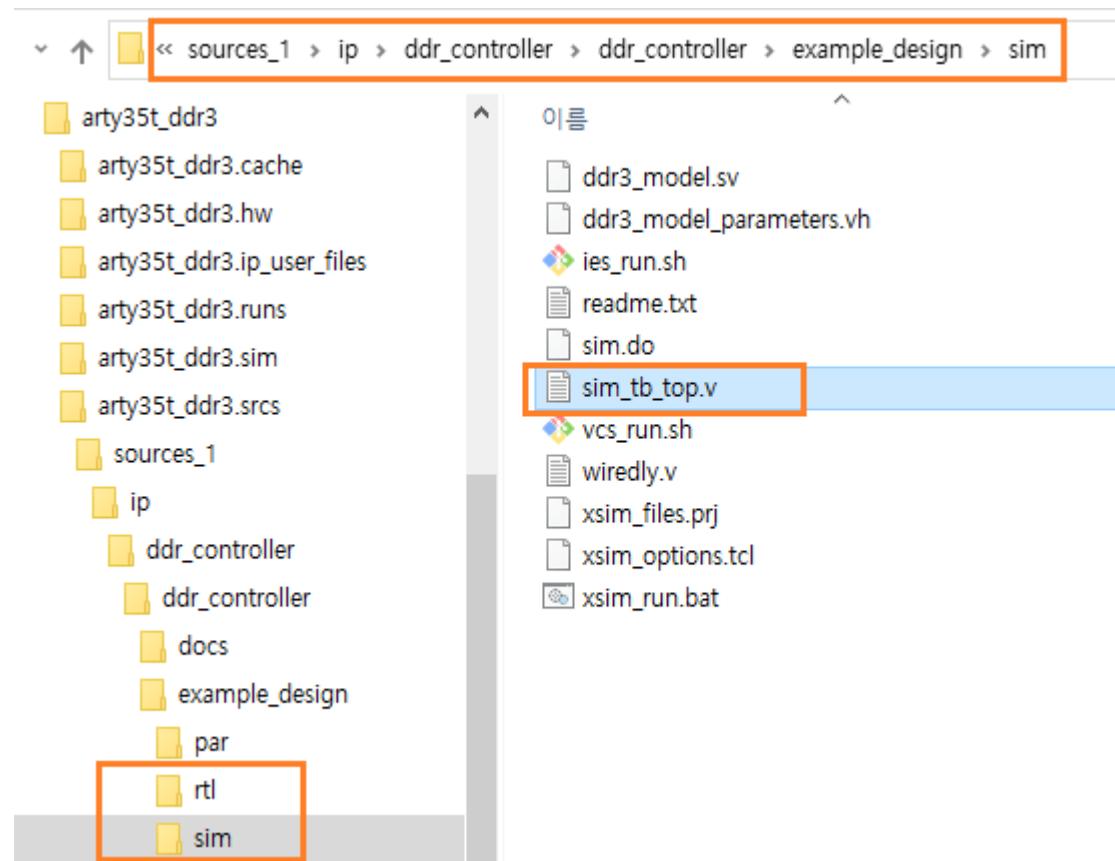


4. Simulation

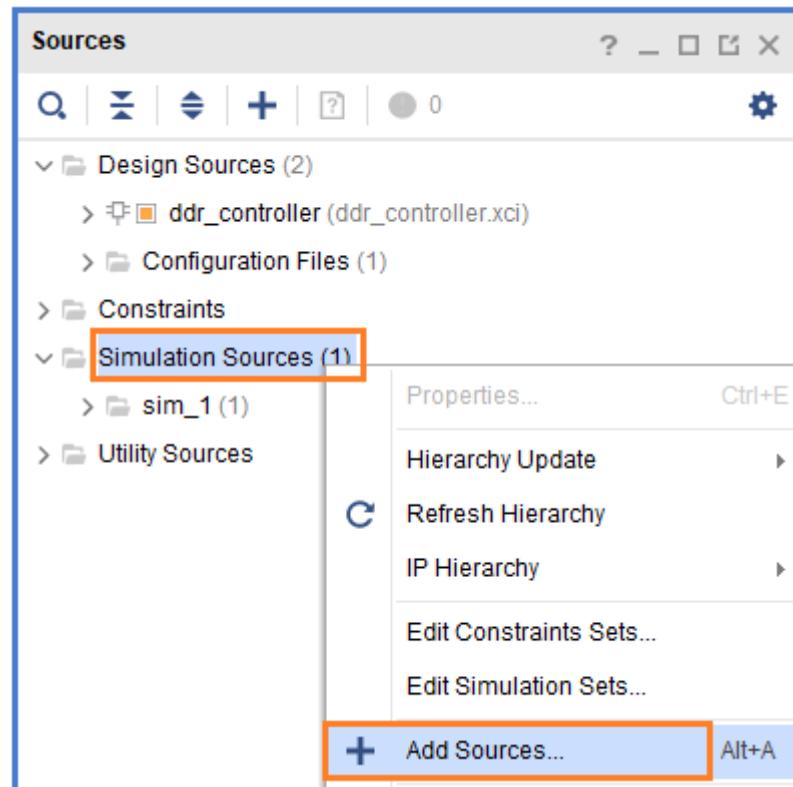
이번장에서는 생성된 Memory IP의 동작을 이해하기 위하여 Simulation을 진행합니다. Simulation 결과를 통하여 Memory IP의 동작을 이해하고 이를 통하여 User Logic을 구현합니다. 특히 영상 데이터 처리를 위한 범용으로 사용 가능한 User Logic을 구현하도록 하겠습니다.

4.1 Simulation 환경 설정

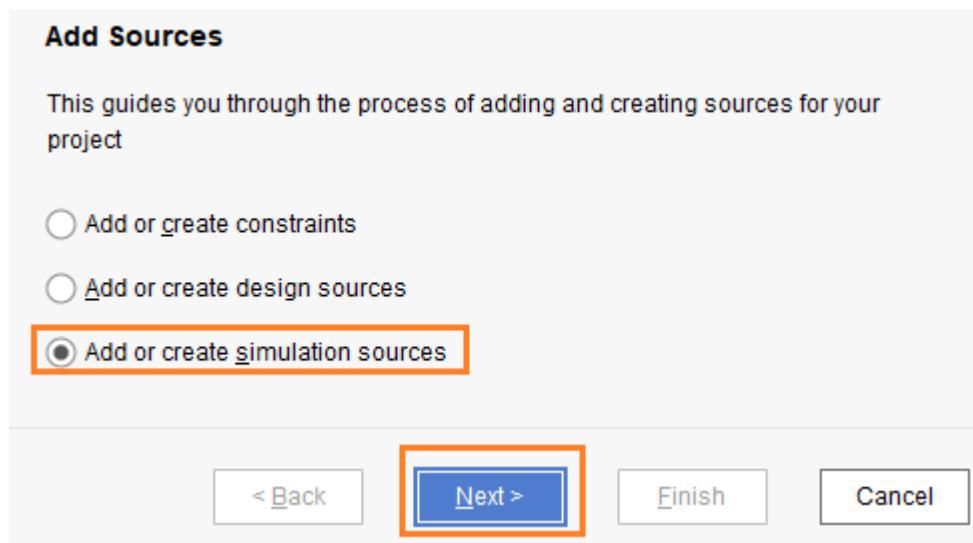
Simulation에 필요한 파일은 생성된 IP 폴더내의 “example_design\sim, rtl” 폴더에 있습니다. Top Module은 sim_tb_top.v입니다.



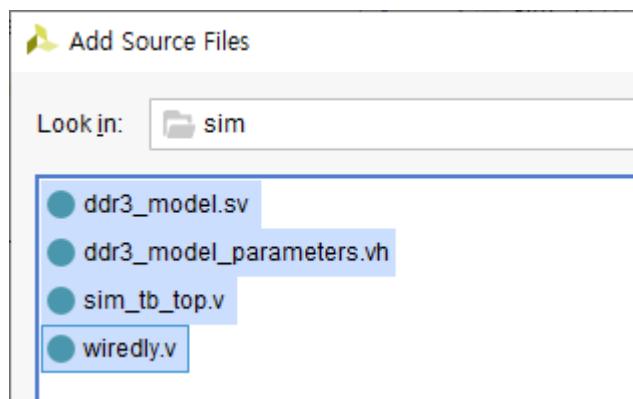
Sources 윈도에서 Simulation Sources - 우클릭 - Add Source...을 클릭합니다.

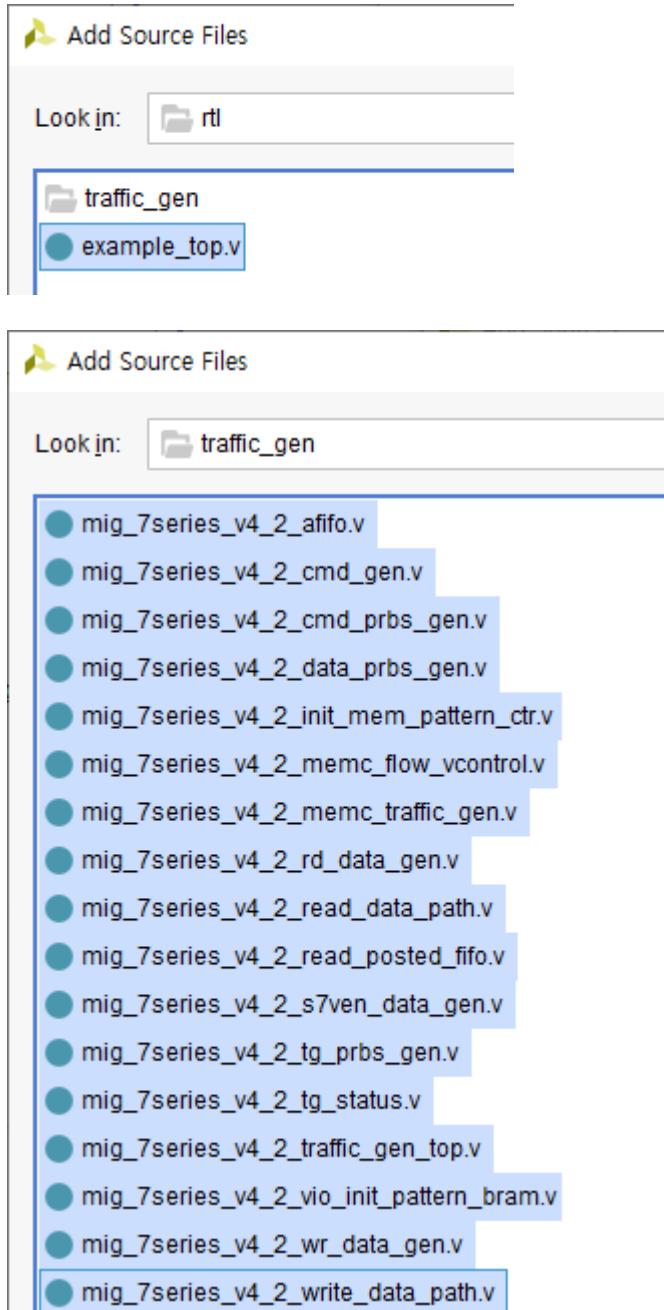


Add Sources 윈도에서 “Add or create simulation sources”을 선택하고 Next 버튼을 클릭합니다.



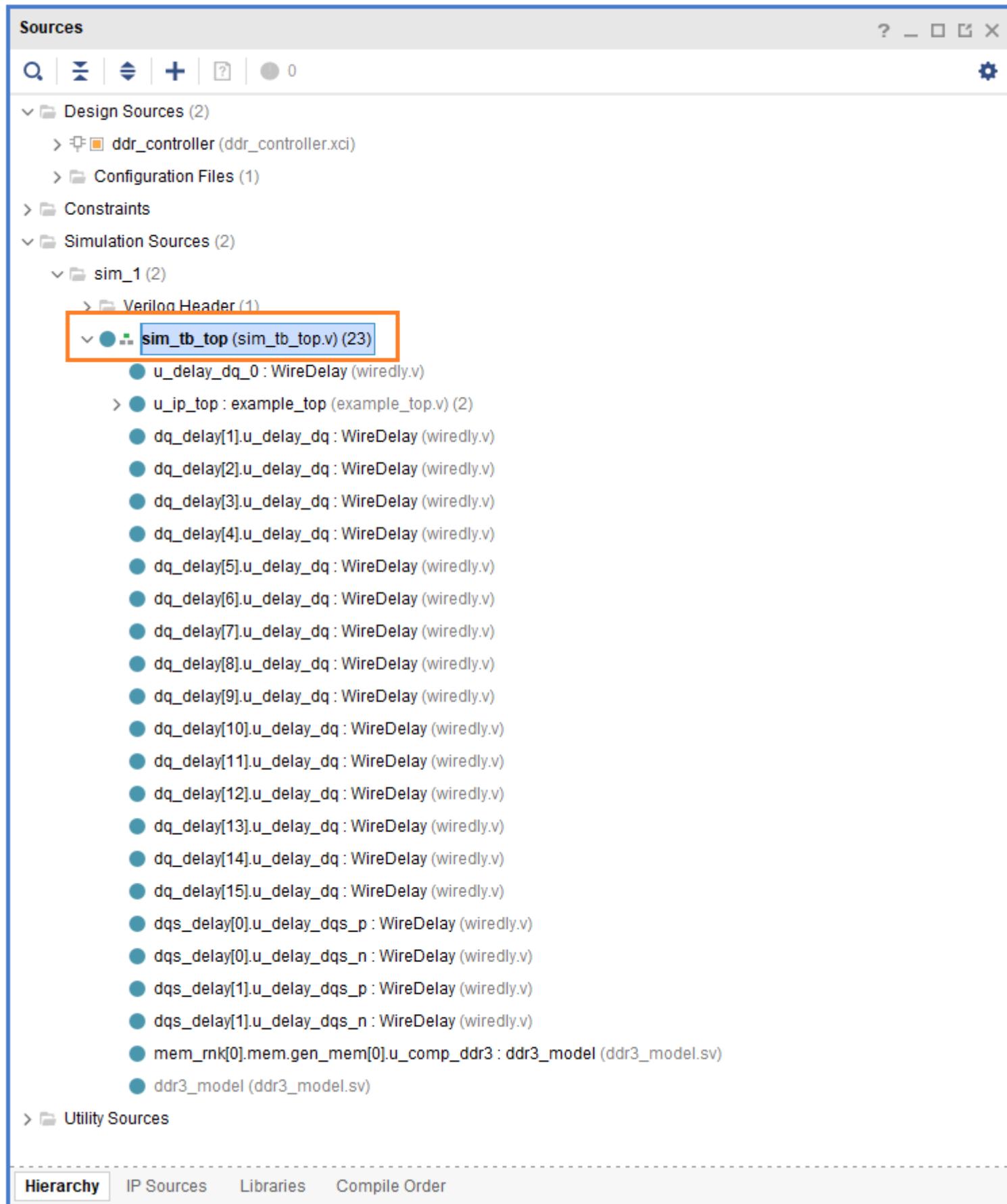
Add Files 버튼을 클릭해서 해당 파일들을 선택하여 추가합니다. 먼저 sim 폴더에 있는 모든 파일들을 선택하고 OK 버튼을 클릭해서 추가합니다. Add Files 버튼을 다시 클릭해서 rtl 폴더에 있는 모든 파일들을 선택하고 OK 버튼을 클릭합니다. rtl 폴더안의 모든 파일들을 선택할 때까지 이과정을 반복합니다.





모든 파일들이 추가되었으면 하단의 Finish 버튼을 클릭합니다.

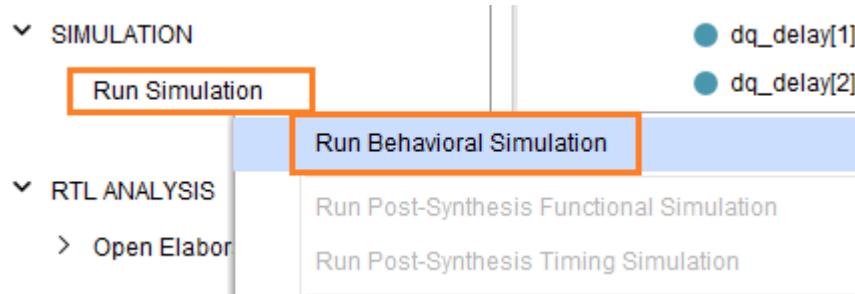
simulation에 필요한 모든 파일들이 추가되고 sim_tb_top 모듈이 Top Module로 지정되었습니다. 만일 sim_tb_top 모듈이 Top Module로 지정되지 않았다면 sim_tb_top 모듈을 선택하고 우클릭 후 “Set as Top”을 클릭하면 Top Module로 지정됩니다. (Top Module로 지정되면 모듈 옆에 아이콘 (■■) 이 표시됩니다)



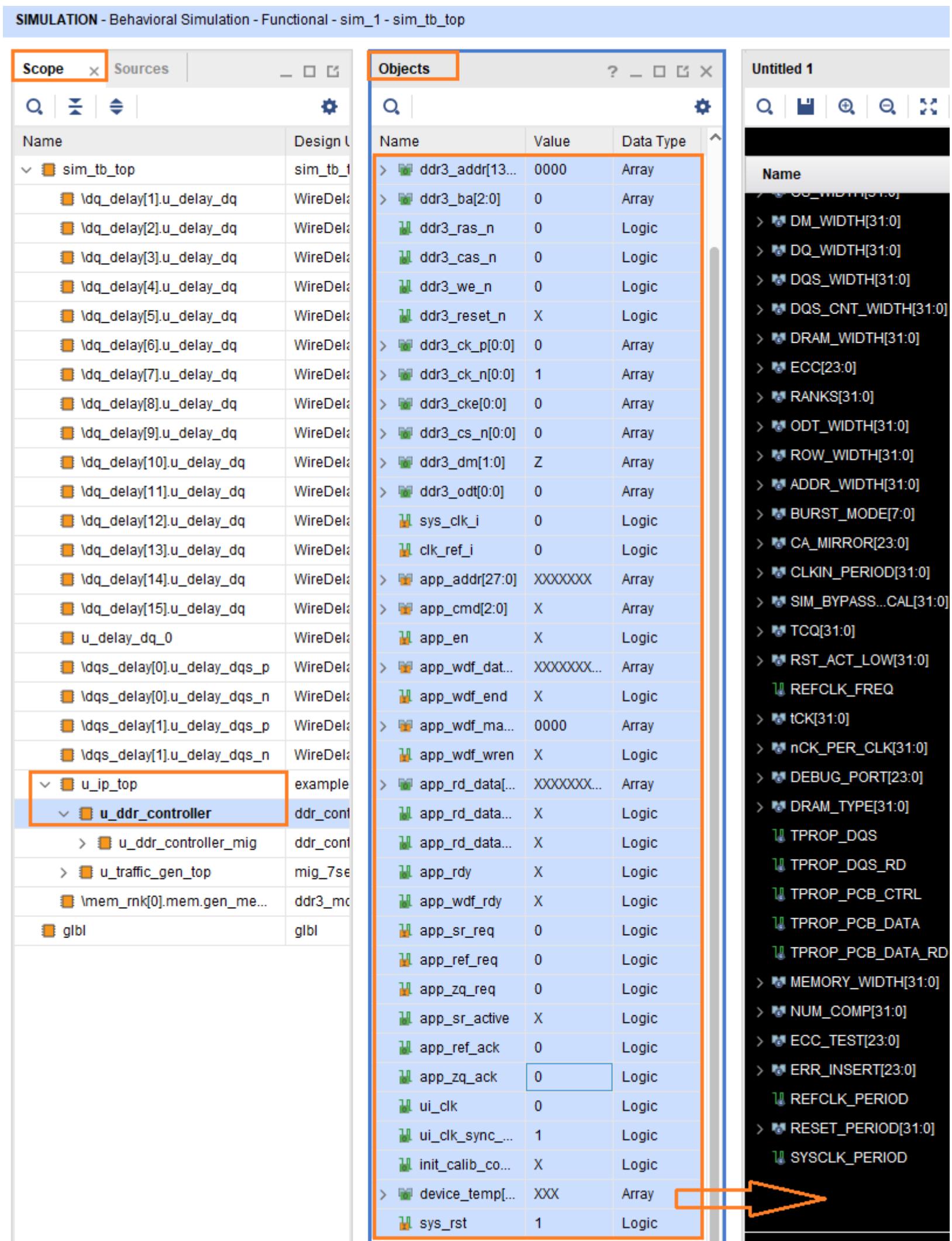
다음 절에서는 Simulation을 진행하고 그 결과를 확인하도록 합니다.

4.2 Simulation

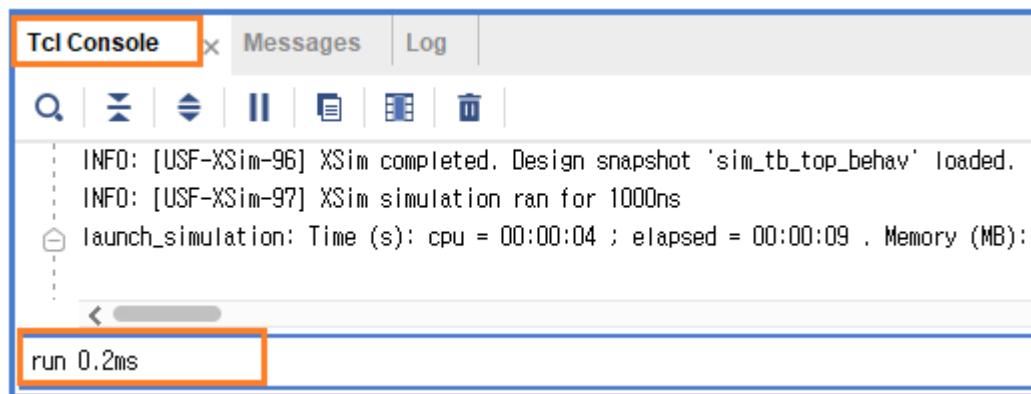
PROJECT MANAGER - SIMULATION - Run Simulation - Run Behavioral Simulation을 클릭해서 Simulation을 진행합니다.



Simulation이 진행되고 Wave 윈도가 나타납니다. Wave 윈도에는 기본적으로 Top Module (sim_tb_top)의 신호들이 선택되어 있습니다. 여기에 우리에게 필요한 신호들을 추가합니다. Scope 윈도에서 sim_tb_top - u_ip_top - u_ddr_controller를 선택하고 Object 윈도에 나타난 모든 신호들을 선택해서 마우스로 드래그해서 기존의 신호 제일 하단에 신호들을 추가해 줍니다. 또는 u_ddr_controller - 우클릭 - Add to Wave Window 클릭합니다.



Tcl Console 창에 “run 0.2ms”을 입력 후 엔터를 입력해서 Simulation을 진행합니다.



0.2ms simulation을 진행하는데 생각보다 많은 시간이 소요됩니다. DDR3 메모리는 고속으로 동작하기 때문에 timescale 을 매우 작을 값으로 설정하고 simulation을 진행했기 때문입니다. “sim_tb_top.v” 파일을 열어보면 timescale 이 “1ps/100fs”로 설정되어 있는 것을 알 수 있습니다.

```

74
75 `timescale 1ps/100fs
76
77 module sim_tb_top;
78

```

Simulation 진행중 대략 176.24us 에서 Simulation이 끝나는 됩니다. 이는 메모리 초기화가 끝나고 일정시간 Memory Read/Write 테스트가 끝나면 Simulation을 Finish 하라는 명령이 “sim_tb_top.v”에 포함되어 있기 때문입니다. 중지 없이 계속해서 Simulation을 진행하고 싶으면 “\$finish” 부분을 주석으로 처리하고 진행하면 됩니다. 176us 정도 simulation이 진행되면 우리가 목표로 하는 Memory IP의 User Interface 신호들을 확인하고 동작을 확인할 수 있습니다.

```

586         disable calib_not_done;
587         $finish;
588     end

```

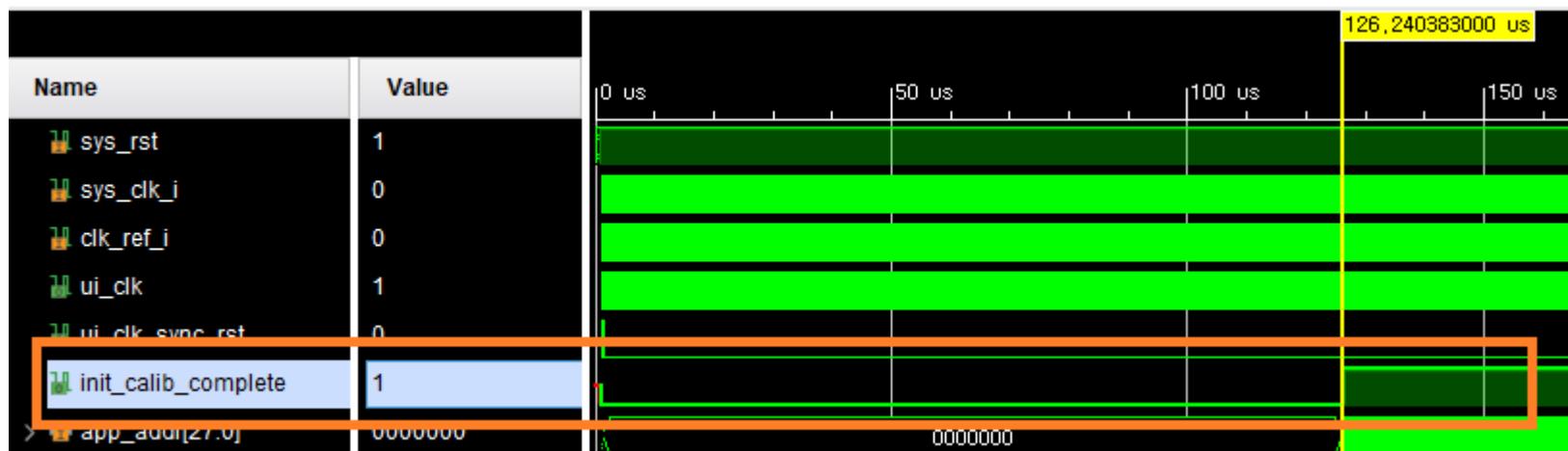
4.3 Simulation 결과 확인

이번 장에서는 Simulation 결과를 통하여 User Interface 신호들이 어떻게 동작하는지를 확인합니다. 이를 바탕으로 다음장에서는 영상 데이터 처리를 위한 범용으로 사용할 수 있는 User Interface Logic을 구현합니다.

Wave 윈도에서 sim_tb_top.v 파일이 열려 있으면 닫고 파형을 확인합니다.

4.3.1 init_calib_complete

init_calib_complete 신호는 메모리 초기화 설정이 완료되었음을 알려주는 신호입니다. 파형을 보면 약 126.24us에서 메모리 설정이 완료되었음을 알 수 있습니다.



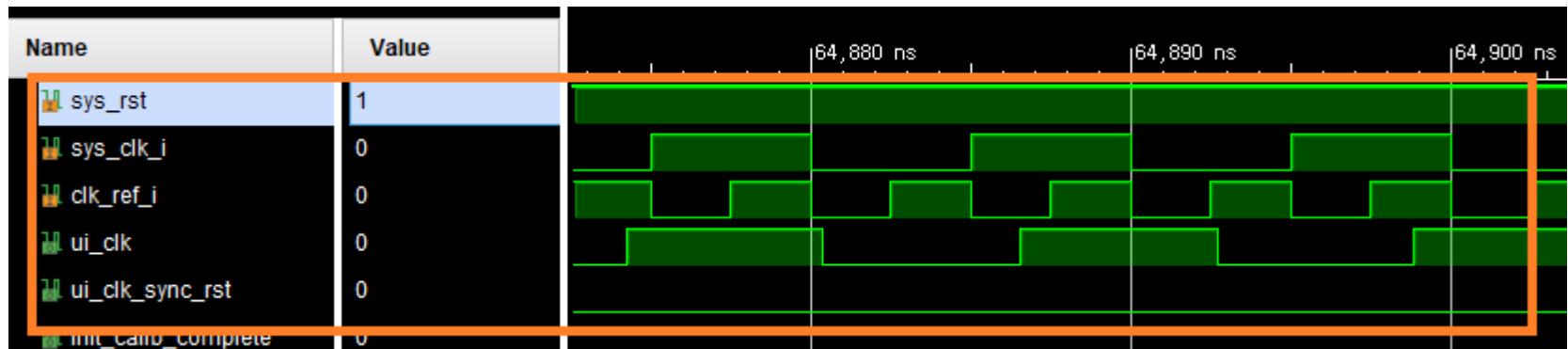
4.3.2 reset, clock

ddr_controller (생성된 Memory IP)의 신호들 중에 reset, clock 신호들이 있습니다. 이 신호들이 어떻게 동작하는지를 이해하는 것은 매우 중요한 일입니다.

- ✓ sys_rst : ddr_controller의 Main Reset 신호입니다. Memory IP 생성시 C2 핀을 통해 입력해 주었습니다.
- ✓ sys_clk_i : Memory IP 생성시, No Buffer로 설정하고 내부에서 생성된 100Mhz를 입력해 주기로 하였습니다. sim_tb_top 모듈에서는 100Mhz Clock 생성 후 입력해 주었습니다.
- ✓ clk_ref_i : Memory IP에서 사용하는 Reference Clock으로 200Mhz를 사용합니다. Memory IP 생성시 No Buffer로 설정하였고, sim_tb_top 모듈에서는 200Mhz Clock 생성후 입력해 주었습니다.
- ✓ ui_clk_sync_rst : User Interface Logic에서 사용하는 Reset 신호입니다. 이는 ddr_controller에서 생성해서 출력해 주고 있습니다. High Active 신호입니다. User Interface Logic에서 사용할 때에는 Low Active로 변환해서 사용합니다.
- ✓ ui_clk : User Interface Logic에서 사용하는 Clock입니다. 이는 ddr_controller에서 생성해서 출력해 주고 있습니다. 대략적으로 약 81.2Mhz 정도 됩니다.

여기에서 중요한 것은 User Interface Logic을 구성할 때, ui_clk_sync_rst, ui_clk을 사용해야 한다는 것입니다. 그렇게 해야만 Memory IP와 User Interface Logic 간에 동기를 맞출 수 있습니다.

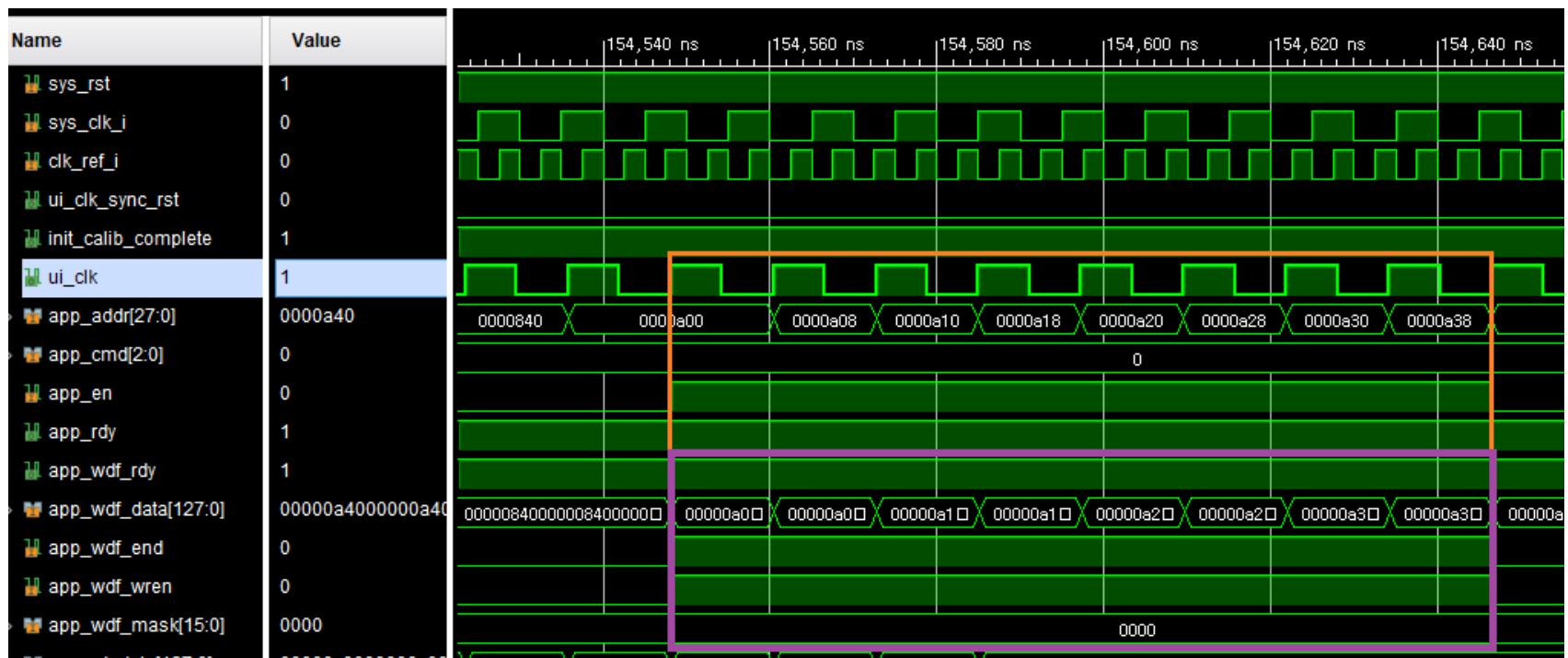
아래는 각 신호들을 보여주고 있습니다.



sys_clk_i 는 주기가 10ns (100 Mhz), clk_ref_i 는 5ns (200 Mhz), ui_clk은 12.308ns (81.24 Mhz) 입니다. sys_rst 는 Low Active, ui_clk_sync_rst 는 High Active 입니다.

4.3.3 Write Timing 분석

이번에는 write timing을 분석합니다. Read / Write 을 반복하고 있기 때문에 임의의 Write 부분을 확대해서 세부 신호들을 확인합니다. 0.154us 근처의 신호를 확인합니다.



Address / Command 신호를 확인합니다. 주황색 부분입니다. app_rdy 신호가 “H”일 때, app_addr, app_cmd(read : 0), app_en 신호를 출력하고 있습니다. 총 8번에 걸쳐 address는 8씩 증가하였습니다. Write 동작은 128bits 데이터를 8번 전송합니다. Write에 사용되는 데이터(app_wdf_data[127:0])은 현재 address 값을 4번씩 반복하여 데이터로 구성하였습니다. 즉 0x0000a00 주소에는 0x0000a00 을 4번 반복해서 데이터를 생성하였습니다.

- ✓ app_addr : 0x0000a00 -> app_wdf_data : 0x00000a00_00000a00_00000a00_00000a00
- ✓ app_addr : 0x0000a08 -> app_wdf_data : 0x00000a08_00000a08_00000a08_00000a08
- ✓ 마지막으로 app_addr : 0x0000a38 -> app_wdf_data : 0x00000a38_00000a38_00000a38_00000a38

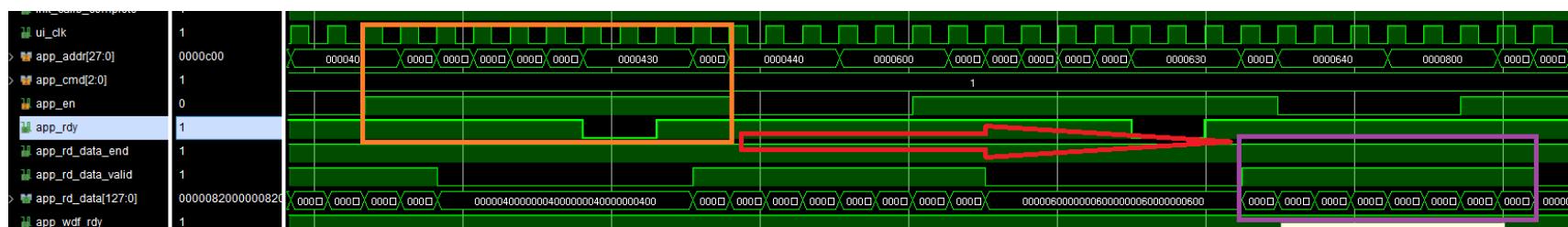
모든 신호들은 (app_addr ~ app_wdf_mask) ui_clk에 동기화 되어 있음을 주의하시길 바랍니다. 또한 ui_clk의 positive edge에 동기화 되어 있습니다. 우리는 다음장에서 User Interface Logic을 구성할 때 다음과 같은 기본 Write 동작을 정의해서 구현합니다.

- ✓ Write 동작의 기본은 8번 연속해서 Write 합니다.
- ✓ Address는 8씩 증가하고, Data는 128bits로 구성합니다.
- ✓ Address / Cmd는 app_rdy 가 High 일 때 출력됩니다.
- ✓ Data는 app_wdf_wren, app_wdf_end 신호가 High 일 때 출력됩니다.
- ✓ app_wdf_mask는 사용하지 않습니다. 0으로 출력합니다.
- ✓ 모든 신호는 ui_clk 의 Positive Edge 에서 동작합니다.

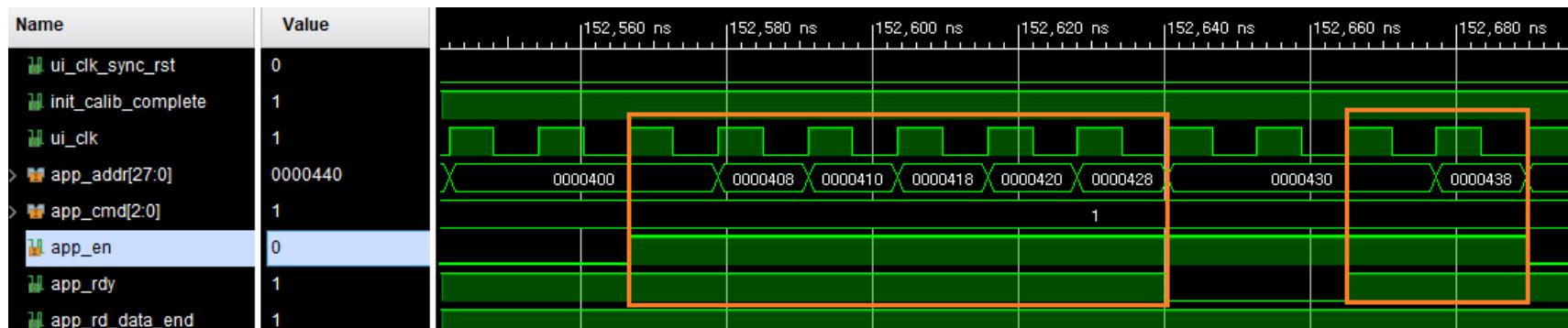
세부적인 구현내용은 다음장에서 좀더 상세히 설명하도록 하겠습니다.

4.3.4 Read Timing 분석

Read Timing 도 임의의 Read 구간을 확대해서 세부 신호들을 확인합니다. 0.152us 근처에서 Read Timing을 확인합니다.

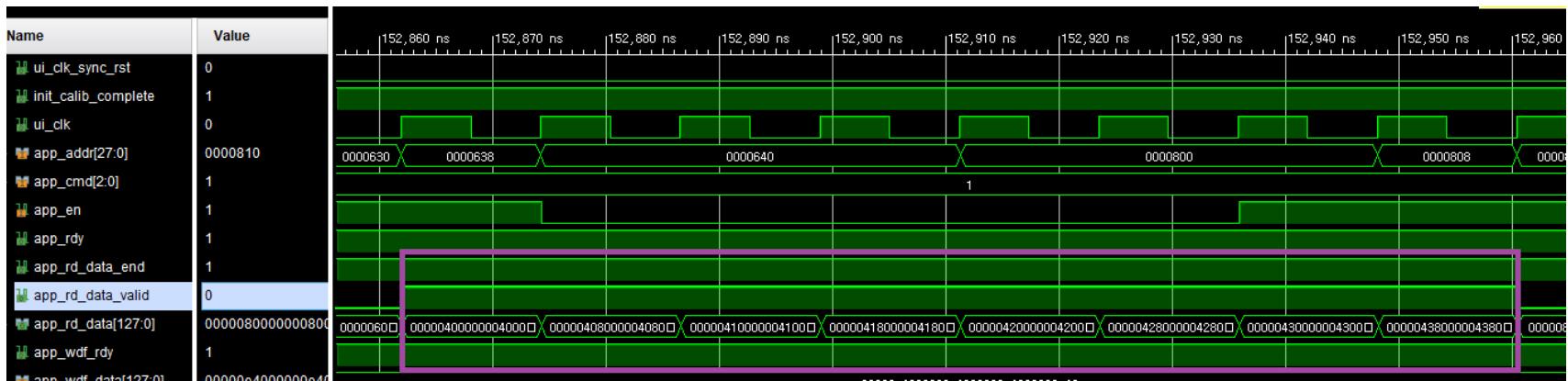


주황색으로 표시된 부분이 Read를 위한 Address / Command입니다. app_rdy 인 구간에서 8번 read address / cmd를 전송하였습니다. 보라색으로 표시된 부분은 Read Data가 나오는 부분입니다. 좀더 신호를 확대해서 살펴보겠습니다. 아래는 address / cmd 부분을 확대하여 보여줍니다.



app_addr은 0x0000400 ~ 0x0000438까지 8번 전송합니다. app_cmd는 1입니다. app_addr, app_cmd, app_en 신호는 app_rdy 신호가 High 인 구간에서만 유효한 신호로 전달됩니다. app_rdy=1, app_en=1인 구간이 총 8번 있습니다.

다음은 Data Read 구간을 확대하여 보여줍니다.



app_rd_data_valid가 High 인 구간이 총 8-clock이고 이 구간에서의 app_rd_data가 유효한 데이터입니다. 순서대로 0x00000400_00000400_00000400_00000400 ~ 0x00000438_00000438_00000438_00000438 까지 데이터가 나오고 있습니다. Address에서 설정된 값(0x0000400 ~ 0x0000438)과 동일한 값이 나오고 있습니다. 이는 Write 시에 Address와 동일한 값으로 Write 했기 때문입니다. Read Timing과 동일하게 모든 신호들은 ui_clk의 Positive Edge에 동기화되어 있습니다.

User Interface Logic을 구현하기 위한 Read 기본 동작을 정의합니다.

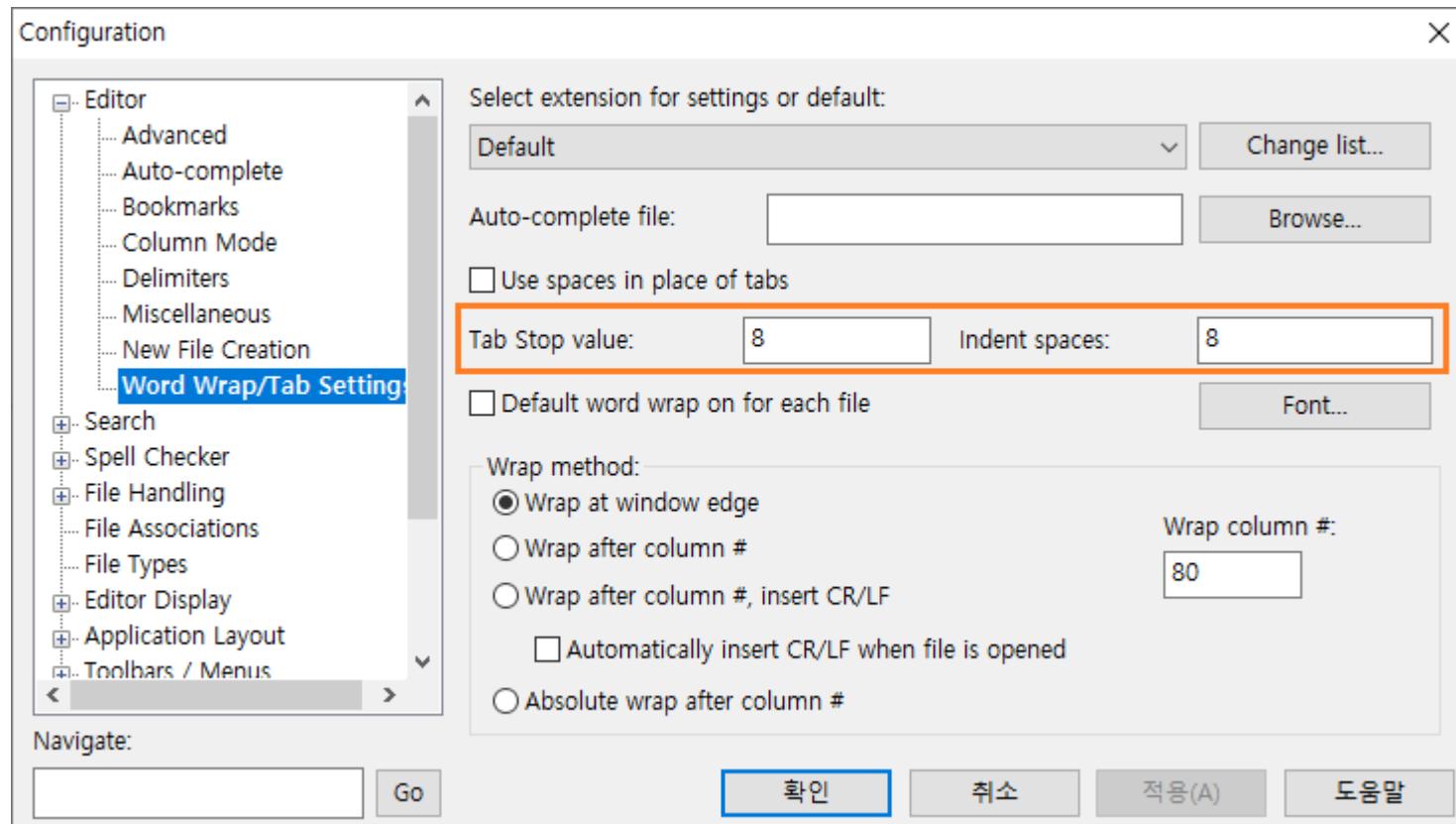
- ✓ Read 동작의 기본은 8번 연속해서 Read 합니다.
- ✓ Address는 8씩 증가하고, Data는 128bits로 구성합니다.
- ✓ Address / Cmd는 app_rdy 가 High 일 때 출력됩니다.
- ✓ Data는 app_rd_data_valid가 High 일 때 유효한 데이터입니다.
- ✓ 모든 신호는 ui_clk의 Positive Edge에서 동작합니다.

User Interface Logic 구성을 위한 Memory IP의 동작 확인이 끝났습니다. 다음 장에서는 이장에서의 내용을 바탕으로 영상 데이터 처리를 위한 범용으로 사용 가능한 User Interface Logic을 구현합니다. 또한 마지막에는 구현된 User Interface Logic을 이용하여 영상이미지 데이터 처리를 위한 Frame Buffer를 구현합니다.

5. User Interface Logic 구현

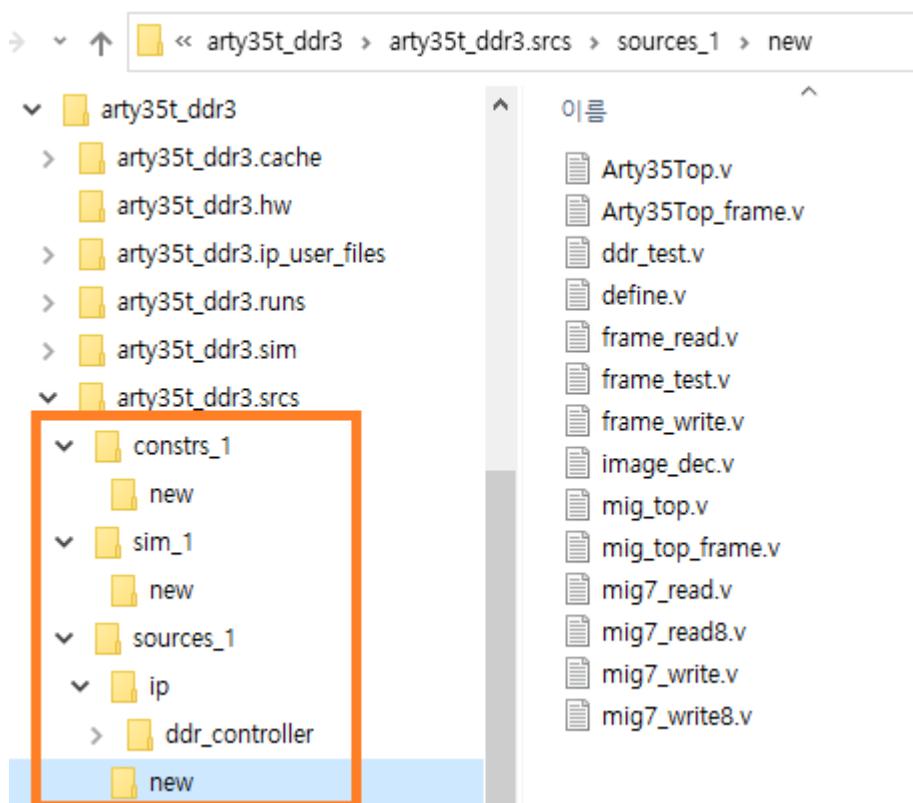
이번 장에서는 4장에서 simulation을 통하여 Memory Controller의 동작을 확인한 것을 바탕으로 User Interface Logic을 구현합니다. 다음장에서는 이를 바탕으로 영상데이터 처리를 위한 프레임 버퍼를 구현하도록 하겠습니다.

저자는 코드 편집을 위한 에디터 툴로 울트라 에디터를 사용합니다. 탭을 맞추기 위하여 탭 설정을 아래와 같이 하면 코드가 보기 좋게 정렬됩니다.



5.1 개요

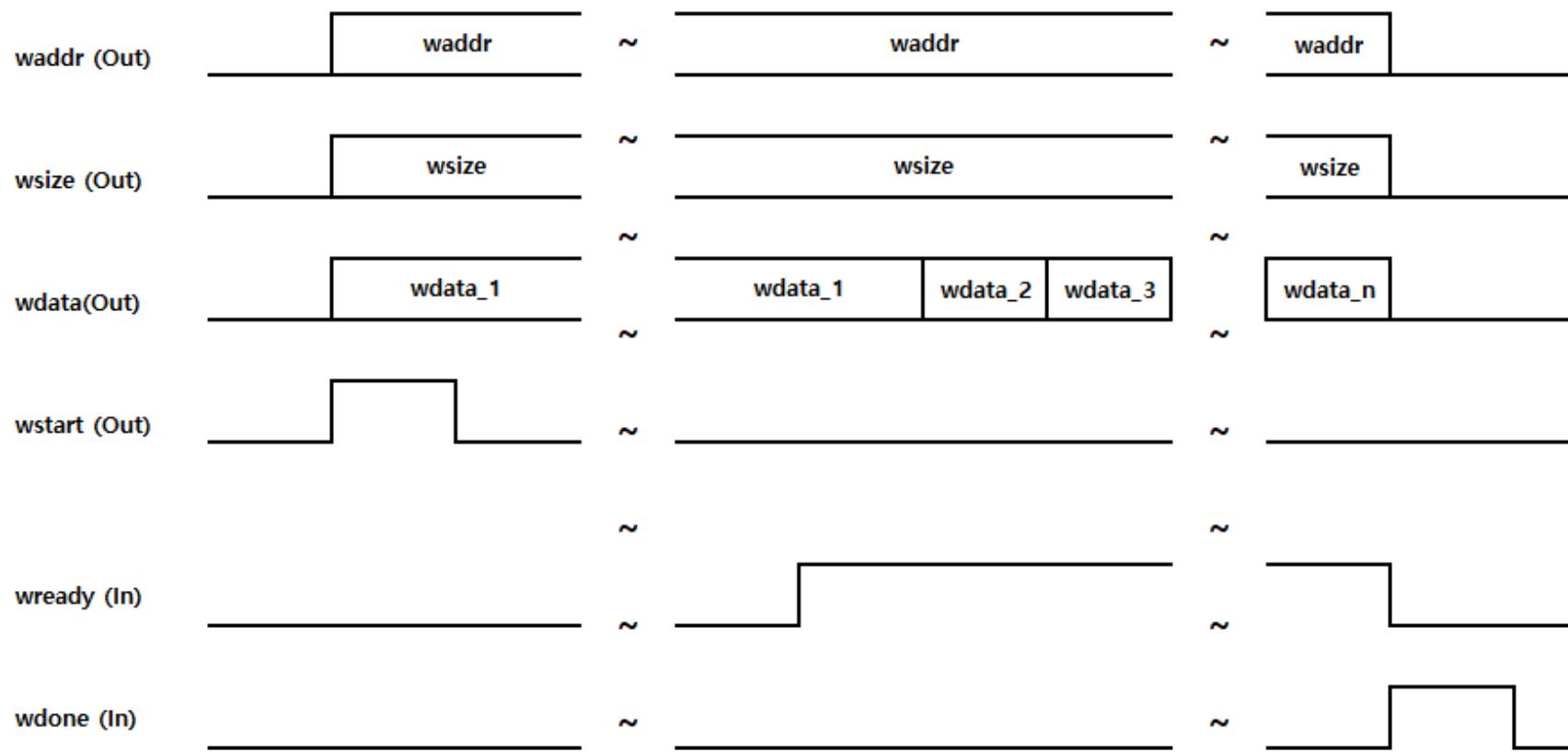
코드를 분석하기 전에 제공받은 자료에서 파일들을 추가하도록 하겠습니다. 제공받은 자료의 arty35t_ddr3.srcs 폴더에서 “consts_1”, “sim_1” 폴더를 복사하여 자신이 생성한 프로젝트의 arty35t_ddr3.srcs 폴더에 추가합니다. IP를 생성했기 때문에 sources_1 폴더는 생성되어 있을 것입니다. 제공받은 자료의 “arty35t_srcs\sources_1” 폴더에서 new 폴더를 복사하여 자신이 생성한 프로젝트의 같은 패스에 추가합니다. “consts_1\new” 폴더에는 xdc 파일이 있고, “sim-1\new” 폴더에는 Test bench 파일이 있고, “sources_1\ip” 폴더에는 생성한 IP들이 있고, “sources_1\new”에는 verilog 소스 코드들이 있습니다. 복사 후 폴더 구조는 아래와 같습니다.



Read, Write 모듈에 대한 스펙을 정의합니다. 이 부분은 사용자가 정의할 수 있습니다. 그러나 최대한 간단하게 구성하는 것을 추천합니다. 본서에서 정의된 것은 저자의 경험에 의해 최대한 간단하게 구성한 것입니다. Memory Controller가 익숙해지면 자신만의 스펙을 정의해서 구현하는 것도 많은 도움이 됩니다. 우선은 저자가 구현한 것을 바탕으로 진행하시길 바랍니다.

5.1.1 Write Timing 정의

아래 그림은 Write Timing 입니다.

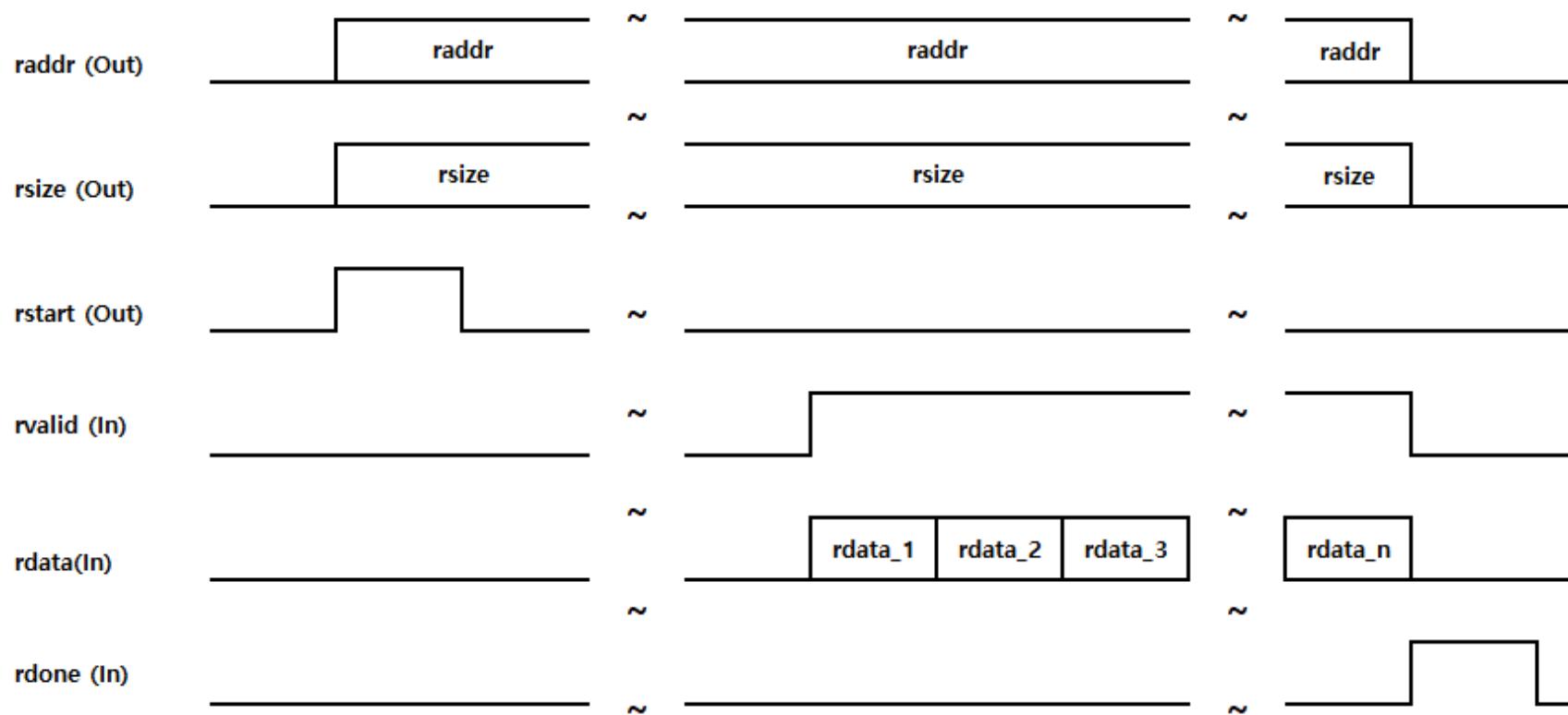


User가 DDR에 데이터를 Write 하기 위해서는 Write 하기 위한 시작 주소(waddr), Write 하려는 Size (wszie), Write 하려는 첫번째 데이터(wdata_1)을 준비한 후에 wstart 신호를 1-clock(ui_clock) 동안 “H”로 만들어 줍니다. 그리고 wready 신호가 “H” 가 될 때마다 다음 Write 할 데이터를 전송합니다. 데이터가 모두 전송이 되면 wdone 신호가 High 가 됩니다.

| port | size | in/out | description |
|--------|---------|--------|---|
| waddr | [27:0] | Output | write start address |
| wszie | [9:0] | Output | write size, 8 x 128bits 단위 예를 들면, 1로 설정 시 8 x 128 bits를 write 합니다. |
| wdata | [127:0] | Output | write data, wready 가 High가 되면 순차적으로 데이터를 전송합니다. |
| wstart | [0] | Output | write start strobe |
| wready | [0] | Input | 데이터를 write 할 준비가 되었음을 알려줍니다. wready가 High가 되면 순차적으로 데이터를 전송합니다. |
| wdone | [0] | Input | wszie 에 설정한 데이터가 모두 write 되었음을 알려줍니다. |

5.1.2 Read Timing 정의

아래 그림은 Read Timing 입니다.

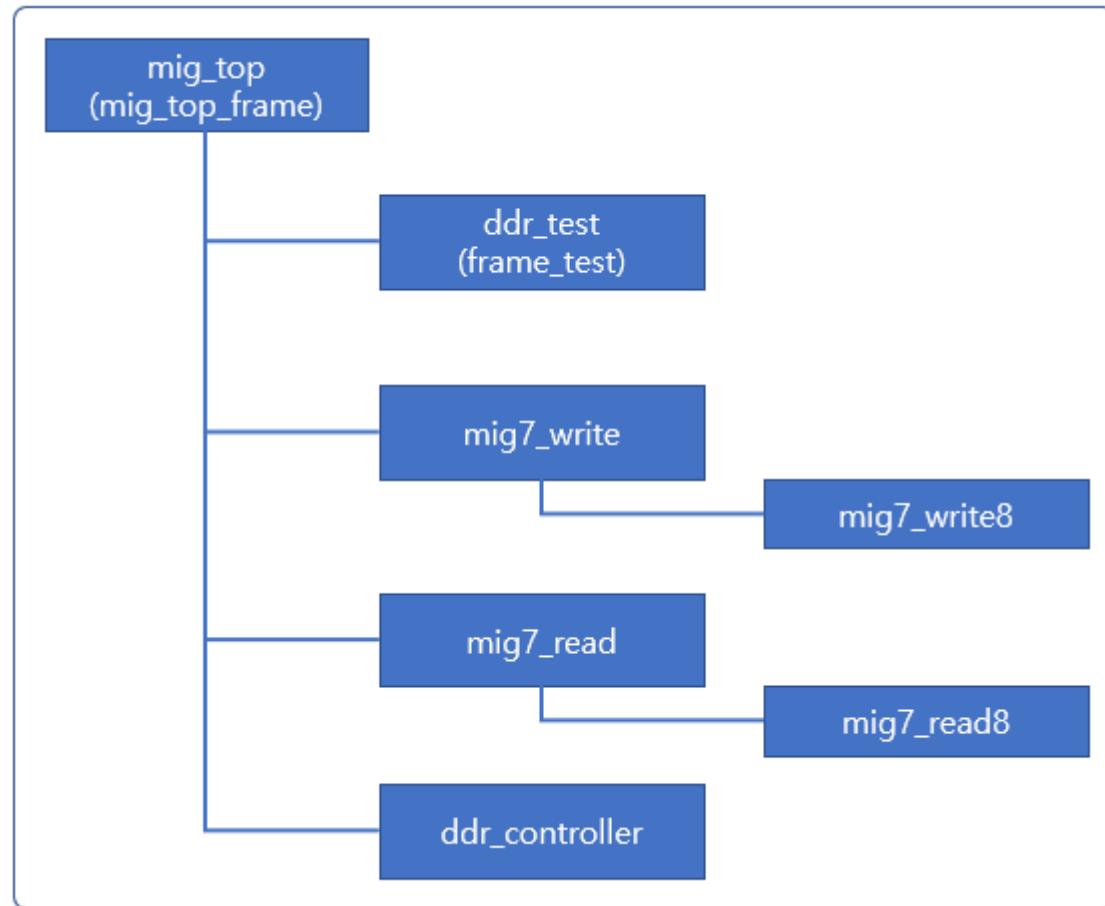


User가 DDR에서 데이터를 Read 하기 위해서는 Read 하기 위한 시작 주소(raddr), Read 하려는 Size (rszie)을 준비한 후에 rstart 신호를 1-clock(ui_clock) 동안 “H”로 만들어 줍니다. 그리고 rvalid 신호가 “H” 가 될 때마다 rdata 값을 읽으면 됩니다. 데이터를 모두 읽으면 rdone 신호가 High 가 됩니다.

| port | size | in/out | description |
|--------|---------|--------|--|
| raddr | [27:0] | Output | read start address |
| rszie | [9:0] | Output | read size, 8 x 128bits 단위 예를 들면, 1로 설정 시 8 x 128 bits를 read 합니다. |
| rstart | [0] | Output | read start strobe |
| rvalid | [0] | Input | read 데이터가 유효함을 알려줍니다. ddr은 데이터를 read 후에 rvalid 신호와 함께 rdata에 데이터를 전송합니다. |
| rdata | [127:0] | Input | read 데이터가입니다. rvalid 가 High 일 때 rdata를 읽으면 됩니다. |
| rdone | [0] | Input | rszie 에 설정한 데이터가 모두 read 되었음을 알려줍니다. |

5.1.3 코드 구조

이번 절에서는 코드 구조에 대해서 설명합니다. 아래는 코드 구조를 보여줍니다.



`mig_top` 은 Top Module 입니다. `ddr_test` 는 사용자가 데이터를 read/write 하기 위한 코드입니다. `ddr_test` 는 5.1.1, 5.1.2에서 설명한 Timing 대로 메모리에 Access(read/write) 하는 코드를 구현합니다. `mig7_write`, `mig7_write8` 모듈은 메모리에 write 하기 위한 코드입니다. 사용자가 메모리에 write 하기 위하여 5.1.1에 설명한 대로 신호를 주면, 실제로 메모리에 데이터를 write 하는 코드입니다. `mig7_write8`은 8x128bits 단위의 write 모듈이고, `mig7_write`는 `mig7_write8`을 이용하여 사용자가 설정한 size 만큼 메모리에 write 하는 코드입니다. `mig7_read`, `mig7_read8` 모듈은 메모리에서 read 하기 위한 코드입니다. 사용자가 메모리에 read 하기 위해서 5.1.2에 설명된 대로 신호를 주면, 실제로 메모리에서 데이터를 read 해서 넘겨주는 코드입니다. `mig7_read8`은 8x128bits 단위의 read 모듈이고, `mig7_read`는 사용자가 설정한 size만큼 read 하는 코드입니다.

`mig_top`, `ddr_test` 는 임의의 데이터를 메모리의 모든 영역에 Write 하고, Read 한 후에 Verification까지 수행하는 코드입니다. 임의의 데이터는 아래와 같이 구성됩니다.

- ✓ 첫번째 128 bits : 0x33333333_33333333_33333333_33333333
- ✓ 두번째 128 bits : 0xCCCCCCCC_CCCCCCCC_CCCCCCCC_CCCCCCCC
- ✓ 세번째 128 bits : 0x55555555_55555555_55555555_55555555
- ✓ 네번째 128 bits : 0xAFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF

위 데이터가 반복해서 Write 됩니다.

메모리의 모든 영역을 위 데이터로 Write 한 후에, 다시 메모리의 모든 영역을 Read 하면서 Read 된 데이터가 Write 한 데이터와 동일한지를 확인해서 에러를 검출합니다. 최종적으로 에러가 없으면 녹색 LED를 점등하고, 에러가 있으면 빨간색 LED를 점등합니다.

mig_top_frame, frame_test 는 영상데이터(640x360 데이터)를 위한 프레임 버퍼를 구현하는 모듈입니다. 640x360 사이즈의 영상 데이터를 메모리에 저장 후, 다시 읽어서 영상 데이터 출력을 구현합니다. 이 부분은 HW로 구현하지는 않고 Simulation 결과로 확인합니다. 추후 적당한 이미지 센서 모듈이 구해지면 HW 검증까지 진행할 예정입니다.

코드 구조에서 알 수 있듯이, 범용 Memory Controller 를 구현하면 단지 사용자 코드 (ddr_test, frame_test) 부분만 수정하면 나머지 코드들은 (mig7_write, mig7_write8, mig7_read, mig7_read8) 수정할 필요 없이 그대로 사용 가능합니다. 이 코드들은 현업에서 그대로 사용가능한 코드들입니다. 다음 절부터 코드를 모듈 별로 구현하고 Simulation을 통하여 검증하는 과정을 진행합니다. 코드 내용이 가벼운 내용은 아닙니다. 저 또한 코드 구현하고, Simulation으로 확인하고 수정하고 또 Simulation 확인하고, 이러한 과정을 여러 번 반복해서 오류를 수정하고, Delay를 맞추고 하는 일을 하였습니다. 내용이 쉽지는 않지만 인내심을 가지고 여러분 코드를 분석하시고, Simulation으로 결과를 확인하여 충분히 본인의 코드로 만드시길 바랍니다. 본서에서 소개되고 제공되는 코드는 FPGA 개발 15년 이상(토탈 개발경험 25년)의 경험자가 구현한 내용입니다. 본 코드를 레퍼런스 삼아 앞으로 Verilog 개발자가 되시는 것도 좋을 것으로 생각됩니다. 그럼, 다음절부터 코드를 구현하고 분석하고 simulation을 통하여 동작을 확인하도록 하겠습니다.



5.2 Write Module 구현

메모리에 Write 하는 코드를 구현합니다. 여기에서 사용하는 reset, clock 은 ddr_controller 에서 출력되는 ui_clk_sync_rst, ui_clk 을 사용합니다. reset은 ~ui_clk_sync_rst 를 사용합니다.

5.2.1 mig7_write8 모듈 구현

mig7_write8 모듈은 128bits 씩 총 8번 Write 를 진행합니다. mig7_write 모듈은 “4.3.3 Write Timing 분석”의 내용을 코드로 구현하였습니다. “4.3.3. Write Timing 분석”을 참조해서 코드를 분석하시길 바랍니다.

```

23  module mig7_write8(
24      input      reset      ;          // ui_clk_sync_rst form ddr controller
25      input      mclk      ;          // ui_clk from ddr controller
26
27      output     wstart    ;
28      output     [27:0]  waddr   ;
29      output     [127:0] wdata   ;
30
31      output     wready   ;
32      output     wdone    ;
33
34      output     app_addr  ;
35      output     app_cmd   ;
36      output     app_en    ;
37      output     app_rdy   ;
38      output     app_wdf_data;
39      output     app_wdf_wren;
40      output     app_wdf_end ;
41      output     app_wdf_mask;
42      output     app_wdf_rdy;
43  );
44
45  input      reset      ;
46  input      mclk      ;
47
48  input      wstart    ;
49  input      [27:0]  waddr   ;
50  input      [127:0] wdata   ;
51  output     wready   ;
52  output     wdone    ;
53
54  output     [27:0]  app_addr  ;
55  output     [2:0]   app_cmd   ;
56  output     app_en    ;
57  input      app_rdy   ;
58  output     [127:0] app_wdf_data;
59  output     app_wdf_wren;
60  output     app_wdf_end ;
61  output     [15:0]  app_wdf_mask;
62  input      app_wdf_rdy;

```

- ✓ reset : ddr_controller에서 출력되는 ui_clk_sync_rst 신호를 inverting 해서 사용합니다.
- ✓ mclk : ddr_controller에서 출력되는 ui_clk 을 사용합니다.
- ✓ wstart : User Interface 모듈(ddr_test or frame_test) 에서 생성하는 write start strobe 입니다. 이 신호가 “H”가 되면 write 를 시작합니다.
- ✓ waddr[27:0] : 메모리에 write 하기 위한 Start Address 입니다.
- ✓ wdata[127:0] : 메모리에 write 하기 위한 데이터 입니다. 첫번째 데이터는 wstart 명령을 출 때 입력되고, 그 다음 부터는 wready 신호가 High 일 때 데이터를 넘겨 줍니다.

- ✓ wready : 메모리에 데이터를 write 하기 위한 준비가 되었음을 User Interface에 알려줍니다.
- ✓ wdone : 모든 데이터가 write 되었음을 User Interface에 알려 줍니다.
- ✓ app_addr : app_xxx 로 시작하는 신호들은 ddr_controller와 연결되는 신호들입니다. memory address 입니다.
- ✓ app_cmd : read (1) / write (0) command 입니다.
- ✓ app_en : Address / Command를 전달하는 신호입니다.
- ✓ app_rdy : Address / Command를 받을 준비가 되었음을 알려줍니다.
- ✓ app_wdf_data : write data 입니다.
- ✓ app_wdf_wren, app_wdf_end : app_wdf_data를 전송할 때, 이 신호를 High 로 만들어 줍니다.
- ✓ app_wdf_rdy : 메모리가 write 준비가 되었음을 알려줍니다. app_wdf_rdy 신호가 High일 때, app_addr, app_wdf_data, app_wdf_wren, app_wdf_end를 전송합니다.

```

65 // State Parameter
66 parameter      M_IDLE          = 2'd0;
67 parameter      M_WRITE         = 2'd1;
68 parameter      M_DONE          = 2'd2;
69
70
71 // -----
72 // State Control
73 reg      [1:0]  m_state;
74 wire      s_idle   = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
75 wire      s_write  = (m_state==M_WRITE) ? 1'b1 : 1'b0;
76 wire      s_done   = (m_state==M_DONE ) ? 1'b1 : 1'b0;

```

- ✓ 라인 66 - 68 : SM(State Machine) 을 위한 parameter 정의
- ✓ 라인 73 - 76 : SM을 위한 신호 상태 정의
- ✓ SM 상태는 3개입니다. IDLE 상태에서 wstart 신호가 들어오면 WRITE 상태가 되어 write를 진행합니다. 128bits 를 모두 8번 write 가 끝나면 DONE 상태가 되고, wdone 신호를 전송한 후 다시 IDLE 상태가 됩니다.

```

78 reg [3:0] addr_cnt ;
79 always @(posedge mclk or negedge reset)
80 begin
81 if(!reset) addr_cnt <= 4'b0;
82 else addr_cnt <= s_idle ? 4'b0 : (app_rdy & app_wdf_rdy) ? addr_cnt + 1'b1 : addr_cnt;
83 end
84
85 wire [27:0] app_addr = {waddr[27:6], addr_cnt[2:0], 3'b0};
86 wire [2:0] app_cmd = 3'b000;
87
88 wire [127:0] app_wdf_data = wdata;
89
90 wire [15:0] app_wdf_mask = 16'b0 ;
91
92 wire app_en = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[3]) ? 1'b1 : 1'b0;
93 wire app_wdf_wren = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[3]) ? 1'b1 : 1'b0;
94 wire app_wdf_end = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[3]) ? 1'b1 : 1'b0;
95 wire wready = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[3]) ? 1'b1 : 1'b0;
96
97 reg wdone;
98 always @(posedge mclk or negedge reset)
99 begin
100 if(!reset) wdone <= 1'b0;
101 else wdone <= s_done ;
102 end

```

- ✓ 라인 78 - 83 : address counter, (app_rdy & app_wdf_rdy) 일 때마다 1개씩 증가합니다. 이 값이 7이면 8개 모두 전송된 것을 알 수 있습니다.
- ✓ 라인 85 : app_addr : address는 8의 배수로 증가합니다. waddr 부터 시작해서 8씩 증가합니다.
- ✓ 라인 86 : app_cmd : write를 위한 app_cmd는 0입니다.
- ✓ 라인 88 : app_wdf_data : User Interface에서 받은 wdata를 그대로 전달합니다.
- ✓ 라인 90 : app_wdf_mask : 사용하지 않으므로 0를 전달합니다.
- ✓ 라인 92 - 95 : app_en, app_wdf_wren, app_wdf_end, wready 모두 (app_rdy & app_wdf_rdy & ~addr_cnt[3]) 일 때에는 1입니다.
- ✓ 라인 97 - 102 : wdone 신호를 생성합니다.

여기에서 주의할 것은 mig7_write8(or mig7_write) 모듈은 User Interface Logic 신호(waddr, wdata, wready)와 ddr_controller 신호(app_xxx 신호)를 전달해 주는 역할을 하게 됩니다. 따라서 일부부분을 Flip/Flip을 추가한 Register로 구현하면 원치 않는 딜레이가 추가되어, 제대로 동작하지 않습니다. 라인 85 - 95 사이에 wire로 처리된 것에 주의 하시길 바랍니다. 또한 8개의 데이터를 전송할 때에만 app_en, app_wdf_wren, app_wdf_end, wready 신호가 Active가 되어야 합니다. 라인 92 - 95에서 “~addr_cnt[3]”이 빠지지 않도록 주의해야 합니다.

```

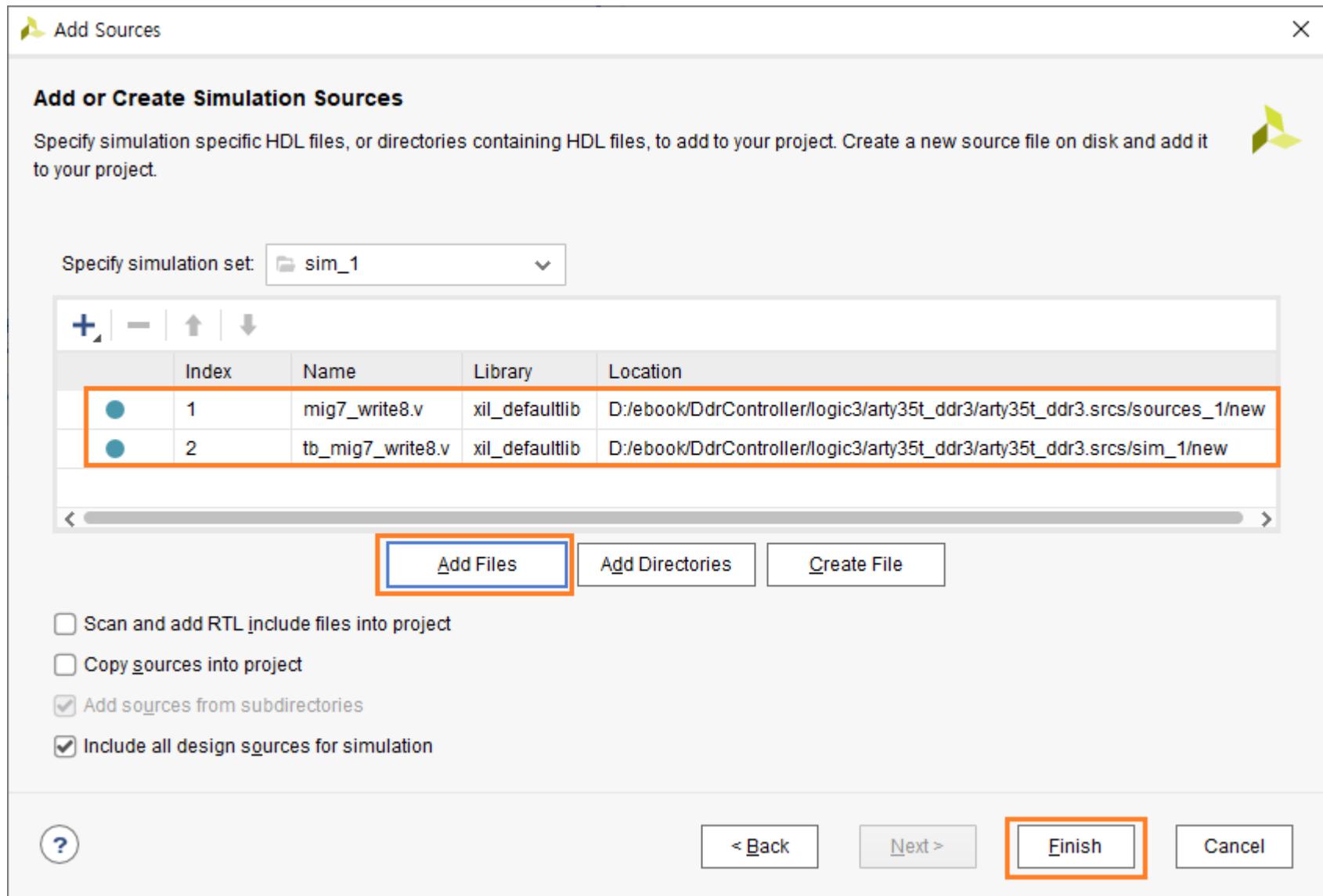
105 always @(posedge mclk or negedge reset)
106 begin
107 if(!reset) begin
108 m_state <= 2'b0;
109 end
110 else begin
111 m_state <= (s_idle & wstart) ? M_WRITE : 
112 (s_write & app_rdy & (addr_cnt==4'd7)) ? M_DONE : 
113 (s_done) ? M_IDLE : m_state ;
114 end
115 end
116
117
118 endmodule

```

- ✓ 라인 105 - 115 : SM 상태 전이를 구현합니다. IDLE 상태에서는 wstart 신호가 들어오면 WRITE 상태가 되고, WRITE 상태에서는 8개의 데이터를 전송하면 DONE 상태가 되고, DONE 상태에서는 wdone 신호를 전송 후 IDLE 상태가 됩니다.

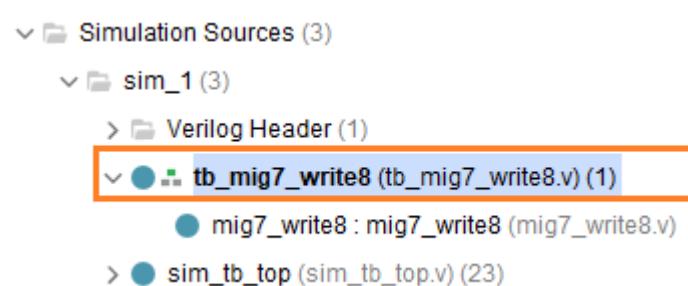
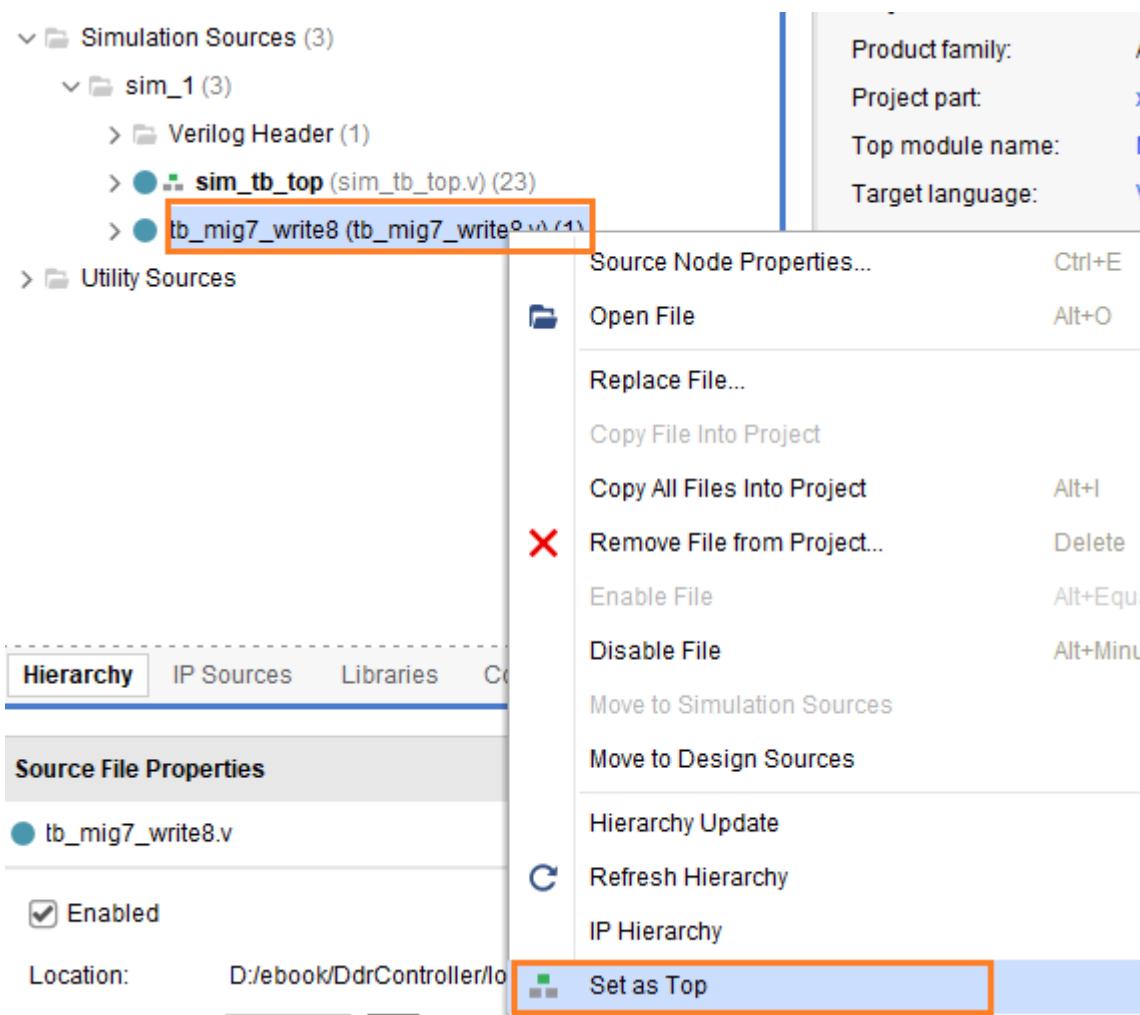
5.2.2 mig7_write8 모듈 simulation

mig7_write8 모듈의 Test bench 는 tb_mig7_write8입니다. Simulation Sources에 2개의 파일(tb_mig7_write8.v mig7_write8.v)을 추가합니다.



Add Files 버튼을 클릭해서 2개의 파일을 추가한 후 Finish 버튼을 클릭합니다.

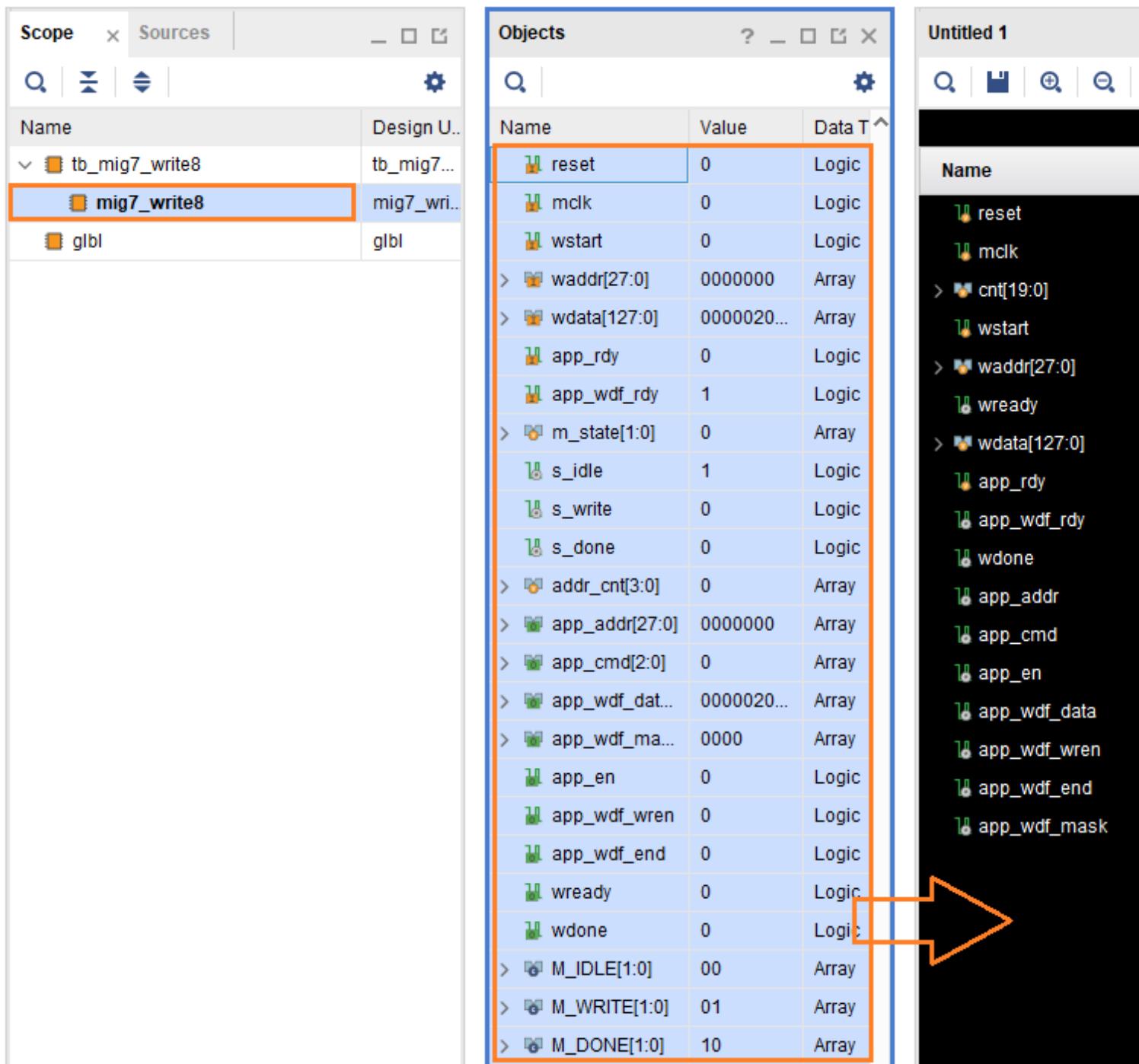
tb_mig7_write8 을 선택하고 우 클릭해서 나오는 메뉴 중에 “Set as Top”을 선택해서 Top Module로 지정합니다. (Top Module로 지정하지 않고 Simulation을 진행하면 다른 모듈이 진행됩니다. 주의하셔야 합니다. 저도 여러 번 당했습니다.)



SIMULATION - Run Simulation - Run Behavioral Simulation 을 클릭하면 Simulation이 진행됩니다.

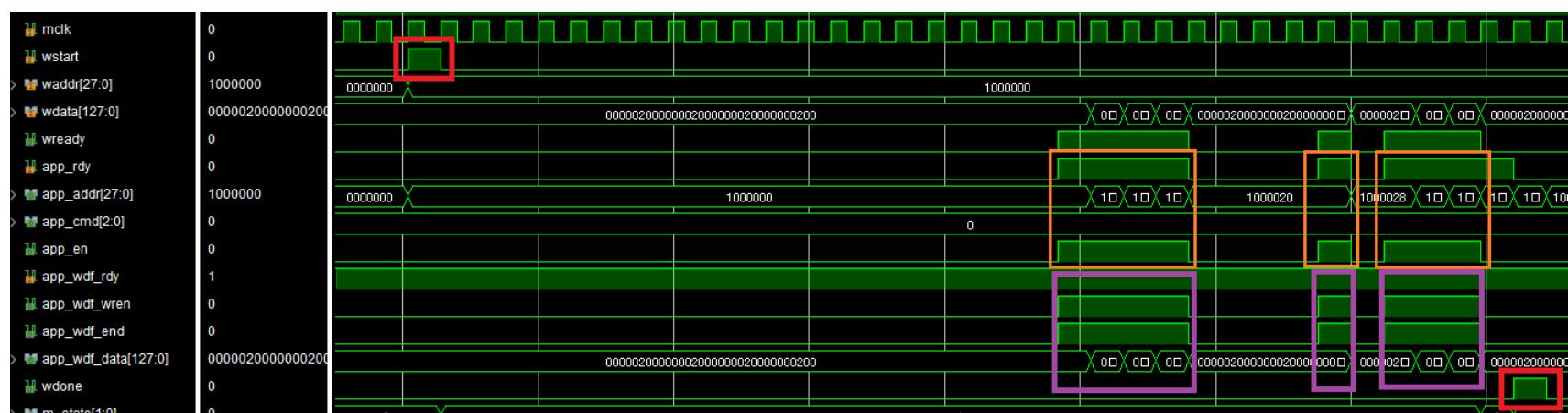
mig7_write8을 선택 후 모든 신호들을 Wave 윈도에 추가합니다. 또는 mig7_write8 - 우클릭 - Add to Wave Window 를 클릭합니다.

Tcl Console 윈도의 Command 입력창에 “run 0.1ms” 입력하고 엔터키를 눌러서 0.1ms 동안 simulation을 진행합니다.



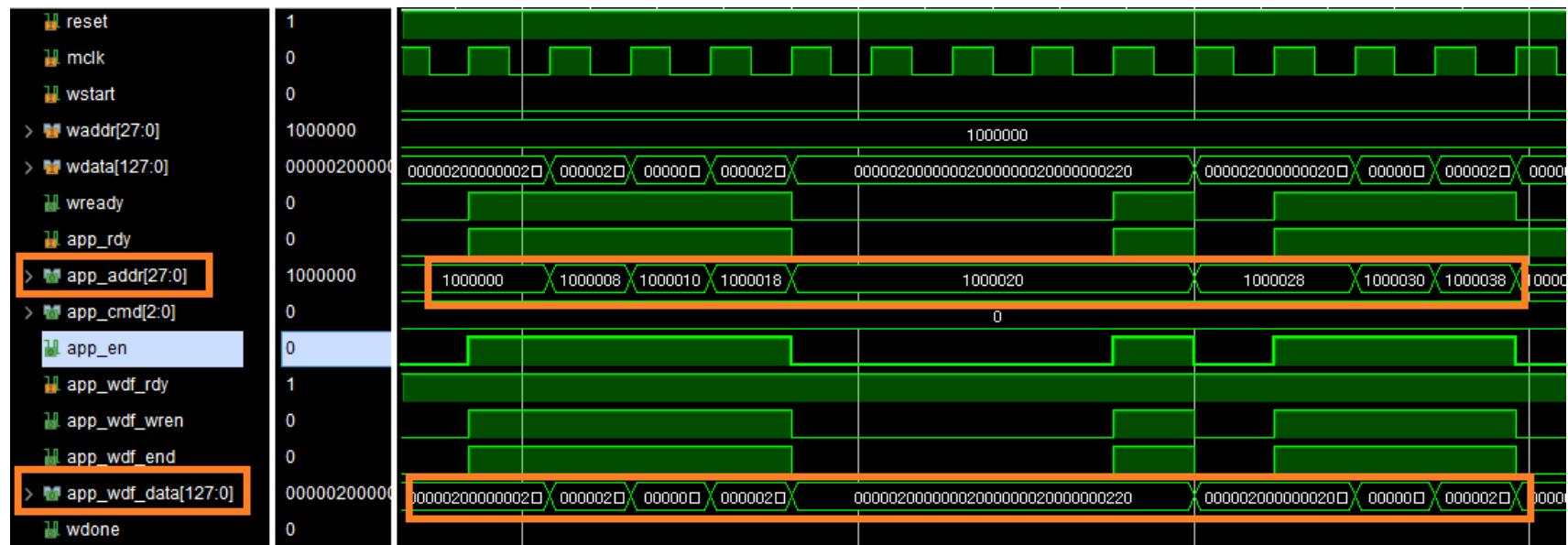
tb_mig7_write() Test bench 는 mig7_write8() 모듈의 동작을 확인하기 위하여 mig7_write8 모듈의 입력 값을 임의로 만들어서 mig7_write8의 동작을 확인하도록 구성되어 있습니다.

파형을 통해 mig7_write8 모듈이 제대로 동작하는지 확인합니다.



wstart 신호를 받아 write를 시작합니다. app_rdy 신호가 High 가 될 때까지 기다렸다가 High 가 되면 app_addr, app_cmd, app_en 값을 총 8번 (4번, 1번, 3번) 전송합니다. app_rdy & app_wdf_rdy 가 High 가 될 때, app_wdf_wren, app_wdf_end_app_wdf_data 를 총 8번 전송합니다. 8번 전송후에는 wdone 신호가 발생합니다.

아래 그림은 app_addr, app_wdf_data 부분을 확대하여 보여줍니다.



app_addr은 0x100_0000 부터 8씩 증가해서 0x100_0038까지 증가했습니다. app_wdf_data는 wdata 값을 그대로 전달하고 있습니다. “4.3.3 Write Timing 분석” 과 동일하게 동작함을 확인하시길 바랍니다.

5.2.3 mig7_write 모듈 구현

mig7_write 모듈은 사용자가 설정한 wsize(8x128bits 단위) 만큼 메모리에 Write 하는 모듈입니다. mig7_write 모듈은 mig7_write8 모듈을 사용하여 구현합니다. mig7_write 모듈은 wsize에 설정된 값만큼, waddr, wstart 신호를 생성해서 mig7_write8 모듈로 전달합니다. mig7_write8 모듈에서 wdone 신호가 오면, waddr+64 후에 wstart 신호를 다시 생성해서 mig7_write8 모듈로 전달합니다. wsize에 설정된 blocks(8x128bits) 만큼 전송을 완료하면 User Interface에 wdone 신호를 전달합니다.

```

23  module mig7_write (
24      reset      ,          // ui_clk_sync_rst from ddr controller
25      mclk       ,          // ui_clk from ddr controller
26
27      mig_wstart ,
28      mig_waddr  ,
29      mig_wsize   ,          // 64 unit
30
31      mig_wdata  ,
32      mig_wrty   ,
33      mig_wdone  ,
34
35      app_addr   ,
36      app_cmd    ,
37      app_en     ,
38      app_rdy    ,
39      app_wdf_data,
40      app_wdf_wren,
41      app_wdf_end ,
42      app_wdf_mask ,
43      app_wdf_rdy
44 );
45
46      input      reset      ;
47      input      mclk       ;
48
49      input      mig_wstart ;
50      input [27:0] mig_waddr ;
51      input [9:0]  mig_wsize ;
52      input [127:0] mig_wdata ;
53      output     mig_wdone ;
54      output     mig_wrty  ;
55
56      output [27:0] app_addr ;
57      output [2:0]  app_cmd   ;
58      output     app_en    ;
59      input      app_rdy   ;
60      output [127:0] app_wdf_data;
61      output     app_wdf_wren;
62      output     app_wdf_end ;
63      output [15:0] app_wdf_mask ;
64      input      app_wdf_rdy

```

In/Out 포트는 mig7_write8 모듈과 거의 비슷합니다. mig_xxx 신호들은 User Interface 와 연결된 신호들이고, app_xxx 신호들은 ddr_controller 와 연결된 신호들입니다. mig_wsize 입력포트가 추가되었습니다.

```

66 // State Parameter
67 parameter      MM_IDLE        = 2'd0;
68 parameter      MM_WRITE       = 2'd1;
69 parameter      MM_DONE        = 2'd2;
70
71 // -----
72 // State Control
73 reg   [1:0] mm_state;
74 wire    ss_idle  = (mm_state==MM_IDLE ) ? 1'b1 : 1'b0;
75 wire    ss_write = (mm_state==MM_WRITE) ? 1'b1 : 1'b0;
76 wire    ss_done  = (mm_state==MM_DONE ) ? 1'b1 : 1'b0;

```

SM 도 mig7_write8 모듈과 거의 비슷합니다. IDLE, WRITE, DONE 3개의 상태가 있습니다.

```

78 wire          wdone;
79 reg           wdone_ld;
80 always @(posedge mclk or negedge reset)
81 begin
82     if(!reset)    wdone_ld <= 1'b0;
83     else          wdone_ld <= wdone ;
84 end
85
86 reg   [9:0]  block_cnt ;
87 always @(posedge mclk or negedge reset)
88 begin
89     if(!reset)    block_cnt <= 10'b0;
90     else          block_cnt <= ss_idle ? 10'b0 : wdone ? block_cnt + 1'b1 : block_cnt;
91 end
92
93 reg   [27:0]  waddr ;
94 always @(posedge mclk or negedge reset)
95 begin
96     if(!reset)    waddr <= 28'b0;
97     else          waddr <= ss_idle ? mig_waddr : wdone_ld ? waddr + 7'd64 : waddr;
98 end
99
100 reg        wstart;
101 always @(posedge mclk or negedge reset)
102 begin
103     if(!reset)    wstart <= 1'b0;
104     else          wstart <= (ss_idle & mig_wstart) ? 1'b1 :
105                           wdone_ld ? ((block_cnt==mig_wsize) ? 1'b0 : 1'b1) : 1'b0 ;
106 end

```

- ✓ 라인 78 - 84 : 내부적으로 타이밍을 맞추기 위하여 1-clock delay가 추가되었습니다.
- ✓ 라인 86 - 91 : 8x128 bits 전송(block)을 카운트합니다. wsize 만큼 전송되었는지를 확인하는 용도로 사용됩니다.
- ✓ 라인 93 - 98 : waddr 을 생성합니다. 처음에는 User Interface 에서 전달받은 값(mig_waddr)을 사용하고, block 이 증가할 때마다 64씩 증가합니다. 1-block 당 8x128bits 씩 전송, address 는 128bits 마다 8씩 증가, 따라서 1-block 당 64씩 증가합니다.
- ✓ 라인 100 - 106 : wstart를 생성합니다. 처음에는 User Interface 에서 전달받은 값(mig_wstart)을 사용하고, mig7_write8 모듈에서 전송이 완료될 때마다 다시 wstart 신호를 생성합니다.

```

108   always @ (posedge mclk or negedge reset)
109 begin
110     if (!reset)      begin
111       mm_state <= 2'b0;
112     end
113     else    begin
114       mm_state <= (ss_idle  & mig_wstart          ) ? MM_WRITE :
115                               (ss_write & wdone_ld & (block_cnt==mig_wsize)) ? MM_DONE :
116                               (ss_done           ) ? MM_IDLE : mm_state ;
117     end
118 end

```

- ✓ 라인 108 - 118 : SM 상태 전이를 구현합니다. IDLE 상태에서는 mig_wstart 신호가 High 가 되면 WRITE 상태가 되고, WRITE 에서는 전송된 block 수와 mig_wsize 값이 같으면 전송이 완료되었으므로 DONE 상태가 됩니다. DONE 상태에서는 mig_wdone 신호를 전송하고 IDLE 상태로 돌아갑니다.

```

120 wire  [27:0] app_addr      ;
121 wire  [2:0]  app_cmd       ;
122 wire  app_en            ;
123 wire  [127:0] app_wdf_data ;
124 wire  app_wdf_wren      ;
125 wire  app_wdf_end        ;
126
127 wire  mig_wrdy;
128 wire  [15:0] app_wdf_mask ;
129
130 reg   mig_wdone;
131 always @ (posedge mclk or negedge reset)
132 begin
133   if (!reset)      mig_wdone <= 1'b0;
134   else             mig_wdone <= ss_done ;
135 end
136
137 mig7_write8 mig7_write8 (
138   .reset      (reset      ),
139   .mclk       (mclk      ),
140
141   .wstart     (wstart     ),
142   .waddr      (waddr      ),
143   .wdata      (mig_wdata  ),
144
145   .wready     (mig_wrdy  ),
146   .wdone      (wdone      ),
147
148   .app_addr   (app_addr   ),
149   .app_cmd    (app_cmd    ),
150   .app_en     (app_en     ),
151   .app_rdy    (app_rdy    ),
152   .app_wdf_data (app_wdf_data),
153   .app_wdf_wren (app_wdf_wren),
154   .app_wdf_end (app_wdf_end),
155   .app_wdf_mask (app_wdf_mask),
156   .app_wdf_rdy (app_wdf_rdy)
157 );
158
159 endmodule

```

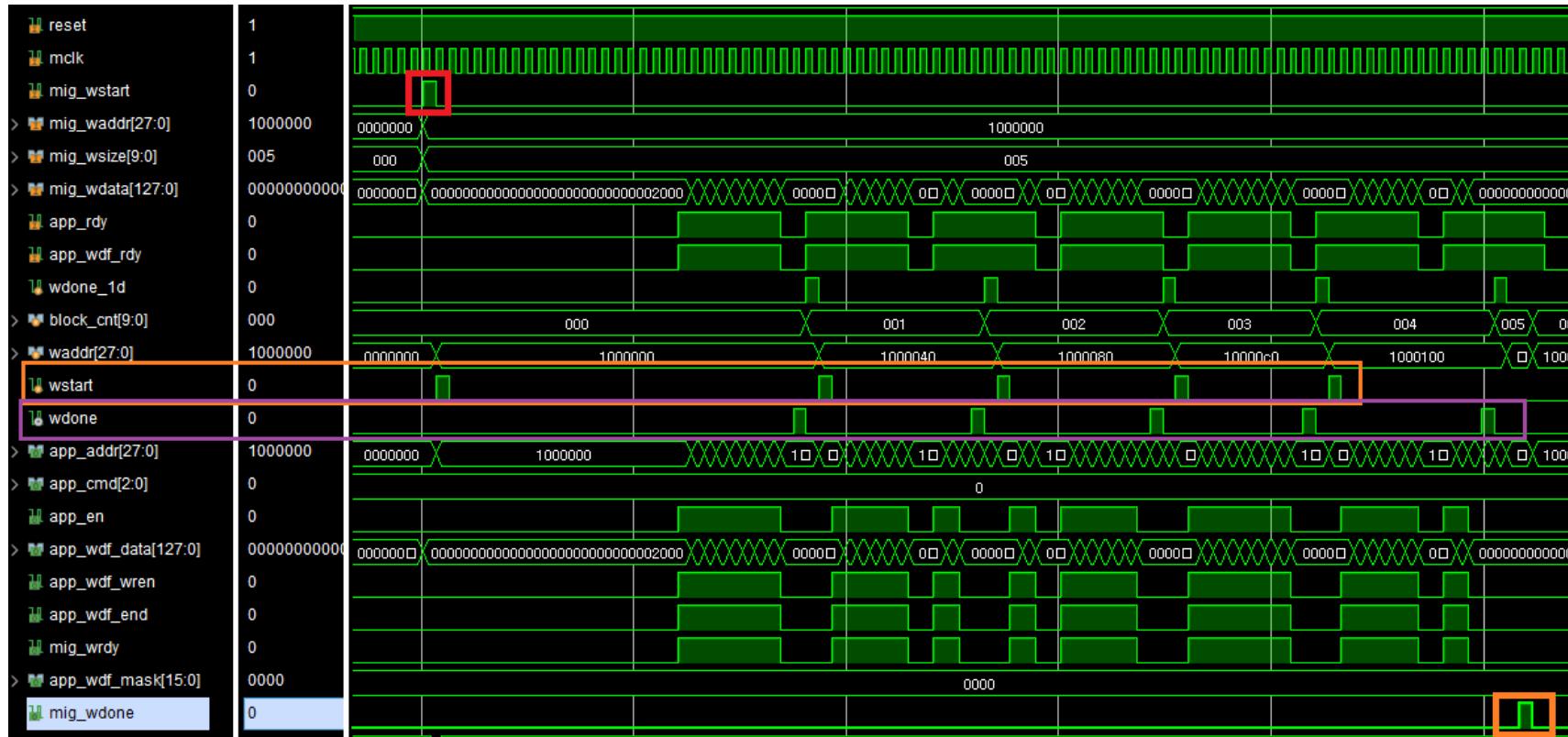
- ✓ 라인 120 - 125 : ddr_controller 와 연결되는 신호입니다.
- ✓ 라인 130 - 135 : mig_wdone 신호를 생성합니다.
- ✓ 라인 137 - 157 : mig7_write8 모듈을 사용합니다.

5.2.4 mig7_write 모듈 Simulation

mig7_write 모듈의 Test bench는 tb_mig7_write() 입니다. Simulation Sources에 mig7_write.v, tb_mig7_write.v 파일을 추가하고, tb_mig7_write 모듈을 Top Module로 지정하고, Simulation을 진행합니다.

mig7_write 신호들을 Wave 윈도에 추가하고 “run 0.1ms”를 입력해서 0.1ms 동안 simulation을 진행합니다.

아래는 simulation 결과를 보여줍니다.



mig_wsiz = 5 로 설정되었습니다. mig_wstart가 발생하면, wstart 가 5번 발생하고, mig7_write8 모듈에서 데이터 전송을 완료할 때마다 wdone 이 발생되어 wdone이 5번 발생합니다. wdone 5번째에 최종적으로 mig_wdone 이 발생합니다. waddr 값은 mig_waddr 값을 받아 wdone 이 High가 될 때마다 64씩 증가합니다. (0x100_0000, 0x100_0040 ~ 0x100_0100)

5.3 Read Module 구현

메모리의 데이터를 Read 하는 코드를 구현합니다. Write 모듈과 많은 부분이 비슷합니다.

5.3.1 mig7_read8 모듈 구현

mig7_read8 모듈은 128bits 씩 총 8번 Read를 진행합니다. “4.3.4 Read Timing 분석”에 있는 내용을 그대로 구현한 모듈입니다.

```

23  module mig7_read8 (
24      reset           ; // ui_clk_sync_rst from ddr controller
25      mclk            ; // ui_clk from ddr controller
26
27      rstart           ;
28      raddr            ;
29
30      rdata             ;
31      rdata_valid       ;
32      rdone            ;
33
34      app_addr          ;
35      app_cmd           ;
36      app_en            ;
37      app_rdy           ;
38      app_rd_data        ;
39      app_rd_data_end    ;
40      app_rd_data_valid  ;
41  );
42
43  input      reset           ;
44  input      mclk            ;
45
46  input      rstart           ;
47  input  [27:0] raddr          ;
48  output  [127:0] rdata         ;
49  output      rdata_valid       ;
50  output      rdone            ;
51
52  output  [27:0] app_addr        ;
53  output  [2:0]  app_cmd         ;
54  output      app_en            ;
55  input      app_rdy           ;
56  input  [127:0] app_rd_data      ;
57  input      app_rd_data_end    ;
58  input      app_rd_data_valid  ;

```

- ✓ reset, mclk : ddr_controller에서 출력되는 ui_clk_sync_rst, ui_clk 신호를 사용합니다.
- ✓ rstart : User Interface 모듈(ddr_test or frame_test)에서 생성하는 read start strobe입니다. 이 신호가 “H”가 되면 read를 시작합니다.
- ✓ raddr[27:0] : 메모리에서 Read 하는 Start Address입니다.
- ✓ rdata[127:0] : 메모리에서 Read 한 데이터를 User Interface 모듈로 전송합니다. rdata_valid 일 때에만 유효한 데이터입니다.
- ✓ rdata_valid : 이 신호가 “H” 일 때에만 rdata 데이터가 유효한 데이터입니다.
- ✓ rdone : 모든 데이터가 Read 되었음을 User Interface 모듈에 알려줍니다.

- ✓ app_addr[27:0] : app_xxx 로 시작하는 신호들은 ddr_controller와 연결되는 신호들입니다. Memory Address입니다.
- ✓ app_cmd[2:0] : read (1) / write (0) command 입니다.
- ✓ app_en : Address / Command 를 전달하는 신호입니다. (enable 신호)
- ✓ app_rdy : ddr_controller 에서 Address / Command를 받을 준비가 되었음을 알려줍니다.
- ✓ app_rd_data [127:0] : Memory에서 Read한 데이터입니다.
- ✓ app_rd_data_valid : 이 신호가 “H” 일 때, app_rd_data는 유효한 데이터입니다.
- ✓ app_rd_data_end : app_rd_data_valid 와 거의 동일한 신호로 생각하면 됩니다. PHY to Controller Clock Ratio의 값이 2:1 일 때에는 의미가 있지만, 4:1 일 때에는 무시해도 됩니다.

```

60 // State Parameter
61 parameter      M_IDLE          = 2'd0;
62 parameter      M_READ          = 2'd1;
63 parameter      M_DONE          = 2'd2;
64
65 // -----
66 // State Control
67 reg      [1:0]  m_state;
68 wire      s_idle  = (m_state==M_IDLE) ? l'b1 : l'b0;
69 wire      s_read   = (m_state==M_READ) ? l'b1 : l'b0;
70 wire      s_done   = (m_state==M_DONE) ? l'b1 : l'b0;

```

- ✓ 라인 60 - 68 : SM을 위한 parameter 정의
- ✓ 라인 67 - 70 : SM 상태 정의 신호

```

72 reg [3:0] addr_cnt ;
73 always @(posedge mclk or negedge reset)
74 begin
75 if(!reset) addr_cnt <= 4'b0;
76 else addr_cnt <= s_idle ? 4'b0 : app_rdy ? ((addr_cnt==4'd8) ? 4'd8 : addr_cnt + 1'b1) : addr_cnt;
77 end
78
79 wire [27:0] app_addr = {raddr[27:6], addr_cnt[2:0], 3'b0};
80 wire [2:0] app_cmd = 3'b001;
81 wire app_en = s_idle ? 1'b0 : ~addr_cnt[3] ? 1'b1 : 1'b0 ;
82
83 reg [3:0] data_cnt ;
84 always @(posedge mclk or negedge reset)
85 begin
86 if(!reset) data_cnt <= 4'b0;
87 else data_cnt <= s_idle ? 4'b0 : (app_rd_data_valid & app_rd_data_end) ? (data_cnt + 1'b1) : data_cnt;
88 end
89
90 reg [127:0] rdata ;
91 always @(posedge mclk or negedge reset)
92 begin
93 if(!reset) rdata <= 128'b0;
94 else rdata <= s_idle ? 128'b0 : (app_rd_data_valid & app_rd_data_end) ? app_rd_data : rdata ;
95 end
96
97 reg rdata_valid ;
98 always @(posedge mclk or negedge reset)
99 begin
100 if(!reset) rdata_valid <= 1'b0;
101 else rdata_valid <= s_idle ? 1'b0 : (app_rd_data_valid & app_rd_data_end & ~data_cnt[3]) ? 1'b1 : 1'b0;
102 end
103
104 reg rdone;
105 always @(posedge mclk or negedge reset)
106 begin
107 if(!reset) rdone <= 1'b0;
108 else rdone <= s_done ;
109 end

```

- ✓ 라인 72 - 77: address counter, app_rdy 신호가 “H” 일 때 값을 증가합니다. 이 값을 사용하여 app_addr 값을 생성합니다.
- ✓ 라인 79 - 81 : app_addr, app_cmd, app_en 을 생성합니다. 불필요한 delay를 없애기 위하여 wire로 구현합니다.
- ✓ 라인 83 - 88 : data counter, Memory에서 읽은 데이터를 수신할 때마다 1씩 증가합니다. 이 값이 7이되면 8개의 데이터를 모두 읽은 것이므로 Read Sequence를 종료합니다.
- ✓ 라인 90 - 95 : 메모리에서 읽은 데이터를 User Interface 모듈로 전달합니다.
- ✓ 라인 97 - 102 : app_rd_data_valid 신호를 User Interface 모듈로 전달합니다.
- ✓ 라인 104 - 109 : 데이터 8개를 모두 읽으면 rdone 신호를 user Interface 모듈로 전달합니다.

```

111 always @(posedge mclk or negedge reset)
112 begin
113 if(!reset) begin
114 m_state <= 2'b0;
115 end
116 else begin
117 m_state <= (s_idle & rstart ) ? M_READ :
118 (s_read & app_rd_data_valid & app_rd_data_end & (data_cnt==4'd7)) ? M_DONE :
119 (s_done ) ? M_IDLE : m_state ;
120 end
121 end

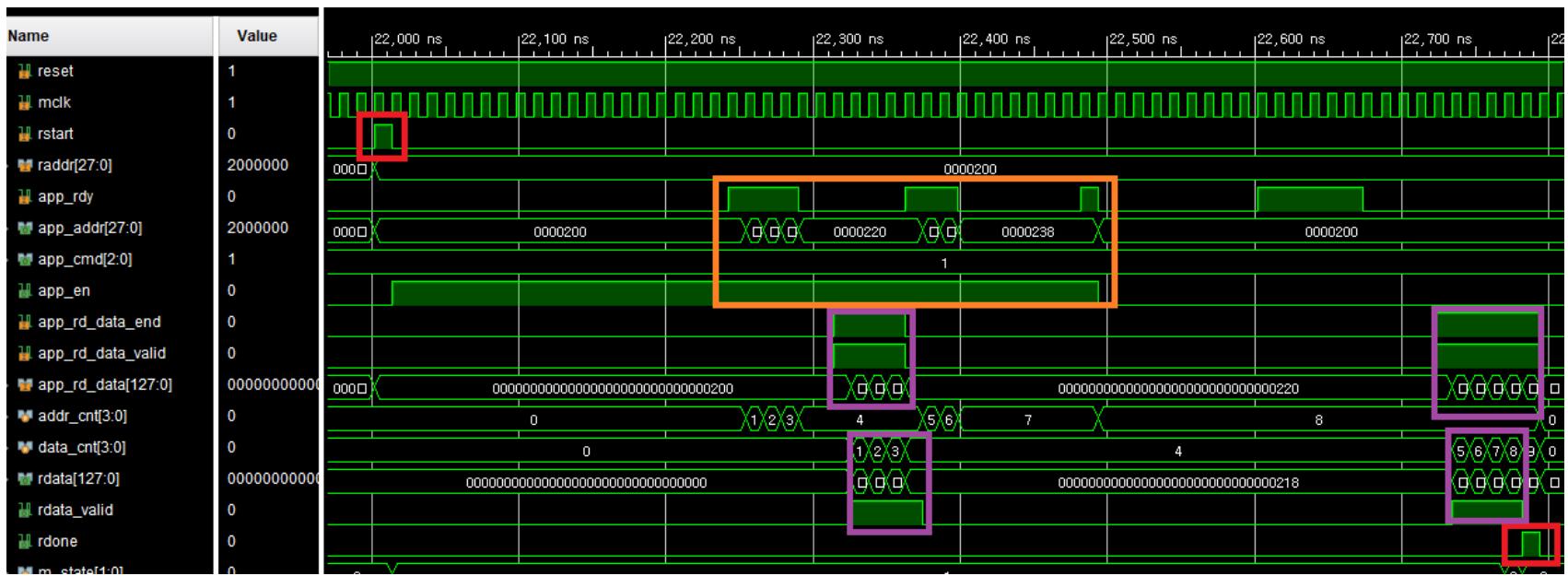
```

- ✓ 라인 111 - 121 : SM 상태 전이를 구현합니다. IDLE 상태에서는 rstart 신호가 Active 되면 READ 상태가 되고, READ 상태에서는 8개의 데이터를 모두 읽으면 DONE 상태가 되고, DONE 상태에서는 rdone 신호를 User Interface 모듈로 전송 후 IDLE 상태가 됩니다.

5.3.2 mig7_read8 모듈 Simulation

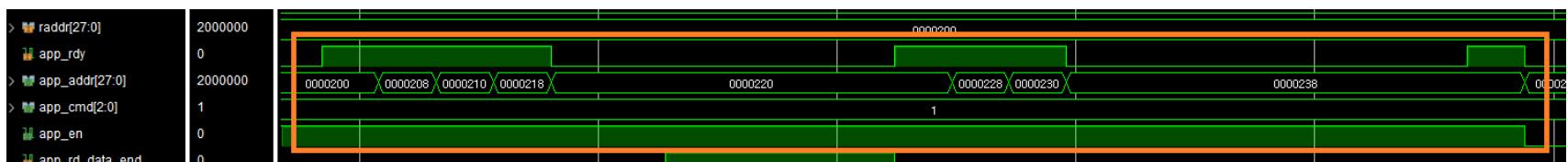
mig7_read8 모듈의 Test bench 는 tb_mig7_read8() 입니다. Simulation Sources에 2개의 파일 (tb_mig7_read8.v, mig7_read8.v)을 추가하고 tb_mig7_read8 모듈을 Top 모듈로 지정합니다. Run Simulation - Run Behavioral Simulation을 클릭하고, mig7_read8 신호들을 Wave 윈도에 추가하고 0.1ms 동안 Simulation을 진행합니다.

아래 그림은 결과를 보여줍니다.

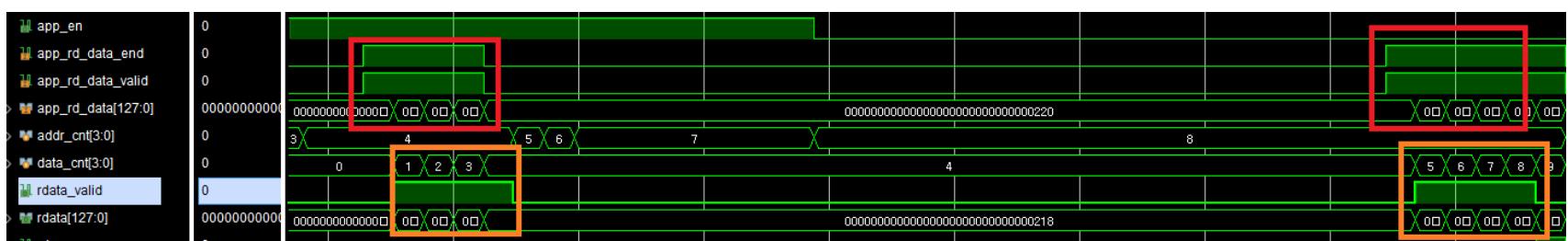


rstart 신호가 “H” 가 될 때 Read를 진행합니다. 주황색 부분에 app_rdy 가 Active(H)인 구간에 8번 app_addr 을 생성해서 ddr_controller 에 전달합니다. Address 는 0x000_0200 ~ 0x000_0238 입니다. app_rd_dava_valid 신호가 Active(H) 일 때, app_rd_data 을 읽습니다. app_rd_data_valid, app_rd_data 는 1-clock delay 를 거쳐서 rdata_valid, rdata 를 생성후 User Interface 모듈로 전달합니다. rdata 를 8개 읽은 후에 rdone 신호가 생성되어 User Interface 모듈로 전달됩니다.

아래 그림은 app_addr 부분을 확대하여 보여줍니다.



아래 그림은 app_rd_data 부분을 확대해서 보여줍니다.



app_rd_data_valid, app_rd_data 신호가 1-clock delay 되어 rdata_valid, rdata 신호로 전달됩니다. 데이터를 8개 Read 한 후에는 rdone 신호가 발생됩니다

5.3.3 mig7 read 모듈 구현

mig7 read 모듈은 사용자가 설정한 rsize (8x128bits 단위) 만큼 메모리의 데이터를 Read 하는 모듈입니다. mig7 read

모듈은 mig7_read8 모듈을 사용하여 구현합니다. mig7_read 모듈은 rstart, raddr 생성해서 mig7_read8 모듈로 전달하고 mig7_read8에서 전달받은 rdata_valid, rdata 를 User Interface 모듈로 전달합니다.

```

23  module mig7_read(
24      reset           ; // ui_clk_sync_rst from ddr controller
25      mclk            ; // ui_clk from ddr_controller
26
27      mig_rstart       ;
28      mig_raddr        ;
29      mig_rsize         ;
30
31      mig_rdata        ;
32      mig_rvalid       ;
33      mig_rdone         ;
34
35      app_addr          ;
36      app_cmd           ;
37      app_en            ;
38      app_rdy           ;
39      app_rd_data        ;
40      app_rd_data_end    ;
41      app_rd_data_valid  ;
42 );
43
44 input      reset           ;
45 input      mclk            ;
46
47 input      mig_rstart       ;
48 input [27:0] mig_raddr        ;
49 input [9:0]  mig_rsize         ;
50
51 output     [127:0] mig_rdata        ;
52 output     mig_rvalid       ;
53 output     mig_rdone         ;
54
55 output     [27:0] app_addr          ;
56 output     [2:0]  app_cmd           ;
57 output     app_en            ;
58 input      app_rdy           ;
59 input     [127:0] app_rd_data        ;
60 input      app_rd_data_end    ;
61 input      app_rd_data_valid  ;

```

In/Out 포트는 mig7_read8 모듈과 거의 비슷합니다. mig_xxx 신호들은 User Interface 모듈과 연결된 신호들이고, app_xxx 신호들은 ddr_controller와 연결된 신호들입니다.

```

63 // State Parameter
64 parameter   MM_IDLE      = 2'd0;
65 parameter   MM_READ      = 2'd1;
66 parameter   MM_DONE      = 2'd2;
67
68 // -----
69 // State Control
70 reg   [1:0]  mm_state;
71 wire      ss_idle = (mm_state==MM_IDLE) ? 1'b1 : 1'b0;
72 wire      ss_read = (mm_state==MM_READ) ? 1'b1 : 1'b0;
73 wire      ss_done = (mm_state==MM_DONE) ? 1'b1 : 1'b0;

```

SM을 정의합니다. IDLE, READ, DONE 3개의 상태를 정의합니다.

```

75   wire          rdone;
76   reg           rdone_ld;
77   always @(posedge mclk or negedge reset)
78   begin
79       if(!reset)      rdone_ld <= 1'b0;
80       else           rdone_ld <= rdone ;
81   end
82
83   reg [9:0]    block_cnt ;
84   always @(posedge mclk or negedge reset)
85   begin
86       if(!reset)      block_cnt <= 10'b0;
87       else           block_cnt <= ss_idle ? 10'b0 : rdone ? block_cnt + 1'b1 : block_cnt;
88   end
89
90   reg [27:0]   raddr ;
91   always @(posedge mclk or negedge reset)
92   begin
93       if(!reset)      raddr <= 28'b0;
94       else           raddr <= ss_idle ? mig_raddr : rdone_ld ? raddr + 7'd64 : raddr ;
95   end
96
97   reg          rstart;
98   always @(posedge mclk or negedge reset)
99   begin
100      if(!reset)     rstart <= 1'b0;
101      else           rstart <= (ss_idle & mig_rstart ) ? 1'b1 :
102                           rdone_ld ? ((block_cnt==mig_rsize) ? 1'b0 : 1'b1) : 1'b0 ;
103  end

```

- ✓ 라인 75 - 81 : 내부적으로 타이밍을 맞추기 위하여 1-clock delay가 추가되었습니다.
- ✓ 라인 83 - 88 : block을 카운트 합니다. 8 x 128 bits의 데이터를 읽으면 (mig7_read8) block 카운트 값이 증가합니다. 이 값이 rsize와 같으면 Read Sequence를 종료합니다.
- ✓ 라인 90 - 95 : raddr 을 생성합니다.
- ✓ 라인 97 - 103 : rstart 을 생성합니다.

```

105  always @(posedge mclk or negedge reset)
106 begin
107     if(!reset)      begin
108         mm_state <= 2'b0;
109     end
110     else           begin
111         mm_state <= (ss_idle & mig_rstart ) ? MM_READ  :
112                           (ss_read & rdone_ld & (block_cnt==mig_rsize)) ? MM_DONE  :
113                           (ss_done ) ? MM_IDLE  : mm_state ;
114     end
115 end

```

- ✓ 라인 105 - 115 : SM 상태 전이를 구현합니다. IDLE 상태에서는 mig_rstart 신호가 Active(H)가 되면 READ 상태가 되고, READ 상태에서는 block_cnt 값이 mig_rsize와 같아지면 DONE 상태가 됩니다. DONE 상태에서는 mig_rdone 신호를 전송하고 IDLE 상태가 됩니다.

```

117   reg          mig_rdone;
118   always @(posedge mclk or negedge reset)
119 begin
120     if(!reset)      mig_rdone <= 1'b0;
121     else           mig_rdone <= ss_done ;
122 end
123
124 wire      mig_rvalid      ;
125 wire [127:0] mig_rdata      ;
126 wire [27:0] app_addr       ;
127 wire [2:0]  app_cmd        ;
128 wire      app_en          ;
129
130 mig7_read8 mig7_read8 (
131   .reset      (reset          ),
132   .mclk       (mclk          ),
133
134   .rstart     (rstart         ),
135   .raddr      (raddr          ),
136
137   .rdata      (mig_rdata      ),
138   .rdata_valid (mig_rvalid    ),
139   .rdone      (rdone          ),
140
141   .app_addr   (app_addr      ),
142   .app_cmd    (app_cmd       ),
143   .app_en    (app_en        ),
144   .app_rdy   (app_rdy       ),
145   .app_rd_data (app_rd_data  ),
146   .app_rd_data_end (app_rd_data_end),
147   .app_rd_data_valid (app_rd_data_valid)
148 );

```

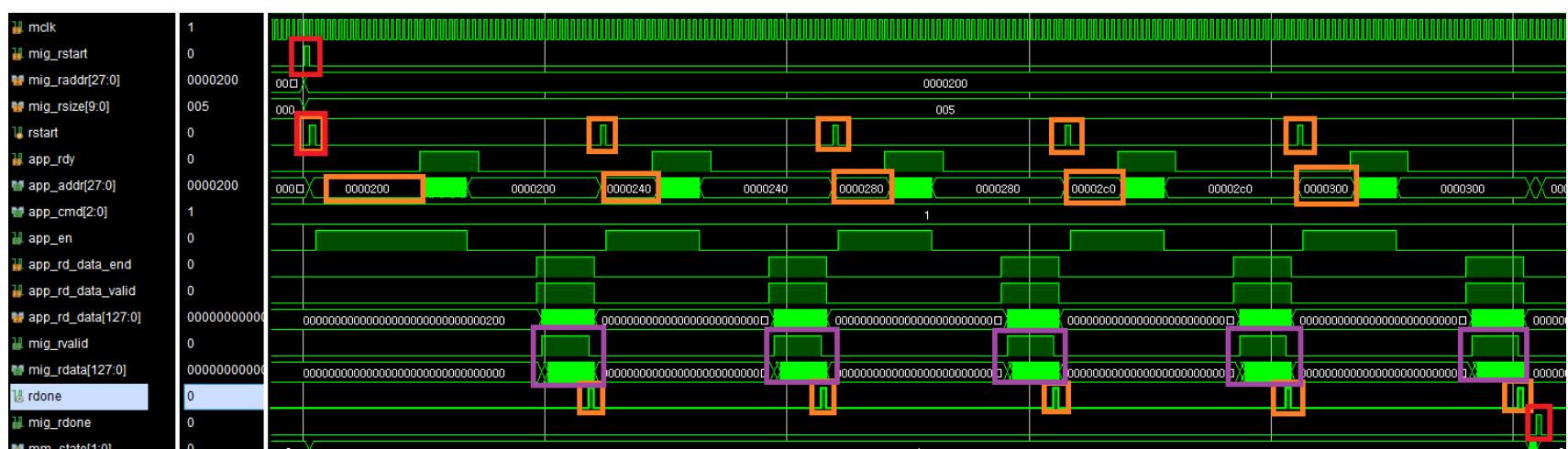
- ✓ 라인 117 - 122 : mig_rdone 을 생성합니다.
- ✓ 라인 124 - 148 : mig7_read8 모듈을 사용합니다.

5.3.4 mig7_read 모듈 Simulation

mig7_read 모듈의 Test bench 는 tb_mig7_read() 입니다. Simulation Sources에 mig7_read.v, tb_mig7_read.v 파일을 추가하고, tb_mig7_read 모듈을 Top Module로 지정하고, Simulation을 진행합니다.

mig7_read 신호들을 Wave 윈도에 추가하고 “run 0.1ms”을 입력해서 0.1ms 동안 simulation을 진행합니다.

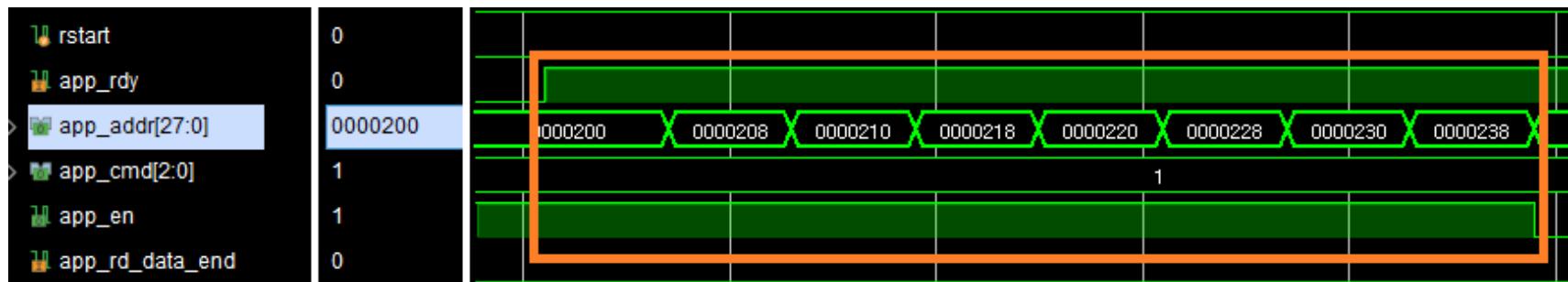
아래는 simulation 결과를 보여줍니다.



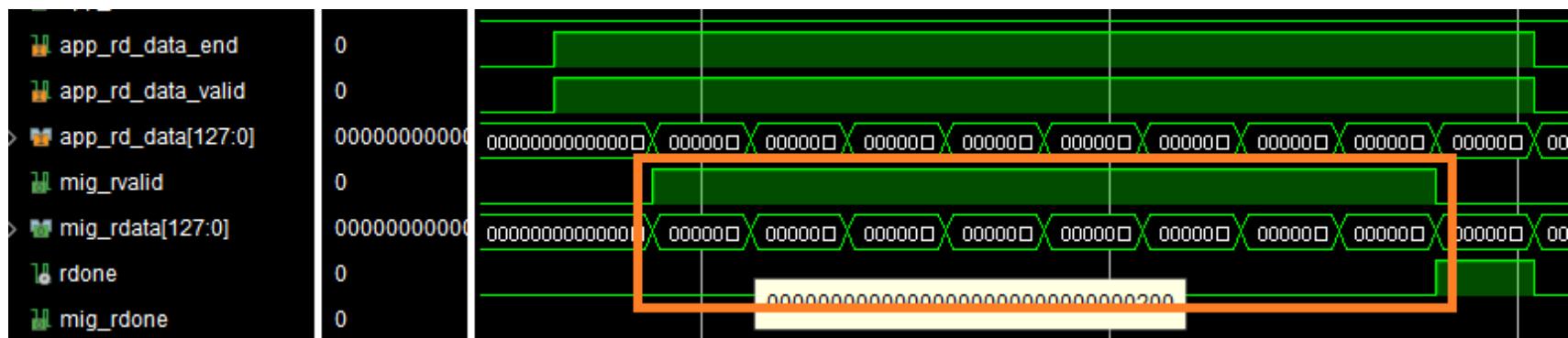
mig_raddr : 0x000_0200, mig_rsize : 0x005 값일 때, mig_rstart가 발생합니다. rstart 가 5번 발생했고, rdone 도 5번

발생하였습니다. rdone이 5번 발생한 후, mig_rdone이 발생합니다. app_addr은 0x000_2000, 0x000_0240, 0x000_0280 ~ 0x000_0x300 입니다. 8 x 128bits 씩 모두 5번 mig_rvalid, mig_rdata 가 발생되어 User Interface 모듈로 전달됩니다.

아래는 첫번째 Block의 app_addr 부분을 확대해서 보여줍니다. 0x000_0200 ~ 0x000_0238까지 8번 발생합니다.



아래는 첫번째 Block의 mig_rvalid, mig_rdata 부분을 확대해서 보여줍니다. 128bits 씩 8번 데이터를 Read 한 후에 rdone이 발생합니다.



5.4 User Interface 모듈 구현 (ddr_test 모듈)

이번 장에서는 User Interface 모듈을 구현합니다. User Interface 모듈은 2가지로 구현합니다. 첫번째는 이번장에서 구현하는 메모리 전영역을 테스트 데이터로 Write 하고, 전 영역을 Read 해서 Write 한 데이터와 비교해서 에러가 있는지를 확인하고 그 결과를 LED 로 표시합니다. 이것은 코드로 구현하고 Bitstream을 생성해서 Arty a7 보드에 다운로드해서 결과를 확인하도록 하겠습니다. 두번째는 다음장에서 구현하는 영상데이터 처리를 위한 프레임 버퍼를 구현합니다. 640x360 사이즈의 테스트 이미지 (rgb 24bits)로 프레임 버퍼를 구현한 후 Simulation 결과로 확인합니다. DDR Memory에 2개의 프레임 버퍼를 만들고 이곳에 영상 데이터를 Write 하고, Read 한 후에 데이터를 파일로 저장해서 원본 데이터와 비교를 합니다.

5.4.1 Test Scenario

DDR3 메모리의 전영역을 Write 하기 위해서는 어떻게 구성해야 하는지를 알아보겠습니다. Arty A7 보드에 장착되어 있는 메모리는 Data Width 16bits 인 2Gb 입니다. Address 영역은 0x000_0000 ~ 0x7ff_ffff (x 16bits) 입니다.

- 0x800_0000 * 16bits = 2Gb

mig7_write (or mig7_read) 모듈에서 Access 할 수 있는 Address 영역은 아래와 같습니다.

- Address 영역 = wsize[9:0] * 64 (x 16bits)

만약에 wsize[9:0] = 256 으로 설정하면 Address 영역은 16,385 (x 16bits) 가 됩니다.

- Address 영역 = 256 * 64 = 16,384 (x 16 bits)

전체 영역을 설정하기 위해서는, 0x800_0000 / 16384 = 8192 번 반복해야 합니다. 즉 wsize = 256으로 설정하고 mig7_write 모듈을 8192번 반복해서 write 하면 메모리 전영역을 write 할 수 있습니다.

5.4.2 Write Sequence

Write Sequence는 btn0 가 눌러지면 시작이 됩니다. wsize = 256으로 설정하고 총 8192번 mig7_write 모듈을 실행합니다. Write가 진행되는 동안에는 Led5 가 On이 됩니다. Write 하는 데이터는 4개의 128bits를 사용합니다. 4개의 데이터를 반복해서 Write 합니다.

- ✓ 첫번째 데이터 : 0x33333333_33333333_33333333_33333333
- ✓ 두번째 데이터 : 0xcccccccc_ cccccccc_ cccccccc_ cccccccc
- ✓ 세번째 데이터 : 0x55555555_55555555_55555555_55555555
- ✓ 네번째 데이터 : 0aaaaaaaaa_ aaaaaaaaaa_ aaaaaaaaaa_ aaaaaaaaaa

5.4.3 Read Sequence

Read Sequence는 btn1 이 눌러지면 시작이 됩니다. rsize = 256으로 설정하고 총 8192번 mig7_read 모듈을 실행합니다. Read가 진행되는 동안에는 Led6 가 On이 됩니다. Read를 하는 동안에 Read Data와 Write 할 때 사용했던 Test Data 와 비교해서 Error를 Count 합니다. Read 가 모두 완료가 되면 Error Count 값을 확인해서 Error Count 값이 “0” 이면 LED0의 파란색이 점등되고, Error Count 값이 “0”가 아니면 LED0의 빨간색이 점등이 됩니다.

5.4.4 ddr_test 모듈 구현

ddr_test 모듈의 소스는 양이 많아서 전체적으로 설명하지는 않습니다. 대략적인 내용을 설명합니다. 지금까지 소스를 분석 하였으면 큰 어려움 없이 분석할 수 있을 거라 생각합니다. 혹 질문사항이 있으면 카페나 이메일로 글을 남겨주시길 바랍니다. (카페, 이메일 정보는 문서의 마지막에 있습니다)

```

93 `ifdef RUN_MODE
94     parameter BLOCK_SIZE = 10'd256;
95     parameter BLOCK_MAX = 14'd8191;
96 `elsif SIM_MODE
97     parameter BLOCK_SIZE = 10'd256;
98     parameter BLOCK_MAX = 14'd3;
99 `endif
100
101 parameter test_data1 = 128'h33333333_33333333_33333333_33333333 ;
102 parameter test_data2 = 128'hcccccccc_cccccccc_cccccccc_cccccccc ;
103 parameter test_data3 = 128'h55555555_55555555_55555555_55555555 ;
104 parameter test_data4 = 128'haaaaaaaa_aaaaaaaaaaaaaaaaa ;

```

- ✓ 라인 93 - 99 : ddr_test 모듈에는 2가지 모드가 있습니다. Simulation 진행을 위한 SIM_MODE와 Bitstream 생성 을 위한 RUN_MODE가 있습니다. SIM_MODE는 4개의 Block만 테스트를 진행합니다. RUN MODE는 전체영역을 진행합니다. SIM MODE에서 4개의 Block만 진행하는 이유는, ddr_test 를 Simulation 한 후에 최종적으로 Top Module(Arty35Top, Arty35TTop_frame)에서도 Simulation을 진행합니다. 그런데 앞에서 sim_tb_top 에서도 확인하였듯이 simulation 시간이 너무 많이 소요됩니다. 그래서 시간을 단축하고자 SIM MODE에서는 4 Block만 진행합니다. 이 값의 정의는 define.v 파일에 있습니다.
- ✓ 라인 101 - 104 : Test Data를 정의합니다.

```

106 // State Parameter
107 parameter      M_IDLE          = 3'd0;
108 parameter      M_WRITE1        = 3'd1;
109 parameter      M_WRITE2        = 3'd2;
110 parameter      M_READ1         = 3'd3;
111 parameter      M_READ2         = 3'd4;
112
113 // -----
114 // State Control
115 reg      [2:0]  ddr_state;
116 wire      s_idle    = (ddr_state==M_IDLE ) ? 1'b1 : 1'b0;
117 wire      s_write1  = (ddr_state==M_WRITE1) ? 1'b1 : 1'b0;
118 wire      s_write2  = (ddr_state==M_WRITE2) ? 1'b1 : 1'b0;
119 wire      s_read1   = (ddr_state==M_READ1 ) ? 1'b1 : 1'b0;
120 wire      s_read2   = (ddr_state==M_READ2 ) ? 1'b1 : 1'b0;

```

- ✓ 라인 106 - 120 : SM을 정의합니다. IDLE 상태에서 btn0의 입력이 발생하면 WRITE1 상태로 이동합니다. WRITE1 상태에서는 mig7_write의 입력 값들을 설정한 후에 WRITE2 상태로 이동합니다. WRITE2 상태에서는 mig7_write의 wdone이 “H”가 되면, Block 카운트 값과 BLOCK_MAX 값을 비교해서 Block 카운트 값이 작으면 다시 WRITE1 상태로 이동하고, 이 값들이 같으면 IDLE 상태로 이동합니다. IDLE 상태에서 btn1의 입력이 발생하면 READ1 상태로 이동합니다. READ1 상태에서는 mig7_read의 입력 값들을 설정한 후에 READ2 상태로 이동합니다. READ2 상태에서는 mig7_read의 rdone이 “H”가 되면, Block 카운트 값과 BLOCK_MAX 값을 비교해서 IDLE or READ1 상태로 이동합니다. READ2 상태에서는 메모리에서 read 한 값과 TEST Data 값을 비교해서 Error를 카운트 합니다. Read 가 완료되면 Error 카운트 값에 따라서 LED0의 파란색, 빨간색 LED를 점등합니다.

나머지 소스들의 설명은 생략하도록 하겠습니다.

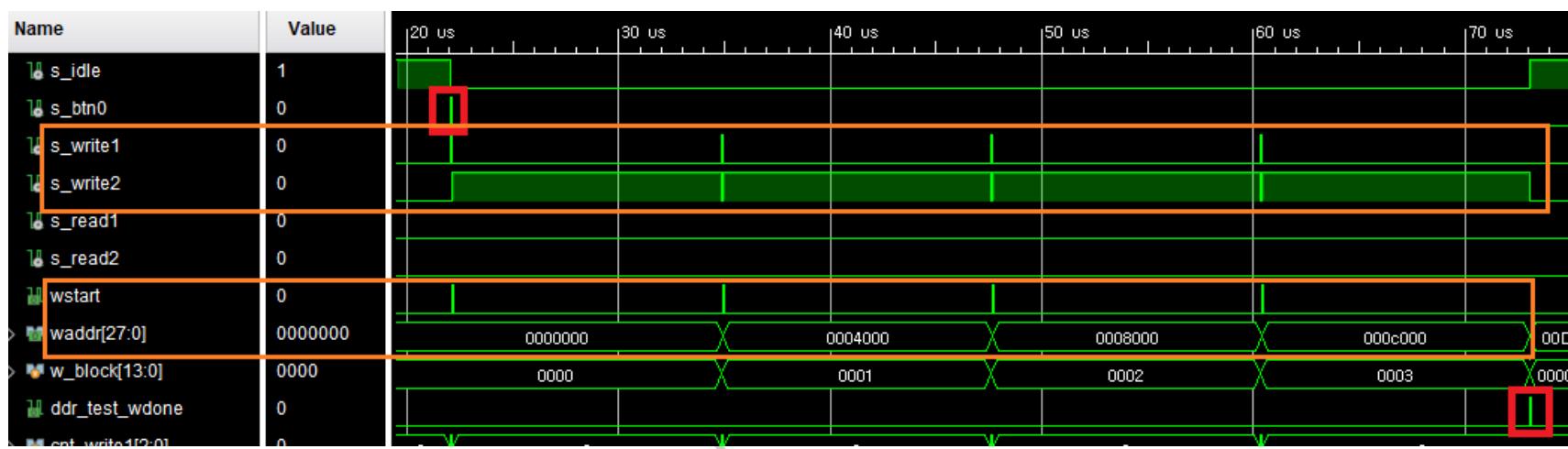
5.4.5 ddr_test 모듈 Simulation

ddr_test 모듈의 Simulation은 데이터가 제대로 Write/Read 되는지를 확인하는 목적이 아니라, mig7_write, mig7_read 모듈의 입력값들이 제대로 설정되는지를 확인하는데 목적이 있습니다. 데이터 값을 확인하는 것은 Top Module Simulation 할 때, 메모리 모델을 추가해서 진행하도록 합니다.

ddr_test 모듈의 Test bench는 tb_ddr_test() 입니다. tb_ddr_test 모듈은 write, read를 나누어서 simulation을 진행합니다. write 를 테스트 하기 위해서는 파일의 50 - 88 라인을 사용하고 (라인 92 - 142 주석처리), read를 테스트 하기 위해서는 92 - 142라인을 사용합니다. (라인 50 - 88 라인 주석처리)

Simulation Sources에 3개의 파일(tb_ddr_test.v, ddr_test.v define.v)을 추가하고 tb_ddr_test 모듈을 Top Module로 지정하고 0.1초 동안 Simulation을 진행합니다.

아래는 Write 테스트 결과를 보여줍니다. (라인 92 - 142 주석처리)



s_btn0가 Active 되면 write를 시작합니다. s_write1, s_write2가 총 4번(4-blocks) 발생합니다. mig7_write 입력신호 wstart, waddr 값이 정상적으로 생성됨을 알 수 있습니다. 4-blocks 후에는 ddr_test_wdone 이 Active가 되고 s_idle 이 Active 됨을 알 수 있습니다.

아래는 Read 테스트 결과를 보여줍니다. (라인 50 - 88 주석처리)



s_btn1이 Active 되면 read를 시작합니다. s_read1, s_read2가 총 4번(4-blocks) 발생합니다. mig7_read 입력신호 rstart, raddr 값이 정상적으로 생성됨을 알 수 있습니다. 4-blocks 후에는 ddr_test_rdone 이 Active가 되고 s_idle 이 Active 됨을 알 수 있습니다

5.5 mig_top Module 구현

mig_top 모듈은 지금까지 구현했던 mig7_write, mig7_read, ddr_test 모듈을 모두 포함하는 Memory 관련 Top 모듈입니다. mig_top 모듈은 simulation은 진행하지 않고, 간단하게 중요한 코드를 중심으로 설명하도록 하겠습니다. “5.1.3 코드 구조”를 참조하시길 바랍니다.

```

115  wire          ui_clk           ;
116  wire          ui_clk_sync_rst  ;
117  wire          rst_du = ~ui_clk_sync_rst;
118
119  wire          clk_du ;
120  BUFG      bufg1  (.O(clk_du), .I(ui_clk));

```

- ✓ 라인 115 - 116 : ui_clk, ui_clk_sync_rst 신호는 ddr_controller에서 생성해주는 신호로 User Interface 모듈에서 사용하는 clock, reset 신호입니다. 우리는 지금까지 ddr_test, mig7_write, mig7_read 모듈에서 이 신호를 사용하였습니다.
- ✓ 라인 117 : ui_clk_sync_rst을 인버팅해서 reset 신호로 사용합니다.
- ✓ 라인 119 - 120 : ui_clk 신호를 Clock 라인으로 사용하기 위해서 BUFG 를 사용합니다.

```

242  reg          ddr_rw_flag ; // 0 : write, 1 : read
243  always @(posedge clk_du or negedge rst_du)
244 begin
245     if(!rst_du)    ddr_rw_flag <= 1'b0;
246     else          ddr_rw_flag <= mig_wstart ? 1'b0 : mig_rstart ? 1'b1 : ddr_rw_flag ;
247 end
248
249
250  wire [27:0] app_addr = ~ddr_rw_flag ? wapp_addr : rapp_addr ;
251  wire [2:0]   app_cmd  = ~ddr_rw_flag ? wapp_cmd  : rapp_cmd  ;
252  wire        app_en   = ~ddr_rw_flag ? wapp_en   : rapp_en  ;

```

- ✓ 라인 242 - 247 : ddr의 read / write flag를 생성합니다. 이는 ddr_controller의 신호들 중에 app_addr, app_cmd, app_en 신호들은 read/write 에 따라 별도로 신호가 구분되어 있지 않고 공통으로 사용합니다. 그런데 우리가 설계한 코드에는 app_addr, app_cmd, app_en 신호를 mig7_write, mig7_read에서 구분되어 사용하였습니다. 따라서 이를 하나의 read/write 에 따라 하나의 신호를 생성하기 위함입니다.
- ✓ 라인 250 - 252 : ddr_controller에 입력되는 app_addr, app_cmd, app_en 신호를 ddr_rw_flag에 따라서 생성합니다.

나머지 코드들은 단순히 모듈들을 연결해 주는 코드들입니다.

5.6 Top Module 구현

이번 장에서는 Top Module 을 구현합니다. 앞에서 모든 부분들을 구현했기 때문에 Top Module 에서는 코드들이 많지 않습니다. Top Module은 Arty35Top.v 입니다. 중요 코드들 중심으로 설명합니다.

```

73 `ifdef RUN_MODE
74     input      mCLOCK ;
75     output     init_calib_complete ;
76 `elsif SIM_MODE
77     input      sys_clk_i      ;
78     input      clk_ref_i      ;
79     output    init_calib_complete ;
80     output    tg_compare_error ;
81     output    clk_du          ;
82     output    rst_du          ;
83 `endif

```

- ✓ 라인 73 - 83 : ddr_test 모듈에서도 설명했듯이 2가지 모드로 진행됩니다. 다음 장에서 언급하겠지만 Top Module simulation은 “sim_tb_top.v” 파일을 참조해서 진행됩니다. sim_tb_top.v 에서는 sys_clk_i, clk_ref_i를 생성해 줍니다. 그러나 RUN_MODE로 Bitstream을 생성할 때에는 Top Module에서 Clock Generator를 이용하여 mCLOCK(100Mhz) 입력으로 sys_clk_i, clk_ref_i 를 생성합니다.

```

112 `ifdef RUN_MODE
113     wire      sys_clk_i ;      // ddr system clock
114     wire      clk_ref_i ;      // ddr reference clock
115     wire      pll_locked ;
116     clk_gen  clk_gen (
117         .clk_out1  (sys_clk_i      ),      // output clk_out1, 100 Mhz
118         .clk_out2  (clk_ref_i      ),      // output clk_out2, 200 Mhz
119
120         .reset     (1'b0          ),      // input reset
121         .locked   (pll_locked    ),      // output locked
122         .clk_inl  (mCLOCK        )
123     );
124 `endif

```

- ✓ 라인 112 - 124 : RUN_MODE시 mCLOCK 입력으로 sys_clk_i, clk_ref_i clock을 생성합니다.

5.7 Top Module simulation

Top Module simulation을 진행합니다. Test bench는 tb_arty35Top.v 입니다. tb_arty35Top.v 는 sim_tb_top.v 파일의 내용을 그대로 가지고 왔습니다. sim_tb_top.v의 라인 432 - 505 까지의 “FPGA Memory Controller” 부분을 빼고 코드의 내용을 모두 가지고 옵니다. 그리고 “FPGA Memory Controller” 가 있던 곳에 Arty35Top 모듈을 추가합니다.

추가된 코드 중심으로 설명합니다. (라인 369 - 470)

```

369 wire LED0_B ;
370 wire LED0_R ;
371 wire LED5, LED6 ;
372
373 wire clk_du;
374 wire rst_du;
375 wire ddr_test_wdone, ddr_test_rdone;
376
377 // State Parameter
378 parameter M_IDLE      = 2'd0;
379 parameter M_WRITE     = 2'd1;
380 parameter M_READ      = 2'd2;
381 parameter M_DONE      = 2'd3;
382
383 // -----
384 // State Control
385 reg [1:0] m_state;
386 wire s_idle  = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
387 wire s_write = (m_state==M_WRITE) ? 1'b1 : 1'b0;
388 wire s_read  = (m_state==M_READ ) ? 1'b1 : 1'b0;
389 wire s_done  = (m_state==M_DONE ) ? 1'b1 : 1'b0;

```

- ✓ 라인 369 - 375 : Arty35Top 모듈의 출력 신호
- ✓ 라인 377 - 389 : SM 상태를 정의합니다.

```

391 reg [10:0] cntw;
392 always @ (posedge clk_du or negedge rst_du)
393 begin
394     if (!rst_du)    cntw <= 11'b0;
395     else           cntw <= ~s_write ? 11'b0 : (cntw==11'd2020) ? 11'd2020 : cntw+1'b1;
396 end
397
398 reg btn0;
399 always @ (posedge clk_du or negedge rst_du)
400 begin
401     if (!rst_du)    btn0 <= 1'b0;
402     else           btn0 <= (cntw==11'd2000) ? 1'b1 : 1'b0;
403 end
404
405 reg [10:0] cntr;
406 always @ (posedge clk_du or negedge rst_du)
407 begin
408     if (!rst_du)    cntr <= 11'b0;
409     else           cntr <= ~s_read ? 11'b0 : (cntr==11'd2020) ? 11'd2020 : cntr+1'b1;
410 end
411
412 reg btnl;
413 always @ (posedge clk_du or negedge rst_du)
414 begin
415     if (!rst_du)    btnl <= 1'b0;
416     else           btnl <= (cntr==11'd2000) ? 1'b1 : 1'b0;
417 end

```

- ✓ 라인 391 - 396, 405 - 410 : 내부에서 사용하는 counter입니다. cntw는 btn0를 생성하기 위한 카운터, cntr은

btn1을 생성하기 위한 카운터입니다.

- ✓ 라인 398 - 403, 412 - 417 : cntw, cntr 값에 따라서 btn0, btn1 Active 신호를 생성합니다.

```

420   always @(posedge clk_du or negedge rst_du)
421 begin
422   if(!rst_du) begin
423     m_state <= 2'b0;
424   end
425   else begin
426     m_state <= (s_idle & init_calib_complete) ? M_WRITE : 
427                 (s_write & ddr_test_wdone) ? M_READ : 
428                 (s_read & ddr_test_rdone) ? M_DONE : 
429                 (s_done) ? M_IDLE : m_state ;
430   end
431 end

```

- ✓ 라인 420 - 431 : SM 상태 전이를 구현합니다. IDLE 상태에서는 메모리 초기화가 끝나면(init_calib_complete=1) WRITE 상태가 됩니다. btn0가 Active 되면 write가 진행되고 완료되면 ddr_test_wdone 신호가 Active 되고, READ 상태가 됩니다. btn1이 Active 되면 read가 진행되고 완료되면 ddr_test_rdone 신호가 Active 되고 DONE 상태가 됩니다. DONE 상태에서는 1-clock 후에 IDLE 상태가 됩니다.

```

434 Arty35Top Arty35Top(
435   .BTN0 (btn0),
436   .BTN1 (btn1),
437
438   .LEDO_B (LEDO_B),
439   .LEDO_R (LEDO_R),
440   .LED5 (LED5),
441   .LED6 (LED6),
442
443   .ddr3_addr (ddr3_addr_fpga),
444   .ddr3_ba (ddr3_ba_fpga),
445   .ddr3_cas_n (ddr3_cas_n_fpga),
446   .ddr3_ck_n (ddr3_ck_n_fpga),
447   .ddr3_ck_p (ddr3_ck_p_fpga),
448   .ddr3_cke (ddr3_cke_fpga),
449   .ddr3_ras_n (ddr3_ras_n_fpga),
450   .ddr3_reset_n (ddr3_reset_n_fpga),
451   .ddr3_we_n (ddr3_we_n_fpga),
452   .ddr3_dq (ddr3_dq_fpga),
453   .ddr3_dqs_n (ddr3_dqs_n_fpga),
454   .ddr3_dqs_p (ddr3_dqs_p_fpga),
455   .ddr3_cs_n (ddr3_cs_n_fpga),
456   .ddr3_dm (ddr3_dm_fpga),
457   .ddr3_odt (ddr3_odt_fpga),
458
459   .sys_clk_i (sys_clk_i),
460   .clk_ref_i (clk_ref_i),
461   .init_calib_complete (init_calib_complete),
462   .tg_compare_error (tg_compare_error),
463   .sys_rst (sys_rst),
464
465   .clk_du (clk_du),
466   .rst_du (rst_du),
467   .ddr_test_wdone (ddr_test_wdone),
468   .ddr_test_rdone (ddr_test_rdone)
469 );

```

Arty35Top 모듈을 추가합니다.

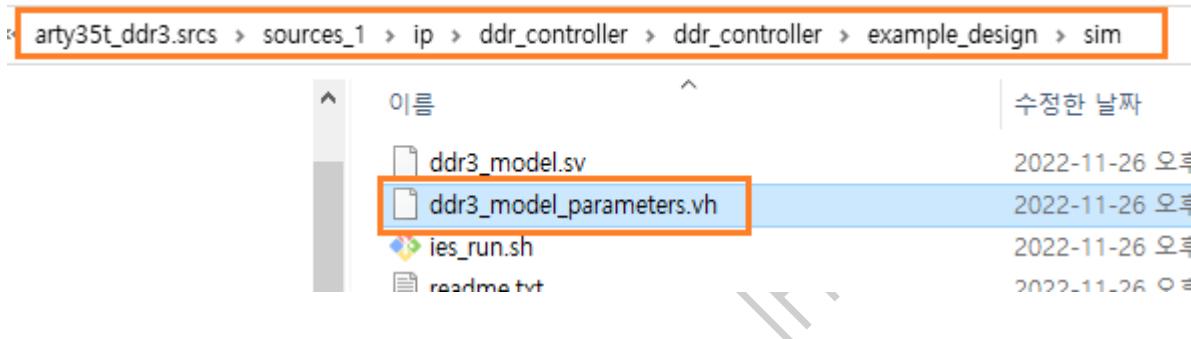
```

538     initial
539     begin : Logging
540         fork
541             begin : calibration_done
542                 wait (init_calib_complete);
543                 $display("Calibration Done");
544                 #500000000.0;
545                 if (!tg_compare_error) begin
546                     $display("TEST PASSED");
547                 end
548                 else begin
549                     $display("TEST FAILED: DATA ERROR");
550                 end
551                 disable calib_not_done;
552             //$/finish;
553         end

```

- ✓ 라인 552 : sim_tb_top 모듈은 메모리 초기화가 끝나고 일정시간이 지나면 simulation이 종료되게 됩니다. 종료 없이 계속해서 진행하기 위하여 552 라인을 주석 처리합니다.

다음으로 ddr3_model_parameters.vh 파일을 수정합니다.



파일을 열어서 “MEM_BITS” 파라미터를 찾습니다. 총 4개가 있습니다. 이 값이 15로 되어 있는데 이 값을 20으로 변경합니다. 이 값은 DDR Memory 의 Access 시에 최대 Address를 설정합니다. 15로 설정되어 있으면 0x000_8000 까지만 Access 할 수 있습니다. 다음장에서 프레임 버퍼를 구현하고 Simulation 할 때 Access 영역이 대략적으로 0x00E_0000까지입니다. 따라서 이 값을 20으로 변경합니다.

```

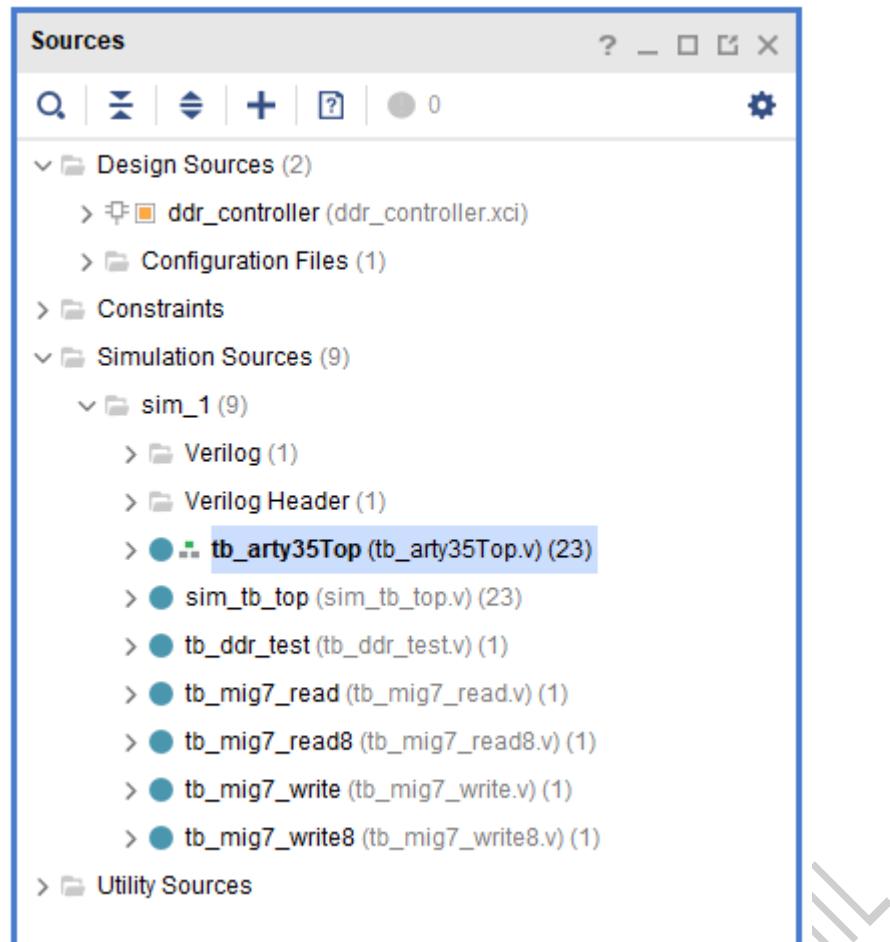
407     // Size Parameters
408     parameter BA_BITS          =      3; // Set this p
409     parameter MEM_BITS          =      20; // Set this p
410     parameter AP                =      10; // the address
411     parameter BC                =      12; // the address
412     parameter BL_BITS          =      3; // the number
413     parameter BO_BITS          =      2; // the number

```

1295, 2250, 3220 라인도 20으로 변경합니다.

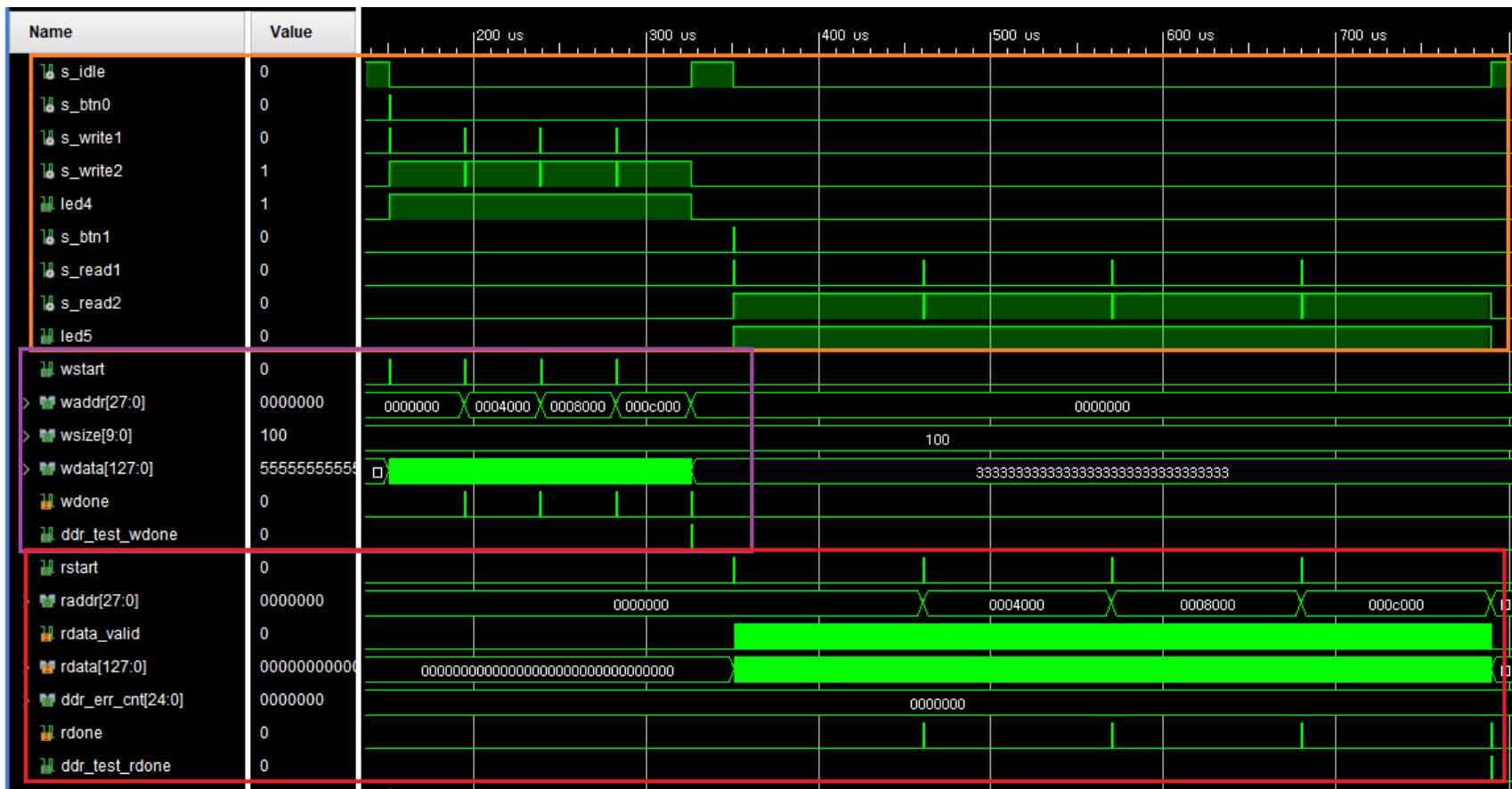
이제 simulation을 진행합니다. Simulation Sources에 3개의 파일 (tb_arty35Top.v Arty35Top.v, mig_top.v)을 추가하고 tb_arty35Top을 Top Module로 지정하고 simulation을 진행합니다.

아래 그림은 지금까지 진행되었던 내용, 추가되었던 파일들을 보여줍니다. 참고하시길 바랍니다.



tb_arty35Top - Arty35Top - mig_top - ddr_test 내의 신호들을 Wave 윈도에 추가합니다. Tcl Console에 “run 0.8ms”를 입력해서 0.8ms 동안 simulation을 진행합니다.

아래 그림은 ddr_test 모듈 신호들의 파형을 보여줍니다. ([코드 수정으로 아래 파형에서 led4 ->led5, led5->led6 으로 변경 되었습니다](#))

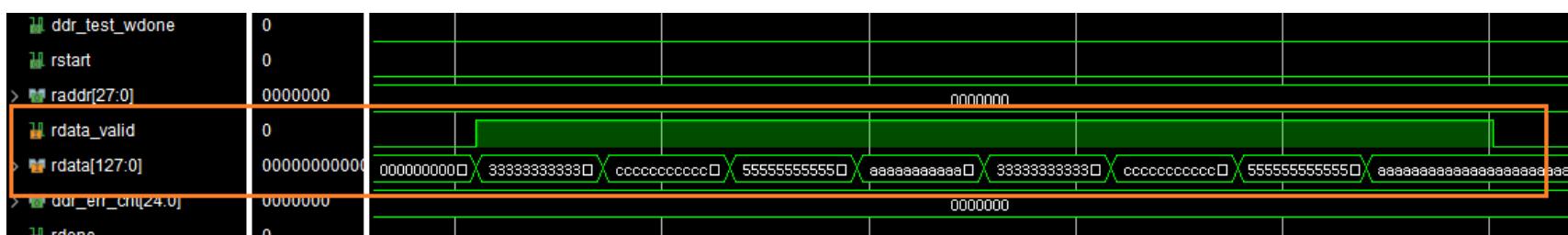


주황색으로 표시된 부분은 SM(State Machine)을 나타냅니다. s_btn0가 Active가 될 때, write 를 진행합니다. 4-blocks 진행 후 종료하고 IDLE 상태가 됩니다. s_btn1이 Active 되면 read 를 진행합니다. 4-blocks 진행 후 종료하고 IDLE 상태가 됩니다.

보라색으로 표시된 부분은 write 부분입니다. wstart, waddr, wdata 신호가 발생하고 4-blocks 후에 ddr_test_wdone 신호가 Active 됩니다.

빨간색으로 표시된 부분은 read 부분입니다. rstart, raddr, rdata_valid, rdata 신호가 정상적으로 나옵니다. ddr_err_cnt 는 항상 “0”임을 알 수 있습니다. ddr_err_cnt는 read 시에 read 한 데이터와 write 했던 데이터를 비교해서 에러를 검출합니다. ddr_err_cnt = 0 은 모든 데이터가 동일하다는 의미입니다.

rdata 부분을 확대해서 보겠습니다.



rdata의 값이 차례대로 33333~, ccccc~, 55555~, aaaaaa~ 가 됨을 알 수 있습니다. 이는 write 했던 데이터와 동일한 값입니다.

5.8 Bitstream 생성

이번 절에서는 Bitsteam을 생성하고 그 결과를 보드에 다운로드해서 확인합니다. define.v 파일에서 SIM_MODE를 주석처리하고 RUN_MODE를 주석해제 합니다.

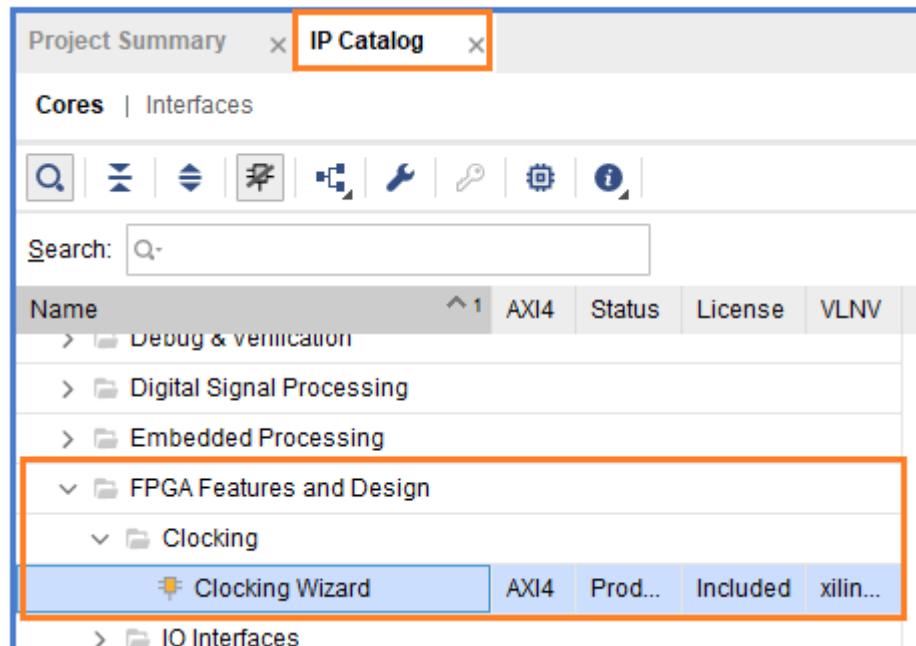
```

4
5 //`define SIM_MODE
6 `define RUN_MODE
7

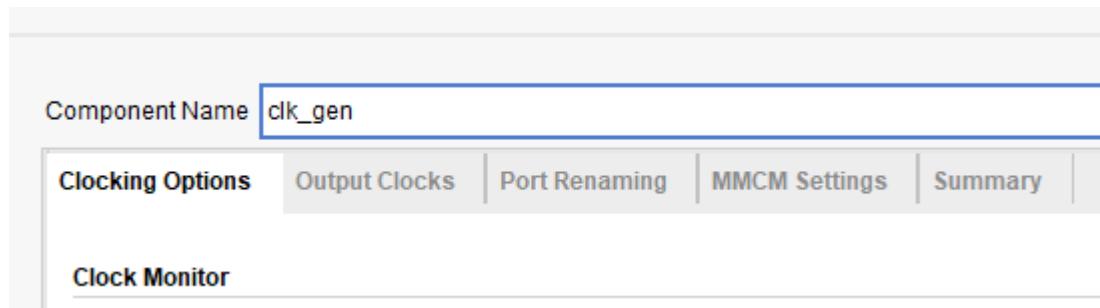
```

Project Manage - Design Sources 에 Arty35Top.v, mig_top.v, ddr_test.v, mig7_write.v, mig7_write8.v, mig7_read.v, mig7_read8.v 파일을 추가합니다.

IP Catalog - FPGA Features and Design - Clocking - Clock Wizard를 실행합니다.



Component Name 을 “clk_gen” 으로 입력합니다.



Clocking Options 탭의 하단에서 Input Clock이 100MHz로 설정된 것을 확인합니다.

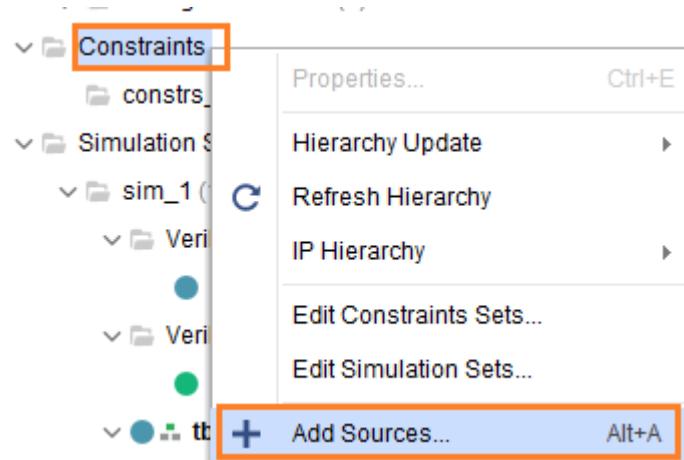
Input Clock Information

| | Input Clock | Port Name | Input Frequency(MHz) | | Jitter Options | Input Jitter | So |
|-------------------------------------|-------------|-----------|----------------------|---|----------------|--------------|-----|
| <input checked="" type="checkbox"/> | Primary | clk_in1 | 100.000 | <input type="button" value="X"/> 10.000 - 800.000 | UI | 0.010 | Sir |
| <input type="checkbox"/> | Secondary | clk_in2 | 100.000 | 60.000 - 120.000 | | 0.010 | Sir |

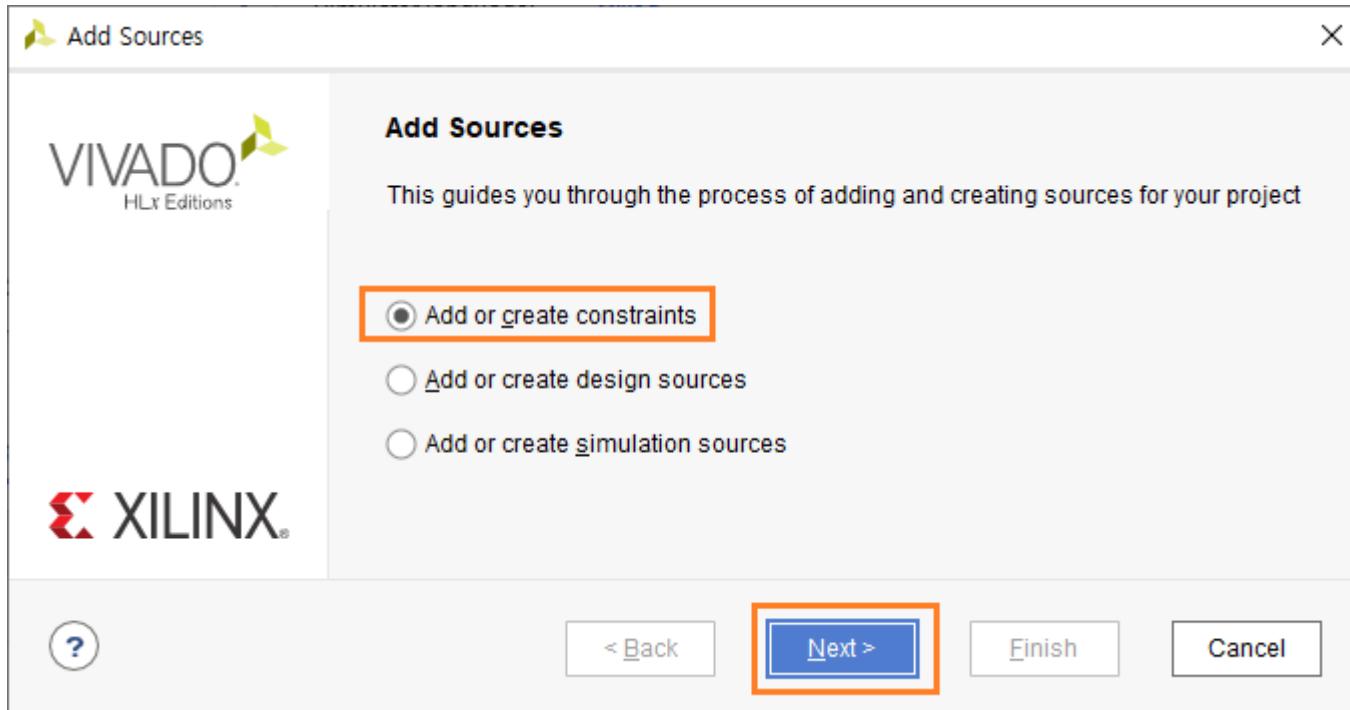
Output Clocks 탭에서 clk_out1 : 100 Mhz, clk_out2 : 200 Mhz 입력하고 하단의 OK 버튼을 클릭해서 Clock을 생성합니다. Generate 버튼을 클릭합니다.

| Output Clock | Port Name | Output Freq (MHz) | | Phase (degrees) | |
|--|-----------|-------------------|--|-----------------|--|
| | | Requested | Actual | Requested | Actual |
| <input checked="" type="checkbox"/> clk_out1 | clk_out1 | 100.000 | <input type="button" value="X"/> 100.000 | 0.000 | <input type="button" value="X"/> 0.000 |
| <input checked="" type="checkbox"/> clk_out2 | clk_out2 | 200 | <input type="button" value="X"/> 200.000 | 0.000 | <input type="button" value="X"/> 0.000 |
| <input type="checkbox"/> clk_out3 | clk_out3 | 100.000 | N/A | 0.000 | N/A |

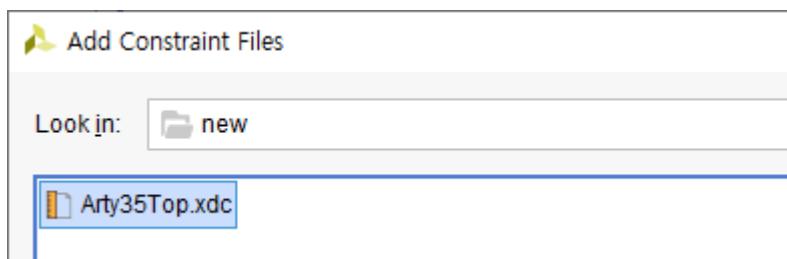
Project Manager - Constraints에 xdc 파일을 추가합니다. Constraints 우클릭 - Add Sources .. 를 클릭합니다.



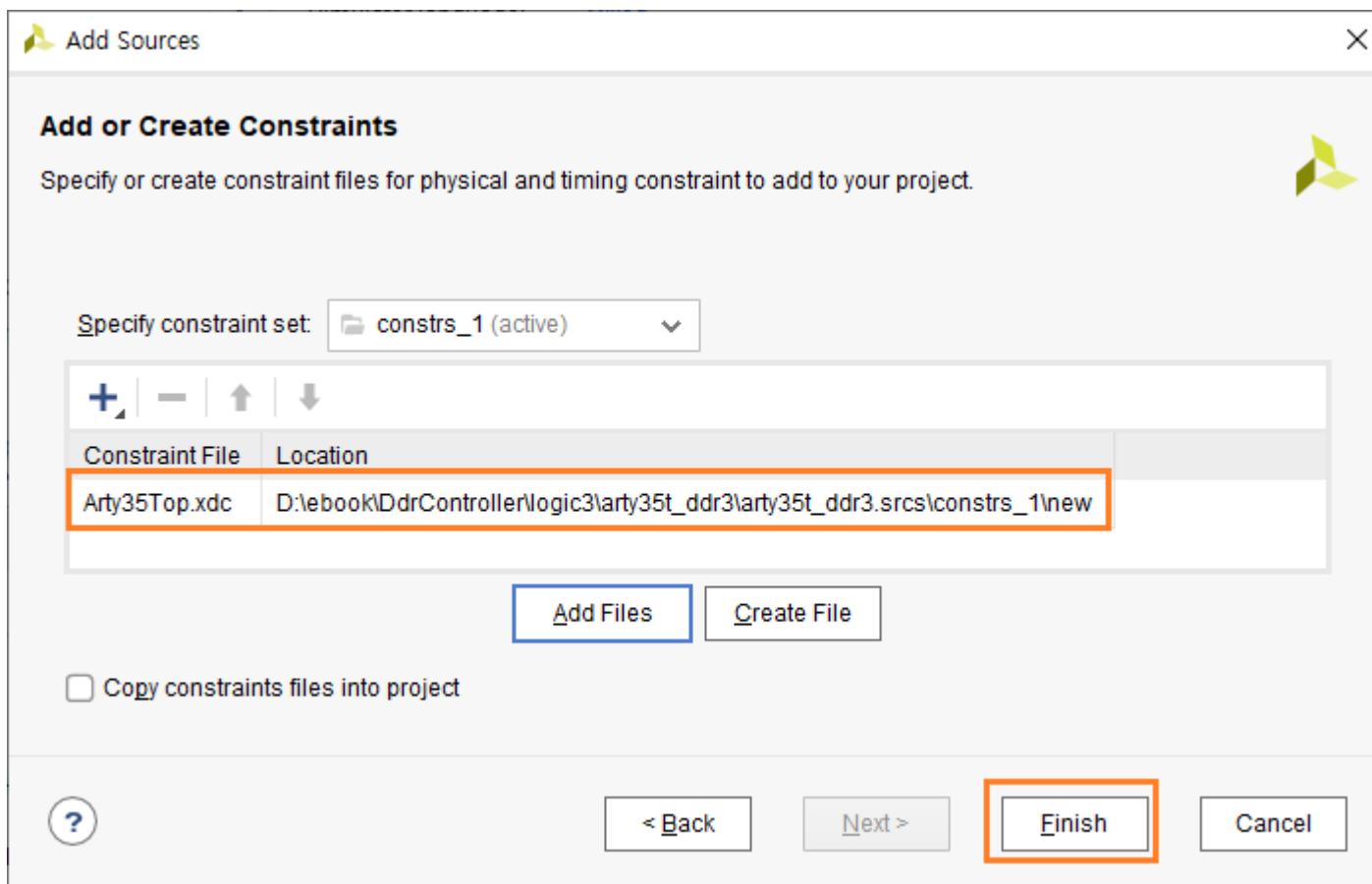
“Add or create constraints”을 선택하고 Next 버튼을 클릭합니다.



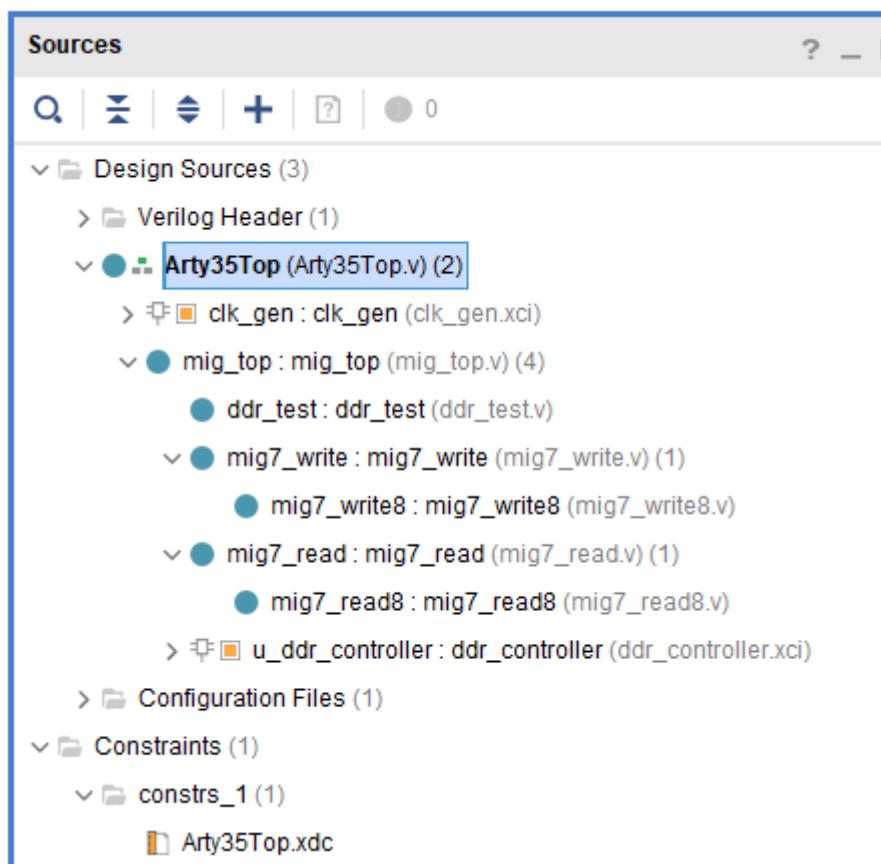
Add Files 버튼을 클릭해서 “arty35t_ddr3.srcts\constrs_1\new\Arty35Top.xdc” 파일을 추가합니다.



Finish 버튼을 클릭합니다.



모든 파일이 준비되었습니다. Arty35Top 모듈이 Top Module로 지정되지 않았으면 Top Module로 지정합니다.

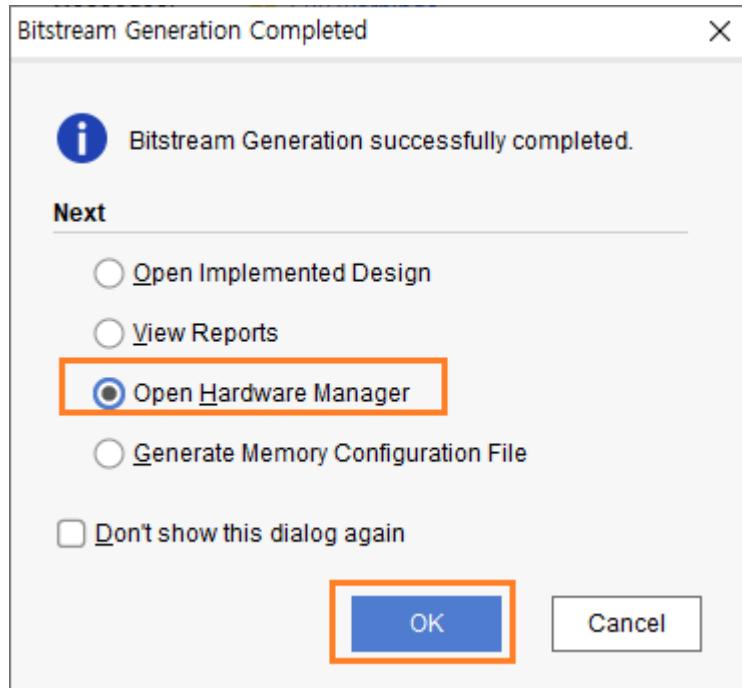


PROGRAM AND DEBUG - Generate Bitstream 을 클릭해서 Bitstream을 생성합니다.

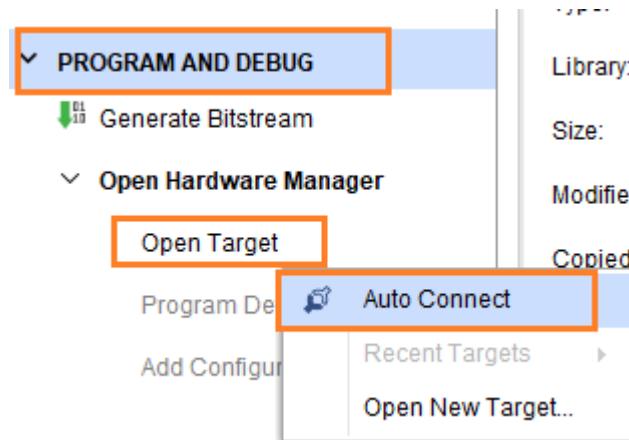
/HIL

5.9 Bitstream 다운로드 & 확인

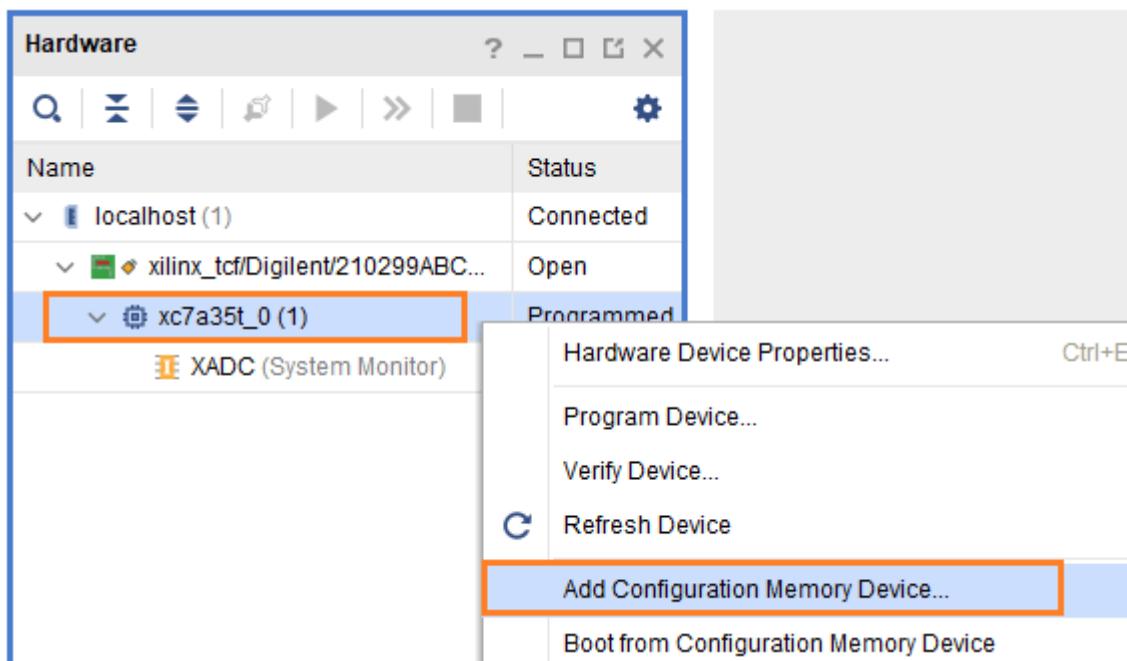
에러가 없이 Bitstream 이 생성되면 아래와 같은 윈도가 나타납니다. Bitstream 파일을 다운로드 하기 위하여 보드와 JTAG-HS2(3)을 연결하고 보드 전원을 인가합니다. “Open Hardware Manager”을 선택하고 OK 버튼을 클릭합니다.



왼쪽의 Flow Navigator - PROGRAM AND DEBUG - Open Hardware Manager - Open Target - Auto Connect 를 클릭합니다.

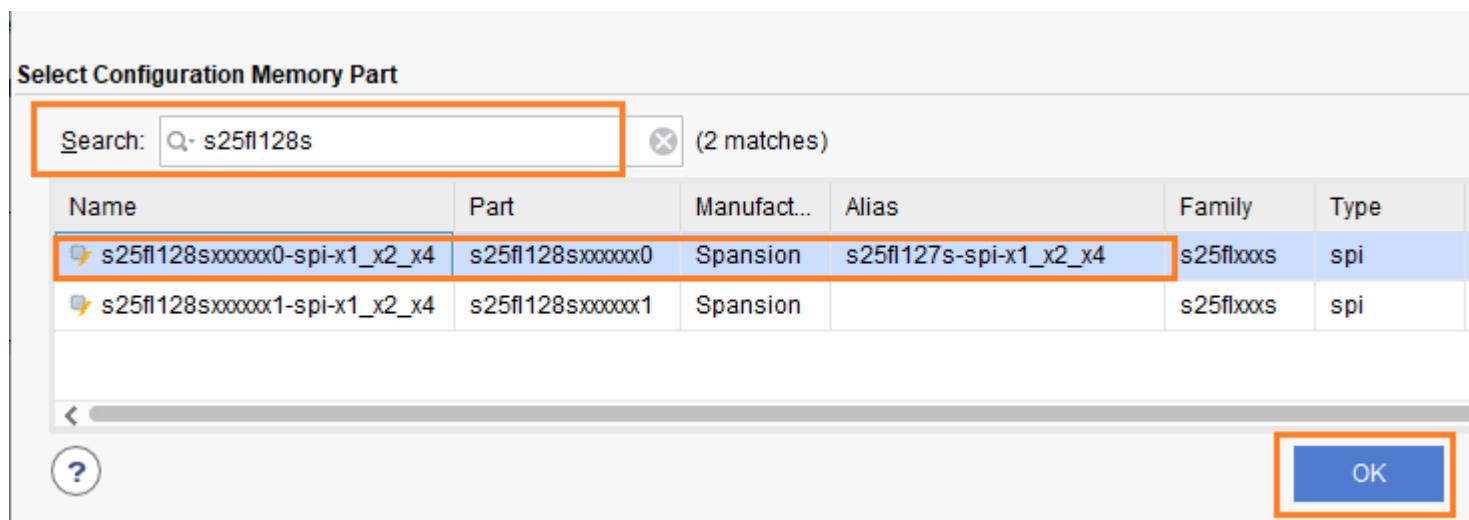


Hardware 윈도에서 xc7a35t_0 선택 후 우클릭 해서 “Add Configuration Memory Device...”를 클릭합니다.

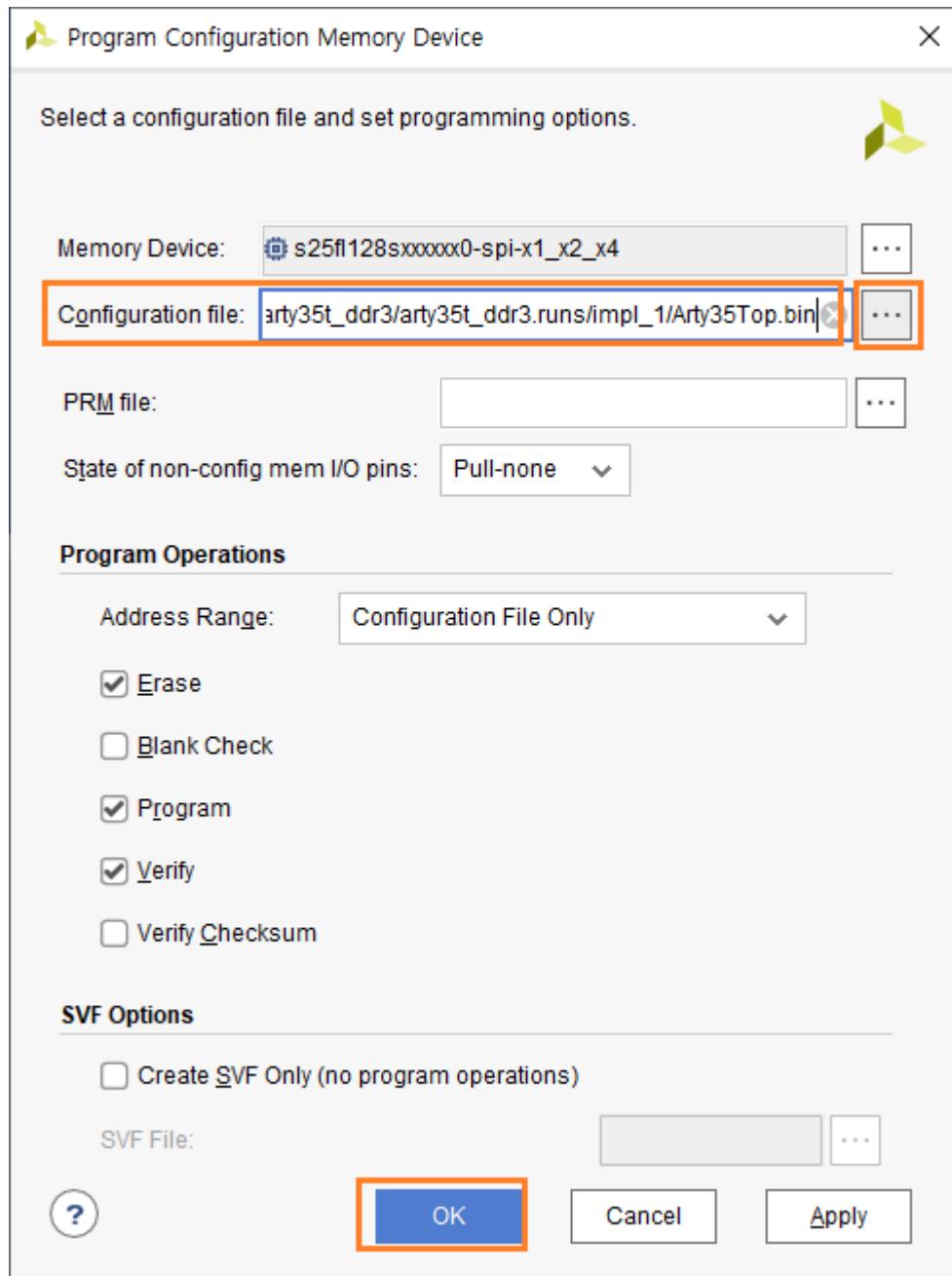


Search에 “s25fl128s”을 입력하고 리스트 중에 첫번째 것을 선택하고 OK 버튼을 클릭합니다.

(보드 버전에 따라 mt25ql128 인 경우도 있습니다)



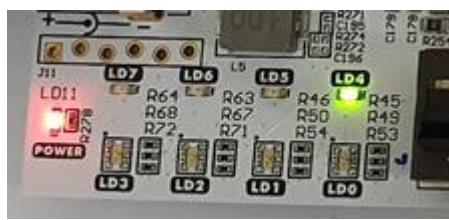
Program Configuration Memory Device 윈도에서 Configuration files : 우측 버튼을 클릭해서 Bitstream 파일 (Arty35Top.bin) 을 선택합니다. 이 파일은 “arty35t_ddr3.runs\impl_1” 폴더 안에 있습니다. OK 버튼을 클릭하면 Bitstream을 보드에 다운로드 합니다.



혹시 “Arty35Top.bin” 파일이 보이지 않으면, 프로젝트 설정을 해 주어야 합니다. “3.2 Memory IP 생성”에서 관련내용을 참조하여 속성을 설정한 후, Hardware Manager 윈도를 닫고, Generate Bitstream을 다시 실행해 줍니다.

다운로드가 완료되면 보드의 전원을 껐다가 다시 인가합니다.

전원을 인가하면 잠시 후에 LD4에 불이 들어옵니다. LD4(LED4)는 init_calib_compete 신호 출력과 연결되었습니다. 이 신호는 메모리 초기화가 완료되었음을 알려줍니다.



BTN0을 누르면 LD5가 잠깐 깜박입니다. 이는 메모리 전영역을 write 하는 것을 의미합니다. 다시 BTN1을 누르면 LD6이 깜박입니다. 이는 메모리 전영역을 read 합니다. read 가 완료되면 LD0에 파란색 불이 들어옵니다. 이는 에러가 없다는 의미입니다.

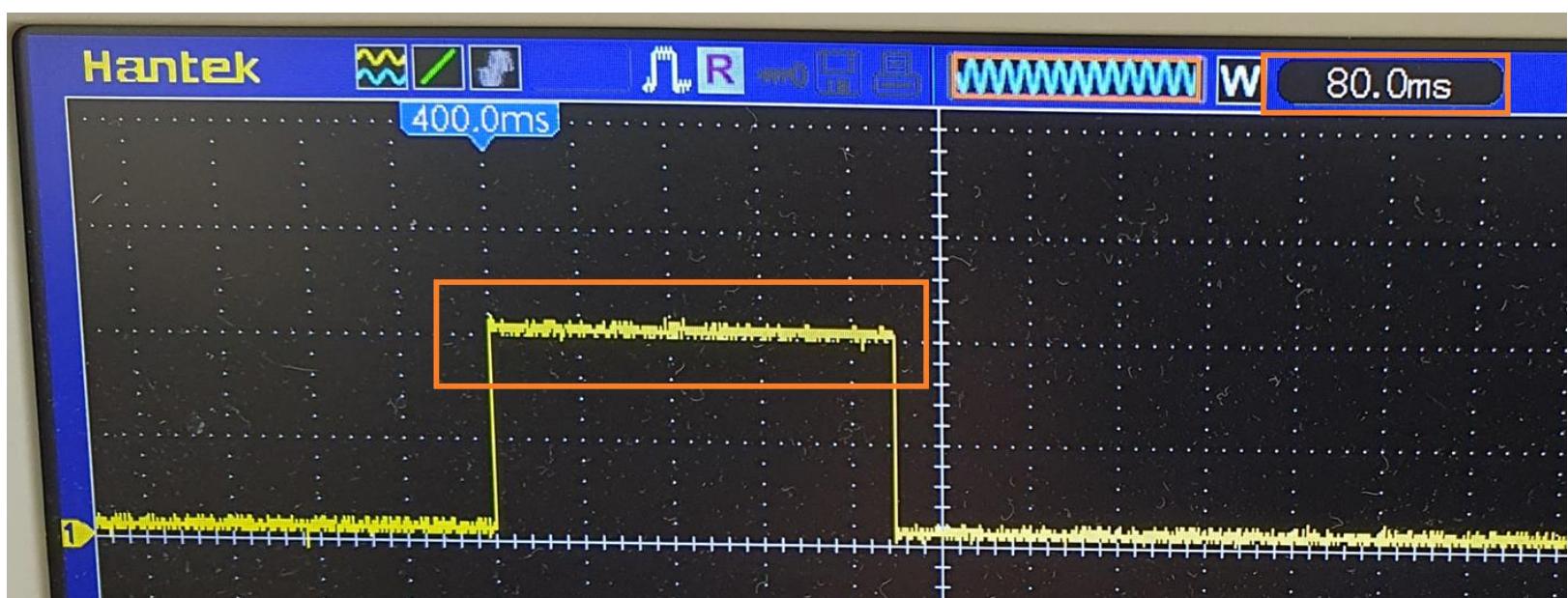


6. DDR3 Memory Access 속도

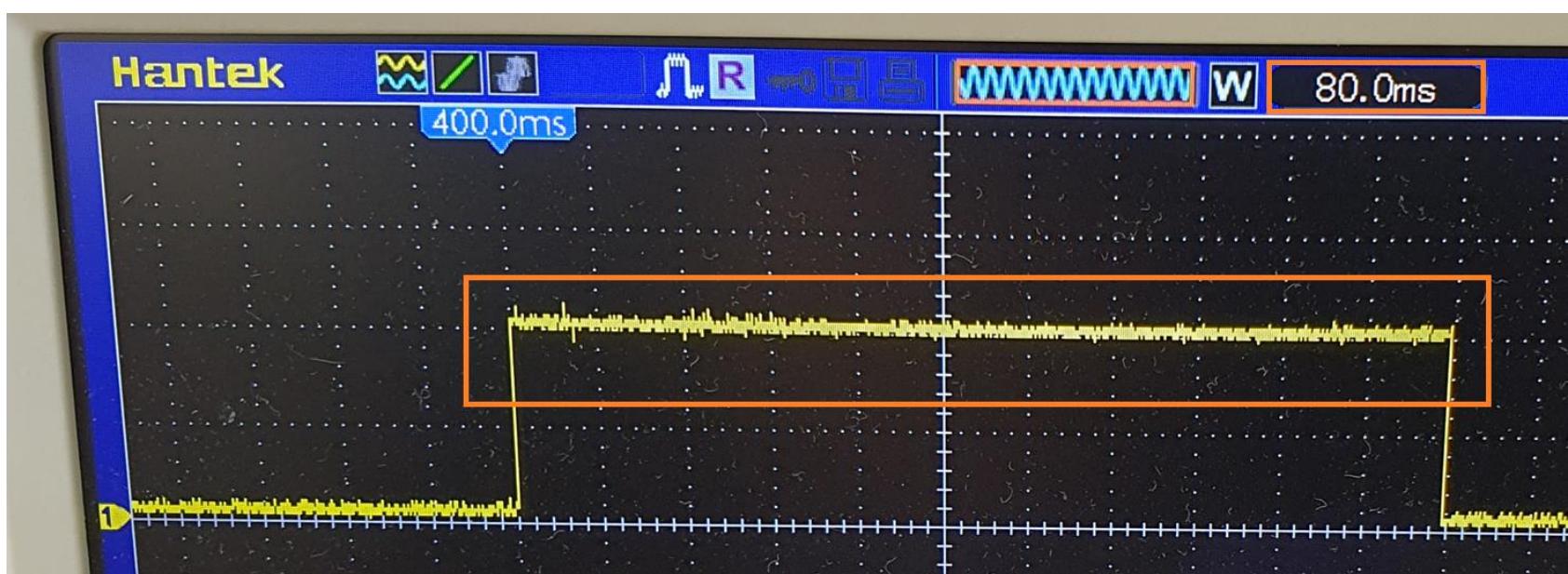
이번장에서는 5장에서 구현된 내용을 바탕으로 read / write 속도를 check하고 속도를 증가시킬 수 있는 방법을 구현합니다. 5장에서는 128bits을 8개씩 write, read 하는 모듈(mig7_write8, mig7_read8)을 구현하고 이 모듈을 이용하여 전영역을 write, read 하는 코드를 구현하였습니다.

먼저 전영역을 write, read하는데 걸리는 시간을 스코프로 측정해 보도록 하겠습니다. 5장에서 구현된 코드는 write 하는 시간에는 LD5가 On되고, read 하는 시간에는 LD6이 On 되도록 구현되었습니다. 따라서 LD5, LD6을 각각 스코프로 확인하면 write, read 하는 시간을 측정할 수 있습니다.

아래 그림은 전영역을 write 하는 시간입니다. BTN0을 누르면서 LD5을 측정하였습니다. 한칸이 80ms 이므로 약 3.5칸으로 360ms 정도 소요됩니다.



아래 그림은 전영역을 read 하는 시간입니다. BTN1을 누르면서 LD6을 측정하였습니다. 약 $11 * 80\text{ms} = 880\text{ms}$ 정도 소요되었습니다.



생각보다 많은 시간이 소요됩니다. 이 시간을 DDR의 속도를 측정하는 단위인 MT/s (Mega Transfer / second)로 환산해 보겠습니다. 사용하는 DDR3은 16bits data width입니다. 따라서 1초에 16bits 단위로 얼마의 전송속도를 나타내는지 측정해 보면 됩니다.

1. write (MT/s)

메모리 전 영역 = $2\text{Gb} = 16\text{bits} \times 128\text{M}$, 전영역을 write 하는 시간이 360ms 이므로 1초로 환산하면, $16\text{bits} \times 128\text{M} \times 1000/360 = 16\text{bits} \times 355\text{M}$, 따라서 355 MT/s 입니다.

2. read (MT/s)

메모리 전 영역 = $2\text{Gb} = 16\text{bits} \times 128\text{M}$, 전영역을 write 하는 시간이 880ms 이므로 1초로 환산하면, $16\text{bits} \times 128\text{M} \times 1000/880 = 16\text{bits} \times 355\text{M}$, 따라서 145 MT/s 입니다.

5장에서 구현된 Memory Interface는 system clock을 325MHz로 사용하고, DDR 모드로 동작하기 때문에 이상적으로 최대값은 $325 \times 2 = 650$ MT/s입니다. 이 값에 비하면 너무 작은 값임을 알 수 있습니다. 이제 다음장에서 속도를 올리고 이상적인 최대값에 근접한 속도를 만들 수 있도록 코드를 수정합니다.

/HIL

6.1 mig7_write8 수정

데이터 전송 속도가 나오지 않는 가장 큰 이유는 작은 단위로 read/write 를 하게 됨으로 잦은 traffic이 발생하기 때문입니다. mig7_write8 (or mig7_read8) 모듈은 128bits x 8 단위를 access 하기 위하여 command를 전송하고 data를 write, read 합니다. mig7_write8 모듈을 반복해서 사용하면, data를 전송하는 시간에 command를 전송하는 등 기타 명령어를 처리하는 시간들이 계속해서 추가되고 이로 인해서 전체 시간이 길어지게 됩니다.

따라서 속도를 올리는 방법은 기본 전송 단위를 크게 하는 것입니다. 본장에서는 128bits x 8, 128bits x 32, 128bits x 64, 128bits x 256, 128bits x 1024 로 코드를 구현하고 전송속도(시간)을 비교하고, 최종적으로 MT/s 를 비교하도록 하겠습니다.

먼저 구현된 mig7_write8 모듈을 분석(복습)하고 이를 바탕으로 mig7_write32(64, 256, 1024)을 구현합니다. mig7_write8은 128bits 씩 총 8개를 전송합니다. address 는 128bits를 전송할 때마다 8씩($128/16=8$) 증가합니다. 따라서 address는 $8 * 8 = 64$ 만큼 증가합니다. (start_address, start_address +8*1 ~ start_address+8*7)

mig7_write32 모듈은 address가 $8*32 = 256$ 까지 증가합니다. (start_address ~ start_address+8*31)

mig7_write8 모듈을 복사해서 이름을 mig7_write32로 변경하고 아래와 같이 코드를 수정합니다.

```

81 reg [5:0] addr_cnt ;
82 always @ (posedge mclk or negedge reset)
83 begin
84     if (!reset)      addr_cnt <= 6'b0;
85     else            addr_cnt <= s_idle ? 6'b0 : (app_rdy & app_wdf_rdy) ? addr_cnt + 1'b1 : addr_cnt;
86 end
87
88 wire [27:0] app_addr = {waddr[27:8], addr_cnt[4:0], 3'b0};

```

- ✓ 라인 81-86 : addr_cnt는 32까지 증가합니다.
- ✓ 라인 88 : app_addr 값은 addr_cnt[4:0] (0 ~ 31) x 8 로 설정됩니다.

```

94
95 wire app_en      = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[5]) ? 1'b1 : 1'b0;
96 wire app_wdf_wren = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[5]) ? 1'b1 : 1'b0;
97 wire app_wdf_end  = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[5]) ? 1'b1 : 1'b0;
98 wire wready       = s_idle ? 1'b0 : (app_rdy & app_wdf_rdy & ~addr_cnt[5]) ? 1'b1 : 1'b0;
99

```

- ✓ 라인 95-98 : app_en ~ wready 신호는 ~addr_cnt[5] (0 ~ 31)까지 active 됩니다.

```

108    always @ (posedge mclk or negedge reset)
109    begin
110        if (!reset)      begin
111            m_state <= 2'b0;
112        end
113        else   begin
114            m_state <= (s_idle & wstart ) ? M_WRITE : :
115            (s_write & app_rdy & (addr_cnt==6'd31)) ? M_DONE : :
116            (s_done ) ? M_IDLE : m_state ;
117        end
118    end

```

- ✓ 라인 115 : addr_cnt=31일 때 종료 됩니다.

mig7_write64, mig7_write256, mig7_write1024 도 동일하게 수정합니다. 자료실의 소스코드를 참조하시길 바랍니다.

6.2 mig7_read8 수정

mig7_read8 모듈을 복사해서 이름을 mig7_read32로 변경하고 아래와 같이 코드를 수정합니다.

```

81    reg      [5:0]  addr_cnt ;
82    always @ (posedge mclk or negedge reset)
83    begin
84        if (!reset)  addr_cnt <= 6'b0;
85        else        addr_cnt <= s_idle ? 6'b0 : app_rdy ? ((addr_cnt==6'd32) ? 6'd32 : addr_cnt + 1'b1) : addr_cnt;
86    end
87
88    wire      [27:0]  app_addr = {raddr[27:8], addr_cnt[4:0], 3'b0};

```

- ✓ 라인 81-86 : addr_cnt는 32까지 증가합니다.
- ✓ 라인 88 : app_addr 값은 addr_cnt[4:0] (0 ~ 31) x 8 로 설정됩니다.

```

93    reg      [5:0]  data_cnt ;
94    always @ (posedge mclk or negedge reset)
95    begin
96        if (!reset)  data_cnt <= 6'b0;
97        else        data_cnt <= s_idle ? 6'b0 : (app_rd_data_valid & app_rd_data_end) ? (data_cnt + 1'b1) : data_cnt;
98    end

```

- ✓ 라인 93 - 98 : data_cnt는 32까지 증가합니다.

```

107   reg          rdata_valid ;
108   always @ (posedge mclk or negedge reset)
109   begin
110       if (!reset)  rdata_valid <= 1'b0;
111       else        rdata_valid <= s_idle ? 1'b0 : (app_rd_data_valid & app_rd_data_end & ~data_cnt[5]) ? 1'b1 : 1'b0;
112   end

```

- ✓ 라인 111 : rdata_valid 는 data_cnt[5]=0, 0 ~ 31까지 active 됩니다.

```

121    always @(posedge mclk or negedge reset)
122    begin
123        if(!reset)      begin
124            m_state <= 2'b0;
125        end
126        else      begin
127            m_state <= (s_idle & rstart
128                        (s_read & app_rd_data_valid & app_rd_data_end & (data_cnt==6'd31)) ? M_READ : 
129                        (s_done
130                        ) ? M_DONE : 
131                        ) ? M_IDLE : m_state ;
132    end

```

- ✓ 라인 128 : data_cnt 값이 31 일 때 종료합니다.

mig7_read64, mig7_read256, mig7_read1024 도 동일하게 수정합니다. 자료실의 소스코드를 참조하시길 바랍니다.

6.3 mig7_write_top 수정

mig7_write_top 모듈을 아래와 같이 수정합니다. (mig7_write 모듈의 이름을 mig7_write_top 으로 변경하였습니다). 주로 수정되는 내용은 address 관련 부분입니다.

```

69 `ifdef DDR3_STEP_128x8
70     parameter ADDR_STEP = 28'd64;
71 `elsif DDR3_STEP_128x32
72     parameter ADDR_STEP = 28'd256;
73 `elsif DDR3_STEP_128x64
74     parameter ADDR_STEP = 28'd512;
75 `elsif DDR3_STEP_128x256
76     parameter ADDR_STEP = 28'd2048;
77 `elsif DDR3_STEP_128x1024
78     parameter ADDR_STEP = 28'd8192;
79 `endif

```

- ✓ 라인 69 - 79 : 각 모듈별로 address step을 설정합니다. DDR3_STEP_128x8 ~ DDR3_STEP_128x1024 는 define.v 파일에 정의합니다.

```

156 `ifdef DDR3_STEP_128x8
157     mig7_write8      mig7_write8 (
158 `elsif DDR3_STEP_128x32
159     mig7_write32    mig7_write32 (
160 `elsif DDR3_STEP_128x64
161     mig7_write64    mig7_write64 (
162 `elsif DDR3_STEP_128x256
163     mig7_write256   mig7_write256 (
164 `elsif DDR3_STEP_128x1024
165     mig7_write1024  mig7_write1024 (
166 `endif
167     .reset          (reset      ),
168     .mclk           (mclk      ),

```

- ✓ 라인 156 - 166 : address step 에 따라 사용하는 module을 설정합니다.

6.4 mig7_read_top 수정

mig7_read_top 모듈을 아래와 같이 수정합니다. (mig7_read 모듈의 이름을 mig7_read_top 으로 변경하였습니다). 주로 수정되는 내용은 address 관련 부분입니다

```

65 `ifdef DDR3_STEP_128x8
66   parameter ADDR_STEP = 28'd64;
67 `elsif DDR3_STEP_128x32
68   parameter ADDR_STEP = 28'd256;
69 `elsif DDR3_STEP_128x64
70   parameter ADDR_STEP = 28'd512;
71 `elsif DDR3_STEP_128x256
72   parameter ADDR_STEP = 28'd2048;
73 `elsif DDR3_STEP_128x1024
74   parameter ADDR_STEP = 28'd8192;
75 `endif

```

- ✓ 라인 65 - 75 : 각 모듈별로 address step을 설정합니다. DDR3_STEP_128x8 ~ DDR3_STEP_128x1024 는 define.v 파일에 정의합니다.

```

145 `ifdef DDR3_STEP_128x8
146   mig7_read8      mig7_read8 (
147 `elsif DDR3_STEP_128x32
148   mig7_read32    mig7_read32 (
149 `elsif DDR3_STEP_128x64
150   mig7_read64    mig7_read64 (
151 `elsif DDR3_STEP_128x256
152   mig7_read256   mig7_read256 (
153 `elsif DDR3_STEP_128x1024
154   mig7_read1024  mig7_read1024 (
155 `endif
156   .reset          (reset           ),
157   .mclk          (mclk            ),

```

- ✓ 라인 145 - 155 : address step 에 따라 사용하는 module을 설정합니다.

6.5 ddr_test 설정

ddr_test 모듈을 아래와 같이 수정합니다. BLOCK_MAX는 mig7_write_top(mig7_read_top)을 수행하는 회수이고, BLOCK_SIZE는 mig7_writexxxx (mig7_readxxxx)을 수행하는 회수입니다. 예를 들어 DDR3_STEP_128x1024의 경우에는 8192(BLOCK_MAX) x 2(BLOCK_SIZE) x 128 x 1024 = 2Gb 가 됩니다.

```

92 `ifdef RUN_MODE
93     parameter BLOCK_MAX = 14'd8191;
94
95 `ifdef DDR3_STEP_128x8
96     parameter BLOCK_SIZE = 10'd256;
97 `elsif DDR3_STEP_128x32
98     parameter BLOCK_SIZE = 10'd64;
99 `elsif DDR3_STEP_128x64
100    parameter BLOCK_SIZE = 10'd32;
101 `elsif DDR3_STEP_128x256
102    parameter BLOCK_SIZE = 10'd8;
103 `elsif DDR3_STEP_128x1024
104    parameter BLOCK_SIZE = 10'd2;
105 `endif
106
107 `elsif SIM_MODE
108     parameter BLOCK_MAX = 14'd3;
109
110 `ifdef DDR3_STEP_128x8
111     parameter BLOCK_SIZE = 10'd256;
112 `elsif DDR3_STEP_128x32
113     parameter BLOCK_SIZE = 10'd64;
114 `elsif DDR3_STEP_128x64
115     parameter BLOCK_SIZE = 10'd32;
116 `elsif DDR3_STEP_128x256
117     parameter BLOCK_SIZE = 10'd8;      ▶
118 `elsif DDR3_STEP_128x1024
119     parameter BLOCK_SIZE = 10'd2;
120 `endif
121
122 `endif

```

- ✓ 라인 92 - 122 : address step에 따라 BLOCK_SIZE를 설정합니다.

6.6 define.v 수정

아래와 같이 DDR3_STEP_128x8 ~ DDR3_STEP_128x1024를 각각 정의하고, 해당하는 값 1개만 사용합니다. 아래는 DDR3_STEP_128x1024를 사용하는 경우입니다. 이 때에는 mig7_write_128x1024, mig7_read_128x1024 모듈을 사용하고, BLOCK_SIZE = 2가 됩니다.

```
7 //`define SIM_MODE
8 `define RUN_MODE
9
10 //`define DDR3_STEP_128x8
11 //`define DDR3_STEP_128x32
12 //`define DDR3_STEP_128x64
13 //`define DDR3_STEP_128x256
14 `define DDR3_STEP_128x1024
15
```

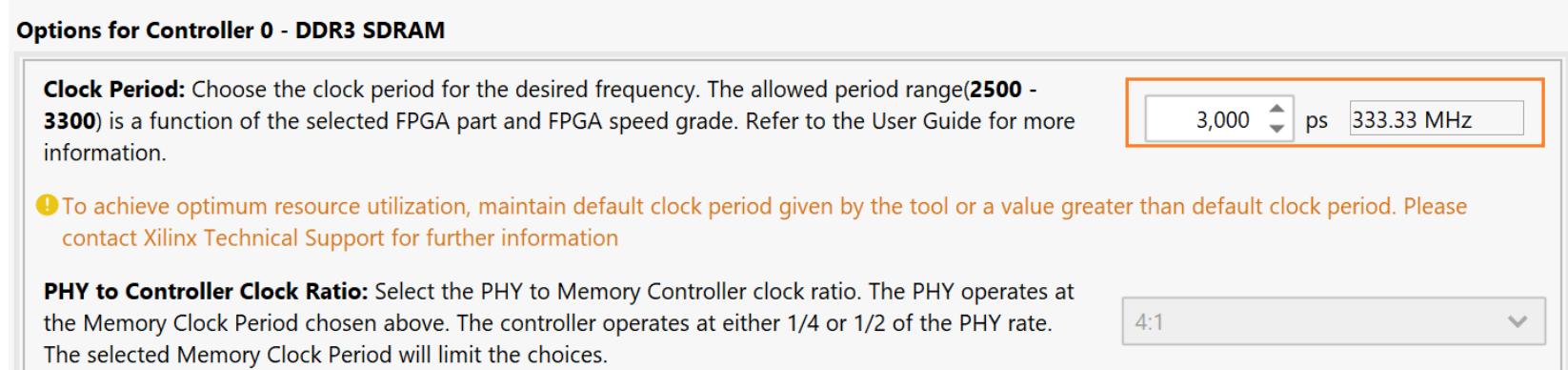
- ✓ 라인 10 - 14 : DDR3_STEP_128x8 ~ DDR3_STEP_128x1024를 정의합니다.

`define 값을 변경했을 때, vivado에서 해당하는 모듈이 바로 반영이 안되는 경우가 있습니다. 이럴 경우에는 프로젝트를 닫고 다시 Open하면 해당 모듈이 정상적으로 올라옵니다.

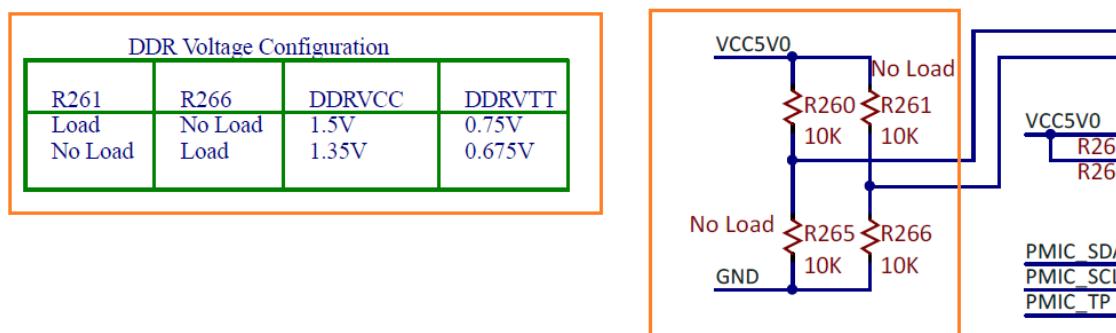
6.7 MIG7 Memory Interface IP 수정

이번 장에서는 Memory Interface IP의 설정을 변경해서 적용하도록 하겠습니다. 5장에서는 system clock을 325Mhz로 사용하고, 외부에서 100Mhz를 입력해서 IP 내부에서 325Mhz를 생성하도록 설정하였는데, 이번장에서는 system clock을 333.333Mhz로 사용하고, 외부에서 333.333Mhz를 생성해서 IP에 입력하도록 하겠습니다. IP에는 system clock : 333.333Mhz와 reference clock : 200Mhz 가 입력됩니다.

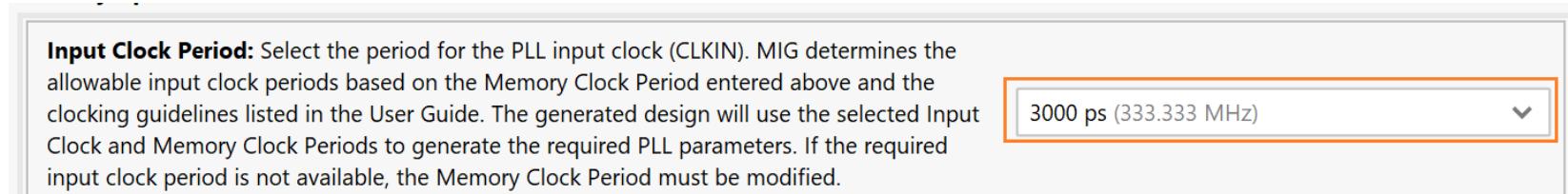
IP를 Re-customize 하기 위하여 생성된 IP를 mig7 memory interface를 double click 합니다. system clock 화면이 나올 때까지 Next를 클릭합니다. Clock Period 에 3,000 입력합니다. 나머지 항목들은 이전 값들을 그대로 사용합니다.



system clock을 400Mhz(clock period : 2500)을 사용하기 위해서는 Memory Voltage를 1.35V → 1.5V로 변경해야 합니다. 그러나 Arty A7 35T의 기본값이 1.35V로 설정되어 있습니다. 이 값을 변경하기 위해서는 회로도 11 페이지를 참조합니다. (보드에서 1.5V로 변경 후, Memory Voltage를 1.5V로 변경하면 clock period의 값을 2500 (400Mhz)으로 설정할 수 있습니다. 이렇게 설정 후 IP를 생성하면 400Mhz로 구현할 수 있습니다)



Next를 클릭합니다. Input Clock Period : 3000ps(333.333 Mhz)를 선택합니다.



Next 버튼을 끝까지 클릭하고, 마지막으로 Generate 버튼을 클릭해서 IP를 생성합니다.

6.8 clk_gen 수정

clk_gen IP를 더블 클릭해서 clock 출력을 아래와 같이 수정합니다.

| Output Clock | Port Name | Output Freq (MHz) | | Phase (degrees) | | Duty Cycle (%) | | Drives |
|--|-----------|-------------------|---------|-----------------|--------|----------------|--------|--------|
| | | Requested | Actual | Requested | Actual | Requested | Actual | |
| <input checked="" type="checkbox"/> clk_out1 | clk_out1 | 333.3333 | 333.333 | 0.000 | 0.000 | 50.000 | 50.0 | BUFG |
| <input checked="" type="checkbox"/> clk_out2 | clk_out2 | 200 | 200.000 | 0.000 | 0.000 | 50.000 | 50.0 | BUFG |
| <input type="checkbox"/> clk_out3 | clk_out3 | 100.000 | N/A | 0.000 | N/A | 50.000 | N/A | BUFG |
| <input type="checkbox"/> clk_out4 | clk_out4 | 100.000 | N/A | 0.000 | N/A | 50.000 | N/A | BUFG |
| <input type="checkbox"/> clk_out5 | clk_out5 | 100.000 | N/A | 0.000 | N/A | 50.000 | N/A | BUFG |
| <input type="checkbox"/> clk_out6 | clk_out6 | 100.000 | N/A | 0.000 | N/A | 50.000 | N/A | BUFG |

clk_out1은 ddr3 IP의 system clock으로 사용되고, clk_out2는 ddr3 IP의 reference clock으로 사용됩니다.

6.9 Arty35Top.v 수정

clock 부분을 아래와 같이 수정합니다.

```

112 `ifdef RUN_MODE
113   wire      sys_clk_i ;      // ddr system clock
114   wire      clk_ref_i ;     // ddr reference clock
115   wire      pll_locked ;
116   clk_gen   clk_gen (
117     .clk_out1  (sys_clk_i      ),      // output clk_out1, 333.333 Mhz
118     .clk_out2  (clk_ref_i      ),      // output clk_out2, 200 Mhz
119
120     .reset    (1'b0          ),      // input reset
121     .locked   (pll_locked    ),      // output locked
122     .clk_inl  (mCLOCK        )
123   );
124 `endif

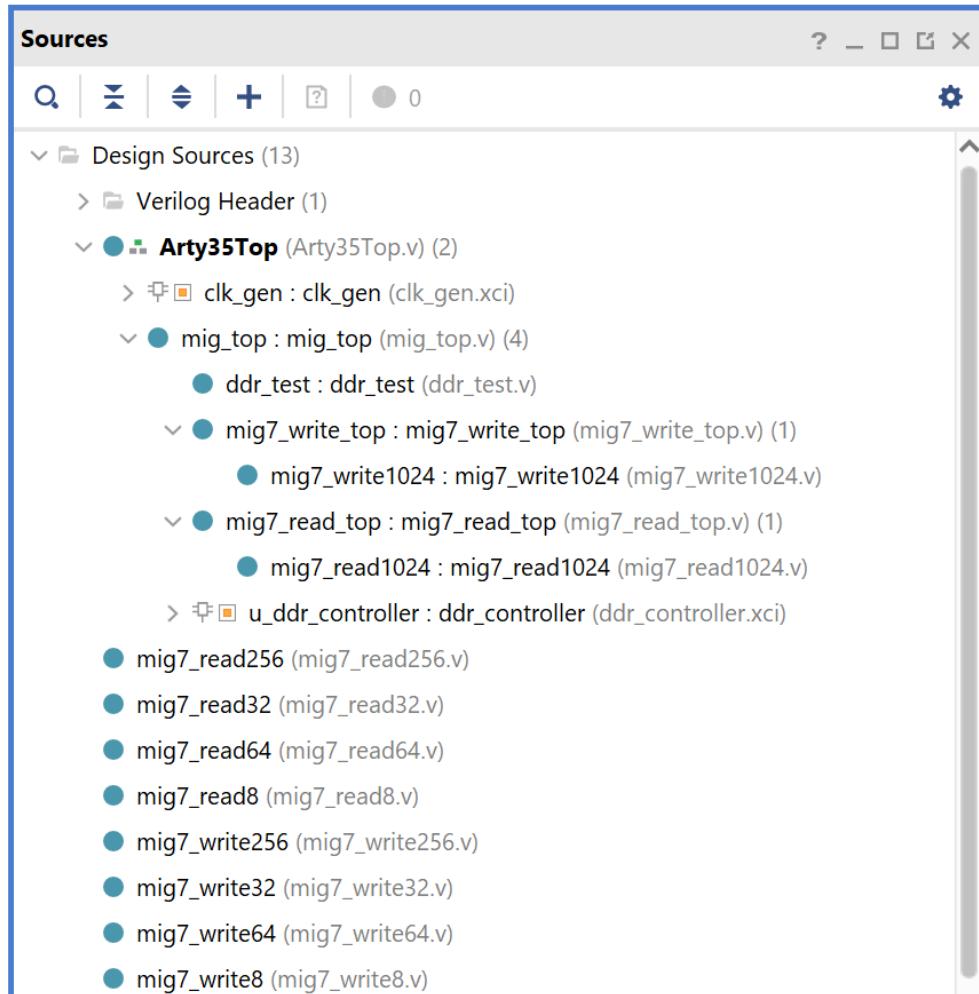
```

라인 112 - 124 : sys_clk_i 는 333.333Mhz를 사용하고, clk_ref_i 는 200Mhz를 사용합니다.

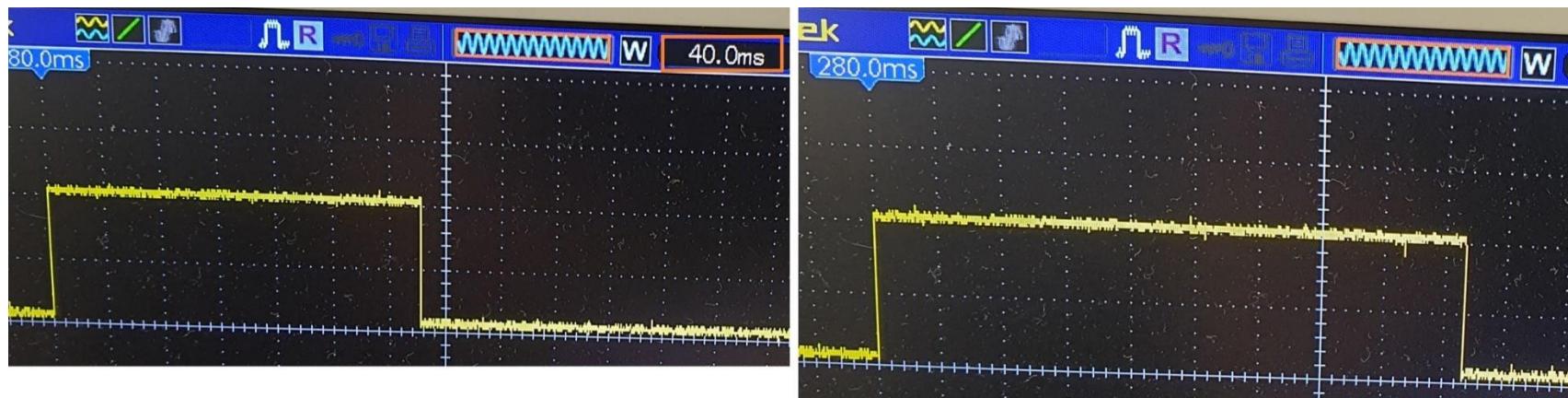
6.10 Bitsteam 생성 및 결과 확인

`define` 을 각각 `DDR3_STEP_128x8` ~ `DDR3_STEP_128x1024` 까지 설정한 후 Bitstream을 생성합니다. 생성된 Bitsteam을 다운로드 해서 전 영역 write, read 시간을 측정합니다. (자료실의 Bitstream 폴더에 Arty35Top_8.bin ~ Arty35Top_1024.bin 참고하세요)

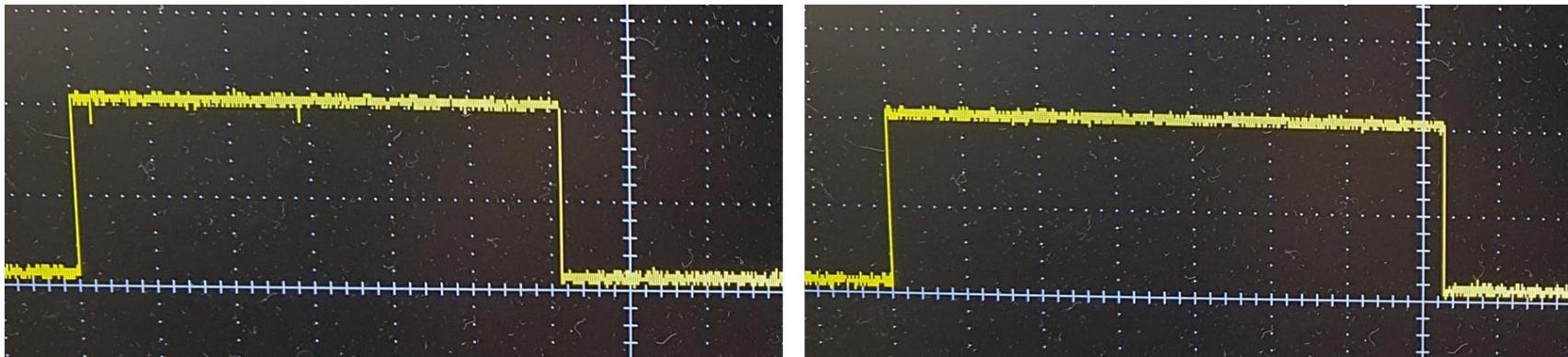
아래는 `DDR3_STEP_128x1024`를 사용할 때의 소스 구조를 보여줍니다.



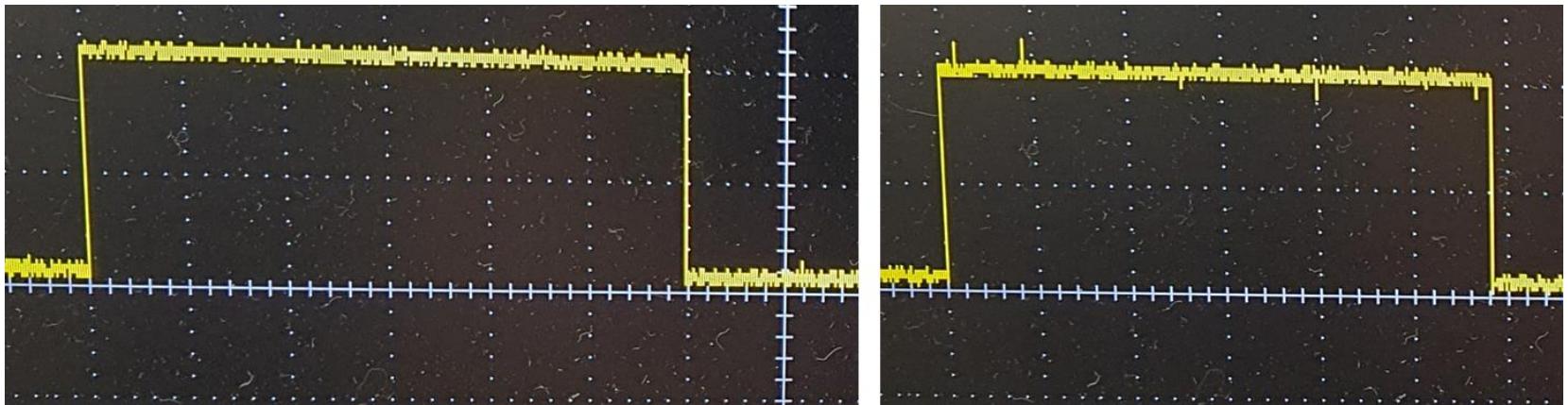
아래 그림은 `DDR3_STEP_128x32`을 사용할 때의 write, read 소요시간을 보여줍니다. 왼쪽이 write, 오른쪽이 read 입니다. 한 칸이 40ms입니다. write 소요시간 : 약 264ms, read 소요시간 : 약 376ms



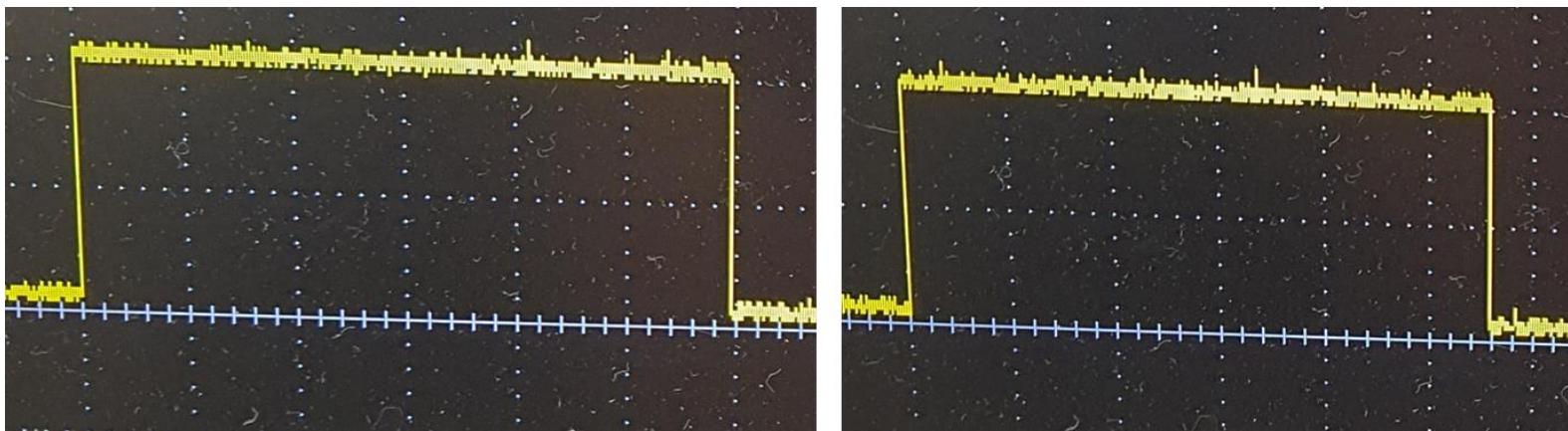
아래 그림은 DDR3_STEP_128x64을 사용할 때의 write, read 소요시간을 보여줍니다. 왼쪽이 write, 오른쪽이 read입니다. 한 칸이 40ms입니다. write 소요시간 : 약 244ms, read 소요시간 : 약 290ms



아래 그림은 DDR3_STEP_128x256을 사용할 때의 write, read 소요시간을 보여줍니다. 왼쪽이 write, 오른쪽이 read입니다. 한 칸이 40ms입니다. write 소요시간 : 약 240ms, read 소요시간 : 약 232ms



아래 그림은 DDR3_STEP_128x1024을 사용할 때의 write, read 소요시간을 보여줍니다. 왼쪽이 write, 오른쪽이 read입니다. 한 칸이 40ms입니다. write 소요시간 : 약 238ms, read 소요시간 : 약 224ms



아래는 지금까지의 결과를 표로 보여줍니다. 최대 MT/s 는 $333 \times 2 = 666$ 인데, 128x1024 module을 사용할 때 비슷한 수치가 나오는 것 같습니다.

| module | 전영역 read/write 소요시간 | | MT/s | | 최대 MT/s |
|------------------------|---------------------|-----------|--------------|-------------|---------|
| | write (ms) | read (ms) | write (MT/s) | read (MT/s) | |
| 128x8 module | 360 | 880 | 356 | 145 | |
| 128x32 module | 264 | 376 | 485 | 340 | |
| 128x64 module | 244 | 290 | 525 | 441 | 666 |
| 128x256 module | 240 | 232 | 533 | 552 | |
| 128x1024 module | 238 | 224 | 538 | 571 | |

ddr memory는 연속해서 read 하거나, 연속해서 write 할 때 높은 전송속도를 나타냅니다. 짧은 transaction은 불필요한 지연시간을 초래합니다. 이미지 처리용으로 ddr을 사용할 때에는 라인 단위로 처리(read, write) 합니다. 한 라인을 연속해서 write, read 하기 때문에 높은 처리속도를 구현할 수 있습니다.

지금까지의 내용을 충분히 학습한다면, DDR Memory를 이용한 어떠한 일도 할 수 있을 것이라 생각합니다. 본 강의의 내용을 충분히 이해하고 활용해서 좋은 개발자 되시길 바랍니다.

7. Frame Buffer 구현

이번 장에서는 앞에서 구현한 범용 Memory Controller (`mig7_write`, `mig7_read`)을 사용하여 영상데이터를 위한 Frame Buffer를 구현하도록 하겠습니다. 적당한 이미지 센서 보드가 없어서 이번장에서는 Simulation 으로 결과를 확인하도록 하겠습니다.

아래는 테스트용으로 사용할 640 x 360 사이즈의 이미지 입니다.

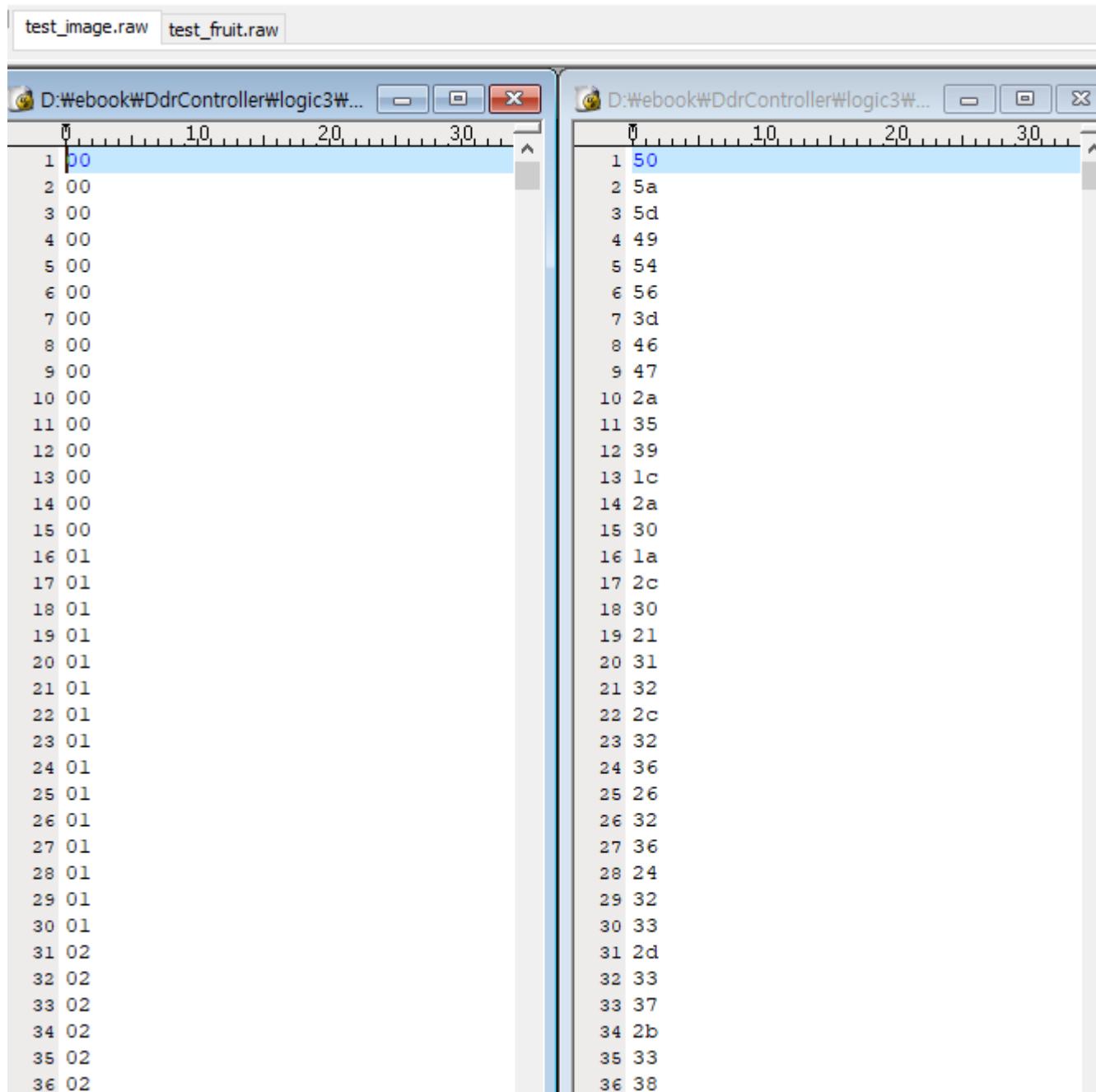


이 이미지 파일을 24bits RGB 값으로 변환해서 파일로 저장합니다. 그리고 메모리 영역에 2개의 Frame Buffer 영역(A, B 라 하겠습니다)을 설정해서, 첫번째 Frame 시간에는 A에는 이미지를 저장하고, B에서는 이미지를 읽습니다. 다음 Frame 시간에는 반대로 A에서는 이미지를 읽고, B에는 이미지를 저장합니다.

7.1 데이터 준비

simulation 용 데이터를 준비합니다. bmp 파일에서 헤더 정보를 제거하고 상하반전 시킨 후, 각각의 RGB 값을 Hexa로 변환하고 String 으로 변환해서 한라인에 8bits hexa 값이 나오도록 파일을 준비합니다. 2개의 파일을 준비합니다. 1개는 실제 영상데이터 파일이고, 나머지 1개는 디버깅을 위해서 값이 순차적으로 증가하는 파일을 준비합니다. 파일이름은 “`test_fruit.raw`”, “`test_image.raw`” 입니다. 파일은 “`arty35t_ddr3\arty35t_ddr3.sim\sim_1\behav\xsim`” 폴더에 있습니다. Simulation 루트 폴더의 위치입니다.

첫번째 라인은 첫번째 픽셀의 R, 두번째 라인은 첫번째 픽셀의 G, 세번째 라인은 첫번째 픽셀의 B, 네번째 라인은 두번째 픽셀의 R, 이런 식으로 데이터가 구성됩니다. (BMP 데이터를 바로 읽어서 데이터를 구성하는 방법도 있습니다. 이것은 이 문서를 보시는 분들이 직접 구현해 보시길 바랍니다)



처음에 simulation을 진행할 때에는 test_image.raw 를 사용합니다. ddr write, read 데이터가 정상적으로 동작하는지 확인하는 용도입니다. 이과정이 끝나면 test_fruit.raw 파일을 이용하여 최종 simulation을 진행합니다. 결과를 확인하기 위해서는 2-frame 동안 simulation을 진행해야 합니다. 이는 매우 많은 시간을 필요로 합니다. PC 성능에 따라 다르겠지만 대략 4~10시간 정도 소요됩니다. 결과를 파일로 저장합니다. 그리고 raw 텍스트 파일을 화면에 보여주는 간단한 Windows Application 프로그램을 이용해서 결과를 확인합니다. 이 프로그램은 제공해 드립니다.

다음장부터 본격적으로 코딩을 진행하도록 하겠습니다.

7.2 Image Decoder

이번 장에서는 Image Decoder를 구현합니다. 이미지 파일을 읽어서 이미지 센서와 같은 신호를 만들어 주는 것입니다. 사양은 아래와 같습니다.

- ✓ Total H x V : 760 x 420
- ✓ Effective H x V : 640 x 360
- ✓ Data width : 24 bits
- ✓ Pixel Clock : 50 Mhz
- ✓ Frame Rate : $50\text{MHz} / (760 \times 420) = 156.6 \text{fps}$

7.2.1 코드 구현

```

25 module image_dec(
26     reset      ,          // ui_clk_sync_rst from ddr controller
27     pclk       ,          // pclk, 50 Mhz
28
29     vsync      ,
30     hsync      ,
31     data       ,
32 );
33
34     input      reset      ;
35     input      pclk       ;
36     output     vsync      ;
37     output     hsync      ;
38     output [23:0] data       ;
39
40     reg      [7:0] buffer[0:3*640*360];
41
42 initial begin
43     $readmemh("test_image.raw", buffer);    // or full path
44 //     $readmemh("test_fruit.raw", buffer);    // or full path
45 end

```

- ✓ 라인 34 : reset
- ✓ 라인 35 : pclk, pixel clock, 50Mhz를 사용합니다.
- ✓ 라인 36 - 37 : vertical sync, horizontal sync, active high
- ✓ 라인 38 : image data, rgb 24bits
- ✓ 라인 40 : 파일에서 읽은 데이터를 저장할 reg를 배열로 선언
- ✓ 라인 42 - 45 : 파일을 Hexa 값으로 읽어서 buffer에 저장합니다. simulation root 폴더는 "arty35t_ddr3#arty35t_ddr3.sim#sim_1#behav#xsim" 입니다. test_image.raw, test_fruit.raw 파일이 반드시 이 폴더에 있어야 합니다. 다른 폴더를 사용하려면 Full Path를 사용하면 됩니다.

```

48  reg      [9:0]  cntP ;
49  always @ (posedge pclk or negedge reset)
50 begin
51      if (!reset)      cntP <= 10'b0;
52      else            cntP <= (cntP==10'd759) ? 10'd0 : cntP+1'b1;
53 end
54
55 reg      [8:0]  cntL ;
56 always @ (posedge pclk or negedge reset)
57 begin
58      if (!reset)      cntL <= 9'b0;
59      else            cntL <= (cntP==10'd759) ? ((cntL==9'd419) ? 9'd0 : cntL+1'b1) : cntL;
60 end
61
62 reg      hsync;
63 always @ (posedge pclk or negedge reset)
64 begin
65      if (!reset)      hsync <= 1'b0;
66      else            hsync <= ((cntL>=9'd30) && (cntL<390)) ? (((cntP>=10'd60) && (cntP<10'd700)) ? 1'b1 : 1'b0) : 1'b0;
67 end
68
69 reg      vsync;
70 always @ (posedge pclk or negedge reset)
71 begin
72      if (!reset)      vsync <= 1'b0;
73      else            vsync <= ((cntL==9'd30) && (cntP==10'd40)) ? 1'b1 :
74                           ((cntL==9'd389) && (cntP==10'd720)) ? 1'b0 : vsync;
75 end
76
77 reg      [19:0]  buf_addr;
78 always @ (posedge pclk or negedge reset)
79 begin
80      if (!reset)      buf_addr <= 20'b0;
81      else            buf_addr <= ((cntL>=9'd30) && (cntL<390)) ? (((cntP>=10'd60) && (cntP<10'd700)) ? buf_addr+1'b1 : buf_addr) : 20'b0;
82 end
83
84 wire    [23:0]  data = hsync ? {buffer[3*(buf_addr-1)], buffer[3*(buf_addr-1)+1], buffer[3*(buf_addr-1)+2]} : 24'b0;

```

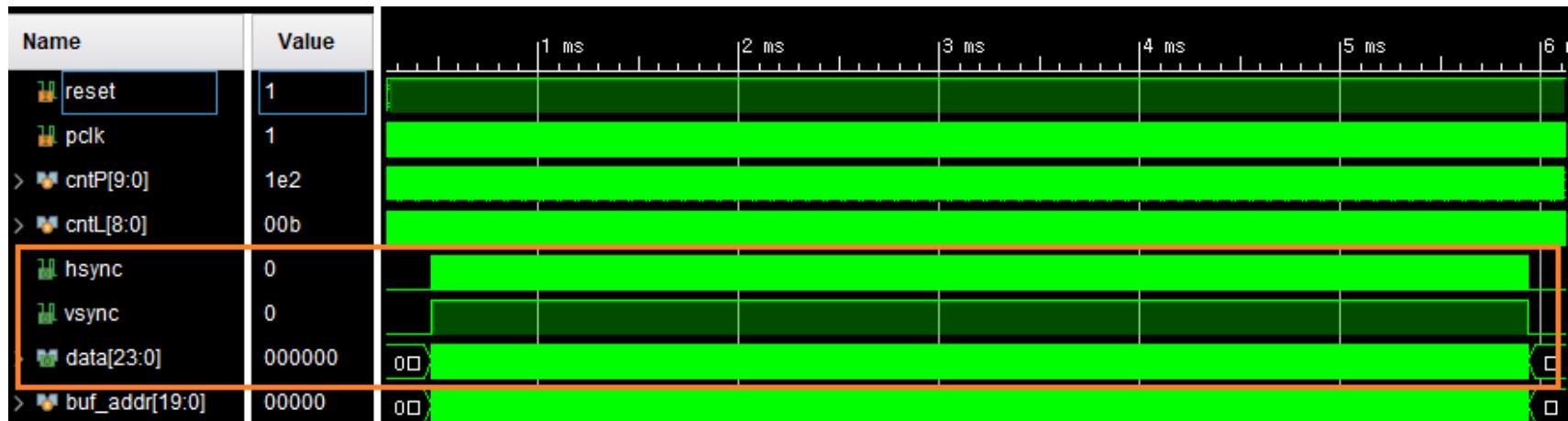
- ✓ 라인 48~60 : pixel counter, line counter를 생성합니다.
- ✓ 라인 62~84 : hsync, vsync, data 를 생성합니다.



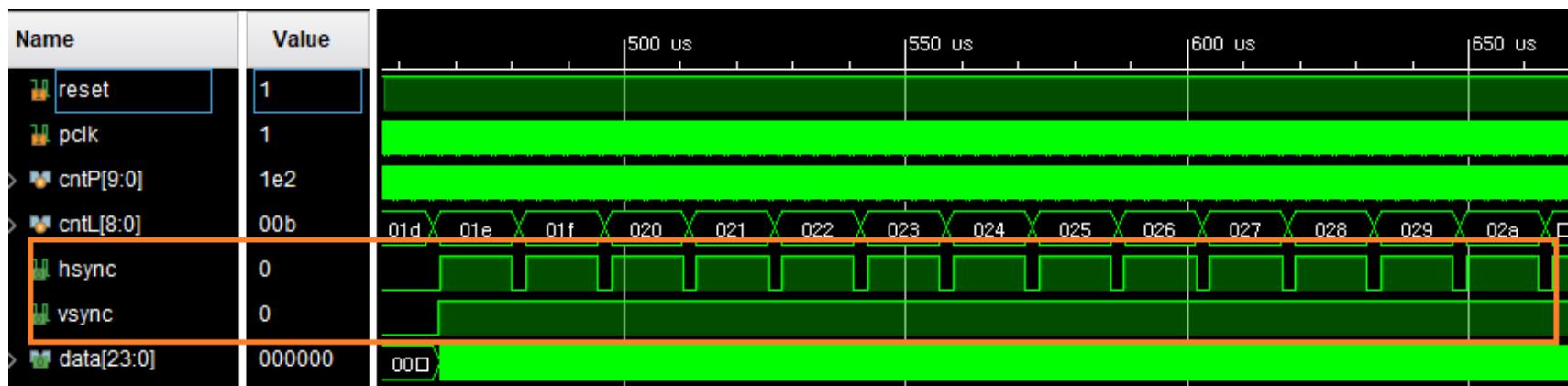
7.2.2 simulation

tb_image_dec.v, image_dec.v 파일을 simulation sources에 추가하고, tb_image_dec를 Top Module로 지정하고 simulation을 진행합니다. 약 6ms 하면 1-frame 데이터가 생성됩니다.

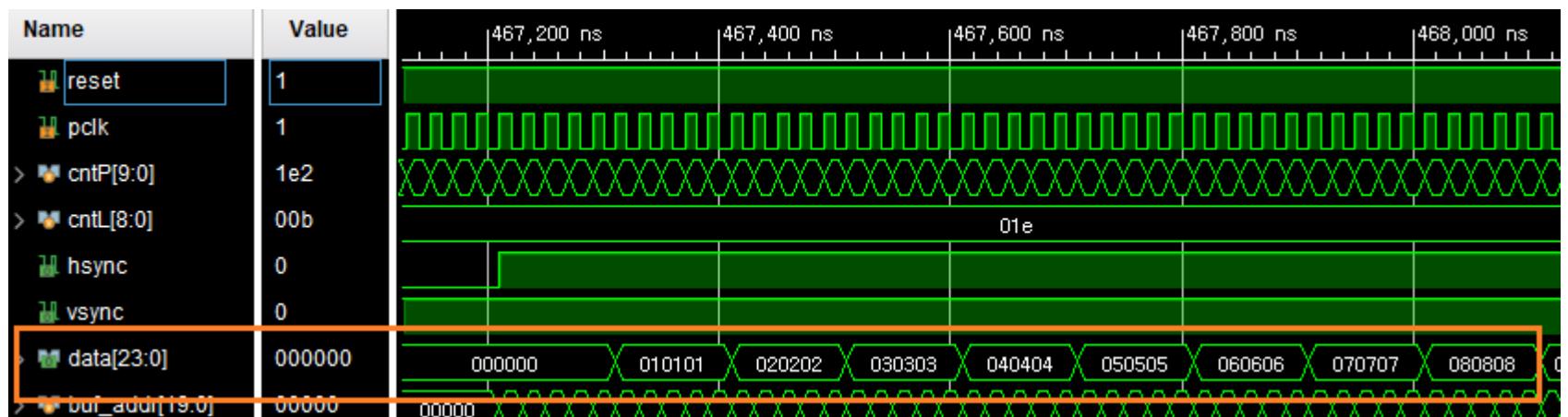
아래는 1-frame vsync 파형입니다.



아래는 hsync 파형을 확인할 수 있습니다.



마지막으로 data 를 확인합니다.



test_image.raw 파일은 00이 15개, 01이 15개, 02가 15개 ~ 이렇게 이루어져 있습니다. data값을 보면 0x000000이 5개, 0x010101이 5개 ~ 입니다. test_image.raw 파일을 제대로 읽고 있습니다.

test_image.raw 파일을 이렇게 구성한 이유를 설명합니다. 우리가 설계한 memory controller는 128bits 단위로 데이터를 access 합니다. 이미지 데이터는 한 픽셀이 24bits입니다. 따라서 5-pixels = 120 bits가 되고, 이미지 데이터를 ddr에 access 할 때에는 5-pixels (120 bits)씩하게 됩니다. 테스트 데이터를 00 : 15개, 01 : 15개, 02 : 15개 이렇게 구성하면 나중에 ddr 데이터를 확인할 때,

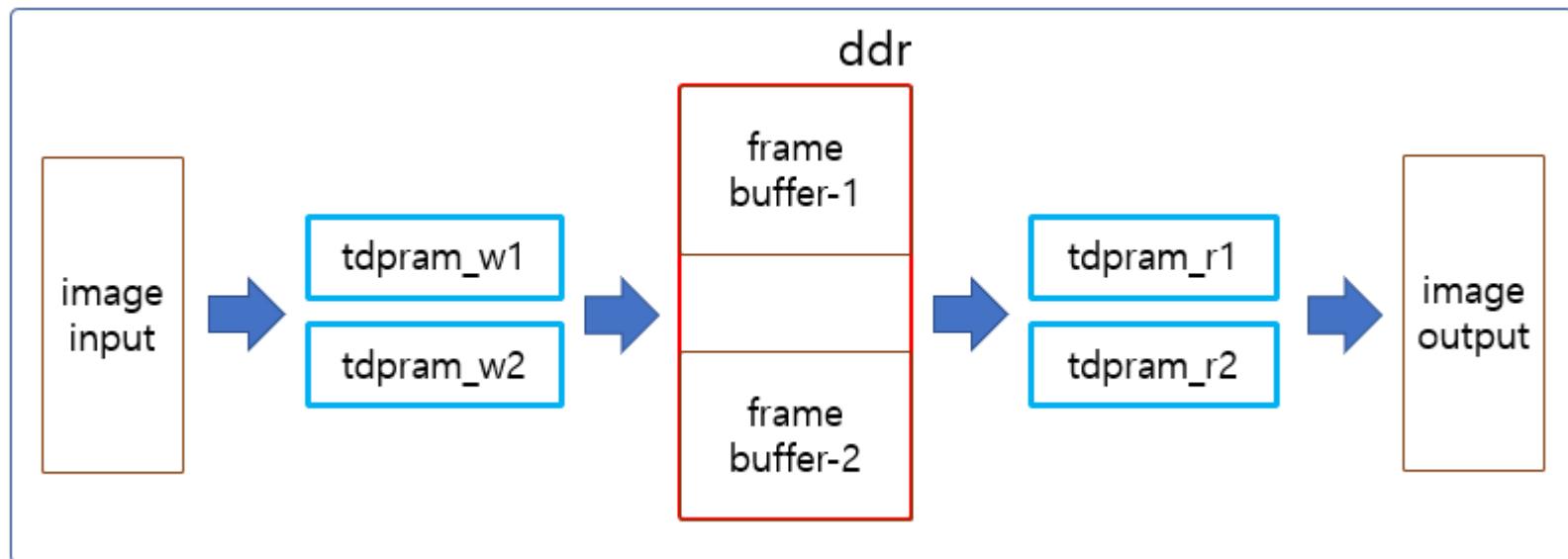
- ✓ 첫번째 데이터 : 0x00000000_00000000_00000000_00000000**00**
- ✓ 두번째 데이터 : 0x01010101_01010101_01010101_010101**00**
- ✓ 세번째 데이터 : 0x02020202_0x02020202_0x02020202_0x020202**00**

로 구성할 수 있습니다. 128bits의 최하위 8-bits는 0 입니다.

AIHIL

7.3 Frame Buffer 개요

이번 장에서는 Frame Buffer를 구현하기 위한 기본적인 개념을 설명합니다. 기본적인 구조를 이해하고 난 후에 구현을 해야 오류가 발생하는 것을 방지할 수 있습니다. 영상처리는 기본적으로 라인 단위로 이루어집니다. 즉 hsync 주기에 맞춰 처리가 됩니다. 실시간으로 들어오는 데이터를 처리하기 위해서는 FPGA 내부의 메모리를 적절히 사용해야 합니다. 다음 그림은 Frame Buffer를 구현하는 전체적인 내용을 보여줍니다.

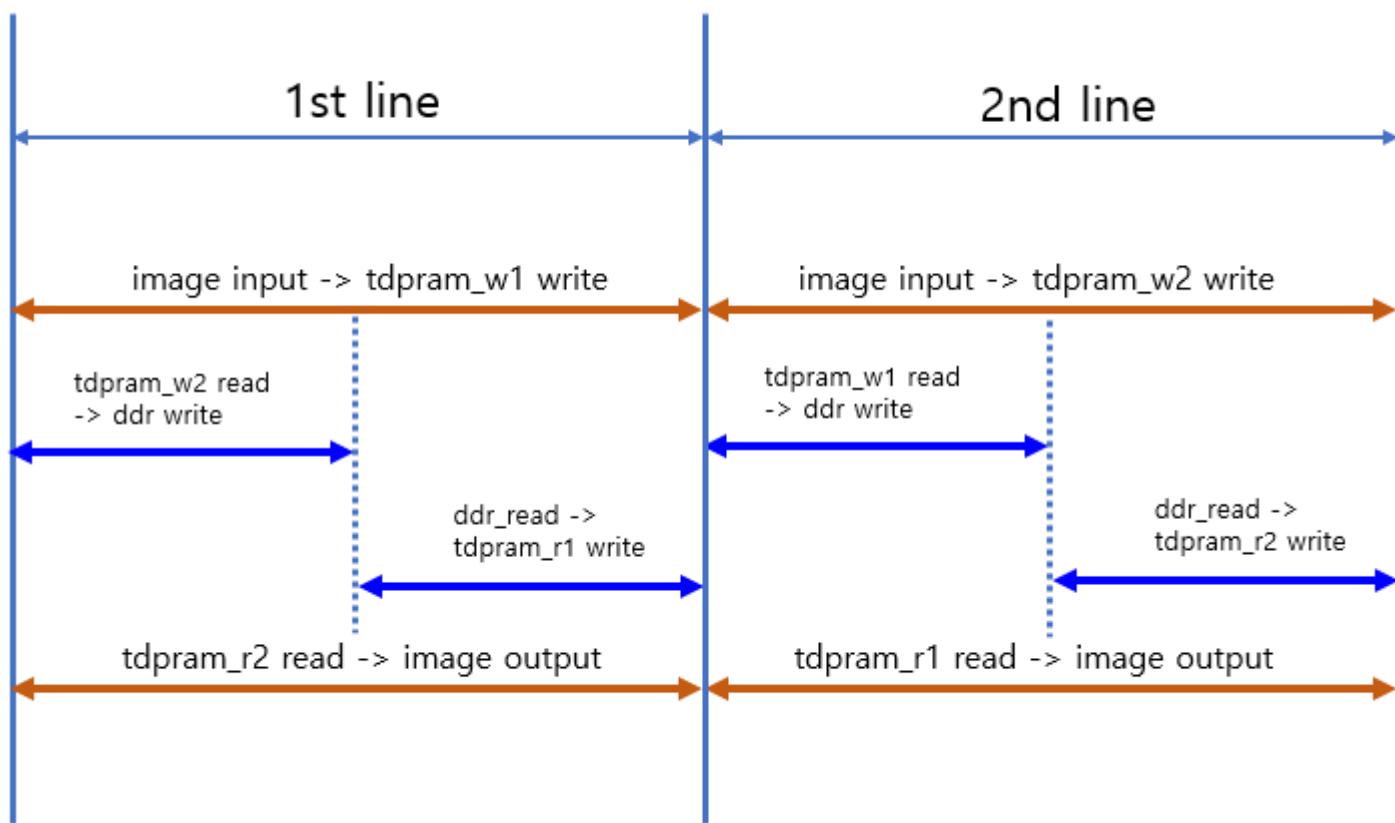


tdpram은 FPGA 내부에서 제공되는 “true dual port sram”을 의미합니다. tdpram은 2개의 포트를 가지고 있고 각각의 포트는 독립적으로 동작합니다. 즉 다른 Clock을 사용할 수 있고 동시에 동작할 수도 있습니다. 우리는 각각의 포트에 다른 클락을 사용하고 동시에 access 하지는 않습니다. “true dual port sram”에 대해서 자세히 알고 싶은 분들은 전자문서 “Verilog를 이용한 FPGA 활용”을 참조하시길 바랍니다.

동작을 간단히 설명합니다.

- 1) Frame 단위 : 첫번째 Frame 구간(vsysn)에서는 frame buffer-1에 image input 데이터를 저장합니다. 그리고 frame buffer-2의 image 데이터를 읽어서 image output 으로 출력합니다. 두번째 Frame 구간에서는 반대로 frame buffer-2에 image input 데이터를 저장하고, frame buffer-1의 image 데이터를 출력합니다.
- 2) 라인 단위 - frame write : 첫번째 라인에서는 tdpram_w1에 image input 데이터를 저장하고, tdpram_w2의 데이터를 frame buffer에 저장합니다. 두번째 라인에서는 tdpram_w2에 image input 데이터를 저장하고, tdpram_w1의 데이터를 frame buffer에 저장합니다.
- 3) 라인 단위 - frame read : 첫번째 라인에서는 frame buffer에 저장된 데이터를 tdpram_r1에 저장하고, tdpram_r2의 데이터는 image output 으로 출력합니다. 두번째 라인에서는 tdpram_r2에 frame buffer의 데이터를 저장하고, tdpram_r1의 데이터를 image output으로 출력합니다.

아래 그림은 라인 단위의 데이터 처리를 나타냅니다.



ddr read / write 구간이 1-라인 주기에서 앞의 1/2주기에는 ddr write, 뒤의 1/2주기에는 ddr read 인 것을 주의하시길 바랍니다. 앞으로 이 타이밍에 기초해서 구현하도록 하겠습니다. 이 타이밍은 영상처리를 하는데 있어서 매우 중요합니다. 대부분의 영상처리가 이러한 방식으로 이루어집니다.

마지막으로 사용하는 frame buffer의 address에 대해서 설명합니다.

- ✓ 1라인 : $640 \times 24\text{bits} \rightarrow 128 \times 5 \times 24\text{bits} \rightarrow 128 \times 120\text{bits} \rightarrow 16 \times 8 \times 120\text{bits}$
- 1-pixel은 24bits입니다. 5-pixels은 120bits입니다. ddr의 access는 128bits씩 처리됨으로 5-pixels (120 bits) 씩 처리합니다. 하위 8bits는 사용하지 않습니다. 1라인을 처리하는데 120bits 기준으로 총 128개가 필요합니다. mig7_write8은 8 x 128bits 씩 처리하기 때문에 wsize는 16으로 설정하면 됩니다. 1라인에 소요되는 총 address는 $16 \times 64 = 1024$ (0x400)입니다. 라인이 증가할 때마다 address를 0x400 만큼 증가하면 됩니다.

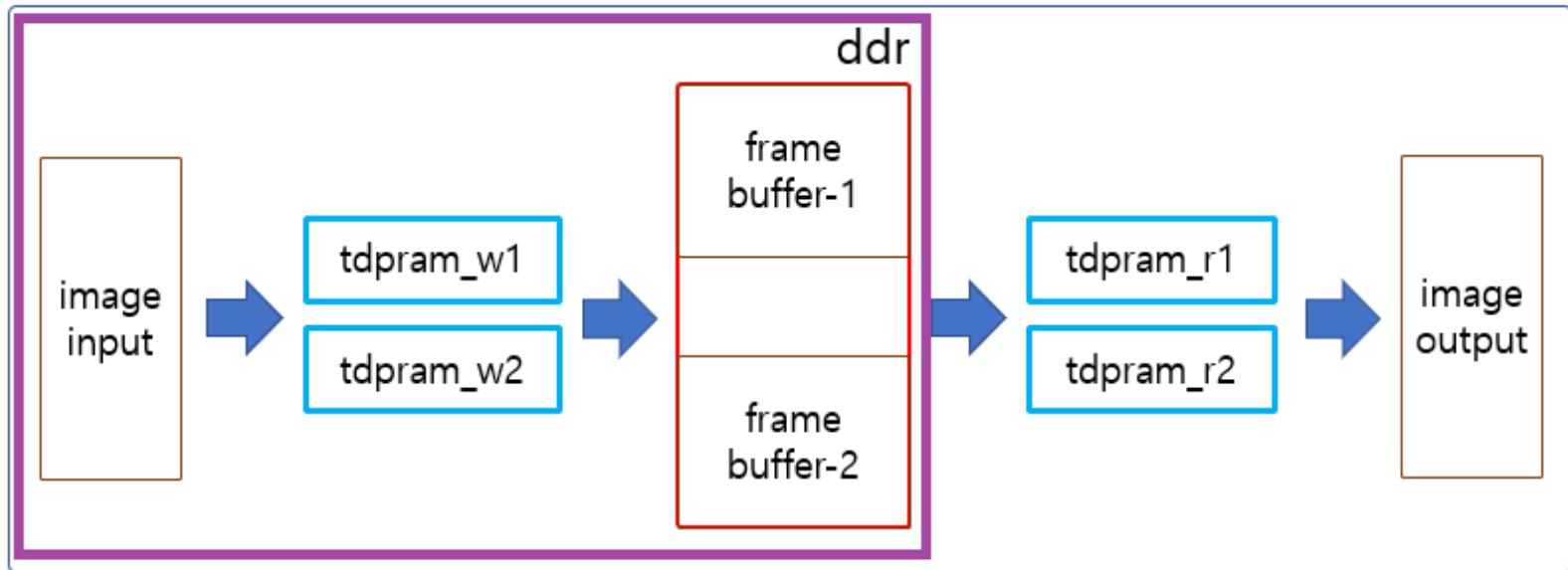
1-frame에 소요되는 address를 계산해 보겠습니다.

- ✓ 1-frame : $360 \times 1024 = 368,640 = 0x5_A000$
- ✓ frame buffer - 1 : $0x000_0000 \sim 0x0005_9FF8$
- ✓ frame buffer - 2 : $0x008_0000 \sim 0x00E_9FF8$

여기에서 계산된 내용들은 ddr write, read 시에 모두 사용됩니다.

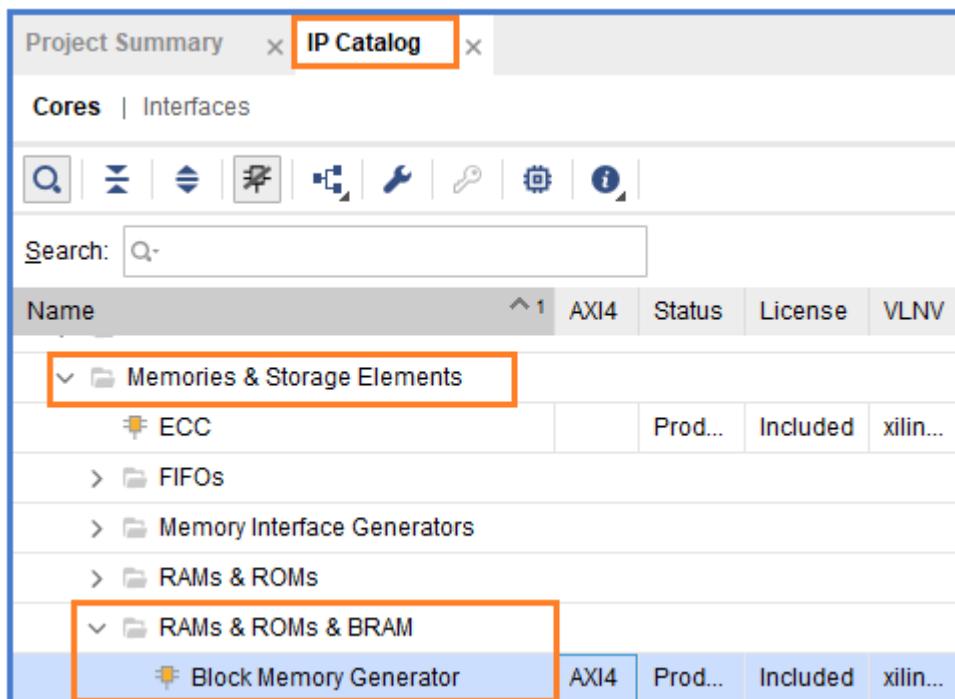
7.4 frame_write

이번 장에서는 이미지 데이터를 ddr에 저장하는 것을 구현합니다. 아래 그림에서 보라색 부분을 구현합니다. 내용이 많고 이해가 어려운 부분이 있을 수 있지만 반복해서 코드를 분석하고, simulation으로 결과를 확인하는 과정을 거쳐서 본인의 것으로 만드시길 바랍니다. 이번 장에서 설명하는 내용은 영상 처리 분야에 아주 많이 사용되는 스킬들이 포함되어 있습니다.



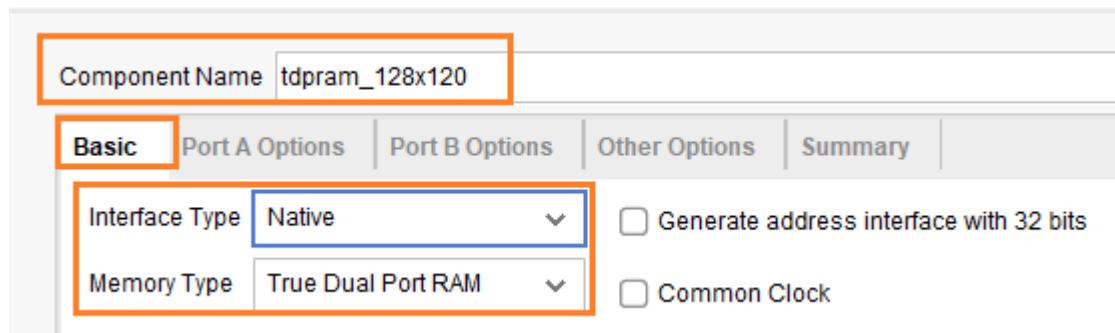
먼저 tdpram 을 생성합니다. tdpram은 1 라인 데이터를 저장할 수 있는 크기로 생성합니다. 128 x 120 bits로 생성하도록 하겠습니다. (메모리 Depth를 여유 있게 좀 크게 생성하는 것이 좋습니다. 본장에서는 정확히 2의 배수로 떨어져서 그대로 진행합니다)

IP Catalog 에서 Memories & Storage Elements - RAMs & ROMs & BRAM - Block Memory Generator 를 더블 클릭 합니다.



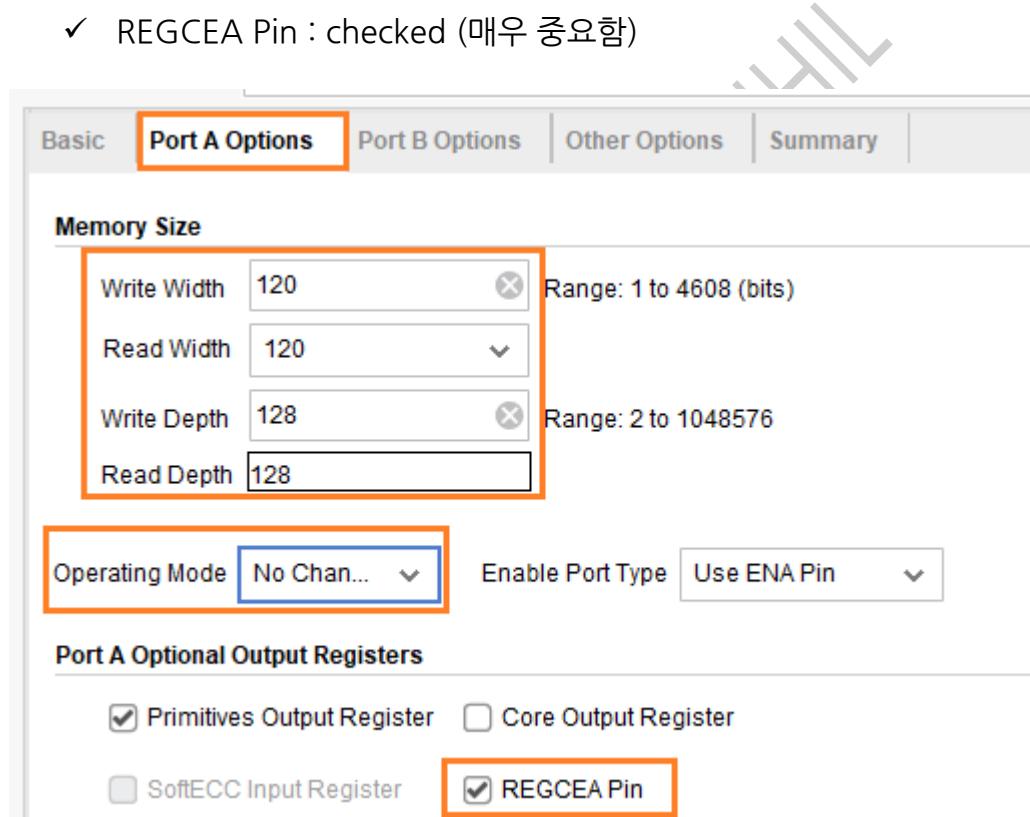
Component Name에 “tdpram_128x120”을 입력하고, Basic 탭에서 다음과 같이 설정합니다.

- ✓ Interface Type : Native
- ✓ Memory Type : True Dual Port RAM

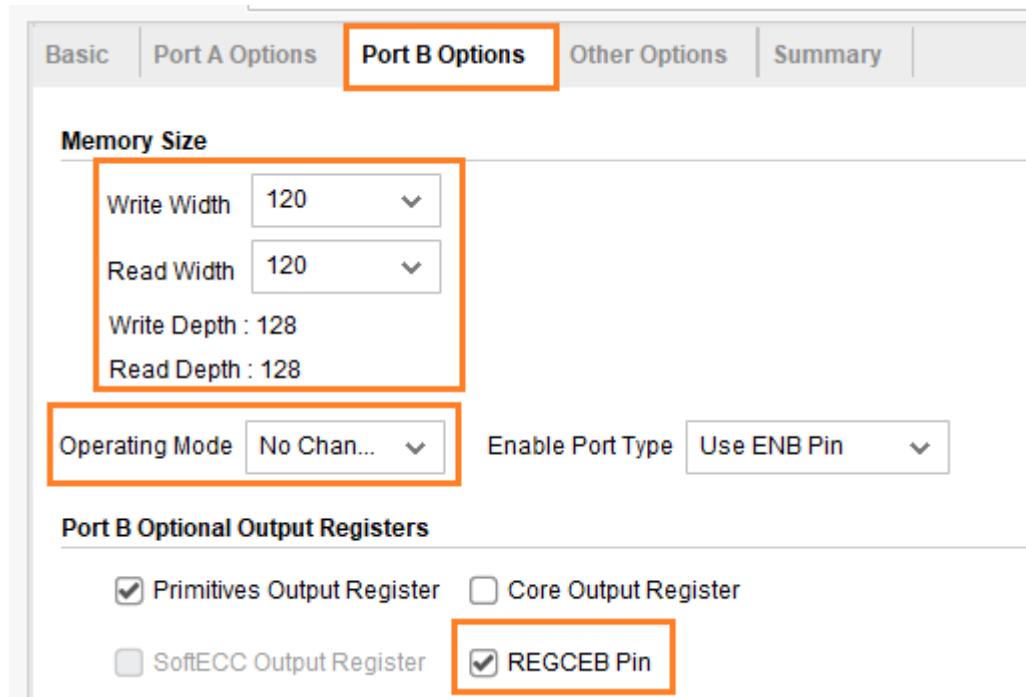


Port A Options 탭에서 다음과 같이 설정합니다.

- ✓ Write Width : 120
- ✓ Read Width : 120
- ✓ Write Depth : 128
- ✓ Read Depth : 128
- ✓ Operating Mode : No Change
- ✓ REGCEA Pin : checked (매우 중요함)



Port B Options 탭도 동일하게 설정합니다.



OK - Generate 버튼을 차례로 클릭해서 메모리를 생성합니다.

7.4.1 frame_write 모듈 구현

```

61  input      reset      ;
62  input      pclk       ;
63  input      vsync      ;
64  input      hsync      ;
65  input [23:0] data       ;
66
67  input      rst_ddr    ;
68  input      clk_ddr    ;
69  output     wstart     ;
70  output [27:0] waddr    ;
71  output [9:0]  wsize    ;
72  output [127:0] wdata   ;
73  input      wready    ;
74  input      wdone     ;
75
76  output     hdd_3d    ;
77  output     vdd_3d    ;
78  output     s_ready   ;
79  output     s_write   ;

```

- ✓ 라인 61 - 65 : reset, pclk (50Mhz), vsync, hsync, pixel data - pclk domain 신호
- ✓ 라인 67 - 74 : ddr에 write 하기 위한 신호, clk_ddr domain 신호
- ✓ 라인 76 - 79 : 디버깅 용도, Test bench에서 사용함.
- ✓ frame_write (or frame_read) 모듈을 2개의 Clock Domain 신호들을 사용합니다. 다른 Clock Domain의 신호들을 사용하려면 최소 3~4 clock delay 후 사용하시길 바랍니다.

```

81  reg          hsync_ld;
82  wire         hsync_pedge = hsync & ~hsync_ld;
83  wire         hsync_nedge = ~hsync & hsync_ld;
84  always @(posedge pclk or negedge reset)
85 begin
86     if(!reset)    hsync_ld <= 1'b0;
87     else         hsync_ld <= hsync;
88 end
89
90
91 reg          flagL ;
92 always @(posedge pclk or negedge reset)
93 begin
94     if(!reset)    flagL <= 1'b0;
95     else         flagL <= ~vsync ? 1'b0 : hsync_nedge ? ~flagL : flagL;
96 end
97
98 reg [2:0]    cntP;
99 always @(posedge pclk or negedge reset)
100 begin
101    if(!reset)   cntP <= 3'b0;
102    else         cntP <= ~hsync ? 3'b0 : (cntP==3'd4) ? 3'd0 : cntP+1'b1 ;
103 end

```

- ✓ 라인 81 - 88 : 내부에서 사용하는 hsync positive edge, negative edge 신호
- ✓ 라인 91 - 96 : 라인 카운터
- ✓ 라인 98 - 103 : Pixel Data 카운터

```

105 // -----
106 // tdpram-1 : write
107 reg [119:0] dinal;
108 always @(posedge pclk or negedge reset)
109 begin
110     if(!reset)    dinal <= 120'b0;
111     else         dinal <= (~hsync | flagL) ? 120'b0 :
112                     (cntP==3'd0) ? {data, dinal[95:0]} :
113                     (cntP==3'd1) ? {dinal[119:96], data, dinal[71:0]} :
114                     (cntP==3'd2) ? {dinal[119:72], data, dinal[47:0]} :
115                     (cntP==3'd3) ? {dinal[119:48], data, dinal[23:0]} :
116                     (cntP==3'd4) ? {dinal[119:24], data} : dinal ;
117 end
118
119 reg [6:0]    addral;
120 always @(posedge pclk or negedge reset)
121 begin
122     if(!reset)    addral <= 7'b0;
123     else         addral <= (~hsync | flagL) ? 7'b0 : (cntP==3'd4) ? addral+1'b1 : addral;
124 end
125
126 reg [6:0]    addral_ld;
127 always @(posedge pclk or negedge reset)
128 begin
129     if(!reset)    addral_ld <= 7'b0;
130     else         addral_ld <= addral;
131 end
132
133 reg          wenl ;
134 always @(posedge pclk or negedge reset)
135 begin
136     if(!reset)    wenl <= 1'b0;
137     else         wenl <= flagL ? 1'b0 : (cntP==3'd4) ? 1'b1 : 1'b0;
138 end
139
140 wire         enal = wenl;
141 wire         weal = wenl;

```

- ✓ tdpram-1의 write 관련 신호들을 생성합니다.

- ✓ 영상 데이터가 들어오면 24bits 씩 5개를 모아서 120bits 만들고 tdpram-1에 저장합니다.
- ✓ 라인 107 - 117 : 5개의 pixels을 모아 120bits로 만들어서 dina1을 생성합니다.
- ✓ 라인 119 - 124 : tdpram-1에 write하기 위한 address를 생성합니다. address는 5-pixels마다 증가합니다.
- ✓ 라인 126 - 131 : 타이밍을 맞추기 위해서 1-clock delay를 생성합니다.
- ✓ 라인 133 - 138 : write enable 신호를 생성합니다. 5-pixels마다 Active 됩니다.
- ✓ 라인 140 - 141 : enable, wea(1이면 write, 0이면 read) 신호를 생성합니다.

```

144 // -----
145 // tdpram-2 : write
146 reg [119:0] dina2;
147 always @ (posedge pcclk or negedge reset)
148 begin
149     if (!reset)      dina2 <= 120'b0;
150     else            dina2 <= (~hsync | ~flagL) ? 120'b0 :
151                     (cntP==3'd0) ? {data, dina2[95:0]} :
152                     (cntP==3'd1) ? {dina2[119:96], data, dina2[71:0]} :
153                     (cntP==3'd2) ? {dina2[119:72], data, dina2[47:0]} :
154                     (cntP==3'd3) ? {dina2[119:48], data, dina2[23:0]} :
155                     (cntP==3'd4) ? {dina2[119:24], data} : dina2 ;
156 end
157
158 reg          wen2 ;
159 always @ (posedge pcclk or negedge reset)
160 begin
161     if (!reset)      wen2 <= 1'b0;
162     else            wen2 <= ~flagL ? 1'b0 : (cntP==3'd4) ? 1'b1 : 1'b0;
163 end
164
165 wire         ena2 = wen2;
166 wire         wea2 = wen2;
167
168 reg [6:0]    addra2;
169 always @ (posedge pcclk or negedge reset)
170 begin
171     if (!reset)      addra2 <= 7'b0;
172     else            addra2 <= (~hsync | ~flagL) ? 7'b0 : (cntP==3'd4) ? addra2+1'b1 : addra2;
173 end
174
175 reg [6:0]    addra2_ld;
176 always @ (posedge pcclk or negedge reset)
177 begin
178     if (!reset)      addra2_ld <= 7'b0;
179     else            addra2_ld <= addra2;
180 end

```

- ✓ tdpram-2의 write 관련 신호들을 생성합니다. tdpram-1과 동일합니다. 단지 flagL의 값이 반전되어 입력됩니다. 이는 라인을 번갈아 가면서 한번은 tdpram-1에 write하고 다음번에는 tdpram-2에 write하기 위해서입니다.

```

183 // -----
184 // tpraml,2 : read, ddr : write
185 reg vdd_1d, vdd_2d, vdd_3d, vdd_4d ;
186 always @ (posedge clk_ddr or negedge rst_ddr)
187 begin
188 if (!rst_ddr) begin
189   vdd_1d <= 1'b0;
190   vdd_2d <= 1'b0;
191   vdd_3d <= 1'b0;
192   vdd_4d <= 1'b0;
193 end
194 else begin
195   vdd_1d <= vsync ;
196   vdd_2d <= vdd_1d;
197   vdd_3d <= vdd_2d;
198   vdd_4d <= vdd_3d;
199 end
200 end
201
202 reg hdd_1d, hdd_2d, hdd_3d, hdd_4d ;
203 always @ (posedge clk_ddr or negedge rst_ddr)
204 begin
205 if (!rst_ddr) begin
206   hdd_1d <= 1'b0;
207   hdd_2d <= 1'b0;
208   hdd_3d <= 1'b0;
209   hdd_4d <= 1'b0;
210 end
211 else begin
212   hdd_1d <= hsync ;
213   hdd_2d <= hdd_1d;
214   hdd_3d <= hdd_2d;
215   hdd_4d <= hdd_3d;
216 end
217 end
218
219 wire vdd_pedge = vdd_3d & ~vdd_4d;
220 wire vdd_nedge = ~vdd_3d & vdd_4d;
221 wire hdd_pedge = hdd_3d & ~hdd_4d;
222 wire hdd_nedge = ~hdd_3d & hdd_4d;
223
224 reg flagLL ;
225 always @ (posedge clk_ddr or negedge rst_ddr)
226 begin
227 if (!rst_ddr) flagLL <= 1'b0;
228 else flagLL <= ~vdd_3d ? 1'b0 : hdd_pedge ? ~flagLL : flagLL;
229 end
230
231 reg flag_frame;
232 always @ (posedge clk_ddr or negedge rst_ddr)
233 begin
234 if (!rst_ddr) flag_frame <= 1'b0;
235 else flag_frame <= vdd_nedge ? ~flag_frame : flag_frame;
236 end

```

- ✓ tpraml,2의 값을 read 하고, ddr 에 write 하기 위한 신호들을 생성합니다. 주의해야 할 것은 tpraml,2에 write 할 때에는 pixel clock (pclk, 50 Mhz)을 사용하였는데, read 할 때에는 clk_ddr (clk_ui, 81.2 Mhz)을 사용하고 있습니다. pclk 영역에서 사용하는 신호를 clk_ddr 영역에서 사용할 때에는 Flip/Flip을 3~4번 정도 거친 후 사용하는 것이 좋습니다. Clock Domain이 변경되면 Glitch 등 노이즈가 발생할 수 있기 때문입니다.
- ✓ 라인 185 ~ 236 : clk_ddr 영역에서 vsync, hsync 신호를 사용하기 위하여 4번 Flip/Flop을 추가하였습니다. 내부에서 사용하는 신호들을 생성합니다.

```

239 parameter      BLOCK_SIZE = 10'd16; // 640 x 24 = 128 x 120 = 16(block size) x 8 x 120
240
241 // State Parameter
242 parameter      M_IDLE      = 2'd0;
243 parameter      M_READY     = 2'd1;
244 parameter      M_START     = 2'd2;
245 parameter      M_WRITE     = 2'd3;
246
247 // State Control
248 reg      [1:0]  ddr_state;
249 wire      s_idle   = (ddr_state==M_IDLE ) ? 1'b1 : 1'b0;
250 wire      s_ready  = (ddr_state==M_READY) ? 1'b1 : 1'b0;
251 wire      s_start   = (ddr_state==M_START) ? 1'b1 : 1'b0;
252 wire      s_write   = (ddr_state==M_WRITE) ? 1'b1 : 1'b0;

```

- ✓ 라인 239 : block size, wszie, rsize 의 값으로 사용됩니다.
- ✓ 라인 242 - 252 : SM을 정의합니다. 각 라인의 hsync의 positive edge에서 tdpram에 있는 데이터를 읽어서 ddr에 write 하는 구조로 되어 있습니다. 자세한 내용은 simulation을 참조하시길 바랍니다.

```

323 reg      [1:0]  wr_ptr;
324 always @ (posedge clk_ddr or negedge rst_ddr)
325 begin
326     if (!rst_ddr)    wr_ptr <= 2'b0;
327     else            wr_ptr <= s_idle ? 2'd0 :
328                           (s_start & (start_cnt==3'd0)) ? 2'd0 :
329                           (enbl_2d | enb2_2d) ? wr_ptr+1'b1 : wr_ptr;
330 end
331
332 wire      [119:0] doutbl, doutb2;
333 reg      [119:0] rdout_0, rdout_1, rdout_2, rdout_3;
334 always @ (posedge clk_ddr or negedge rst_ddr)
335 begin
336     if (!rst_ddr) begin
337         rdout_0 <= 120'b0;
338         rdout_1 <= 120'b0;
339         rdout_2 <= 120'b0;
340         rdout_3 <= 120'b0;
341     end
342     else begin
343         rdout_0 <= (wr_ptr==2'd0) ? (enbl_2d ? doutbl : enb2_2d ? doutb2 : rdout_0) : rdout_0 ;
344         rdout_1 <= (wr_ptr==2'd1) ? (enbl_2d ? doutbl : enb2_2d ? doutb2 : rdout_1) : rdout_1 ;
345         rdout_2 <= (wr_ptr==2'd2) ? (enbl_2d ? doutbl : enb2_2d ? doutb2 : rdout_2) : rdout_2 ;
346         rdout_3 <= (wr_ptr==2'd3) ? (enbl_2d ? doutbl : enb2_2d ? doutb2 : rdout_3) : rdout_3 ;
347     end
348 end
349
350 reg      [1:0]  rd_ptr;
351 always @ (posedge clk_ddr or negedge rst_ddr)
352 begin
353     if (!rst_ddr)    rd_ptr <= 2'b0;
354     else            rd_ptr <= (s_idle ) ? 2'd0 :
355                           (s_start) ? ((start_cnt==3'd1) ? 2'd0 : rd_ptr) :
356                           (s_write & wready) ? rd_ptr+1'b1 : rd_ptr;
357 end
358
378 wire      [127:0] wdata = (rd_ptr==2'd0) ? {rdout_0, 8'b0} :
379                               (rd_ptr==2'd1) ? {rdout_1, 8'b0} :
380                               (rd_ptr==2'd2) ? {rdout_2, 8'b0} : {rdout_3, 8'b0} ;

```

위 내용(라인 323 - 380)은 간단하게 4-depth fifo를 구현하는 아주 중요한 내용입니다. 4-depth fifo가 필요한 이유를 먼저 설명합니다. tdpram에서 데이터를 읽을 때, 데이터가 출력될 때까지 2-clock delay가 발생합니다. 앞에서 살펴보았듯이 ddr 메모리에 데이터를 write 하기 위해서는 wready 신호가 Active 되면 바로 데이터를 보내 주어야 합니다. 그런데 tpdram의 read redundancy가 2-clock 발생해서 타이밍을 맞출 수가 없습니다. 그래서 ddr에 데이터를 write 하기 전에 미리 4개 정도를 읽어서 준비해 두어야 합니다. 이를 구현하는 것이 “4-depth simple fifo”입니다. simulation에서 확인하

면 4-depth면 2-clock delay로 인한 타이밍을 맞출 수 있습니다. 이 부분은 simulation에서 확인하도록 합니다.

```

390    always @ (posedge clk_ddr or negedge rst_ddr)
391    begin
392        if (!rst_ddr)      begin
393            ddr_state <= 2'b0;
394        end
395        else begin
396            ddr_state <= (vdd_nedge ) ? M_IDLE : ;
397            (s_idle & vdd_pedge ) ? M_READY : ;
398            (s_ready & hdd_pedge ) ? M_START : ;
399            (s_start & (start_cnt==3'd7)) ? M_WRITE : ;
400            (s_write & wdone ) ? M_READY : ddr_state ;
401        end
402    end

```

- ✓ 라인 390 - 402 : 상태 전이를 구현합니다. READY 상태와 WRITE 상태 사이에 START 상태를 약 7-clcok 만큼 두었습니다. 이는 앞에서 설명한 “4-depth simple fifo” 를 구현하기 위함입니다.

```

404    wire          web1 = 1'b0;
405    wire          web2 = 1'b0;
406
407    tdpram_128x120 tdproml (
408        .clka      (pclk      ), // input wire clka
409        .ena       (enal      ), // input wire ena
410        .regcea   (1'bl      ), // input wire regcea
411        .wea       (weal      ), // input wire [0 : 0] wea
412        .addr     (addr1_l1d ), // input wire [6 : 0] addr
413        .dina     (dinal      ), // input wire [119 : 0] dina
414        .douta   (douta      ), // output wire [119 : 0] douta
415        .clkb     (clk_ddr    ), // input wire clk
416        .enb      (enbl      ), // input wire enb
417        .regceb   (1'bl      ), // input wire regceb
418        .web      (web1      ), // input wire [0 : 0] web
419        .addrb   (addrb1    ), // input wire [6 : 0] addr
420        .dinb     (120'b0    ), // input wire [119 : 0] dinb
421        .doutb   (doutb1    ), // output wire [119 : 0] doutb
422    );
423
424
425    tdpram_128x120 tdprom2 (
426        .clka      (pclk      ), // input wire clka
427        .ena       (ena2      ), // input wire ena
428        .regcea   (1'bl      ), // input wire regcea
429        .wea       (wea2      ), // input wire [0 : 0] wea
430        .addr     (addr2_l1d ), // input wire [6 : 0] addr
431        .dina     (dina2      ), // input wire [119 : 0] dina
432        .douta   (douta2    ), // output wire [119 : 0] douta
433        .clkb     (clk_ddr    ), // input wire clk
434        .enb      (enb2      ), // input wire enb
435        .regceb   (1'bl      ), // input wire regceb
436        .web      (web2      ), // input wire [0 : 0] web
437        .addrb   (addrb2    ), // input wire [6 : 0] addr
438        .dinb     (120'b0    ), // input wire [119 : 0] dinb
439        .doutb   (doutb2    ), // output wire [119 : 0] doutb
440    );

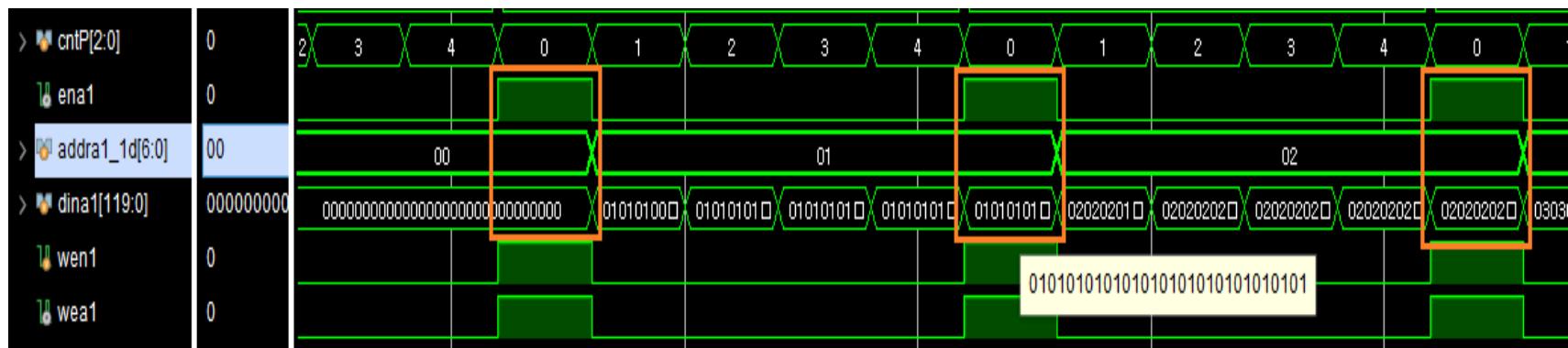
```

- ✓ 라인 404 - 405 : tdpram의 PortB는 항상 Read 로만 동작하기 때문에 web 신호는 0로 만들었습니다.
- ✓ 라인 407 - 440 : tdpram-1, 2를 사용합니다. clka에는 pclk 가, clkb에는 clk_ddr 이 사용됨을 주의하시길 바랍니다.

7.4.2 frame_write 모듈 simulation

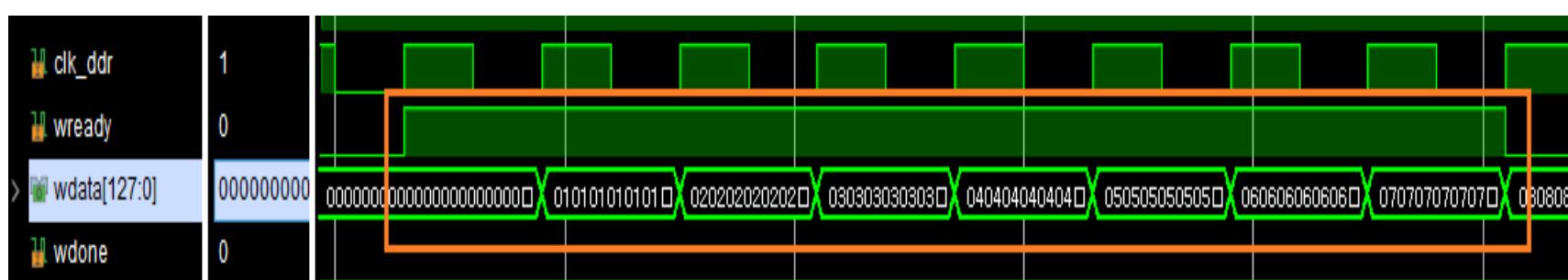
frame_write 모듈의 Test bench 는 tb_frame_write 입니다. Simulation Sources 에 2개의 파일 (tb_frame_write.v, frame_write.v)을 추가하고, tb_frame_write 를 Top Module 로 지정하고 simulation을 진행합니다. frame_write 신호들을 Wave 윈도에 추가하고 “run 6ms”을 입력해서 6ms동안 simulation을 진행합니다.

아래 그림은 tdpram-1에 write 파형입니다.



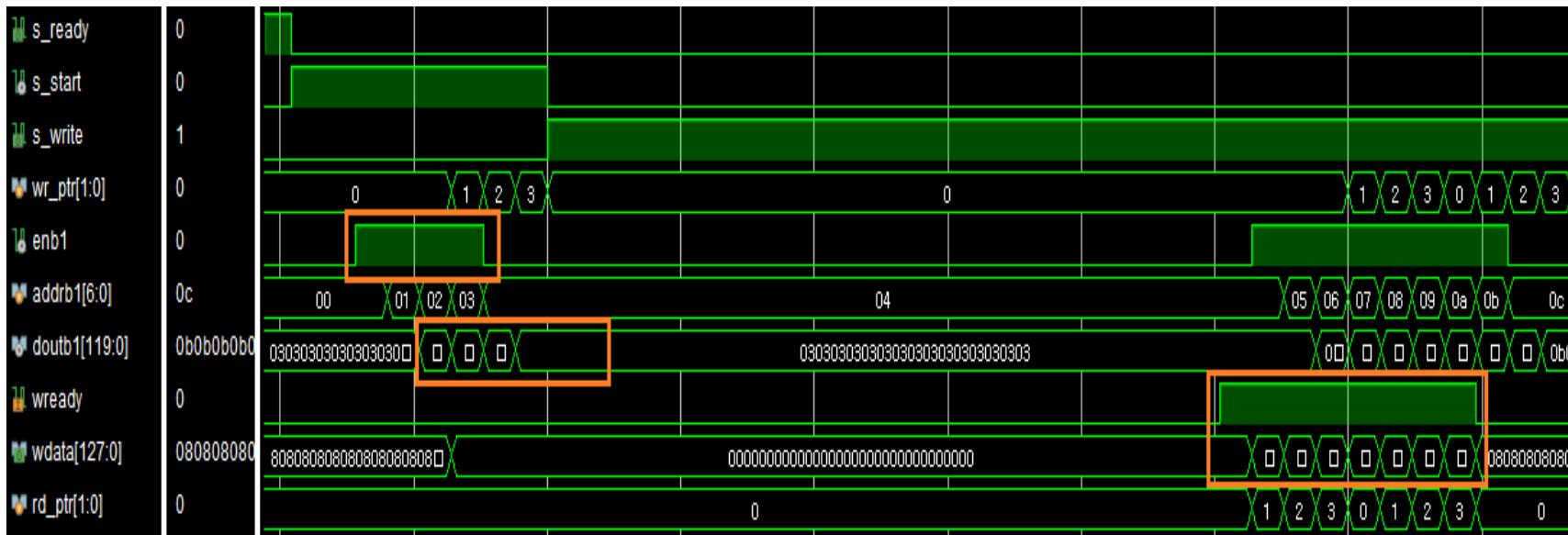
매 5 픽셀마다 데이터가 저장되고 있습니다. 0번지에 0x000~ , 1번지에 0x010101~ , 2번지에 0x020202~ 가 순서대로 저장되고 있습니다. 우리가 사용하고 있는 테스트 데이터 파일 “test_image.raw”은 00이 15개, 01이 15개, 02가 15개로 구성되어 있습니다. 즉 처음 5픽셀 값은 0x000000~ , 그 다음 5 픽셀 값은 0x010101~ , 그 다음 5 픽셀 값은 0x020202~ 입니다. 예상했던 대로 데이터가 tpdram에 잘 써지고 있습니다.

아래 그림은 tdprma-1,2의 데이터를 read 한 후에, ddr로 write 하기 위한 파형입니다.



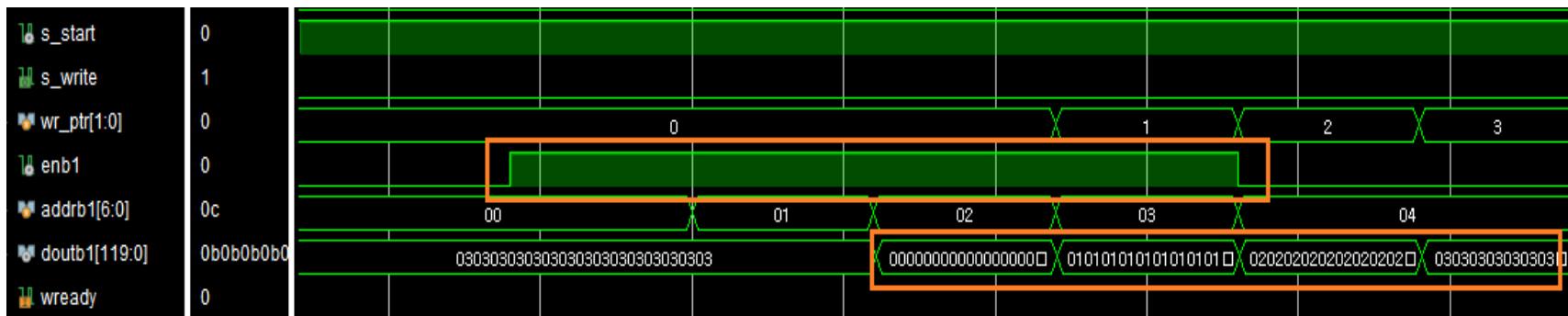
clk_ddr에 따라서 wready가 Active 일 때 wdata 값이 0x0000~ , 0x0100~ , 0x0202~ , 0x0707~ 까지 잘 전달되고 있습니다.

아래 그림은 “4-depth simple fifo”을 보여줍니다.

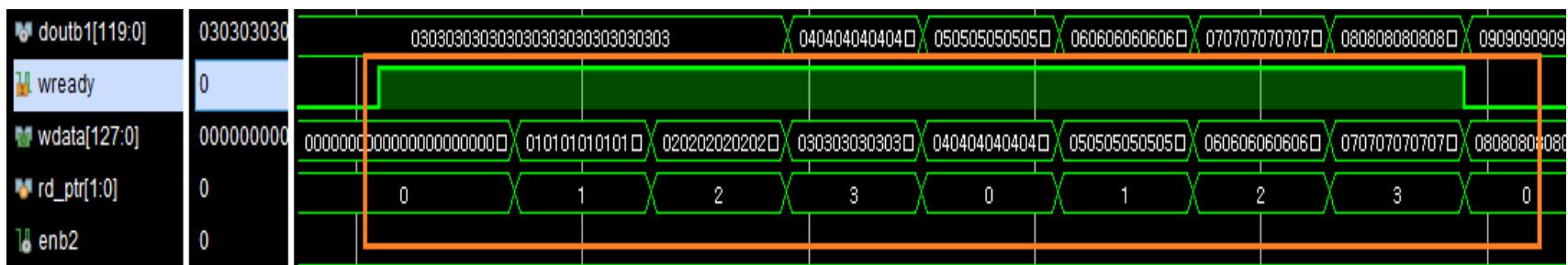


s_start 구간에서 tdpram의 데이터를 미리 4개를 읽습니다. enb1이 4-clock 동안 Active 되었고, 2-clock 후에 doutb1 데이터가 0x0000~, 0x0101~, 0x0202~, 0x0303~ 출력됩니다. ddr write 하기 위한 wready 가 발생하면 거기에 맞게 wdata의 값이 0x0000~, 0x0101~, ~~ 0x0707~ 까지 잘 전달되는 것을 확인할 수 있습니다.

좀 더 확대해서 보겠습니다. 처음 4개를 미리 읽는 파형입니다.



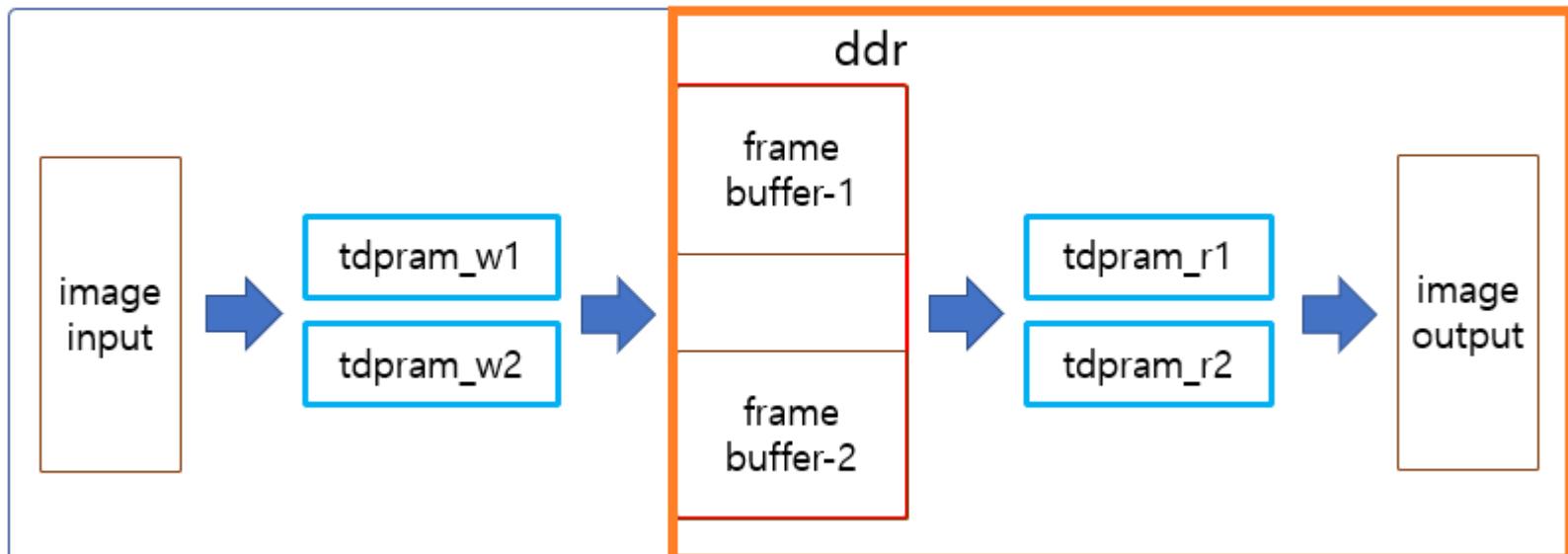
ddr에 전달되는 곳입니다.



그 외의 simulation 파형은 독자들이 직접 분석하고 확인해 보시길 바랍니다.

7.5 frame_read

이번 장에서는 ddr에 저장되어 있는 이미지 데이터를 read 해서 이미지를 출력하는 것을 구현합니다. frame_write에서 구현했던 것과 유사한 부분들이 많이 있습니다. 아래 그림에서 주황색 부분을 구현합니다. 이번장에서도 tdpram 2개를 사용하여 구현합니다.



7.5.1 frame_read 모듈 구현

```

62  input      reset      ;
63  input      pcclk     ;
64  input      vsync      ;
65  input      hsync      ;
66
67  input      rst_ddr   ;
68  input      clk_ddr   ;
69  output     rstart    ;
70  output     [27:0] raddr  ;
71  output     [9:0]  rsize  ;
72  input      [127:0] rdata  ;
73  input      rdata_valid ;
74  input      rdone     ;
75
76  output     vsync_o   ;
77  output     hsync_o   ;
78  output     [23:0] data_o  ;
79
80  output     hdd_3d   ;
81  output     vdd_3d   ;
82  output     s_ready  ;
83  output     s_start  ;
84  output     s_read   ;
  
```

- ✓ 라인 62 - 65 : reset, pcclk, vsync, hsync - pcclk domain 신호
- ✓ 라인 67 - 74 : ddr에서 read 하기 위한 신호, clk_ddr domain 신호
- ✓ 라인 76 - 78 : Image Data Outut
- ✓ 라인 80 - 84 : 디버깅 용도, Test bench에서 사용함.

```

87 // -----
88 // ddr read
89 reg vdd_1d, vdd_2d, vdd_3d, vdd_4d ;
90 always @ (posedge clk_ddr or negedge rst_ddr)
91 begin
92 if (!rst_ddr) begin
93   vdd_1d <= 1'b0;
94   vdd_2d <= 1'b0;
95   vdd_3d <= 1'b0;
96   vdd_4d <= 1'b0;
97 end
98 else begin
99   vdd_1d <= vsync ;
100  vdd_2d <= vdd_1d;
101  vdd_3d <= vdd_2d;
102  vdd_4d <= vdd_3d;
103 end
104 end
105
106 reg hdd_1d, hdd_2d, hdd_3d, hdd_4d ;
107 always @ (posedge clk_ddr or negedge rst_ddr)
108 begin
109 if (!rst_ddr) begin
110   hdd_1d <= 1'b0;
111   hdd_2d <= 1'b0;
112   hdd_3d <= 1'b0;
113   hdd_4d <= 1'b0;
114 end
115 else begin
116   hdd_1d <= hsync ;
117   hdd_2d <= hdd_1d;
118   hdd_3d <= hdd_2d;
119   hdd_4d <= hdd_3d;
120 end
121 end
122
123 wire vdd_pedge = vdd_3d & ~vdd_4d;
124 wire vdd_nedge = ~vdd_3d & vdd_4d;
125 wire hdd_pedge = hdd_3d & ~hdd_4d;
126 wire hdd_nedge = ~hdd_3d & hdd_4d;
127
128 reg [10:0] cntP;
129 always @ (posedge clk_ddr or negedge rst_ddr)
130 begin
131 if (!rst_ddr) cntP <= 11'b0;
132 else cntP <= ~hdd_3d ? 11'b0 : cntP + 1'b1 ;
133 end
134
135 reg flagLL ;
136 always @ (posedge clk_ddr or negedge rst_ddr)
137 begin
138 if (!rst_ddr) flagLL <= 1'b0;
139 else flagLL <= ~vdd_3d ? 1'b0 : hdd_pedge ? ~flagLL : flagLL;
140 end
141
142 reg flag_frame;
143 always @ (posedge clk_ddr or negedge rst_ddr)
144 begin
145 if (!rst_ddr) flag_frame <= 1'b0;
146 else flag_frame <= vdd_nedge ? ~flag_frame : flag_frame;
147 end

```

- ✓ 라인 89 - 126 : pclk 영역에서 사용하는 vsync, hsync 신호를 clk_ddr 영역에서 사용하기 위하여 flip/flop 추가함,
- 내부에서 사용하는 cntP (pixel counter), flagLL (line flag), flag_frame 을 생성합니다.

```

149 // -----
150 // ddr read
151 parameter      BLOCK_SIZE  = 10'd16;    // 640 x 24 = 128 x 12
152
153 // State Parameter
154 parameter      M_IDLE     = 2'd0;
155 parameter      M_READY    = 2'd1;
156 parameter      M_START    = 2'd2;
157 parameter      M_READ     = 2'd3;
158
159 // State Control
160 reg   [1:0]  ddr_state;
161 wire   s_idle  = (ddr_state==M_IDLE ) ? 1'b1 : 1'b0;
162 wire   s_ready = (ddr_state==M_READY) ? 1'b1 : 1'b0;
163 wire   s_start = (ddr_state==M_START) ? 1'b1 : 1'b0;
164 wire   s_read  = (ddr_state==M_READ ) ? 1'b1 : 1'b0;

```

- ✓ 라인 151 : block size = 16, rsize 로 사용됩니다.
- ✓ 라인 154 - 164 : SM을 정의합니다. ddr에서 data를 읽어서 tdpram에 저장하고, tdpram에서 읽어서 이미지 포맷에 맞추어 출력합니다.

```

200 // -----
201 // ddr read -> tpram write
202 reg   ena;
203 always @ (posedge clk_ddr or negedge rst_ddr)
204 begin
205     if (!rst_ddr)    ena <= 1'b0;
206     else            ena <= rdata_valid;
207 end
208
209 reg   [119:0] dina;
210 always @ (posedge clk_ddr or negedge rst_ddr)
211 begin
212     if (!rst_ddr)    dina <= 128'b0;
213     else            dina <= rdata[127:8];
214 end
215
216 reg   [6:0]  addra;
217 always @ (posedge clk_ddr or negedge rst_ddr)
218 begin
219     if (!rst_ddr)    addra <= 7'b0;
220     else            addra <= s_start ? 7'b0 : rdata_valid ? addra + 1'b1 : addra;
221 end
222
223 wire   enal   = ~flagLL ? ena   : 1'b0;
224 wire   ena2   = flagLL ? ena   : 1'b0;
225
226 wire   [119:0] dinal  = ~flagLL ? dina  : 120'b0;
227 wire   [119:0] dina2  = flagLL ? dina  : 120'b0;
228
229 wire   [6:0]  addral = ~flagLL ? addra : 7'b0;
230 wire   [6:0]  addra2 = flagLL ? addra : 7'b0;

```

ddr에서 읽은 데이터를 tdpram에 저장하기 위하여 신호들을 생성합니다.

```
232 // -----
233 // tpram read
234 reg [2:0] cntPP;
235 always @ (posedge pclk or negedge reset)
236 begin
237     if (!reset)      cntPP <= 3'b0;
238     else            cntPP <= ~hsync ? 3'b0 : (cntPP==3'd4) ? 3'b0 : cntPP + 1'bl ;
239 end
240
241 reg          enb;
242 always @ (posedge pclk or negedge reset)
243 begin
244     if (!reset)      enb <= 1'b0;
245     else            enb <= ~hsync ? 1'b0 : (cntPP==3'd0) ? 1'bl : 1'b0;
246 end
247
248 reg [6:0]    addrb;
249 always @ (posedge pclk or negedge reset)
250 begin
251     if (!reset)      addrb <= 7'b0;
252     else            addrb <= ~hsync ? 7'b0 : (cntPP==3'd0) ? addrb+1'bl : addrb;
253 end
254
255 reg          flagLL_1d, flagLL_2d, flagLL_3d, flagLLx;
256 always @ (posedge pclk or negedge reset)
257 begin
258     if (!reset)      begin
259         flagLL_1d <= 1'b0;
260         flagLL_2d <= 1'b0;
261         flagLL_3d <= 1'b0;
262         flagLLx   <= 1'b0;
263     end
264     else            begin
265         flagLL_1d <= flagLL    ;
266         flagLL_2d <= flagLL_1d ;
267         flagLL_3d <= flagLL_2d ;
268         flagLLx   <= flagLL_3d ;
269     end
270 end
271
272 wire          enb1 = flagLLx ? enb    : 1'b0;
273 wire          enb2 = ~flagLLx ? enb    : 1'b0;
274 wire [6:0]    addrbl = flagLLx ? addrb : 7'b0;
275 wire [6:0]    addrb2 = ~flagLLx ? addrb : 7'b0;
```

tpram에서 데이터를 read 하기 위한 신호들을 생성합니다.

```

278 // -----
279 // image output
280 reg hddx_1d, hddx_2d, hddx_3d, hsync_o;
281 always @ (posedge pclk or negedge reset)
282 begin
283 if (!reset) begin
284     hddx_1d <= 1'b0;
285     hddx_2d <= 1'b0;
286     hddx_3d <= 1'b0;
287     hsync_o <= 1'b0;
288 end
289 else begin
290     hddx_1d <= hsync ;
291     hddx_2d <= hddx_1d;
292     hddx_3d <= hddx_2d;
293     hsync_o <= hddx_3d ;
294 end
295 end
296
297 reg [2:0] cntPx;
298 always @ (posedge pclk or negedge reset)
299 begin
300     if (!reset) cntPx <= 3'b0;
301     else cntPx <= ~hddx_3d ? 3'b0 : (cntPx==3'd4) ? 3'b0 : cntPx + 1'bl ;
302 end
303
304 wire [119:0] doutbl, doutb2;
305 wire [119:0] doutb = flagLLx ? doutbl : doutb2 ;
306
307 reg [23:0] data_o;
308 always @ (posedge pclk or negedge reset)
309 begin
310     if (!reset) data_o <= 24'b0;
311     else data_o <= ~hddx_3d ? 24'b0 :
312             (cntPx==3'd0) ? doutb[119:96] :
313             (cntPx==3'd1) ? doutb[95:72] :
314             (cntPx==3'd2) ? doutb[71:48] :
315             (cntPx==3'd3) ? doutb[47:24] : doutb[23:0] ;
316 end
317
318
319 reg vddx_1d, vddx_2d, vddx_3d, vsync_o;
320 wire vddx_nedge = ~vddx_3d & vsync_o;
321 always @ (posedge pclk or negedge reset)
322 begin
323 if (!reset) begin
324     vddx_1d <= 1'b0;
325     vddx_2d <= 1'b0;
326     vddx_3d <= 1'b0;
327     vsync_o <= 1'b0;
328 end
329 else begin
330     vddx_1d <= vsync ;
331     vddx_2d <= vddx_1d;
332     vddx_3d <= vddx_2d;
333     vsync_o <= vddx_3d;
334 end
335 end

```

- ✓ 라인 278 - 335 : 이미지 출력 포맷에 맞게 데이터를 출력합니다.

```

338   always @ (posedge clk_ddr or negedge rst_ddr)
339   begin
340     if (!rst_ddr)      begin
341       ddr_state <= 2'b0;
342     end
343     else    begin
344       ddr_state <= (vdd_nedge           ) ? M_IDLE  :
345                   (s_idle   & vdd_pedge      ) ? M_READY :
346                   (s_ready & (cntP==11'd600) ) ? M_START :
347                   (s_start & (start_cnt==2'd3)) ? M_READ  :
348                   (s_read   & rdone        ) ? M_READY : ddr_state ;
349     end
350   end

```

- ✓ 라인 338 - 350 : 상태 전이를 구현합니다.

```

353   tdpram_128x120  tdpraml (
354     .clka      (clk_ddr      ), // input wire clka
355     .ena       (ena          ), // input wire ena
356     .regcea   (l'bl         ), // input wire regcea
357     .wea       (l'bl         ), // input wire [0 : 0] wea
358     .addr     (addr1        ), // input wire [6 : 0] addr
359     .dina     (dinal        ), // input wire [119 : 0] dina
360     .douta   (              ), // output wire [119 : 0] douta
361     .clkb     (pclk         ), // input wire clkb
362     .enb      (enbl         ), // input wire enb
363     .regceb   (l'bl         ), // input wire regceb
364     .web      (l'b0         ), // input wire [0 : 0] web
365     .addrb   (addrb1        ), // input wire [6 : 0] addrb
366     .dinb     (120'b0       ), // input wire [119 : 0] dinb
367     .doutb   (doutb1        ), // output wire [119 : 0] doutb
368   );
369
370   tdpram_128x120  tdpram2 (
371     .clka      (clk_ddr      ), // input wire clka
372     .ena       (ena2         ), // input wire ena
373     .regcea   (l'bl         ), // input wire regcea
374     .wea       (l'bl         ), // input wire [0 : 0] wea
375     .addr     (addr2         ), // input wire [6 : 0] addr
376     .dina     (dina2        ), // input wire [119 : 0] dina
377     .douta   (              ), // output wire [119 : 0] douta
378     .clkb     (pclk         ), // input wire clkb
379     .enb      (enb2         ), // input wire enb
380     .regceb   (l'bl         ), // input wire regceb
381     .web      (l'b0         ), // input wire [0 : 0] web
382     .addrb   (addrb2        ), // input wire [6 : 0] addrb
383     .dinb     (120'b0       ), // input wire [119 : 0] dinb
384     .doutb   (doutb2        ), // output wire [119 : 0] doutb
385   );

```

- ✓ 라인 353 - 385 : tdpram 2개를 사용합니다.

```

388 // -----
389 // frame data : save file
390 integer frame0, frame1, frame2;
391 initial frame0 = $fopen("frame0.raw");
392 initial frame1 = $fopen("frame1.raw");
393 initial frame2 = $fopen("frame2.raw");
394
395 reg [1:0] frame_cnt;
396 always @ (posedge pclk or negedge reset)
397 begin
398 if (!reset) begin
399     frame_cnt <= 2'b0;
400 end
401 else begin
402     frame_cnt <= vddx_nedge ? frame_cnt+l'bl : frame_cnt;
403
404 if (hsync_o) begin
405     if (frame_cnt==2'd0)
406         $fwrite(frame0, "%02X\n%02X\n%02X\n", data_o[23:16], data_o[15:8], data_o[7:0]);
407     else if (frame_cnt==2'd1)
408         $fwrite(frame1, "%02X\n%02X\n%02X\n", data_o[23:16], data_o[15:8], data_o[7:0]);
409     else if (frame_cnt==2'd2)
410         $fwrite(frame2, "%02X\n%02X\n%02X\n", data_o[23:16], data_o[15:8], data_o[7:0]);
411 end
412 end
413 end

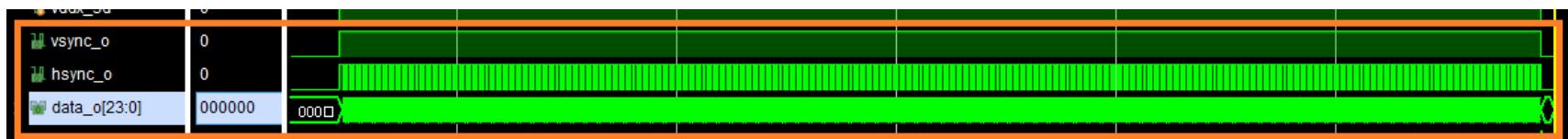
```

- ✓ 라인 388 - 413 : 이미지 출력 데이터를 파일로 저장합니다. 첫번째 프레임 데이터(frame0.raw)는 가비지 데이터가 들어 있습니다. 두번째 프레임(frame1.raw)부터 정상적인 영상데이터가 들어 있습니다.

7.5.2 frame_read 모듈 simulation

frame_read 모듈의 Test bench는 tb_frame_read입니다. Simulation Sources에 2개의 파일(tb_frame_read.v, frame_read.v)을 추가하고, tb_frame_write를 Top Module로 지정하고 simulation을 진행합니다. frame_read 신호들을 Wave 윈도에 추가하고 6ms simulation을 진행합니다.

아래 그림은 최종 이미지 출력을 나타냅니다.



tb_frame_read를 simulation 한 결과에서 data 값은 큰 의미가 없습니다. 왜냐하면 tb_frame_read에서 임의 데이터를 생성해서 입력했기 때문입니다. 이곳에서 확인해야 할 사항은 tdpram의 Write, Read 관련 신호들과 ddr read 관련 신호들이 제대로 동작하는지를 확인하는 것이 중요합니다. 아무튼 대부분은 독자분들이 확인해 보시길 바랍니다. 저자도 이 코드를 예러 없이 구현하기 위하여 수많은 simulation 확인과 코드 수정의 과정을 거쳤습니다.

다음 장에서는 최종적으로 영상 이미지 데이터를 이용한 Frame Buffer 검증입니다.

7.6 영상 데이터를 이용한 Frame Buffer 확인

이번 장에서는 “test_fruit.raw” 테스트 파일을 이용하여 Top Module에서 Simulation을 진행하고 그 결과를 확인합니다. 원래는 “test_image.raw” 파일로 먼저 simulation을 해서 ddr access 데이터가 정상적으로 동작하는지 확인해야 합니다. 저자도 “test_image.raw” 파일로 먼저 simulation을 진행한 후 데이터가 모두 오류 없이 동작함을 확인하였습니다. Top Module을 simulation하면 시간이 너무 많이 소요됩니다. 소스 폴더에 “image_dec_short.v” 파일이 있습니다. 이 파일은 simulation 시간을 단축하기 위하여 이미지 사이즈를 640x30, 640x20으로 구성하였습니다. 독자들은 이 파일을 이용해서 simulation을 진행하고 ddr access 데이터를 반드시 확인해 보시길 바랍니다.

7.6.1 image_dec.v

“test_fruit.raw” 파일로 simulation을 진행하기 위하여 아래와 같이 수정합니다.

```
42 initial begin
43     // $readmemh("test_image.raw", buffer);    // or full path
44     $readmemh("test_fruit.raw", buffer);      // or full path
45 end
```

7.6.2 mig_top_frame.v

mig_top.v에서는 ddr_test 모듈을 사용하였습니다. 프레임 버퍼를 확인하기 위해서는 ddr_test 모듈 대신에 frame_test 모듈을 사용합니다. 그 외의 것은 모두 동일합니다. (clock, reset, 이 추가되고, 필요없는 btn, led는 빠졌습니다)

```
113 frame_test      frame_test (
114     .reset        (rst_du      ),
115     .pclk         (pclk_i     ),
116     .clk_ddr     (clk_du      ),
117
118     .wstart       (mig_wstart   ),
119     .waddr        (mig_waddr   ),
120     .wslice       (mig_wsize   ),
121
122     .wdata        (mig_wdata   ),
123     .wready       (mig_wrdy   ),
124     .wdone        (mig_wdone   ),
125
126     .rstart       (mig_rstart   ),
127     .raddr        (mig_raddr   ),
128     .rslice       (mig_rsize   ),
129
130     .rdata        (mig_rdata   ),
131     .rdata_valid  (mig_rvalid  ),
132     .rdone        (mig_rdone   )
133 );
```

- ✓ 라인 113 - 133 : ddr_test 대신 frame_test 을 사용합니다.

7.6.3 Arty35Top_frame.v

Arty35Top 모듈 대신에 mig_top_frame을 사용하는 Arty35Top_frame 모듈을 사용합니다. 또한 Image 데이터를 위한 Pixel Clock (50 Mhz)를 추가합니다.

```

54  input      sys_clk_i      ;
55  input      clk_ref_i     ;
56  input      pclk_i      ;
57  output     init_calib_complete ;
58  output     tg_compare_error ;

```

- ✓ 라인 56 : pclk_i 를 추가합니다. Test bench에서 50Mhz를 생성해서 인가합니다.

```

94  mig_top_frame  mig_top_frame (
95      .sys_rst          (sys_rst           ),
96      .sys_clk_i         (sys_clk_i          ),
97      .clk_ref_i         (clk_ref_i          ),
98      .pclk_i            (pclk_i            ),
99
100     .ddr3_addr        (ddr3_addr          ),
101     .ddr3_ba          (ddr3_ba           ),
102     .ddr3_cas_n       (ddr3_cas_n         ),
103     .ddr3_ck_n        (ddr3_ck_n          ),
104     .ddr3_ck_p        (ddr3_ck_p          ),
105     .ddr3_cke         (ddr3_cke           ),
106     .ddr3_ras_n       (ddr3_ras_n         ),
107     .ddr3_reset_n     (ddr3_reset_n        ),
108     .ddr3_we_n        (ddr3_we_n          ),
109     .ddr3_dq           (ddr3_dq            ),
110     .ddr3_dqs_n       (ddr3_dqs_n         ),
111     .ddr3_dqs_p       (ddr3_dqs_p          ),
112     .ddr3_cs_n        (ddr3_cs_n          ),
113     .ddr3_dm          (ddr3_dm           ),
114     .ddr3_odt         (ddr3_odt          ),
115
116     .init_calib_complete (init_calib_complete),
117     .tg_compare_error   (tg_compare_error  )
118 );

```

- ✓ 라인 94 - 118 : mig_top_frame 모듈을 사용합니다.

7.6.4 tb_arty35Top_frame.v

Test bench는 tb_arty35Top_frame.v 입니다. tb_arty35Top.v 와 거의 비슷하고 아래 몇가지만 수정되었습니다.

```
92     parameter PCLK_PERIOD          = 10000;
93                      // Input Clock Period, 50Mhz
```

- ✓ 라인 92 : pclk_i 를 생성하기 위한 주기를 설정합니다.

```
185     reg                  pclk_i;
```

- ✓ 라인 185 : pclk_i 를 reg 로 선언합니다.

```
264     initial
265         pclk_i = 1'b0;
266     always
267         pclk_i = #PCLK_PERIOD ~pclk_i;
```

- ✓ 라인 264 - 267 : pclk_i clock 을 생성합니다.

```
378 Arty35Top_frame      Arty35Top_frame(
379     .ddr3_addr          (ddr3_addr_fpga      ),
380     .ddr3_ba             (ddr3_ba_fpga       ),
381     .ddr3_cas_n          (ddr3_cas_n_fpga    ),
382     .ddr3_ck_n           (ddr3_ck_n_fpga    ),
383     .ddr3_ck_p           (ddr3_ck_p_fpga    ),
384     .ddr3_cke            (ddr3_cke_fpga     ),
385     .ddr3_ras_n          (ddr3_ras_n_fpga   ),
386     .ddr3_reset_n        (ddr3_reset_n_fpga ),
387     .ddr3_we_n           (ddr3_we_n_fpga    ),
388     .ddr3_dq              (ddr3_dq_fpga      ),
389     .ddr3_dqs_n           (ddr3_dqs_n_fpga   ),
390     .ddr3_dqs_p           (ddr3_dqs_p_fpga   ),
391     .ddr3_cs_n            (ddr3_cs_n_fpga    ),
392     .ddr3_dm              (ddr3_dm_fpga      ),
393     .ddr3_odt             (ddr3_odt_fpga    ),
394
395     .sys_rst             (sys_rst          ),
396     .sys_clk_i            (sys_clk_i         ),
397     .clk_ref_i            (clk_ref_i         ),
398     .pclk_i               (pclk_i           ),
399     .init_calib_complete (init_calib_complete),
400     .tg_compare_error     (tg_compare_error  )
401 );
402 );
```

- ✓ 라인 378 - 402 : Arty35Top_frame 을 추가합니다.

7.6.5 simulation

Simulation Sources에 4개의 파일(tb_arty35Top_frame.v, Arty35Top_frame.v, mig_top_frame.v, frame_test.v)을 추가하고, tb_arty35Top_frame을 Top Module로 지정하고 simulation을 진행합니다. 2-frame을 진행해야 출력 이미지 데이터를 확인할 수 있습니다. 약 12.8ms 정도 진행하면 됩니다. simulation 시간이 생각보다 많이 소요됩니다. 시간을 단축하기 위해서 tb_arty35Top_frame.v의 timescale을 변경합니다.

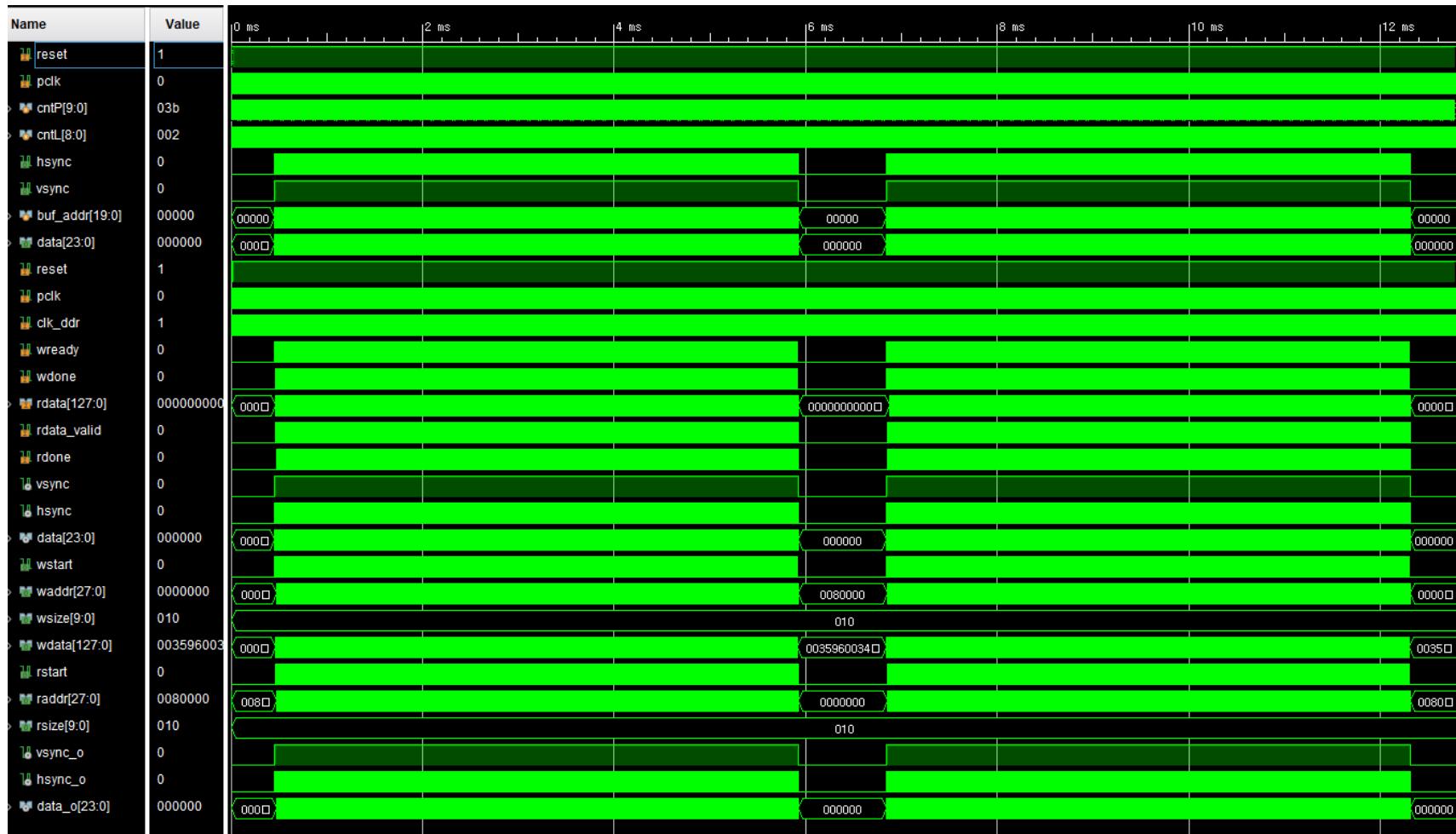
```

11
12 //`timescale 1ps/100fs
13 `timescale 1ps / 1ps
14
15 module tb_arty35Top_frame();
16

```

simulation의 resolution을 100fp → 1ps으로 변경하고 simulation을 진행합니다. 에러가 없이 simulation이 진행되면 frame_test 모듈의 신호들을 Wave 윈도에 추가하고 (필요한 신호들은 더 추가합니다) “run 12.8ms”하고 퇴근했다가 다음날 확인하시는 것이 좋습니다.

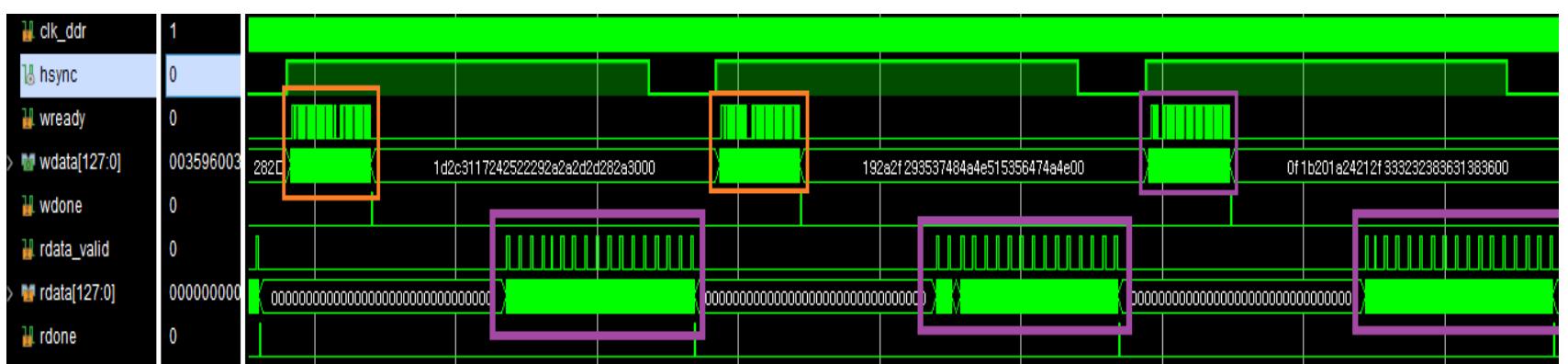
아래는 12.8ms 진행한 결과입니다. frame_test 모듈의 신호들입니다. 2-frame 정도 simulation이 진행되었습니다.



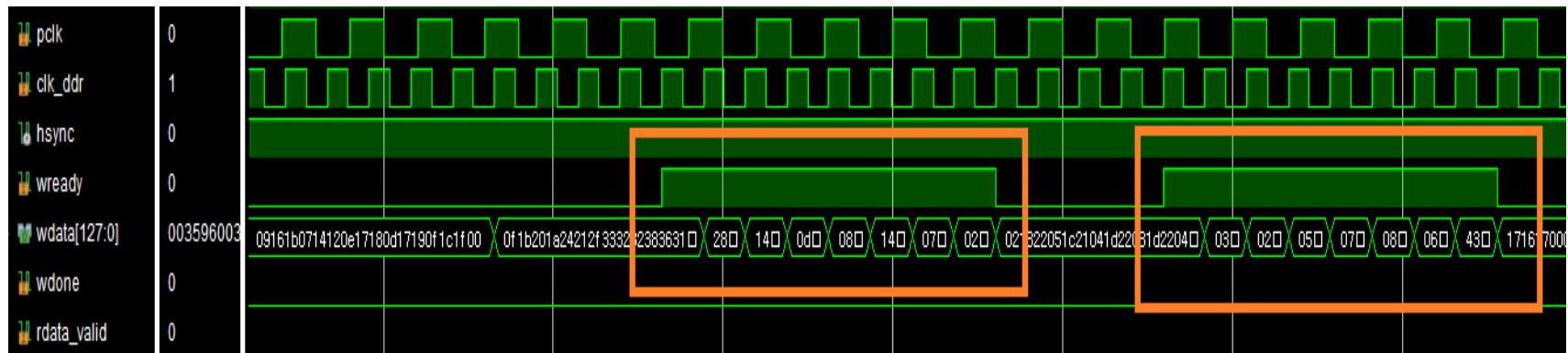
라인 단위의 파형을 확대하였습니다. 입력 Image 와 출력 Image를 표시하였습니다.



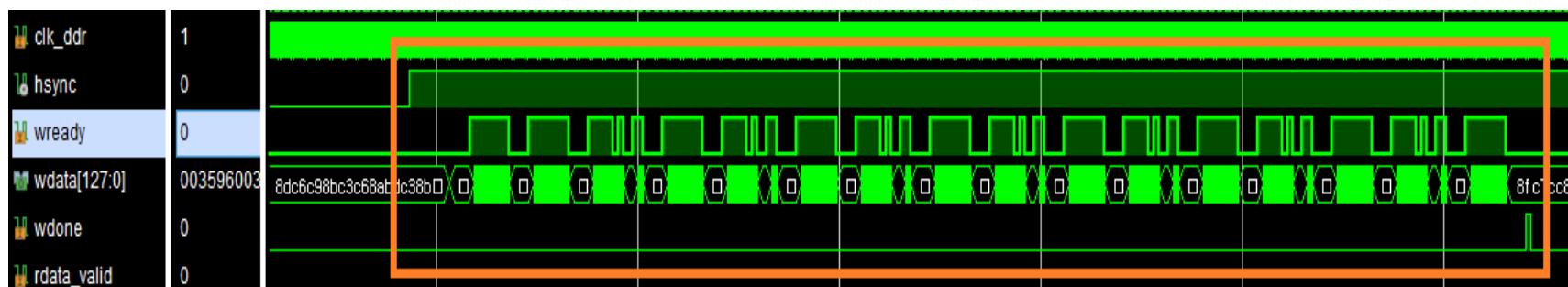
라인 단위에서 ddr access를 확인합니다. 라인 앞쪽 1/2 에서는 ddr write, 라인 뒤쪽 1/2 에서는 ddr read 가 동작합니다. 주황색 부분이 ddr write, 보라색 부분이 ddr read 입니다. 만일 1 라인 안에 ddr write, read를 다 할 수 없으면 다른 방법을 찾아야 합니다. ddr을 2개를 사용하면 속도를 2배로 높일 수 있습니다. ddr clock 속도를 높일 수도 있지만 HW 이슈가 있어서 신중해야 합니다. mig7_read8 을 수정해서 한번에 16개를 read 하게 코드를 수정해도 조금 나아질 수 있을 것 같습니다.



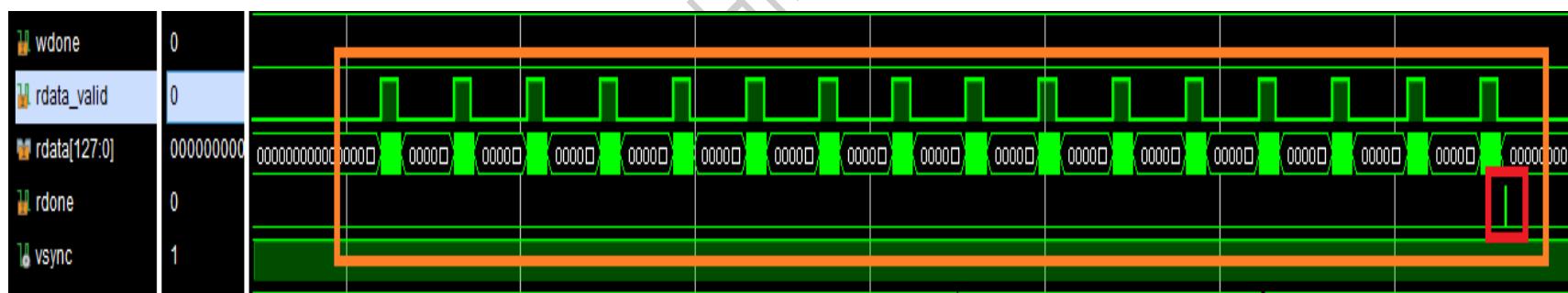
ddr write 부분을 보여줍니다. 128bits x 8 개씩 write 되고 있습니다.



한라인 전체 ddr write 입니다. 뒤에 wdone 이 Active 됩니다.



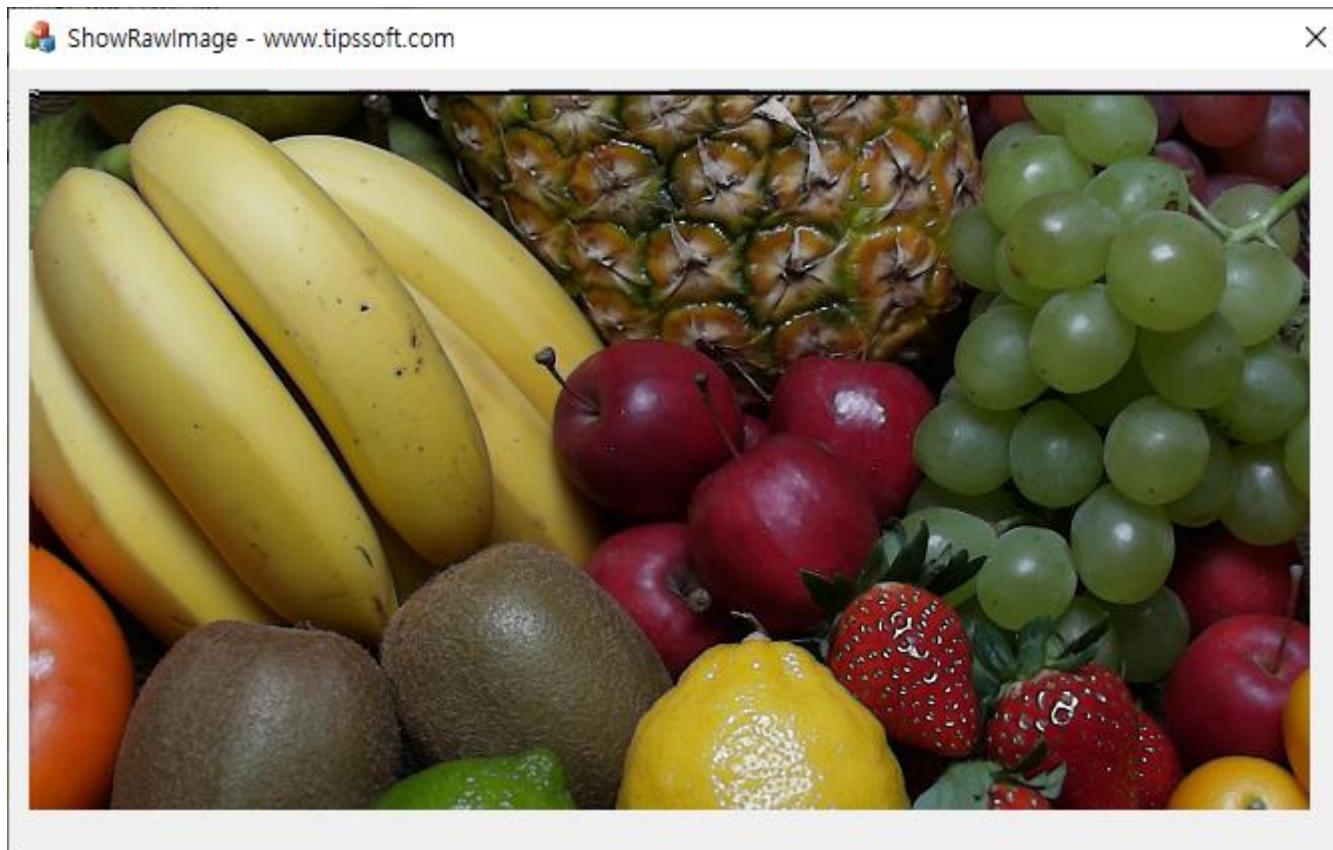
ddr read 부분입니다. 1-라인 전체를 read 후에 rdone 이 Active 됩니다.



simulation을 종료하면 최종적으로 “frame0.raw”, “frame1.raw”, “frame2.raw” 파일이 생성됩니다. “frame0.raw” 파일에는 가비지 데이터가 들어있고, “frame1.raw” 파일에는 정상적인 이미지 데이터가 들어 있습니다. “frame2.raw” 파일은 아직 3번째 frame까지는 진행되지 않아서 데이터가 없을 것입니다. (simulation을 종료하지 않으면 frame1.raw 파일에 아무런 내용도 없습니다. simulation을 종료해야 결과가 저장됩니다)

7.6.6 이미지 확인하기

제공받은 “ShowRawImage.exe” 파일을 실행하면 파일 Open Dialog가 나타납니다. “frame1.raw” 파일을 찾아서 선택하면 아래와 같은 화면이 나타납니다.



이미지의 첫라인의 데이터가 이상하게 보입니다. 이것은 이미지 데이터를 tdpram을 거쳐 ddr에 저장할 때, 처음 첫번째 라인을 처리할 때에는 tdpram에 있는 가비지 데이터가 ddr에 저장되었기 때문입니다. 이를 해결하기 위해서는 ddr에 저장할 때 2번째 라인부터 저장하면 됩니다. 또는 ddr에서 이미지를 읽을 때 2번째 라인부터 읽으면 됩니다. 이 부분은 직접 구현해 보시길 바랍니다.

여기까지 해서 모든 과정을 마치도록 하겠습니다. 여기까지 온다고 수고 많으셨습니다. 본문에 있는 내용들과 소소들은 매우 중요한 내용을 포함하고 있습니다. 충분히 분석하시고 정독하셔서 본인의 것으로 만드시면 좋은 개발자가 될 것으로 기대합니다.

혹 질문사항이 있으시면 카페(<https://cafe.naver.com/worshippt>)나 메일(alex@ihil.co.kr)에 남겨주시길 바랍니다.

감사합니다.

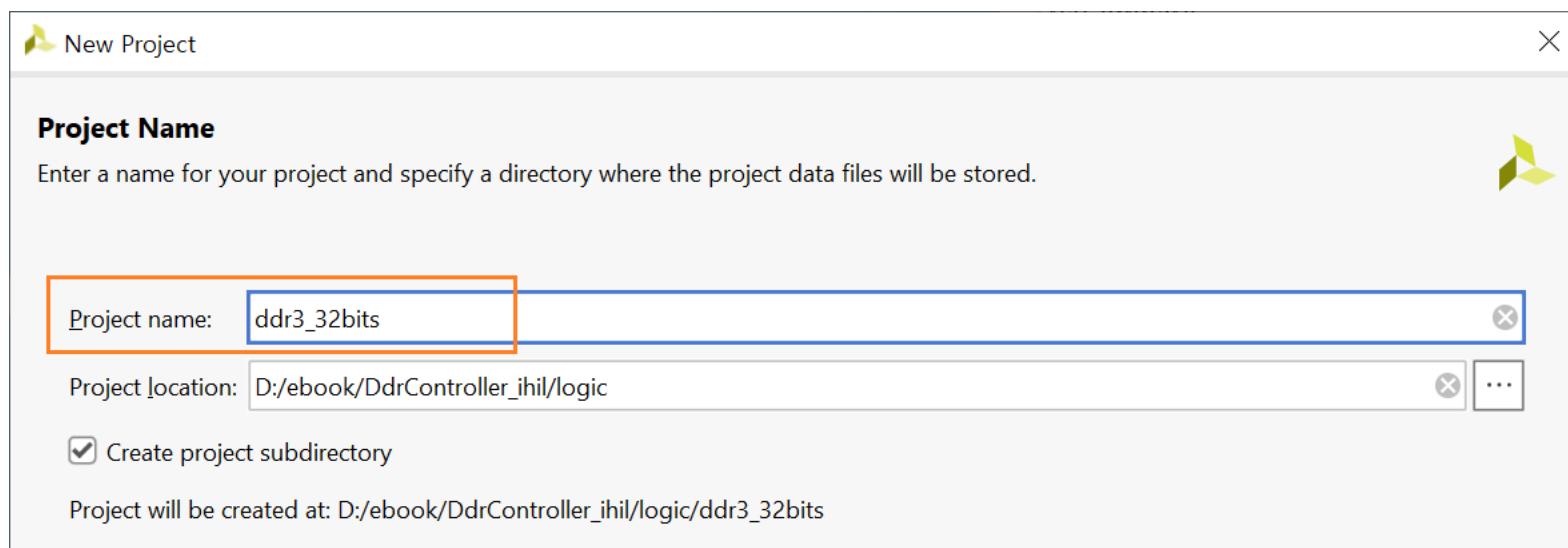
8. 32bits Interface 구현

이번장에서는 32bits Interface를 구현합니다. 고속으로 데이터를 처리해야하는 경우에 16bits ddr 2개를 사용하여 32bits Interface를 구현하여 사용합니다. Address와 기타 제어 신호들은 공통으로 사용하고, data를 16bits에서 32bits로 확장하여 사용합니다. 32bits Interface를 구현하기 위해서는 IP를 생성할 때부터 Data Width를 32bits로 설정해야 합니다. 이번 장의 내용은 보드 검증은 하지 않고, simulation으로 결과를 확인합니다.

이번장은 3-5장에서 구현한 내용을 확장해서 사용합니다. 이미지 처리용으로 사용하는 DDR Controller는 기본적인 내용만 이해하면 그 내용을 바탕으로 응용해서 사용할 수 있고, 기본적인 구성은 동일하다는 것을 이장을 통해서 이해할 수 있습니다.

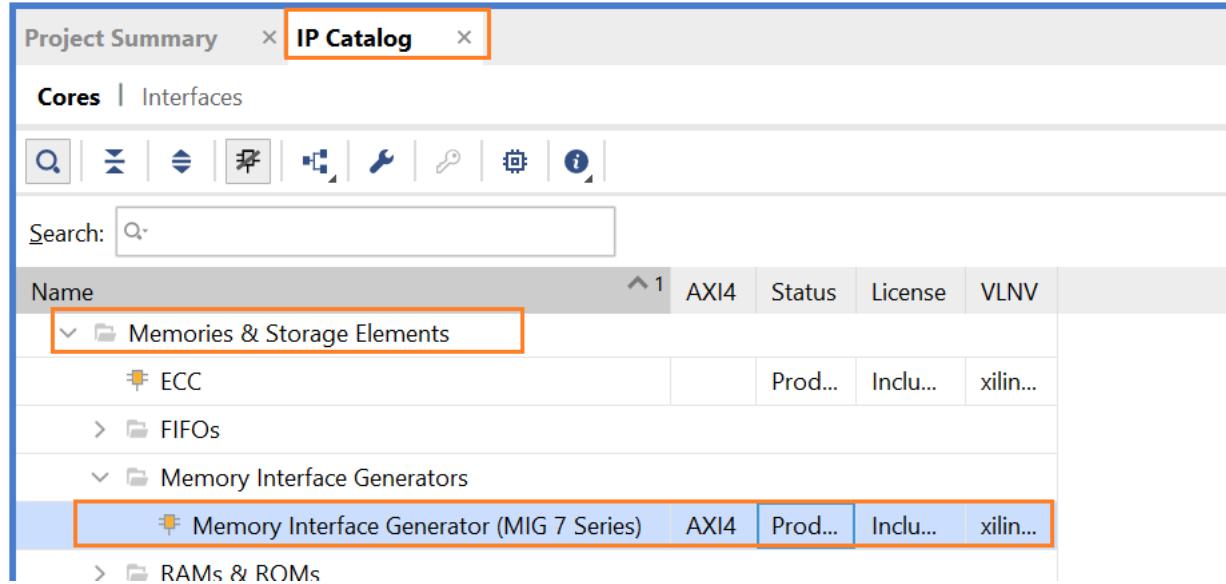
8.1 프로젝트 생성

프로젝트를 생성합니다. “3.1 프로젝트 생성”을 참조해서 프로젝트를 생성합니다. Project Name은 “ddr3_32bits”하고, Part는 “xc7a35tcs324-1”로 합니다.



8.2 Memory IP 생성

Memory IP를 생성합니다. “3.2 Memory IP 생성”을 참조하시길 바랍니다. IP Catalog - Memories & Storage Elements - Memory Interface Generator (MIG 7 Series)를 더블 클릭합니다.



Next를 클릭합니다.

Memory Interface Generator

The Memory Interface Generator (MIG) creates memory controllers for Xilinx FPGAs. MIG creates complete customized Verilog or VHDL RTL source code, pin-out and design constraints for the FPGA selected, and script files for implementation and simulation.

Vivado Project Options

This GUI includes all configurable options along with explanations to aid in generation of the required controller. Please note that some of the options selected in the Vivado Project Options will be used in generation of the controller. It is very important that the correct Vivado Project Options are selected. These options are listed below.

Selected Vivado Project Options:

- Fpga Family : Artix-7
- Fpga Part : xc7a35t-csg324
- Speed Grade : -1
- Synthesis Tool : VIVADO
- Design Entry : VERILOG

If any of these options are incorrect, please click on "Cancel", change the Vivado Project Options, and restart MIG. This version of MIG is tested with Vivado 2018.3 or later, it is not tested with previous versions of Vivado.

Next > **Cancel**

Component Name : ddr32로 입력합니다. Number of controllers는 1을 사용합니다. ddr3를 2개를 사용한다고 해서 2로 입력하지 않습니다. 추후의 설정값에서 Data Width를 32로 설정할 것입니다. Next를 클릭합니다.

MIG Output Options

- Create Design
Select this option to generate a memory controller. Generating a memory controller will create RTL, XDC, implementation and simulation files.
- Verify Pin Changes and Update Design
Selecting this feature verifies the modified XDC for a design already generated through MIG. This option will allow you to change the pin out and validate it instantly. It updates the input XDC file to be compatible with the current version of MIG. While updating the XDC it preserves the pin outs of the input XDC. This option will also generate the new design with the Component Name you selected in this page.

Component Name

Please specify the component name for the memory interface. The design directories will be generated under a directory with this name. Three directories will be created "example_design", "user_design" and "docs". The user_design will contain the generated memory interface. The example_design adds a simple example application connected to the generated memory interface.

Component Name

Multi-Controller

Up to maximum of 8 controllers with a combination of DDR3 SDRAM, QDRII+ SRAM or RLDRAM II can be generated. The number of controllers that can be accommodated may be limited by the data width and the number of banks available in device. Refer user guide for more information

Number of controllers

AXI4 Interface

Enables the AXI4 interface. AXI4 interface is supported only for DDR3 SDRAM and DDR2 SDRAM controllers with Verilog design entry.

AXI4 Interface

Pin Compatible FPGAs에서는 기본값을 그대로 사용하고 Next를 클릭합니다.

Memory Selection은 DDR3 SDRAM을 선택하고 Next를 클릭합니다.

Memory Selection

Select the type of memory interface. Please refer to the User Guide for a detailed list of supported controllers for each FPGA family. The list below shows currently available interface(s) for the specific FPGA, speed grade and design entry chosen.

Select the controller type:

- DDR3 SDRAM**
- DDR2 SDRAM
- LPDDR2 SDRAM

Clock Period는 최대값인 2500 (400 MHz)을 사용하도록 하겠습니다. simulation으로만 검증하기 때문에 최대 주파수를 사용합니다. 만일 보드에서 검증하기 위해서는 400MHz로 구현해보고, 동작이 안되면 주파수를 낮추어서 구현해야 합니다.

Clock Period를 2500으로 설정하면, PHY to Controller Clock Ratio 값은 4:1로 설정됩니다. Memroy Type은 MT41K128M16XX-15E로 설정합니다. Memory Voltage 값은 400 Mhz(2500 ps)로 설정시 1.5V로 설정됩니다. 따라서 HW를 설계할 때, 400MHz로 구현하기 위해서는 1.35V가 아닌 1.5V로 설계해야 합니다. Data Width는 32로 설정합니다. ORDERING은 Normal로 설정합니다. Next를 클릭합니다.

Options for Controller 0 - DDR3 SDRAM

Clock Period: Choose the clock period for the desired frequency. The allowed period range(**2500 - 3300**) is a function of the selected FPGA part and FPGA speed grade. Refer to the User Guide for more information.

PHY to Controller Clock Ratio: Select the PHY to Memory Controller clock ratio. The PHY operates at the Memory Clock Period chosen above. The controller operates at either 1/4 or 1/2 of the PHY rate. The selected Memory Clock Period will limit the choices.

Memory Type: Select the memory type. Type(s) marked with a warning symbol are not compatible with the frequency selection above.

Memory Part: Select the memory part. Part(s) marked with a warning symbol are not compatible with the frequency selection above. Find an equivalent part or create a part using the "Create Custom Part" button if the part needed is not listed here. The "Create Custom Part" feature is not supported for RLDRAM II.

Memory Voltage: Select the Voltage of the Memory part selected.

Data Width: Select the Data Width. Parts marked with a warning symbol are not compatible with the frequency and memory part selected above.

ECC: MIG supports ECC for 72 bit data width configuration. To be able to select ECC, select a data width that has ECC supported.

Data Mask: Enable or disable the generation of Data Mask (DM) pins using this check box. This option can be selectable only if the memory part selected has DM pins. Uncheck this box to not use data masks and save FPGA I/Os that are used for DM signals. ECC designs (DDR3 SDRAM, DDR2 SDRAM) will not use Data Mask.

Number of Bank Machines: This parameter defines the number of bank machines. A given bank machine manages a single DRAM bank at any given time.

Note: Setting a lower value will result in lower resource utilization, but may effect controller efficiency for certain traffic patterns.

ORDERING: Normal mode allows the memory controller to reorder commands to the memory to obtain the highest possible efficiency. Strict mode forces the controller to execute commands in the exact order received.

Memory Details: 2Gb, x16, row:14, col:10, bank:3, data bits per strobe:8, with data mask, single rank, 1.35V, 1.5V

[**< Back**](#) [**Next >**](#) [**Cancel**](#)

기본 설정값을 그대로 사용합니다. Next를 클릭합니다.

Memory Options C0 - DDR3 SDRAM

Input Clock Period: Select the period for the PLL input clock (CLKIN). MIG determines the allowable input clock periods based on the Memory Clock Period entered above and the clocking guidelines listed in the User Guide. The generated design will use the selected Input Clock and Memory Clock Periods to generate the required PLL parameters. If the required input clock period is not available, the Memory Clock Period must be modified.

2500 ps (400 MHz)

Choose the Memory Options for the memory device. Memory Option selections are restricted to those supported by the controller. Consult the memory vendor data sheet for more information.

Read Burst Type and Length

The burst type determines the data ordering within a burst. Consult the memory datasheet for more information. Burst length 8 is the only supported value.

Sequential

Output Driver Impedance Control

Programmable impedance for the output buffer.

RZQ/7

RTT (nominal) - On Die Termination (ODT)

Select the nominal value of ODT for the DQ, DQS/DQS# and DM signals on the component or DIMM interface. This must be set to RZQ/6 (40 ohms) for data rates at 1333 Mbps and above. In 2 slot DIMM configurations this value will be used for the unwritten slot during a write and will also be used for the unselected slot during a read. Use board level simulation to choose the optimum value.

RZQ/4

Controller Chip Select Pin

The Chip Select (CS#) pin can be tied low externally to save one pin in the address/command group when this selection is set to 'Disable'. Disable is only valid for single rank configurations.

Enable

Memory Address Mapping Selection

User Address

ROW BANK COLUMN

BANK ROW COLUMN

< Back Next > Cancel

System Clock, Reference Clock은 내부에서 PLL을 생성해서 사용하기 합니다. No Buffer를 선택합니다. Internal Vref 핀은 사용하지 않습니다. Normal IO Pin으로 사용하기 때문에 Internal Vref 항목을 Check 합니다. 나머지 항목은 기본값을 사용합니다. Next를 클릭합니다.

System Clock
Choose the desired input clock configuration. Design clock can be Differential or Single-Ended.
System Clock No Buffer

Reference Clock
Choose the desired reference clock configuration. Reference clock can be Differential or Single-Ended.
Reference Clock No Buffer

System Reset Polarity
Choose the desired System Reset Polarity.
System Reset Polarity ACTIVE LOW

Debug Signals Control
This feature allows various debug signals present in the IP to be monitored on the ChipScope tool. The debug signals include status signals of various PHY calibration stages. Enabling this feature will connect all the debug signals to the ChipScope ILA and VIO cores in the example design top module. A part of each bus in the debug interface has been grounded so that users can replace the grounded signals with the required signals.
Debug Signals for Memory Controller OFF

Sample Data Depth
This selects the value of Sample Data depth for Chipscope ILA used in Debug logic.
Sample Data Depth 1024

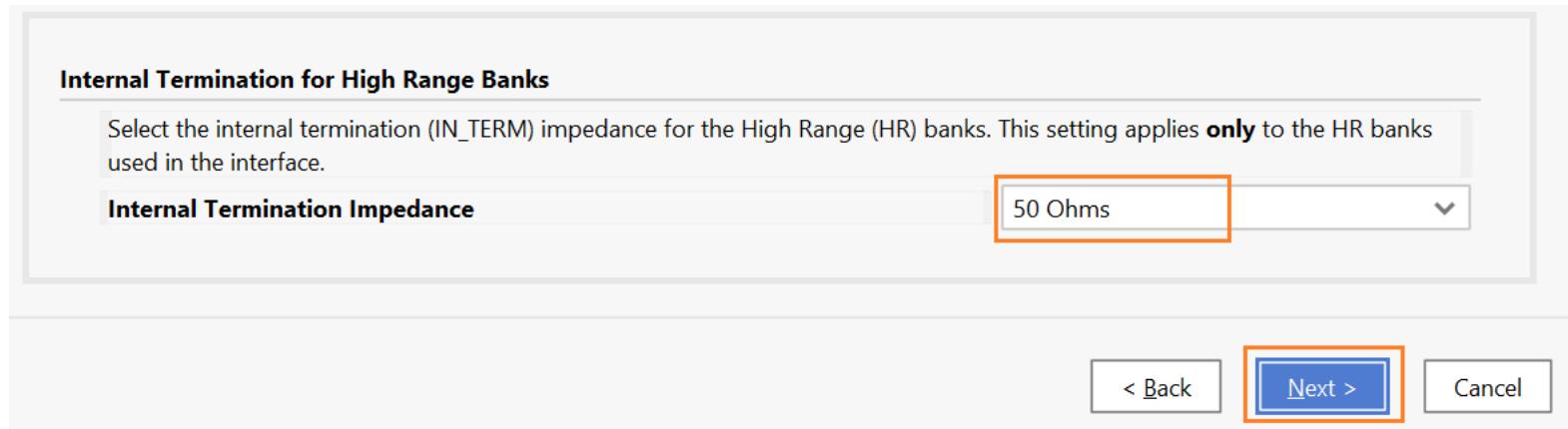
Internal Vref
Internal Vref can be used to allow the use of the Vref pins as normal IO pins. This option can only be used at 800 Mbps and lower data rates. This can free 2 pins per bank where inputs are used. This setting has no effect on banks with only outputs.
Internal Vref

IO Power Reduction
Significantly reduces average IO power by automatically disabling DQ/DQS IBUFs and internal terminations during WRITES and periods of inactivity.
IO Power Reduction ON

XADC Instantiation
The memory interface uses the temperature reading from the XADC block to perform temperature compensation and keep the read DQS centered in the data window. There is one XADC block per device. If the XADC is not currently used anywhere in the design, enable this option to have the block instantiated. If the XADC is already used, disable this MIG option. The user is then required to provide the temperature value to the top level 12-bit device_temp_i input port. Refer to Answer Record 51687 or the UG586 for detailed information.
XADC Instantiation Enabled

< Back **Next >** Cancel

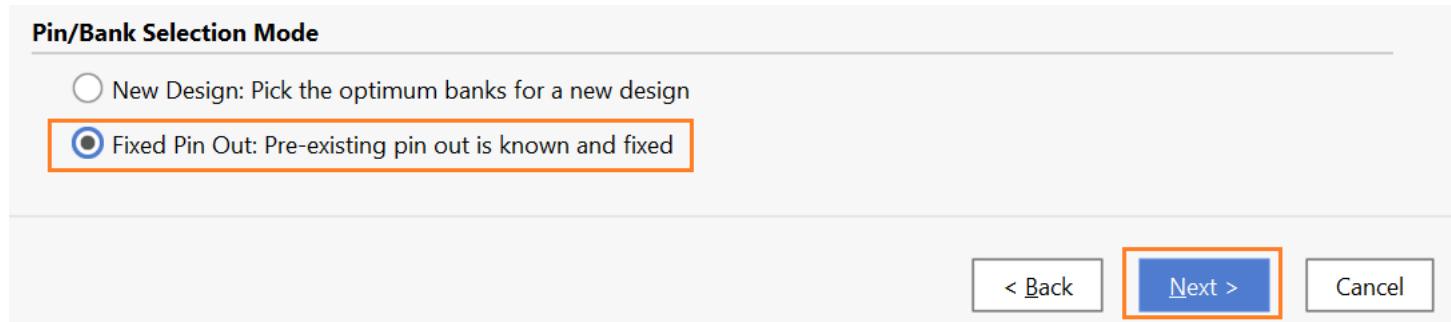
Internal Termination Impedance는 기본값 50 Ohms을 사용합니다. Next를 클릭합니다.



Pin/Bank Selection Mode는 Fixed Pin Out 을 선택합니다. DDR Memory Interface를 구현하기 위해서는 핀 할당이 매우 중요합니다. 아무 핀이나 사용하면 에러가 발생합니다. 따라서 HW를 설계하기 전에 Memory IP를 생성해서 Pin/Bank 항목을 설정하고, Validation 검증을 해서 핀 설정에 오류가 없는지 반드시 확인해야 합니다. 본 강의에서는 이미 설정된 파일을 불러와서 사용하도록 하겠습니다. 핀을 할당할 때, DQS핀은 반드시 핀 중에 DQS로 된 핀을 사용해야 합니다. 또한 DQS[0]와 dq[7:0]는 Bank Number와 Byte Number가 동일해야 합니다. DSQ[1]과 dq[15:8]도 Bank Number, Byte Number가 동일해야 합니다. DSQ[2]는 dq[23:16], DSQ[3]은 DQ[31:24]와 Bank Number Byte Number를 맞추어야 합니다.

본 강의에서는 Bank34, 35를 사용합니다. (HW로 구현한다면, Bank34, 35의 VCCO는 1.5V를 사용해야 합니다)

Next를 클릭합니다.



Read XDC/UCF 버튼을 클릭해서 미리 할당된 파일을 불러옵니다. “ddr3_32bits.ucf” 파일을 불러옵니다.

Pin Selection For Controller 0 - DDR3 SDRAM

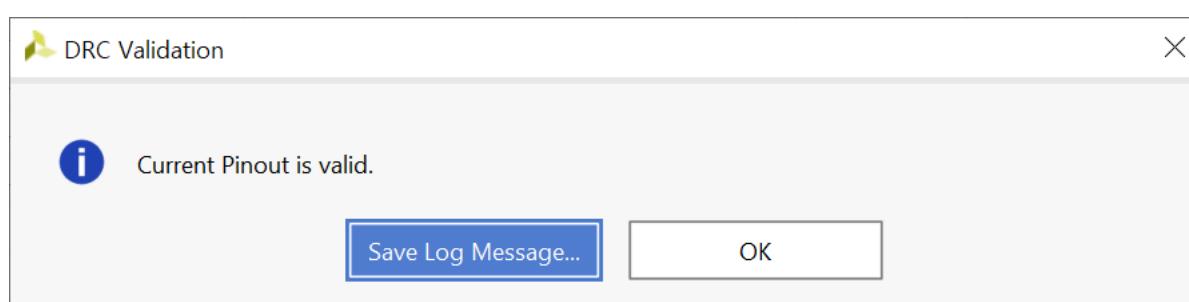
| | Signal Name | Bank Number | Byte Number | Pin Number | IO Standard |
|----|-------------|-------------|-------------|------------|-------------|
| 1 | ddr3_dq[0] | 34 | T0 | K5 | SSTL15 |
| 2 | ddr3_dq[1] | 34 | T0 | L3 | SSTL15 |
| 3 | ddr3_dq[2] | 34 | T0 | K3 | SSTL15 |
| 4 | ddr3_dq[3] | 34 | T0 | L6 | SSTL15 |
| 5 | ddr3_dq[4] | 34 | T0 | M3 | SSTL15 |
| 6 | ddr3_dq[5] | 34 | T0 | M1 | SSTL15 |
| 7 | ddr3_dq[6] | 34 | T0 | L4 | SSTL15 |
| 8 | ddr3_dq[7] | 34 | T0 | M2 | SSTL15 |
| 9 | ddr3_dq[8] | 34 | T1 | V4 | SSTL15 |
| 10 | ddr3_dq[9] | 34 | T1 | T5 | SSTL15 |
| 11 | ddr3_dq[10] | 34 | T1 | U4 | SSTL15 |
| 12 | ddr3_dq[11] | 34 | T1 | V5 | SSTL15 |
| 13 | ddr3_dq[12] | 34 | T1 | V1 | SSTL15 |
| 14 | ddr3_dq[13] | 34 | T1 | T3 | SSTL15 |
| 15 | ddr3_dq[14] | 34 | T1 | U3 | SSTL15 |
| 16 | ddr3_dq[15] | 34 | T1 | R3 | SSTL15 |
| 17 | ddr3_dq[16] | 35 | T0 | C6 | SSTL15 |
| 18 | ddr3_dq[17] | 35 | T0 | C5 | SSTL15 |
| 19 | ddr3_dq[18] | 35 | T0 | B7 | SSTL15 |
| 20 | ddr3_dq[19] | 35 | T0 | B6 | SSTL15 |
| 21 | ddr3_dq[20] | 35 | T0 | D8 | SSTL15 |
| 22 | ddr3_dq[21] | 35 | T0 | C7 | SSTL15 |
| 23 | ddr3_dq[22] | 35 | T0 | E6 | SSTL15 |
| 24 | ddr3_dq[23] | 35 | T0 | E5 | SSTL15 |
| 25 | ddr3_dq[24] | 35 | T1 | C4 | SSTL15 |
| 26 | ddr3_dq[25] | 35 | T1 | B4 | SSTL15 |
| 27 | ddr3_dq[26] | 35 | T1 | A4 | SSTL15 |
| 28 | ddr3_dq[27] | 35 | T1 | A3 | SSTL15 |
| 29 | ddr3_dq[28] | 35 | T1 | B3 | SSTL15 |
| 30 | ddr3_dq[29] | 35 | T1 | B2 | SSTL15 |
| 31 | ddr3_dq[30] | 35 | T1 | D5 | SSTL15 |

Validation successful. Press Next to proceed.

Validate **Read XDC/UCF** **Save Pin Out**

< Back **Next >** **Cancel**

각각의 핀들이 할당됩니다. Validate 버튼을 클릭해서 유효성을 확인합니다.



에러가 없음을 확인하고 OK를 클릭합니다. 만일 에러가 발생하면 해당 핀을 수정한 후 다시 Validate 를 확인합니다.

에러가 없으면 Next 버튼이 활성화 됩니다. Next를 클릭합니다.

System Signals Selection은 나중에 Top Module의 xdc 파일에서 설정합니다. 기본값을 사용합니다. Next를 클릭합니다.

System Signals Selection

Select the system pins below appropriately for the interface. Customization of these pins can also be made in the XDC after the design is generated. For more information see [UG586 Bank and Pin rules](#).

System Clock and Reference Clock pin selections will not be visible if the 'No Buffer' option was selected in the FPGA Options page.

System Signals

These signals may be connected internally to other logic or brought out to a pin.

- **sys_rst**: This input signal is used to reset the interface.
- **init_calib_complete**: This signal indicates that the interface has completed calibration and memory initialization and is ready for commands. LOC constraint will be generated in XDC for Example design only based on "Pin Number" selection below.
- **error**: This output signal indicates that the traffic generator in the Example Design has detected a data mismatch. This signal does not exist in the User Design.

| Signal Name | Bank Number | Pin Number |
|---------------------|-------------|------------|
| sys_rst | Select Bank | No connect |
| init_calib_complete | Select Bank | No connect |
| tg_compare_error | Select Bank | No connect |

All pins must be constrained to specific locations in order to generate a bit file in the implementation phase (this is not required for simulation).

< Back

Next >

Cancel

지금까지 생성된 내용을 보여줍니다. Next를 클릭합니다.

```
Vivado Project Options:
  Target Device           : xc7a35t-csg324
  Speed Grade            : -1
  HDL                    : verilog
  Synthesis Tool          : VIVADO

If any of the above options are incorrect, please click on "Cancel", change the CO

MIG Output Options:
  Module Name             : ddr32
  No of Controllers       : 1
  Selected Compatible Device(s) : --
  Selected Compatible Device(s) : --

FPGA Options:
  System Clock Type       : No Buffer
  Reference Clock Type    : No Buffer
  Debug Port               : OFF
  Internal Vref            : enabled
  IO Power Reduction       : ON
  XADC instantiation in MIG : Enabled

Extended FPGA Options:
  DCI for DQ,DQS/DQS#,DM   : enabled
  Internal Termination (HR Banks) : 50 Ohms

/*****
 *          Controller 0
 */
Controller Options :
  Memory                  : DDR3_SDRAM
  Interface                : NATIVE
  Design Clock Frequency   : 2500 ps (400.00 MHz)
  Phy to Controller Clock Ratio : 4:1
  Input Clock Period       : 2499 ps
  CLKFBOUT_MULT (PLL)      : 2
  DIVCLK_DIVIDE (PLL)       : 1
  VCC_AUX IO               : 1.8V
  Memory Type              : Components
  Memory Part              : MT41K128M16XX-15E
  Equivalent Part(s)        : --
  Data Width                : 32

< Back   Next >   Cancel
```

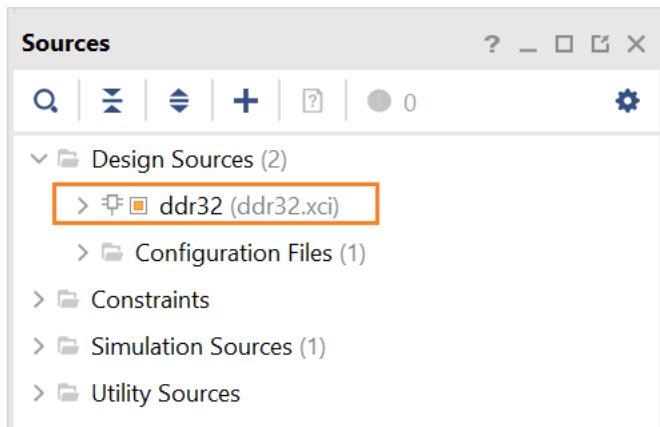
License 관련 내용입니다. Accept를 선택하고 Next를 클릭합니다.

Generate를 클릭해서 IP를 생성합니다.

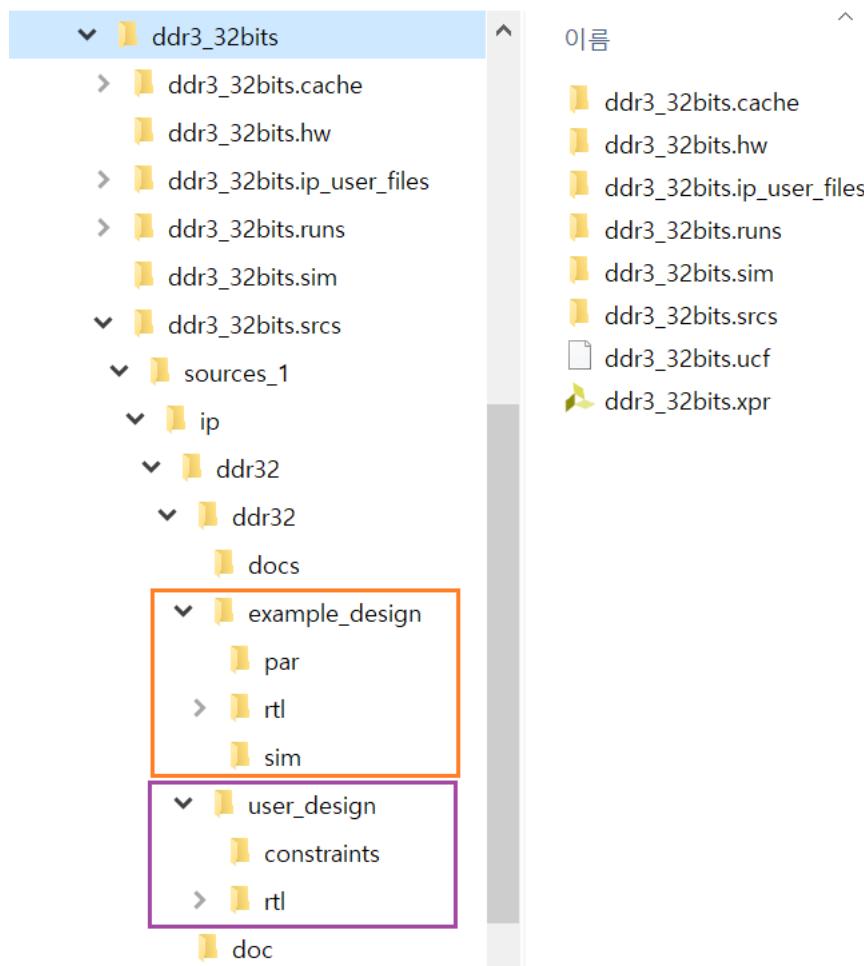
다음 장에서는 생성된 IP의 동작을 이해하기 위하여 simulation을 진행합니다.

8.3 Simulation을 통한 생성된 IP 동작 이해

이번 장에서는 생성된 IP의 동작을 이해합니다. 아래 그림은 생성된 IP를 보여줍니다.

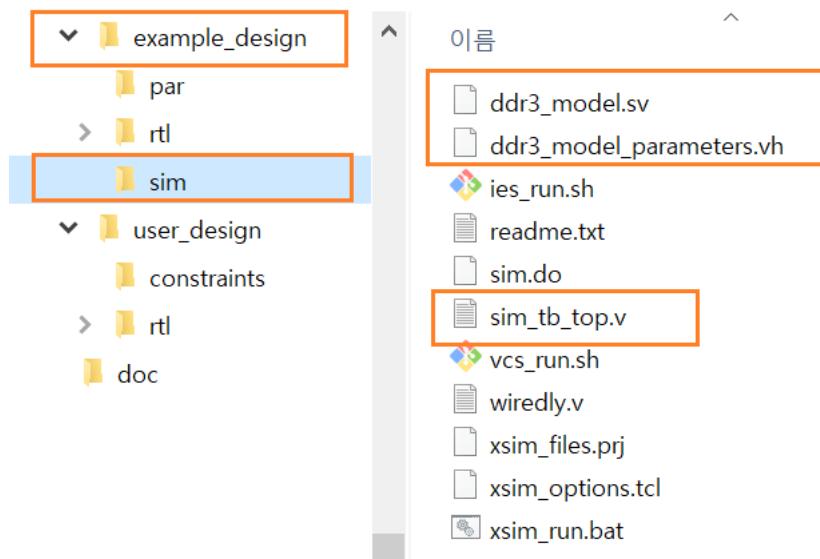


파일 구조는 아래와 같습니다.

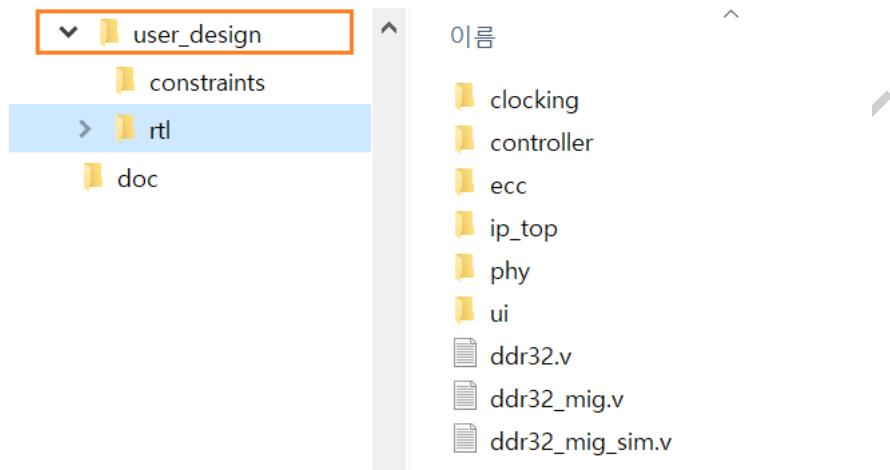


example_design 폴더와 user_design 폴더가 있습니다. example_design 폴더에는 simulation을 할 수 있는 ddr3 memory model 과 simulation top module과 simulation에 필요한 파일들(Test Pattern 생성 등)로 구성되어 있습니다. user_design 폴더는 실제 구현에 필요한 파일들이 있습니다.

아래 그림은 example_design 폴더를 보여줍니다. sim 폴더안에 “sim_tb_top.v”는 simulation Top Module 입니다. ddr3_modle.sv, ddr3_model_parameters.vh 는 ddr3 memory modeling 파일입니다. rtl 폴더에는 simulation을 위한 Test Pattern (Test Data)를 구현하는 파일들이 있습니다.



아래 그림은 user_design 폴더를 보여줍니다. rtl 폴더의 ddr32.v 는 Top module 입니다. 그 외의 design 파일들이 있습니다. constraints 폴더에는 xdc 파일이 있습니다.



vivado의 Design Sources - ddr32 (ddr32.xci) 안에 user_design 폴더의 파일들이 모두 포함되어 있습니다. 따라서 user_design 폴더의 파일들은 별도로 프로젝트에 추가할 필요가 없습니다.

simulation을 진행하기 위해서는 Simulation Sources 에 example_design 폴더 안에 있는 파일들을 추가합니다.

simulation을 진행하기 전에 ddr32의 Top Module 파일(ddr32.v)을 열어서, 이전의 16bits 인터페이스와 다른 점을 확인해 보겠습니다.

```
139 module ddr32 (
140
141     // Inouts
142
143     inout [31:0]      ddr3_dq,
144
145     inout [3:0]       ddr3_dqs_n,
146
147     inout [3:0]       ddr3_dqs_p,
148
149
185     // user interface signals
186
187     input [27:0]      app_addr,
188
189     input [2:0]       app_cmd,
190
191     input             app_en,
192
193     input [255:0]     app_wdf_data,
194
195     input             app_wdf_end,
196
197     input [31:0]      app_wdf_mask,
198
199     input             app_wdf_wren,
200
201     output [255:0]    app_rd_data,
```

대부분의 내용들은 3-5장에서 구현한 내용들과 동일합니다. Memory와 연결되는 data(ddr3_dq)가 16-> 32bits로 변경되었고, user interface의 data (app_wdf_data, app_rd_data)가 128 -> 256bits로 변경되었고, app_wdf_mask가 16->32bits로 변경되었습니다. 나머지 부분들은 Memory IP 생성시 설정하는 Option들에 따라서 singal이 추가되던지 없던지 하는 내용들입니다.

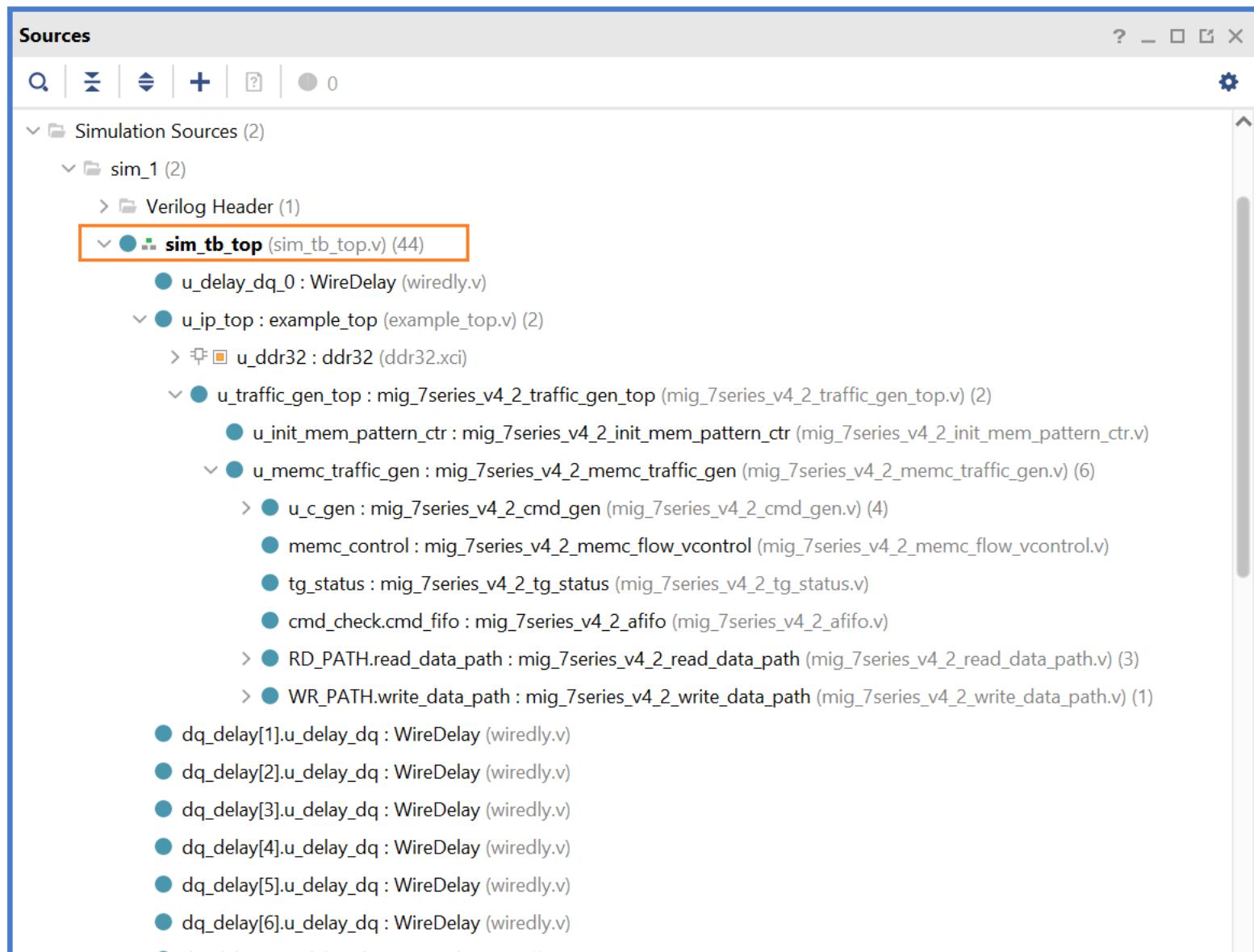
따라서 32bits로 구현하기 위해서는 3-5장에서 구현한 내용들 중에 data, mask 부분만 변경해 주면 됩니다.

user interface data와 memory interface data는 256bits, 32bits로 구성되어 있습니다. 우리가 설정한 DDR3의 clock은 400Mhz로 설정되어 있습니다. fpga내부에서 400Mhz (또한 DDR Mode)로 동작하는 것은 불가능합니다. 따라서 32bits 데이터를 8개씩 묶어서 256bis로 처리합니다. user interface clock은 SDR(single data rate)로 동작하기 때문에, $400/4 = 100\text{Mhz}$ 로 동작합니다.

simulation을 진행하기 위하여 파일들을 추가합니다. “example_design\sim\sim_tb_top.v”을 추가하고 관련된 모든 파일들을 추가합니다.

- ✓ Simulation Sources - 우클릭 - Add Source... 클릭합니다.
- ✓ Add or create simulation sources를 선택하고 Next를 클릭합니다.
- ✓ Add Files를 클릭해서 해당 파일들을 추가합니다.
- ✓ 먼저 sim_tb_top.v를 추가하고, 나머지 파일들도 모두 추가합니다.

아래 그림은 모든 파일들이 추가된 모습을 보여줍니다. 자동으로 sim_tb_top 모듈이 Top 모듈로 설정되었습니다.



simulation을 진행하기 전에 sim_tb_top.v 파일을 수정합니다.

- 1) time scale을 수정합니다. simulation 기본 단위가 100fp로 설정되어 있는데, 시간이 너무 많이 걸려서 1ps로 수정합니다.
- 2) simulation중에 특정 조건(calibration 완료)이 되면 finish 되는데, finish를 주석처리 합니다.

```

74
75 //`timescale 1ps/100fs
76 `timescale 1ps/1ps
77
78 module sim_tb_top;
79
585
586         $display("TEST FAILED: DATA ERROR");
587     end
588     disable calib_not_done;
589 //         $finish;
590
591 begin : calib_not_done
592     if (SIM_BYPASS_INIT_CAL == "SIM_INIT_CAL_FULL")
593         #2500000000.0;
594     else
595         #1000000000.0;
596 if (!init_calib_complete) begin
597     $display("TEST FAILED: INITIALIZATION DID NOT COMPLETE");
598 end
599 disable calibration_done;
600 //         $finish;
601 end
602 join
603 end

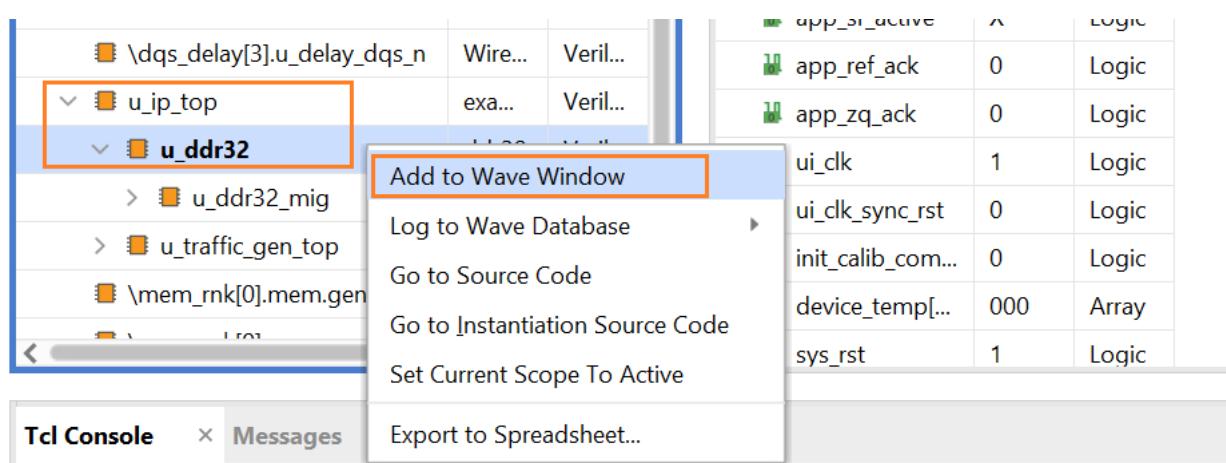
```

sim_tb_top이 Top Module로 설정되어 있음을 확인하고, SIMULATION - Run Simulation - Run Behavioral Simulation 을 클릭해서 simulation을 진행합니다.

sim_tb_top 내부의 신호들은 기본적으로 wave 윈도에 추가되어 있습니다. 우리가 확인해야 하는 user interface 관련된 신호들을 wave 윈도에 추가합니다.

sim_tb_top 아래 부분에 u_ip_top - u_ddr32 - 우클릭 - Add Wave Window를 클릭해서 해당 신호들을 추가합니다.

“run 0.2ms”을 입력해서 200us 동안 simulation을 진행합니다.



8.3.1 clock 신호

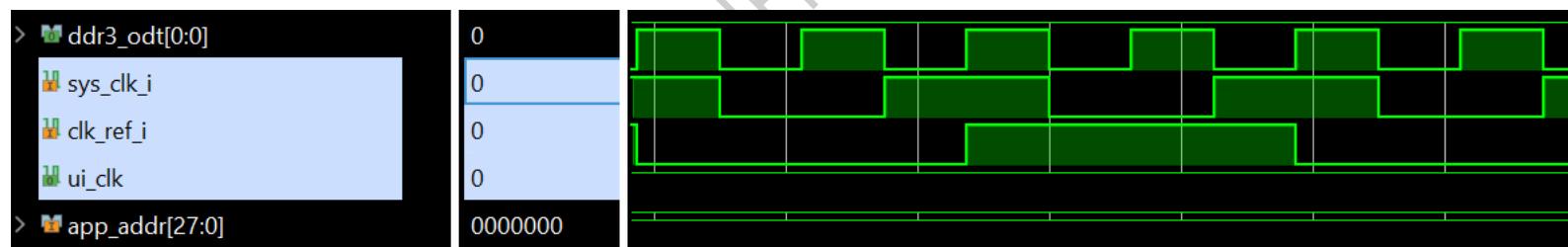
이제 본격적으로 simulation 파형을 분석합니다. 먼저 clock 신호를 확인합니다. 생성한 ddr3 ip는 sys_clk_i, clk_ref_i 2개의 clock 입력을 받습니다. 또한 user interface 모듈에서 사용할 clock (ui_clk)을 출력합니다.

```

139  module ddr32 (
140
175      // Inputs
176
177      // Single-ended system clock
178
179      input          sys_clk_i,
180
181      // Single-ended iodelayctrl clk (reference clock)
182
183      input          clk_ref_i,
184
223      output         ui_clk,
224
225      output         ui_clk_sync_rst,
226

```

아래는 3개의 clock을 보여줍니다. sys_clk_i 는 400Mhz, clk_ref_i 는 200Mhz, ui_clk은 100Mhz 입니다. sys_clk_i와 ui_clk은 phase가 맞는데 (sys_clk_i와 ui_clk의 positive edge가 동일한 timing), sys_clk_i 와 clk_ref_i 는 phase가 90 or 270 차이가 발생합니다. (clk_ref_i의 positive/negative edge가 sys_clk_i의 negative edge 에서 발생함) 이는 실제로 Memory Interface를 구현할 때 매우 중요합니다. clk_ref_i 를 생성할 때, phase를 90 or 270 으로 설정해야 합니다.

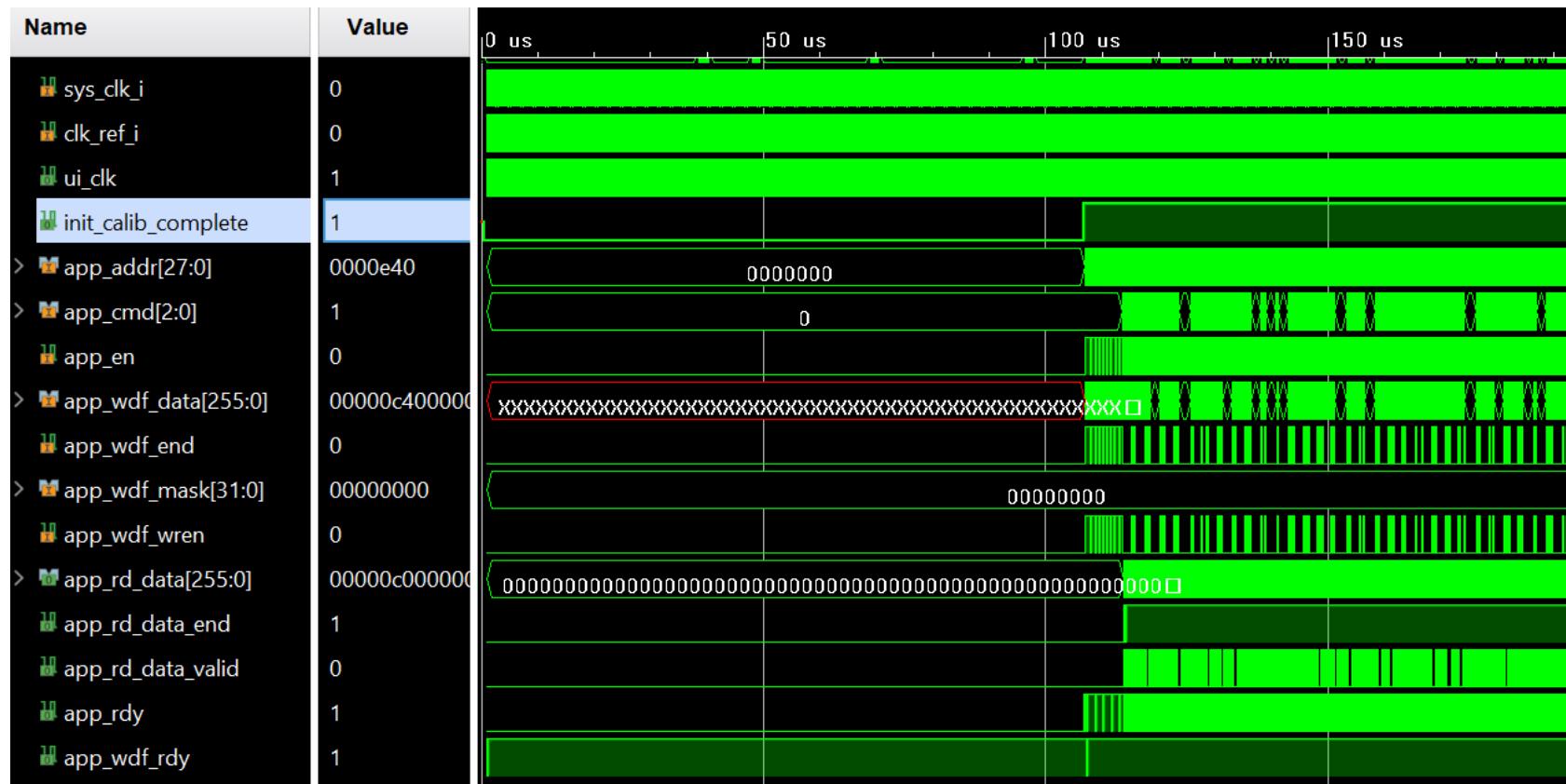


아래 그림은 Clock Generator를 사용하여, 100M 입력으로 400Mhz, 200Mhz(phase : 0, 90, 180, 270)을 생성하여 simulation을 한 결과를 보여줍니다. phase 90 or 270 일 때, clk_400M 의 negative edge에서 clock이 변하는 것을 알 수 있습니다.



8.3.2 init_calib_complete

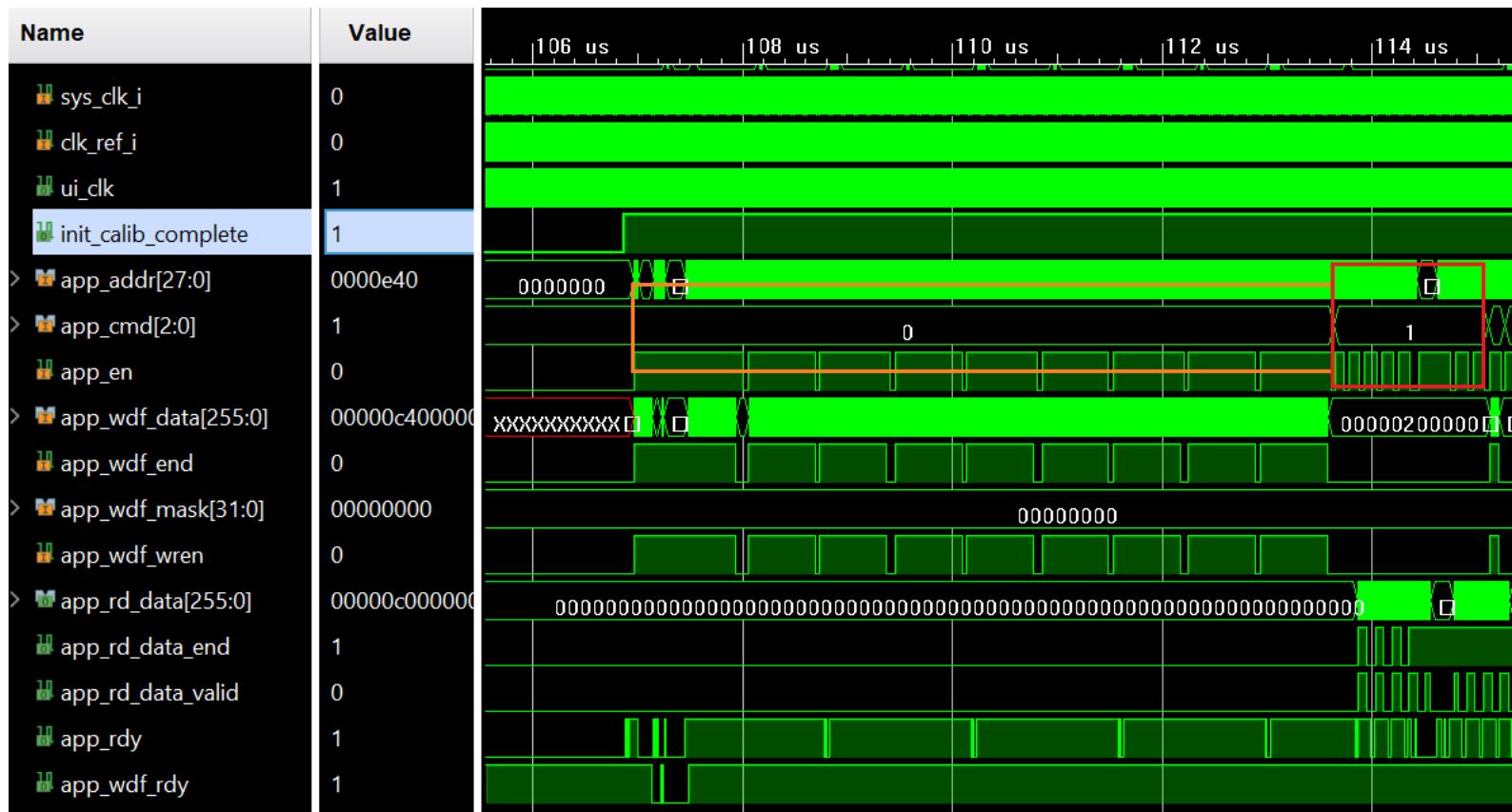
아래는 전체적인 파형을 보여줍니다. 약 130us에서 init_calib_complete 신호가 active 됩니다. 약 130us에서 calibration이 완료되고, memory write / read 동작이 구현됩니다.



/HIL

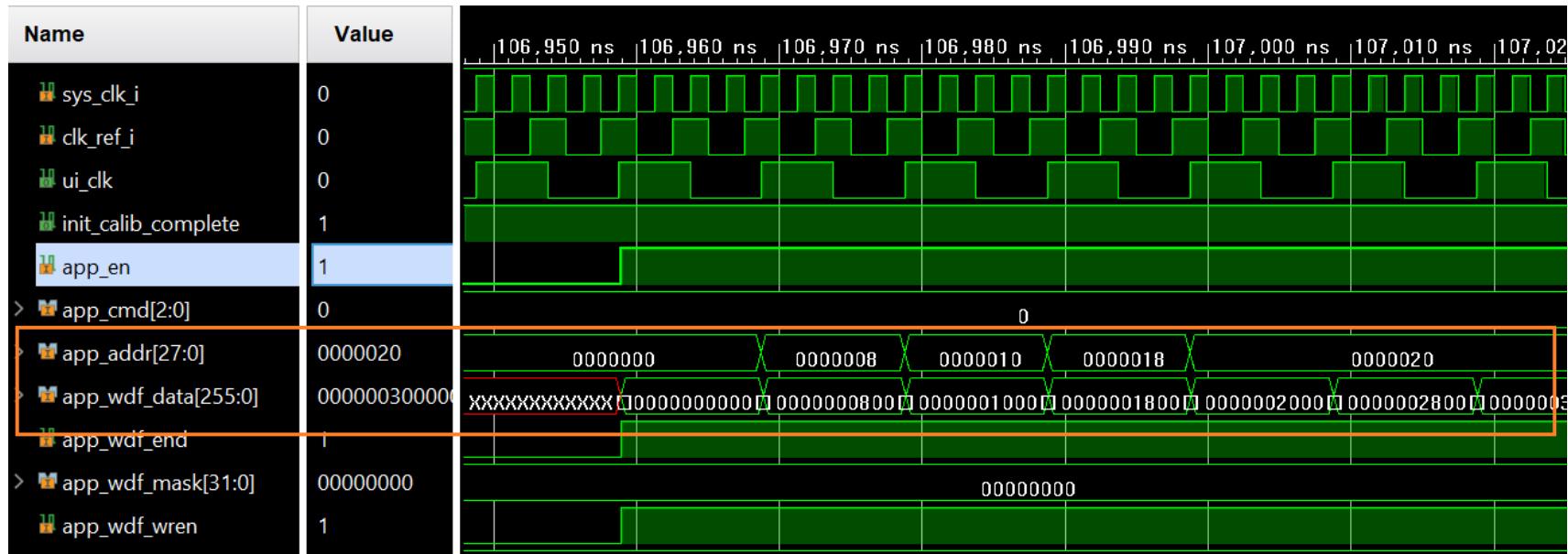
8.3.3 Memory read / write

아래는 calibration 완료후에 memory write / read 부분을 보여줍니다. app_cmd 가 0 일 때가 write, 1 일 때가 read입니다.

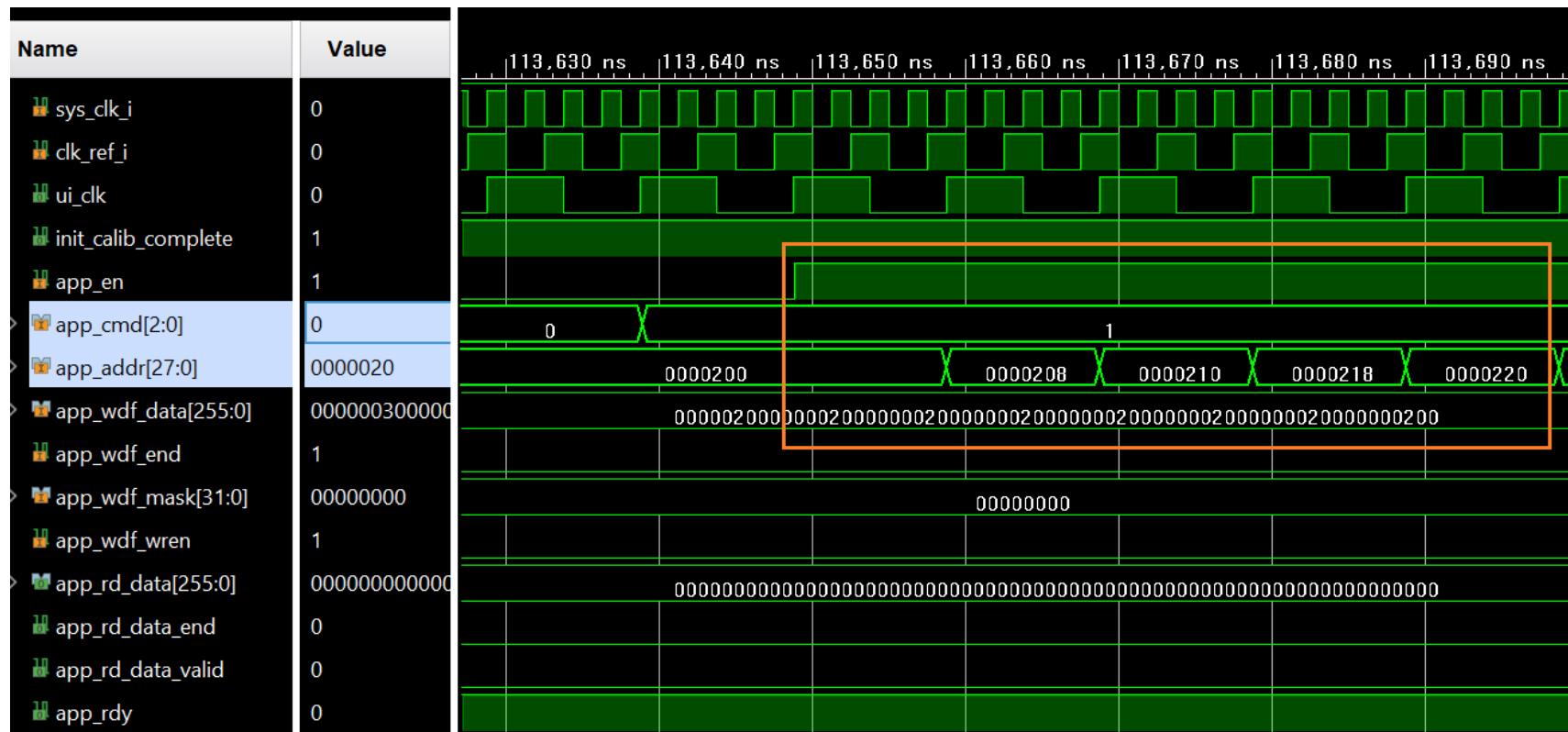


memory write 부분을 자세히 확인합니다. app_en이 High 일 때, app_addr, app_wdf_data 가 생성됩니다. ui_clk에 동기 되어 생성됩니다. address는 8씩 증가합니다. 즉 data 256bits 당 8씩 증가하므로, address 1당 32bits입니다. 데이터 구성(Test Pattern, Test Data)은 address를 8번 반복합니다. 즉 app_addr이 0x00000008 이면, app_wdf_data는 0x00000008을 8번 반복합니다.

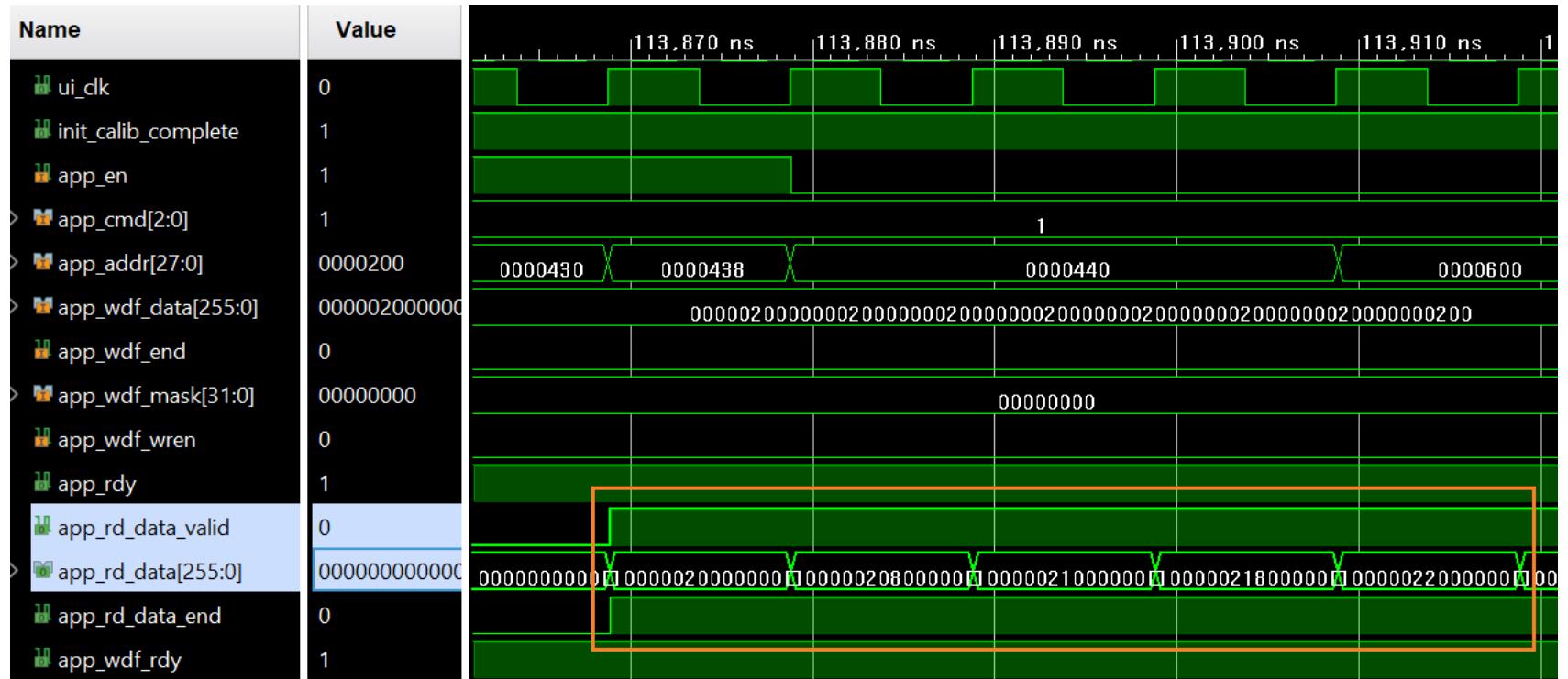
app_wdf_data : 0x00000008_00000008_00000008_00000008_00000008_00000008_00000008_00000008



memory read 부분을 확인합니다. app_cmd 가 1인 곳을 확인합니다. 약 113.6us 에서 첫번째 read command가 전송됩니다. read는 0 번지부터 하지 않고, 0x200 부터 합니다. 0x200, 0x208, 0x210, 0x218, 0x220 ~ 이렇게 read command가 전송됩니다.



아래 그림은 read data가 출력되는 부분을 보여줍니다. app_rd_data_valid가 active(1)될 때, app_rd_data가 출력됩니다. 출력되는 데이터는 0x00000200~, 0x00000208~, 0x00000210~, 0x00000218~, 0x00000220~ 임을 알 수 있습니다. (0x00000200~ : 0x00000200 이 8번 반복됨을 의미합니다)



user interface 관련 신호들은 4-5장에서 구현된 내용들과 동일합니다. 단지 data width가 16 -> 32, 128 -> 256 으로 변경되었습니다. 자세한 신호들은 4-5장을 참조하시길 바랍니다.

8.4 User Interface 구현

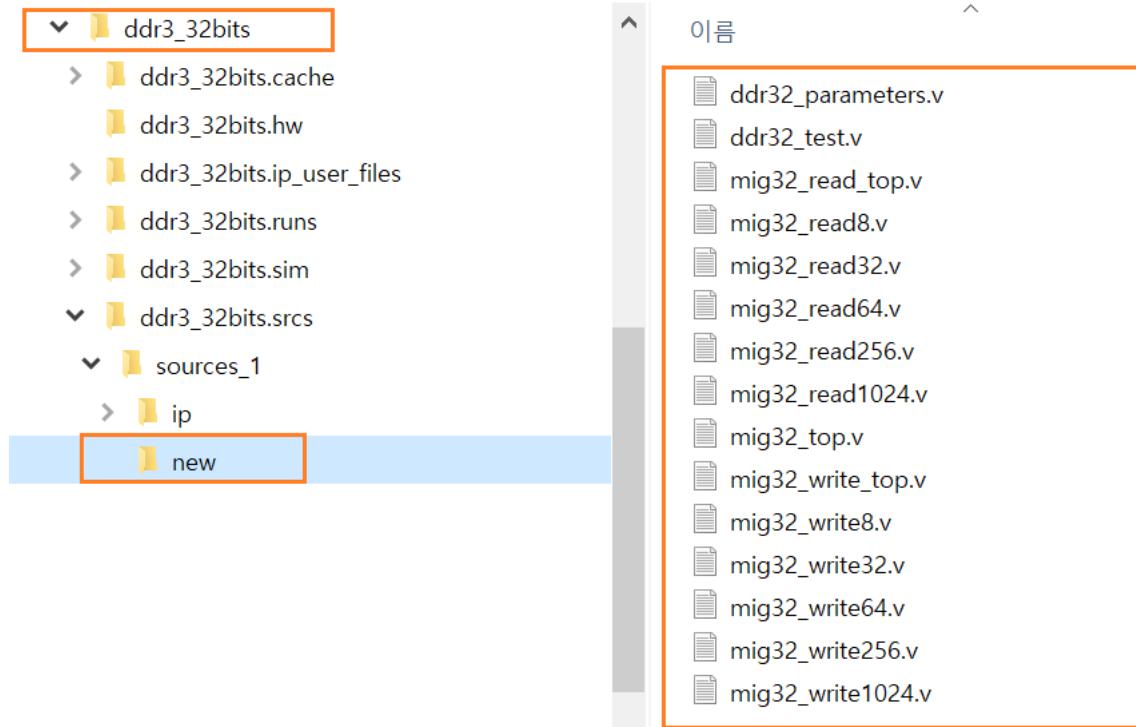
이번 장에서는 User Interface를 구현합니다. 대부분의 내용은 3-5장, 8장에서 구현한 내용들과 거의 동일하고, 단지 data bits 수만 바꾸어 주면 됩니다.

Memory Interface data width : 16 → 32,

User Interface data width : 128 → 256

이번 장에서는 8장에서 구현된 소스를 가져와서 사용합니다. 파일이름(모듈이름)을 구별하기 위하여 mig7~ 을 mig32~로 변경해서 사용하도록 하겠습니다.

아래는 전체적인 소스를 보여줍니다. 8장에서 구현된 소스를 가져와서 이름(파일, 모듈)을 변경하였습니다.



ddr32_parameters.v 는 simulation 조건이 포함된 파일입니다.

mig32_read8, mig32_write8 은 각각 256bits x 8 read / write을 구현하는 모듈입니다. (32, 64, 256, 1024 는 각각 256bits x 32, x 64, x 256, x 1024를 구현합니다. simulation 조건에 따라서 각각의 시간을 비교합니다)

8.4.1 mig32_read8.v, mig32_write8.v 수정

이번장에서는 reset 신호를 asynchronous에서 synchronous로 변경해서 구현합니다. 또한 reset 신호를 active low -> active high로 변경합니다. xilinx user guide를 보면 reset 신호를 synchronous로 사용하는 것이 더 좋다는 내용이 있습니다. (제가 보기에는 큰 차이는 없는 것 같습니다)

in/out port의 data bits를 변경합니다.

```

49  input      rstart      ;
50  input      [27:0] raddr      ;
51  output     [255:0] rdata      ;
52  output     rdata_valid      ;
53  output     rdone      ;
54
55  output     [27:0] app_addr      ;
56  output     [2:0]  app_cmd      ;
57  output     app_en      ;
58  input      app_rdy      ;
59  input      [255:0] app_rd_data      ;
60  input      app_rd_data_end      ;
61  input      app_rd_data_valid      ;

95  reg      [255:0] rdata      ;
96  always @(posedge mclk)
97 begin
98     if(reset) rdata <= 256'b0;
99     else      rdata <= s_idle ? 256'b0 : (app_rd_data_valid & app_rd_data_end) ? app_rd_data : rdata ;
100 end

```

reset을 모두 synchronous로 변경하고, active high로 변경합니다. always 문에서 "or negedge reset"을 제거하고 if (!reset) -> if (reset) 으로 변경합니다.

mig32_write8.v 도 동일하게 변경합니다. 추가로 app_wdf_mask는 16bits -> 32bits로 변경합니다.

```

54  input      [27:0] waddr      ;
55  input      [255:0] wdata      ;
56  output     wready      ;
57  output     wdone      ;
58
59  output     [27:0] app_addr      ;
60  output     [2:0]  app_cmd      ;
61  output     app_en      ;
62  input      app_rdy      ;
63  output     [255:0] app_wdf_data      ;
64  output     app_wdf_wren      ;
65  output     app_wdf_end      ;
66  output     [31:0] app_wdf_mask      ;
67  input      app_wdf_rdy      ;

92  -----
93  wire      [255:0] app_wdf_data = wdata;
94
95  wire      [31:0] app_wdf_mask = 32'b0 ;
96

```

mig32_read32, mig32_read64, mig32_read256, mig32_read1024 도 동일하게 변경합니다.

mig32_write32, mig32_write64, mig32_write256, mig32_write1024도 동일하게 변경합니다.

8.4.2 mig32_read_top, mig32_write_top 수정

mig32_read_top을 수정합니다. data bit, reset 부분만 수정하면 됩니다.

```

51  output [255:0] mig_rdata ;
52  output      mig_rvalid ;
53  output      mig_rdone ;
54
55
56  output [27:0] app_addr ;
57  output [2:0]  app_cmd ;
58  output      app_en ;
59  input       app_rdy ;
60  input [255:0] app_rd_data ;
61  input      app_rd_data_end ;
62  input      app_rd_data_valid ;

140 wire      mig_rvalid ;
141 wire [255:0] mig_rdata ;
142 wire [27:0]  app_addr ;
143 wire [2:0]   app_cmd ;
144 wire      app_en ;

145
146 `ifdef DDR3_STEP_8
147     mig32_read8      mig32_read8 (
148 `elsif DDR3_STEP_32
149     mig32_read32     mig32_read32 (
150 `elsif DDR3_STEP_64
151     mig32_read64     mig32_read64 (
152 `elsif DDR3_STEP_256
153     mig32_read256    mig32_read256 (
154 `elsif DDR3_STEP_1024
155     mig32_read1024   mig32_read1024 (
156 `endif
157     .reset          (reset      ),
158     .mclk           (mclk      ),
159
160     .rstart         (rstart    ),
161     .raddr          (raddr     ),
162

```

mig32_write_top을 수정합니다.

```

52  input [27:0] mig_waddr ;
53  input [9:0]  mig_wsize ;
54  input [255:0] mig_wdata ;
55  output      mig_wdone ;
56  output      mig_wrady ;
57
58  output [27:0] app_addr ;
59  output [2:0]  app_cmd ;
60  output      app_en ;
61  input       app_rdy ;
62  output [255:0] app_wdf_data ;
63  output      app_wdf_wren ;
64  output      app_wdf_end ;
65  output [31:0] app_wdf_mask ;
66  input       app_wdf_rdy ;


```

```
140    wire      app_en      ;
141    wire      [255:0] app_wdf_data  ;
142    wire      app_wdf_wren   ;
143    wire      app_wdf_end    ;
144
145    wire      mig wrdy;
146    wire      [31:0] app_wdf_mask ;
147
148    reg       mig_wdone;
149    always @ (posedge mclk)
150    begin
151        if (reset)      mig_wdone <= 1'b0;
152        else           mig_wdone <= ss_done ;
153    end
154
155    `ifdef DDR3_STEP_8
156        mig32_write8      mig32_write8 (
157    `elsif DDR3_STEP_32
158        mig32_write32      mig32_write32 (
159    `elsif DDR3_STEP_64
160        mig32_write64      mig32_write64 (
161    `elsif DDR3_STEP_256
162        mig32_write256     mig32_write256 (
163    `elsif DDR3_STEP_1024
164        mig32_write1024    mig32_write1024 (
165    `endif
166        .reset            (reset          ),
167        .mclk             (mclk          ),
168    
```

8.4.3 ddr32_test 수정

ddr32_test 모듈을 Test Pattern (Data)을 생성하는 모듈입니다.

```

75   output [9:0] wsize;
76   output [255:0] wdata;
77   input    wready;
78   input    wdone;
79
80   output    rstart;
81   output [27:0] raddr;
82   output [9:0] rszie;
83   input [255:0] rdata;
84   input    rdata_valid;
85   input    rdone;
86
87   output [24:0] ddr_err_cnt;
88   output    ddr_wdone;
89   output    ddr_rdone;
90
91
92 `ifdef DDR3_RUN_MODE
93   parameter DDR3_BLOCK_MAX = 14'd8191;
94
95 `ifndef DDR3_STEP_8
96   parameter DDR3_BLOCK_SIZE = 10'd256;
97 `elsif DDR3_STEP_32
98   parameter DDR3_BLOCK_SIZE = 10'd64;
99 `elsif DDR3_STEP_64
100  parameter DDR3_BLOCK_SIZE = 10'd32;
101 `elsif DDR3_STEP_256
102  parameter DDR3_BLOCK_SIZE = 10'd8;
103 `elsif DDR3_STEP_1024
104  parameter DDR3_BLOCK_SIZE = 10'd2;
105 `endif
106
107 `elsif DDR3_SIM_MODE
108   parameter DDR3_BLOCK_MAX = 14'd3;
109
110 `ifndef DDR3_STEP_8
111   parameter DDR3_BLOCK_SIZE = 10'd256;
112 `elsif DDR3_STEP_32
113   parameter DDR3_BLOCK_SIZE = 10'd64;
114 `elsif DDR3_STEP_64
115   parameter DDR3_BLOCK_SIZE = 10'd32;
116 `elsif DDR3_STEP_256
117   parameter DDR3_BLOCK_SIZE = 10'd8;
118 `elsif DDR3_STEP_1024
119   parameter DDR3_BLOCK_SIZE = 10'd2;
120 `endif
121
122 `endif
123

```

라인 92 - 122 : run_mode에서는 ddr 의 전영역을 read/write 합니다. sim_mode에서는 4개의 Block만 진행합니다.

```

125 parameter test_data1 = 256'h33333333_33333333_33333333_33333333_33333333_33333333_33333333;
126 parameter test_data2 = 256'hcccccccc_cccccccc_cccccccc_cccccccc_cccccccc_cccccccc_cccccccc;
127 parameter test_data3 = 256'h55555555_55555555_55555555_55555555_55555555_55555555_55555555_55555555;
128 parameter test_data4 = 256'haaaaaaaa_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa;

```

test data는 4개의 data로 256bits로 구성되어 있습니다. 순차적으로 test_data1 - 4의 데이터를 write하고, read 해서 read data와 test_data1 - 4를 비교해서 에러가 있는지를 확인합니다.

wdata를 256bits로 수정합니다.

```
203 wire [255:0] wdata = (wready_cnt[1:0]==2'b00) ? test_data1 :  
204 (wready_cnt[1:0]==2'b01) ? test_data2 :  
205 (wready_cnt[1:0]==2'b10) ? test_data3 : test_data4 ;  
206
```

8.4.4 mig32_top 모듈 수정

아래와 같이 수정합니다.

- ✓ ddr3_dq : 16bits → 32bits
- ✓ ddr3_dm : 2bits → 4bits
- ✓ ddr3_dqs_n, p : 2bits → 4bits
- ✓ mig_wdata, mig_rdata : 128bits → 256bits
- ✓ app_wdf_data, app_rd_data : 128bits → 256bits
- ✓ app_wdf_mask : 16bits → 32bits

자세한 사항은 소스 코드 (mig32_top.v)를 참조하시길 바랍니다.



8.5 mig32_top 모듈 simulation

source 구현은 모두 완료되었습니다. 이번 장에서는 mig32_top 모듈을 simulation 통해서 결과를 확인합니다. Test bench는 tb_mig32_top.v 입니다.

- ✓ Simulation Sources - Add Sources... - Add or create simulation sources - Next 클릭합니다.
- ✓ Create File 버튼 클릭 - File name : tb_mig32_top.v 입력하고 OK를 클릭합니다.
- ✓ Finish를 클릭해서 파일을 생성합니다.
- ✓ Module Name : tb_mig32_top (기본값) 확인하고, OK를 클릭합니다.

이제 sim_tb_top.v 파일을 참조해서 tb_mig32_top.v 파일의 코드를 구현합니다.

sim_tb_top.v 의 모든 코드를 tb_mig32_top.v 에 복사해서 붙여 넣습니다. 그리고 상단의 module 이름을 “tb_mig32_top” 으로 수정합니다.

```

23 //`timescale 1ps/100fs
24 `timescale 1ps/1ps
25
26 module tb_mig32_top;
27
28
29 //*****
30 // Traffic Gen related parameters
31 //*****
32 parameter SIMULATION          = "TRUE";
33 parameter PORT_MODE           = "BI_MODE";
34 parameter DATA_MODE           = 4'b0010;
35 parameter TST_MEM_INSTR_MODE = "R_W_INSTR_MODE";
36 parameter EYE_TEST             = "FALSE";

```

코드의 중간 쯤에 example_top 모듈 대신에 mig32_top 모듈을 추가하고, 필요한 신호를 생성합니다.

```

379 wire          ddr_ok      ;
380 wire          ddr_error   ;
381 wire          clk_du     ;
382 wire          rst_du     ;
383 wire          ddr_wdone  ;
384 wire          ddr_rdone  ;
385
386 // State Parameter
387 parameter    M_IDLE      = 2'd0;
388 parameter    M_WRITE     = 2'd1;
389 parameter    M_READ      = 2'd2;
390 parameter    M_DONE      = 2'd3;

```

- ✓ 라인 379 - 384 : mig32_top의 output 신호를 wire로 선언합니다.
- ✓ 라인 386 - 390 : state parameter를 선언합니다.

```

392 // -----
393 // State Control
394 reg [1:0] m_state;
395 wire s_idle = (m_state==M_IDLE) ? 1'b1 : 1'b0;
396 wire s_write = (m_state==M_WRITE) ? 1'b1 : 1'b0;
397 wire s_read = (m_state==M_READ) ? 1'b1 : 1'b0;
398 wire s_done = (m_state==M_DONE) ? 1'b1 : 1'b0;
399
400 reg [10:0] cntw;
401 always @ (posedge clk_du)
402 begin
403     if(rst_du)      cntw <= 11'b0;
404     else           cntw <= ~init_calib_complete ? 11'b0 : (cntw==11'd2020) ? 11'd2020 : cntw+1'b1;
405 end
406
407 reg btn0;
408 always @ (posedge clk_du)
409 begin
410     if(rst_du)      btn0 <= 1'b0;
411     else           btn0 <= (cntw==11'd2000) ? 1'b1 : 1'b0;
412 end
413
414 reg [10:0] cntr;
415 always @ (posedge clk_du)
416 begin
417     if(rst_du)      cntr <= 11'b0;
418     else           cntr <= ~s_read ? 11'b0 : (cntr==11'd2020) ? 11'd2020 : cntr+1'b1;
419 end
420
421 reg btnl;
422 always @ (posedge clk_du)
423 begin
424     if(rst_du)      btnl <= 1'b0;
425     else           btnl <= (cntr==11'd2000) ? 1'b1 : 1'b0;
426 end
427
428
429 always @ (posedge clk_du)
430 begin
431     if(rst_du)      begin
432         m_state <= 2'b0;
433     end
434     else      begin
435         m_state <= (s_idle & init_calib_complete) ? M_WRITE : ;
436         (s_write & ddr_wdone) ? M_READ : ;
437         (s_read & ddr_rdone) ? M_DONE : ;
438         (s_done) ? M_IDLE : m_state ;
439     end
440 end

```

- ✓ 라인 435 : idle에서 init_calib_complete가 active되면 (calibration 완료) ddr write를 진행합니다.
- ✓ 라인 436 : write가 완료되면 (ddr_wdone) ddr read를 수행합니다.
- ✓ 라인 437 : read가 완료되면 done 상태가 됩니다.

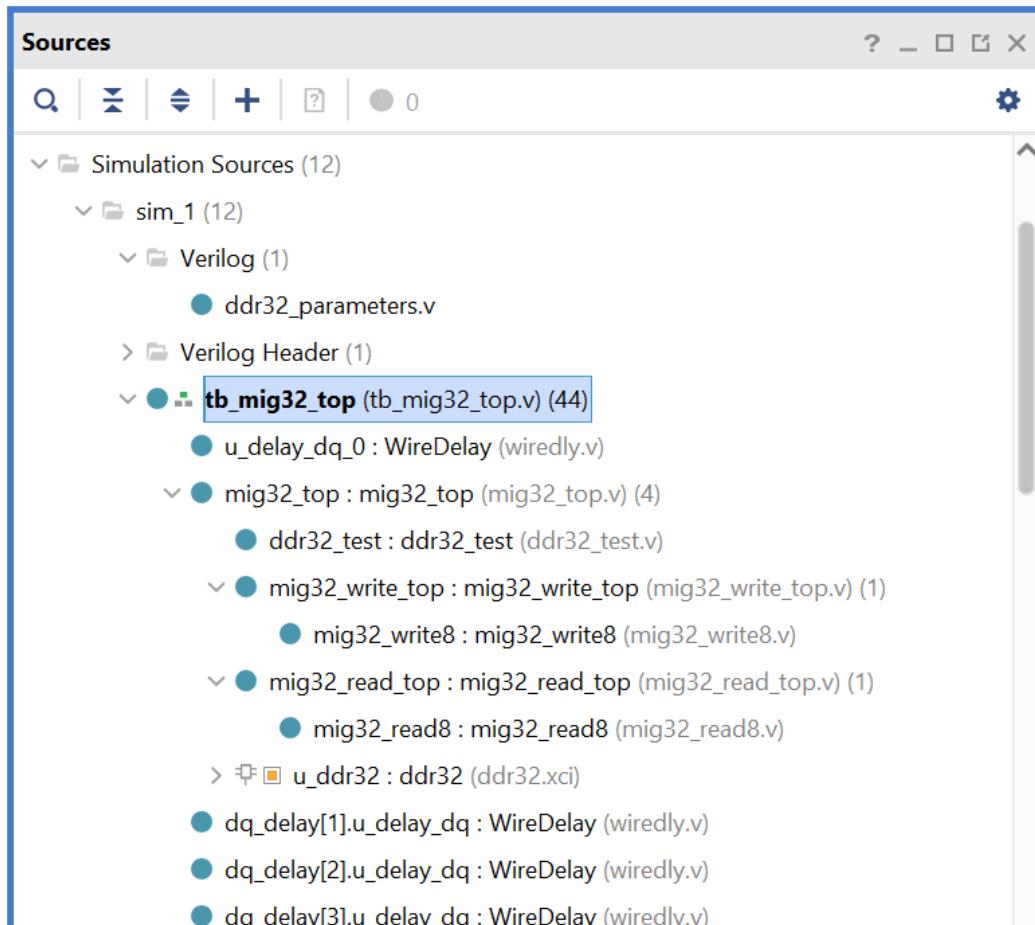
tb_mig32_top 모듈을 Top Module로 지정하고, 지금까지 진행했던 모든 파일들을 추가합니다. 먼저 mig32_write8, mig32_read8을 simulation하기 위해서 ddr32_parameter.v의 내용을 아래와 같이 설정합니다.

```

5 // -----
6 // DDR3
7
8 //`define           DDR3_RUN_MODE
9 `define           DDR3_SIM_MODE
10
11 `define           DDR3_STEP_8
12 //`define          DDR3_STEP_32
13 //`define          DDR3_STEP_64
14 //`define          DDR3_STEP_256
15 //`define          DDR3_STEP_1024
16

```

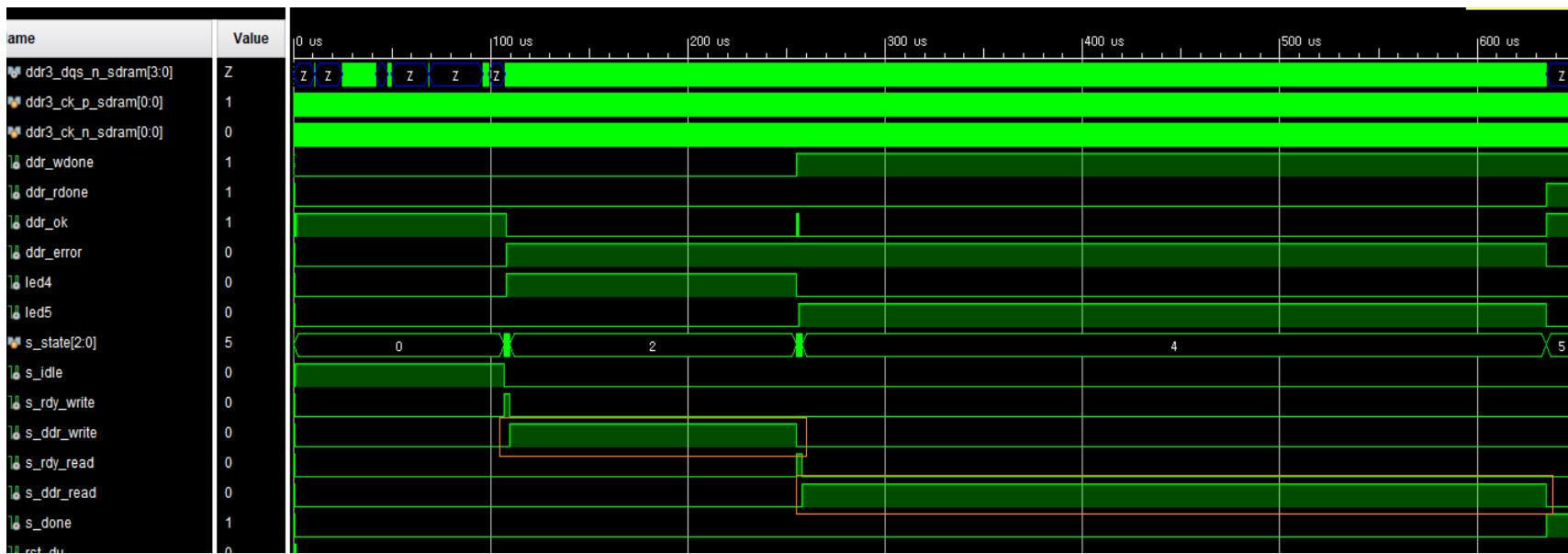
tb_mig32_top - 우클릭 - Refresh Hierarchy 클릭합니다. 아래는 현재까지 진행한 내용을 보여줍니다.



SIMULATION - Run Simulation - Run Behavioral Simulation을 클릭해서 simulation을 진행합니다. Scope 탭에서 mig32_top - 우클릭 - Add to Wave Window 클릭해서 신호들을 추가합니다.

“run 0.7ms” 을 입력해서 simulation을 진행합니다.

아래 그림은 결과를 보여줍니다.

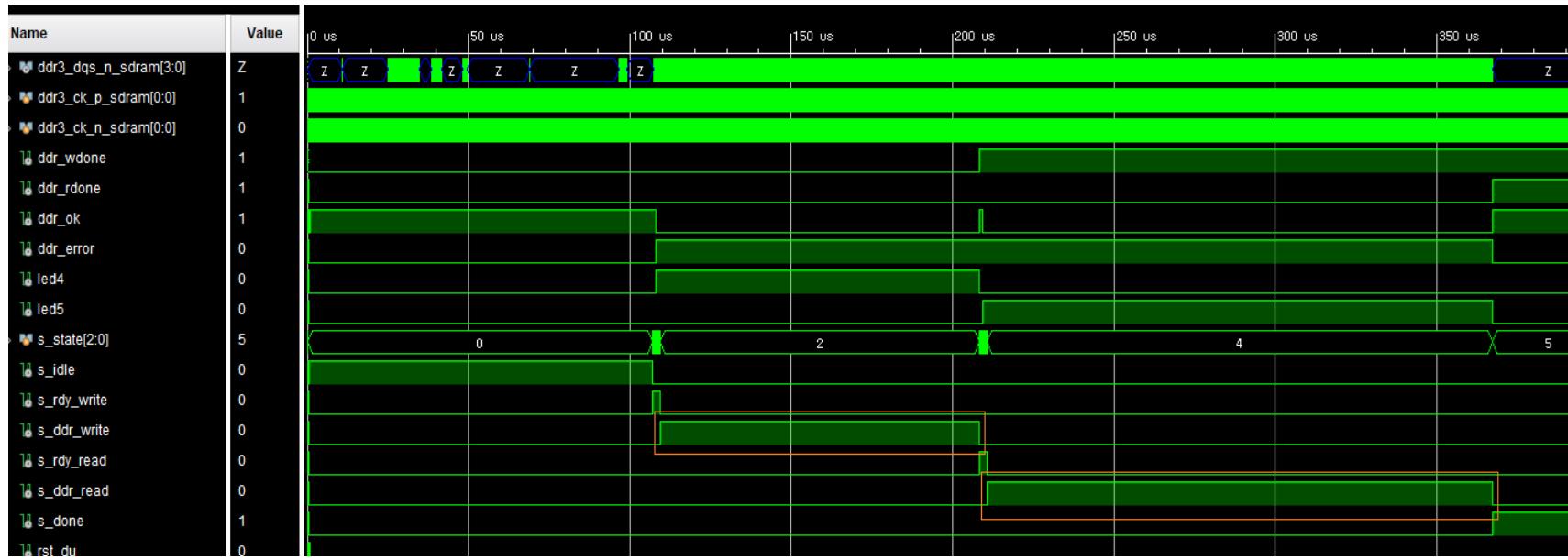


s_ddr_write는 write 구간을 나타내고, s_ddr_read는 read 구간을 나타냅니다. 대략 4개의 block을 write하는 시간은 약 0.14ms, read하는 시간은 0.37ms 정도 소요됩니다. read가 완료된 후에 ddr_ok, ddr_error 신호를 보면 ddr_ok는 1로 되고, ddr_error는 0가 됩니다. 이는 write 한 데이터를 그대로 read 해서 에러가 없음을 보여줍니다.

이번에는 mig32_write32, mig32_read32 모듈로 simulation을 진행합니다. ddr32_parameters.v 파일을 아래와 같이 수정하고, 소스 트리에서 우클릭 - Refresh Hierarchy를 클릭합니다.

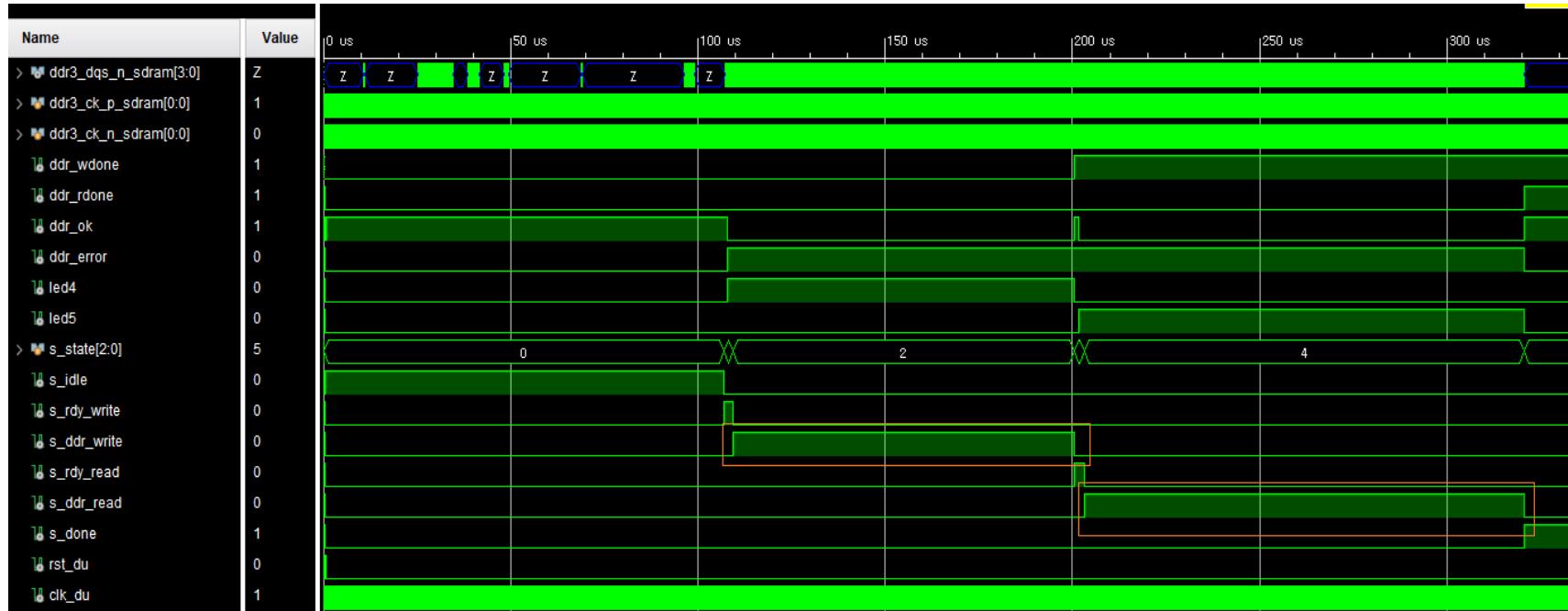


run 0.4ms 입력해서 simulation을 진행합니다. 아래는 그 결과를 보여줍니다.



write : 0.1ms, read : 0.15ms 정도 소요되었습니다.

아래는 mig32_write64, mig32_read64 모듈을 simulation 한 결과를 보여줍니다.



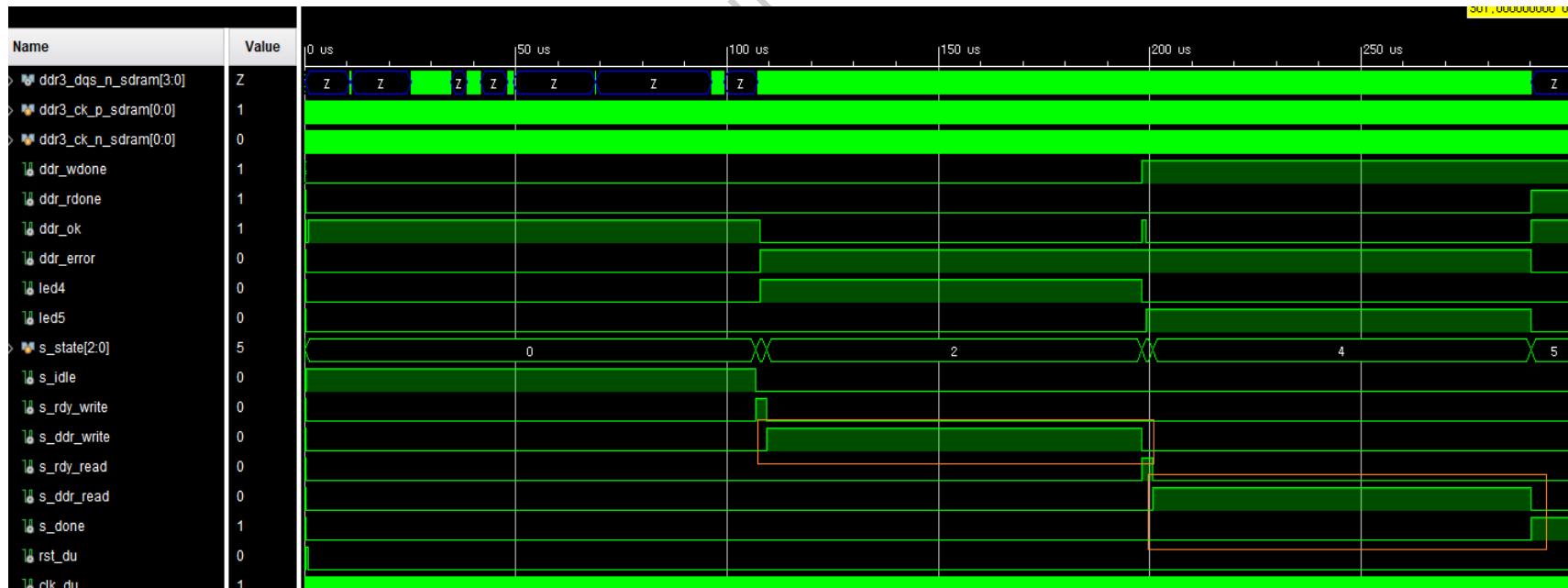
write : 0.09ms, read : 0.12ms 정도 소요되었습니다.

아래는 mig32_write256, mig32_read256 모듈을 simulation 한 결과를 보여줍니다.



write : 0.09ms, read : 0.09ms 정도 소요되었습니다.

아래는 mig32_write1024, mig32_read1024 모듈을 simulation 한 결과를 보여줍니다.



write : 0.09ms, read : 0.09ms 정도 소요되었습니다

8.6 결론

이번 장에서는 16bits DDR3 Memory 2개를 사용하여 32bits Interface를 구현하였습니다. 16bits를 구현한 코드에서 크게 벗어나지 않았습니다. 범용으로 사용가능한 User Interface를 잘 만들면, 32bits에서도 확장해서 사용하는데 큰 어려움 없이 구현할 수 있음을 알 수 있습니다.

본 강의에서 구현한 내용은 영상 처리용 DDR Memory Interface를 구현하는 실무에서 적용가능한 코드입니다. 저도 이 코드들을 활용하여 많은 일들을 진행하였습니다.

실제 업무에서 진행하려면, 최대 동작 Clock을 알아야 합니다. 본 장에서는 최대 Clock인 400Mhz로 구현하였습니다. 이를 실제 보드에 적용해서, 전 영역을 Write 하고, Read 한 후에 에러가 없는지를 확인해야 합니다. 만일 에러가 발생한다면 Clock을 낮추어서 진행해야 합니다.

본 강의를 통하여 영상 처리용 DDR Memory Interface 구현하는 부분을 마스터 할 수 있길 기대하고, 현장에서 활용될 수 있길 바랍니다.

AIHIL

9. Spartan6 DDR Controller 구현

이번 장에서는 Spartan6 FPGA에서 DDR Controller를 구현합니다. Xilinx에서는 개발환경이 6-Series 이하에서는 ISE을 사용하고, 7-Series 부터는 Vivado를 사용합니다. ISE는 14.7버전이 최종 버전입니다. 이번 장에서는 ISE14.7 버전을 사용하여 Spartan6를 위한 DDR Controller를 개발합니다.

ISE와 Vivado를 같은 PC에 설치해서 사용하는 것은 여러 문제를 발생하는 것으로 알고 있습니다. 저는 Vivado가 설치되어 있지 않은 PC에 ISE14.7을 설치하여 진행하였습니다. HW 검증은 프로젝트를 진행하면서 사용했던 보드에서 검증하였습니다. 본 장에서는 simulation으로 검증하는 것까지 진행하도록 하겠습니다.

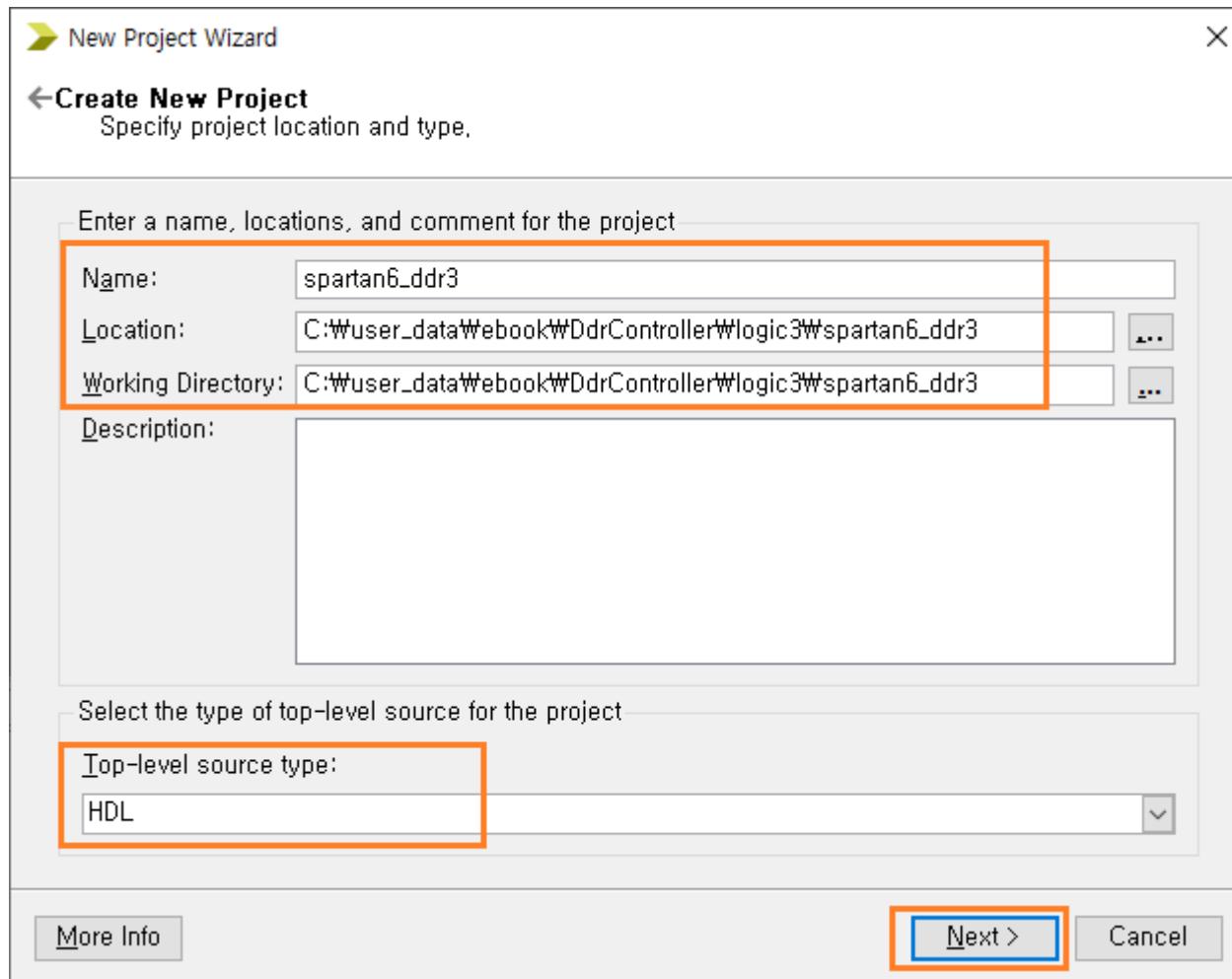
ISE14.7을 windows10에 설치하는 것은 구글링해서 진행하시길 바랍니다.

9.1 프로젝트 생성

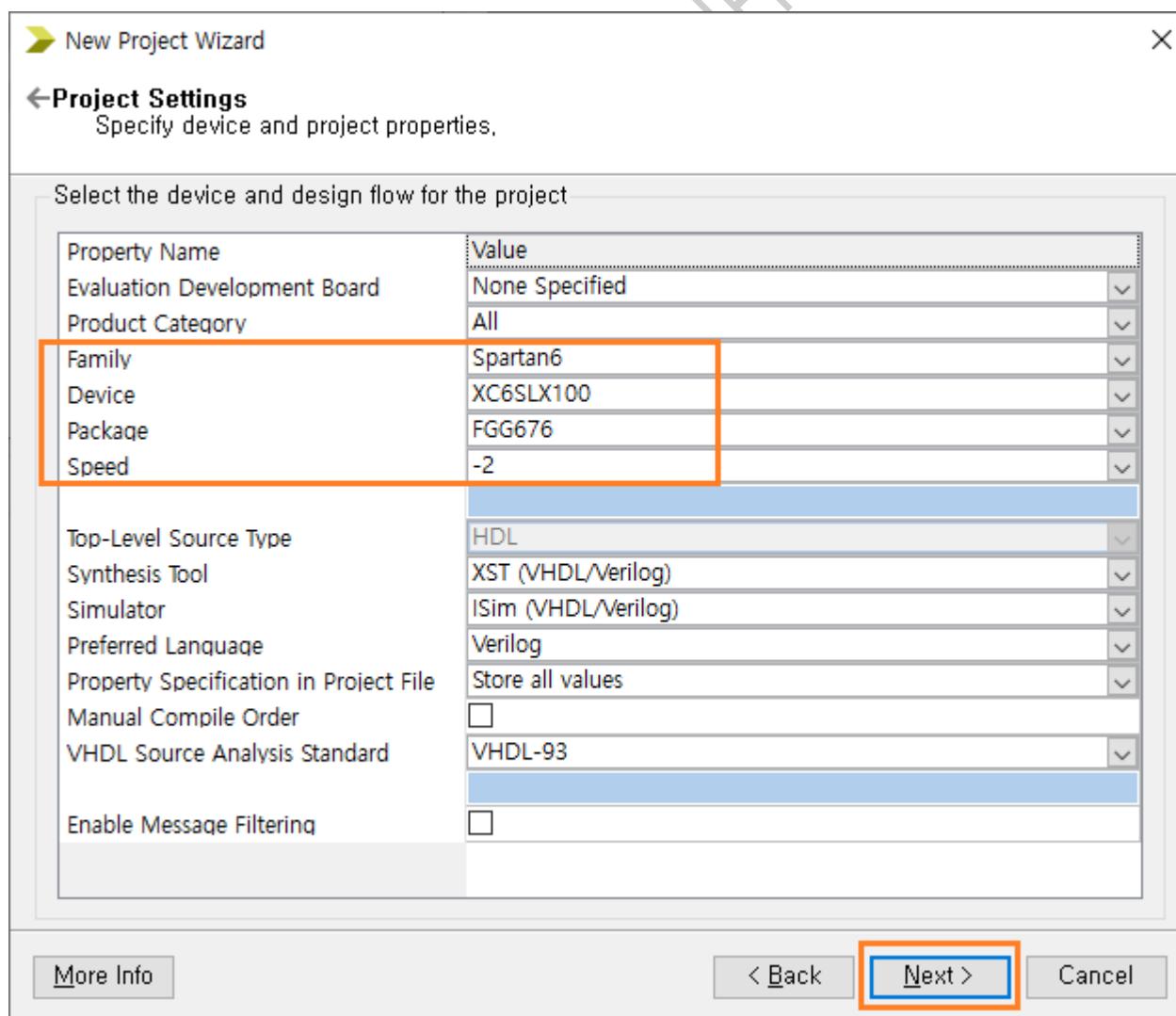
ISE 14.7을 실행합니다. New Project... 클릭합니다.



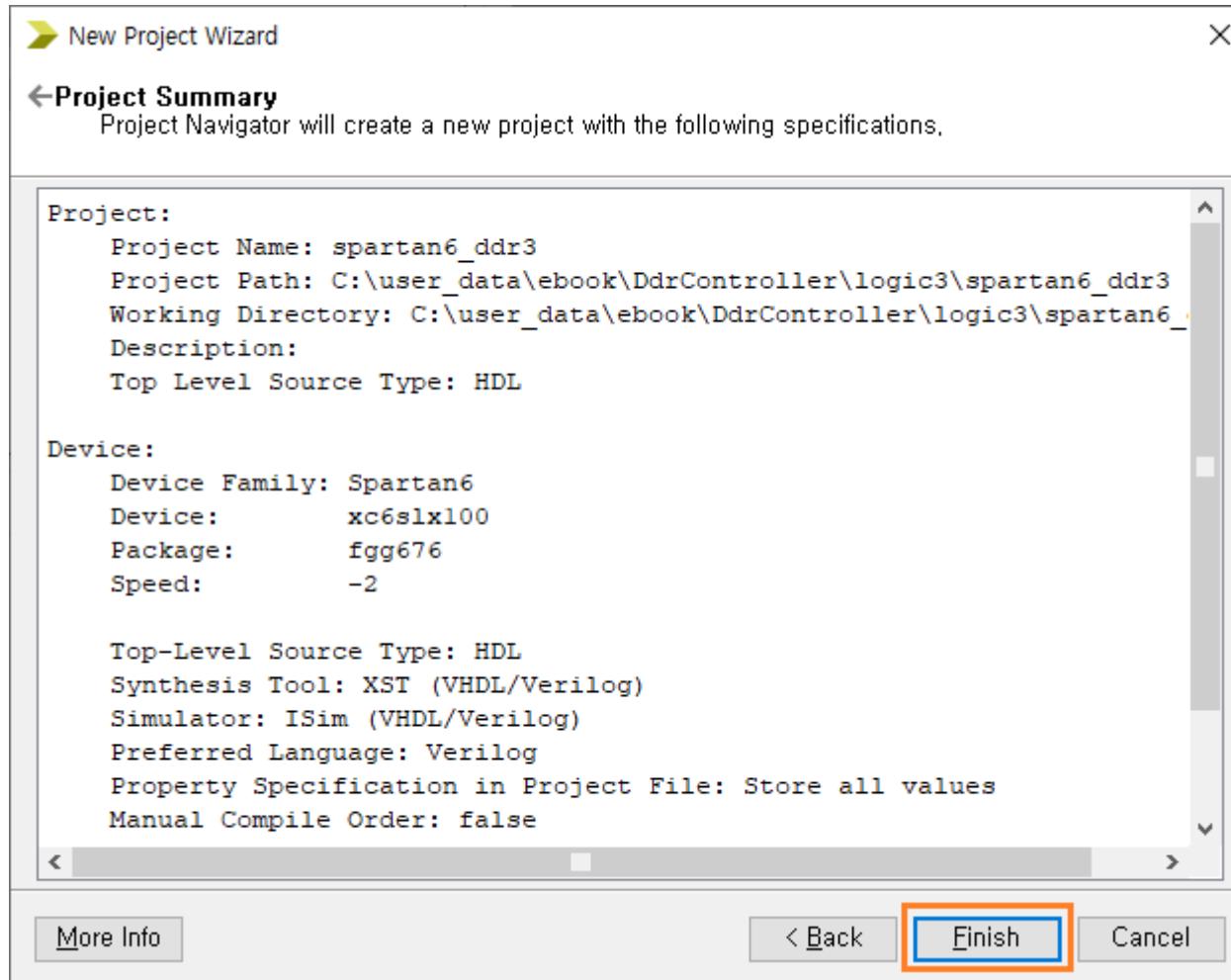
Project Name, Location, Top-level source type ; HDL 설정하고 Next 클릭합니다.



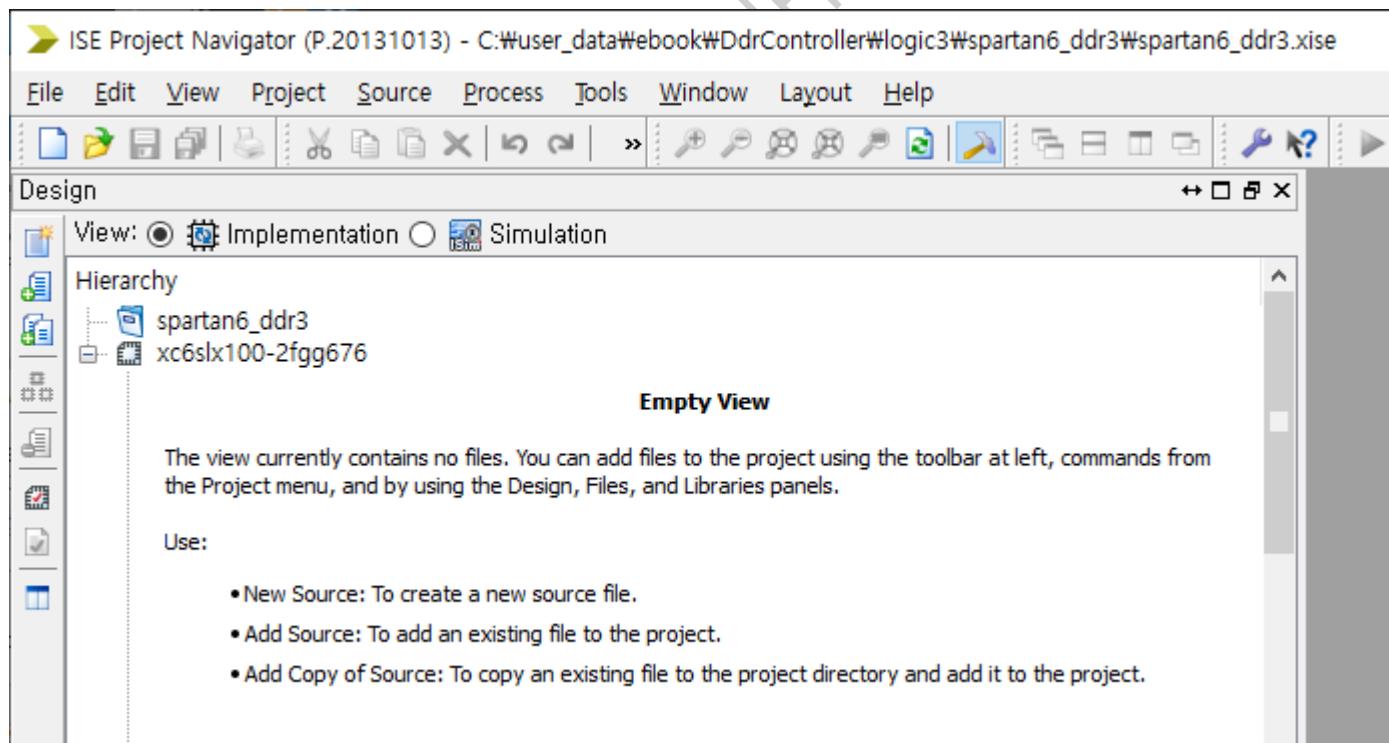
Device를 설정합니다. Family : Spartan6, Device : XC6SLX100, Package : FGG676, Speed : -2. Next 클릭합니다.



Finish를 클릭해서 프로젝트를 생성합니다.

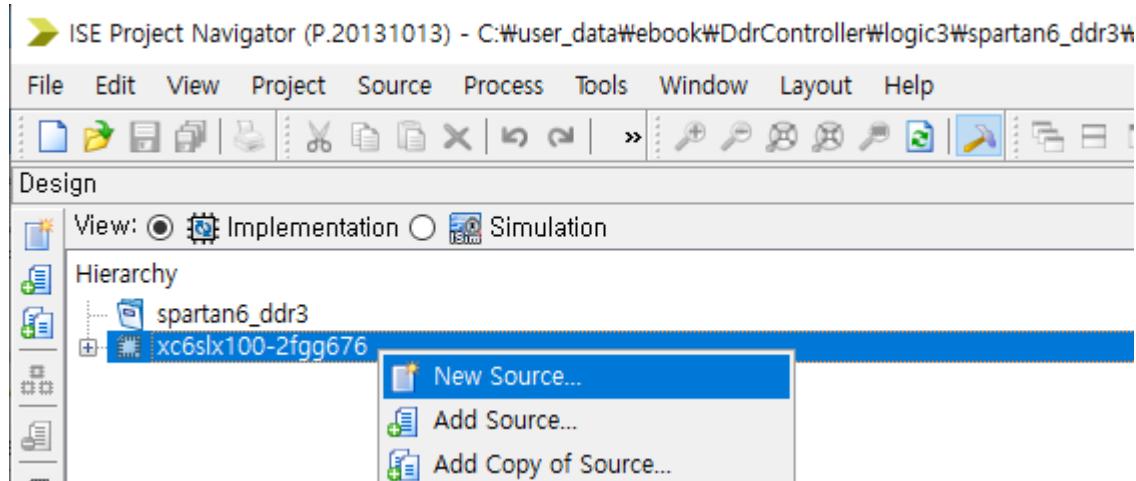


아래는 프로젝트가 생성된 모습을 보여줍니다.

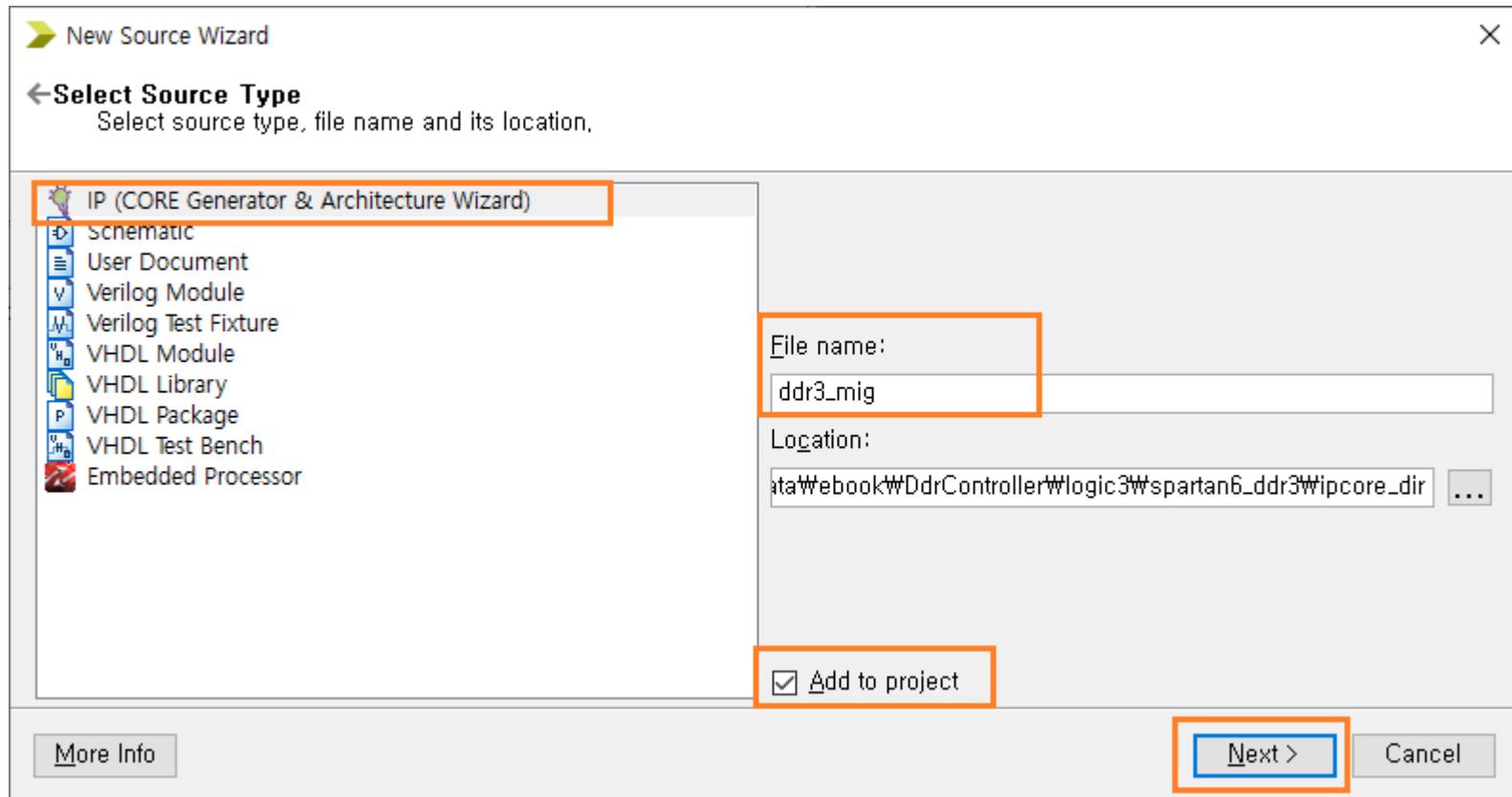


9.2 IP 생성

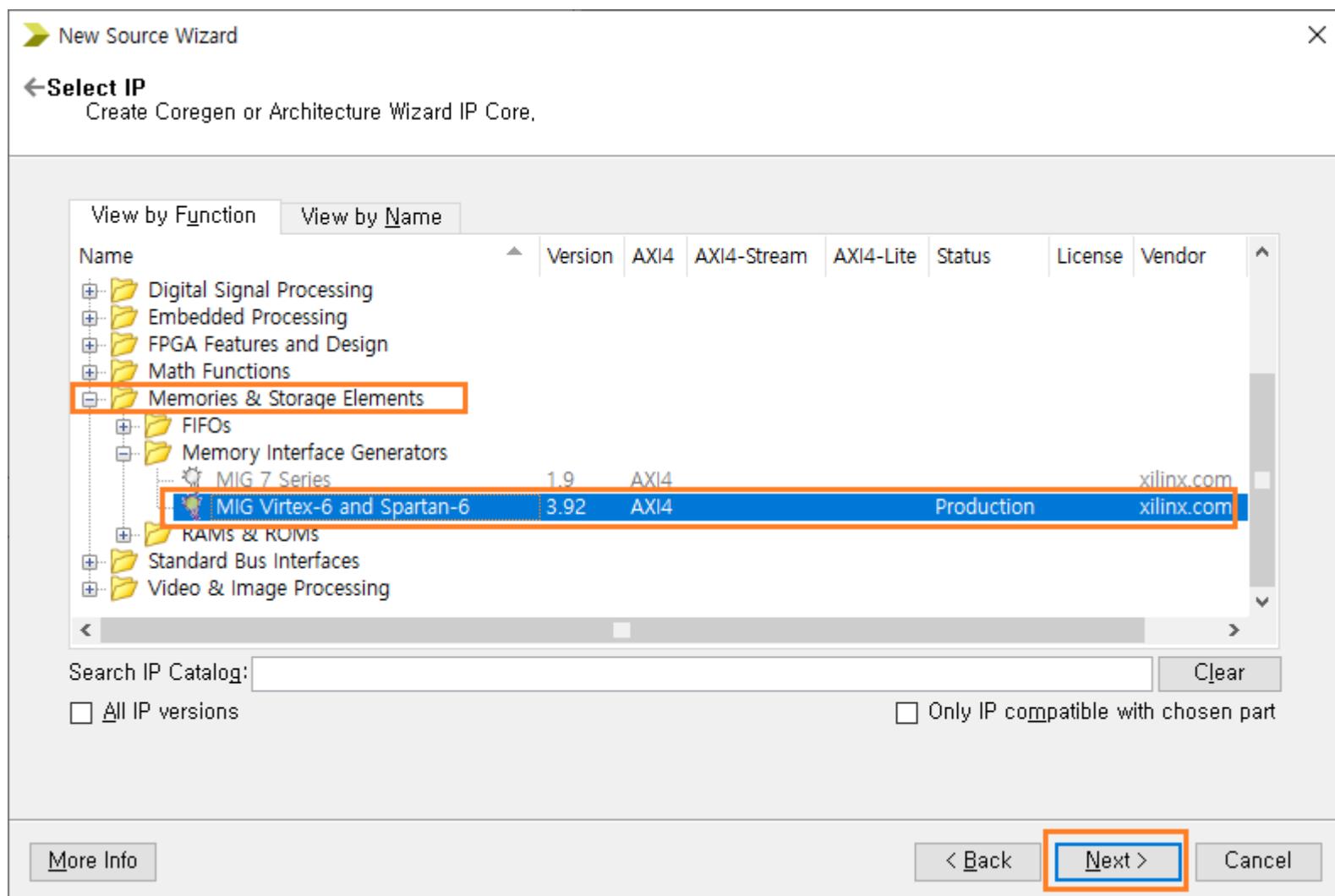
Memory controller를 구현하기 위하여 IP를 생성합니다. xc6slx100-2fgg676 - 우클릭 - New Source 클릭합니다.



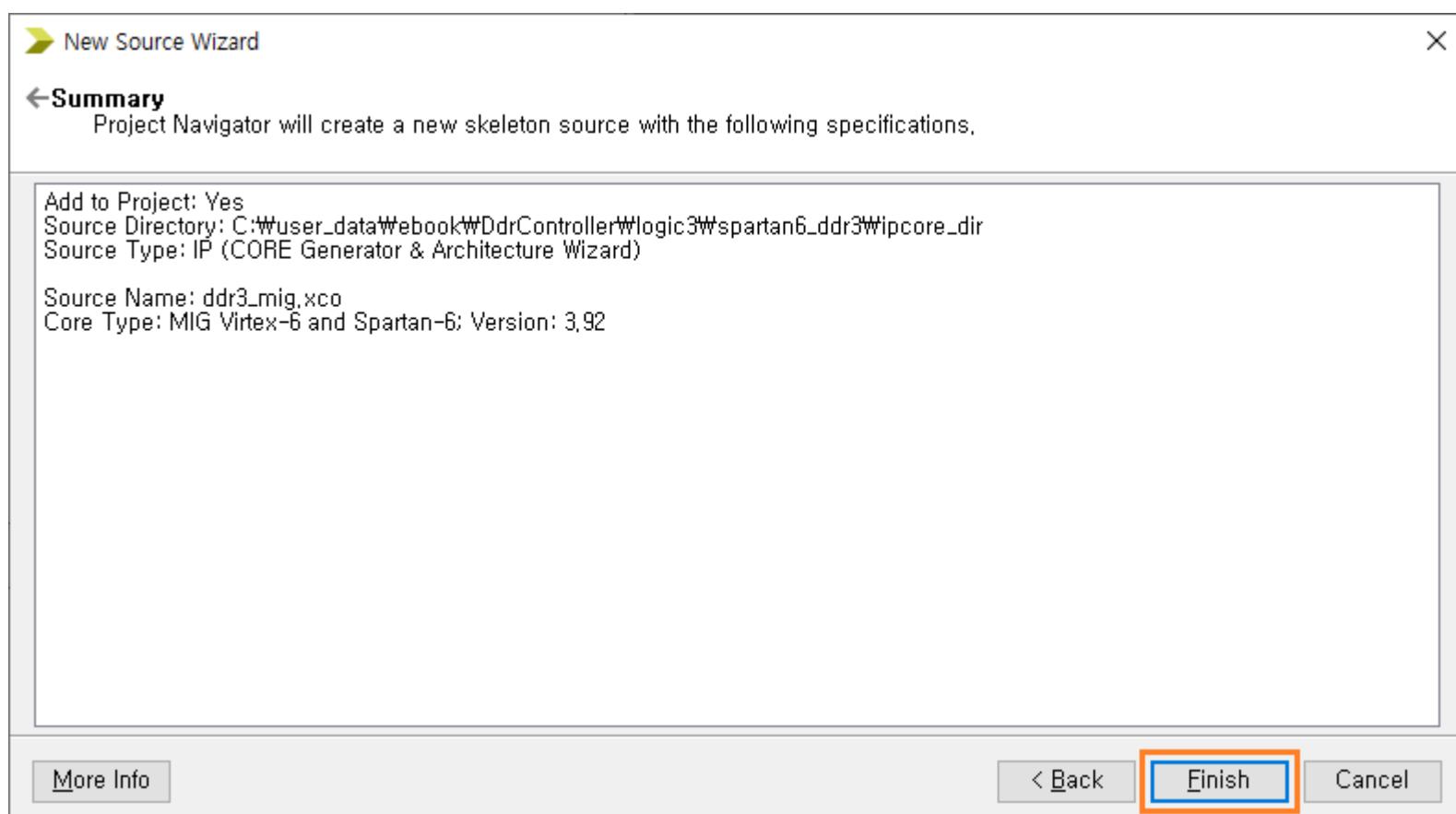
IP (CORE Generator & Architecture Wizard)을 선택하고, File name : ddr3_mig 입력하고 Next 클릭합니다.



IP를 선택합니다. Memories & Storage Elements - Memory Interface Generators - MIG Virtex-6 and Spartan-6 을 선택하고, Next 클릭합니다.

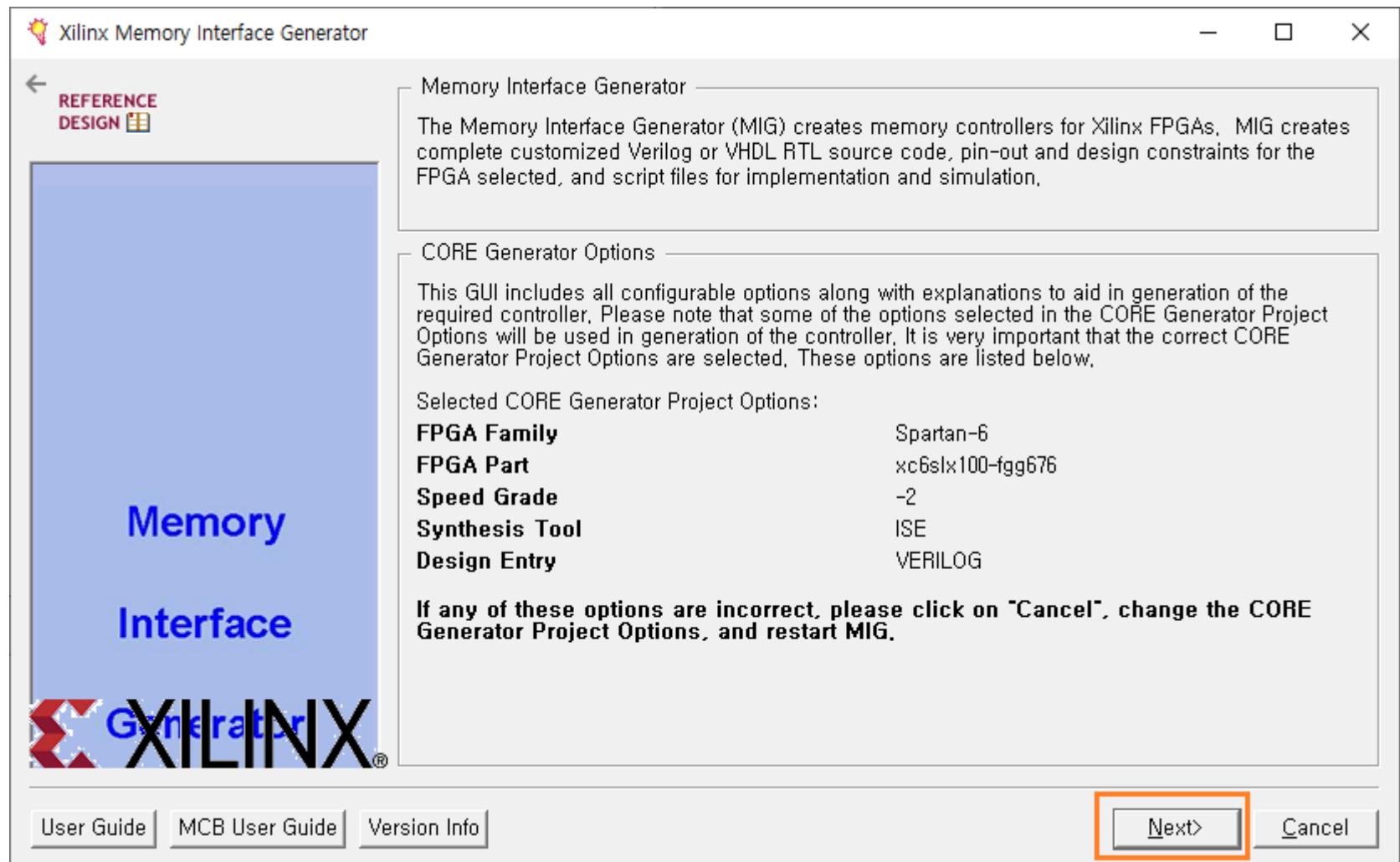


Finish를 클릭합니다.

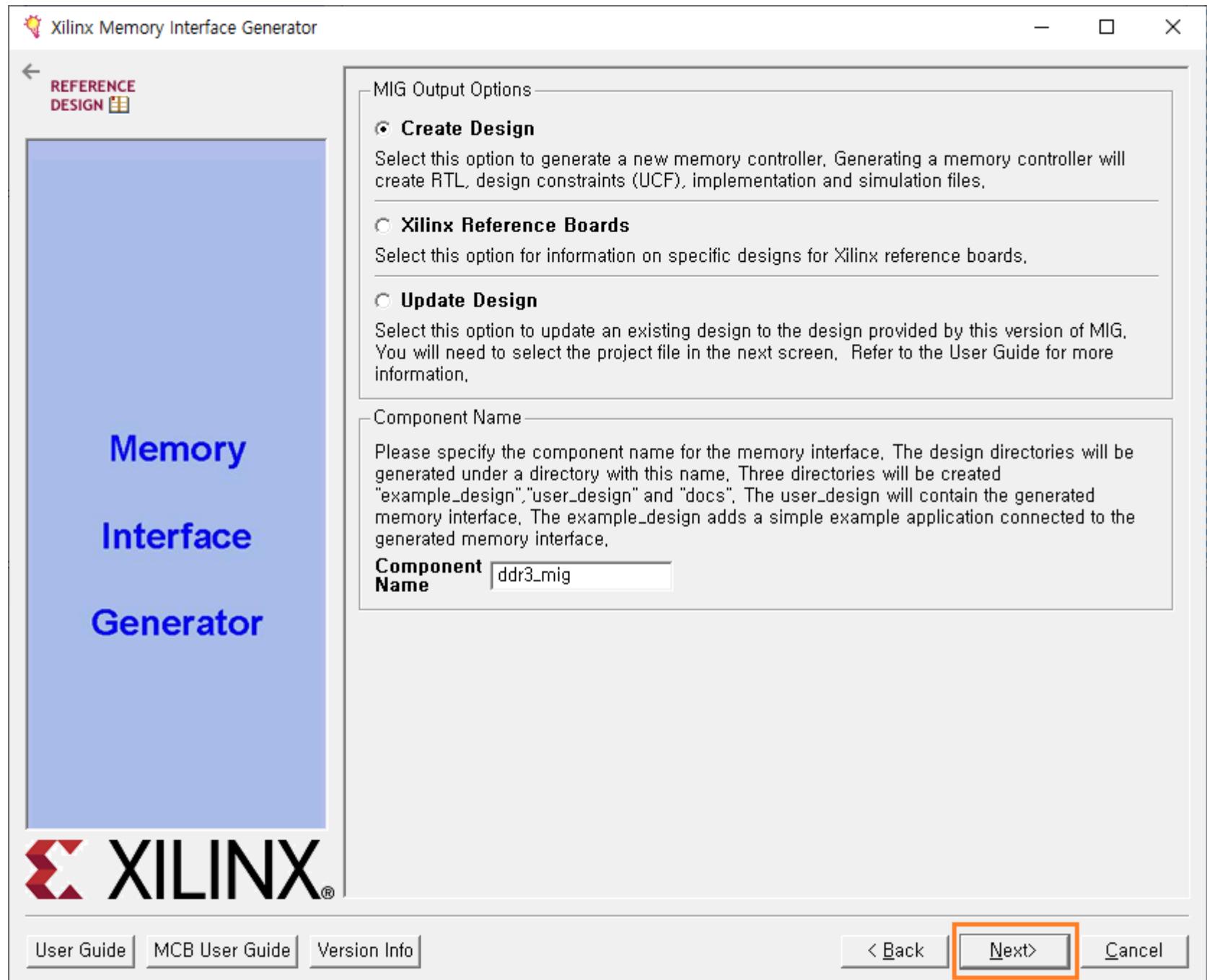


Xilinx Memory Interface Generator 윈도가 생성됩니다. Memory Controller의 세부적인 사항을 설정합니다.

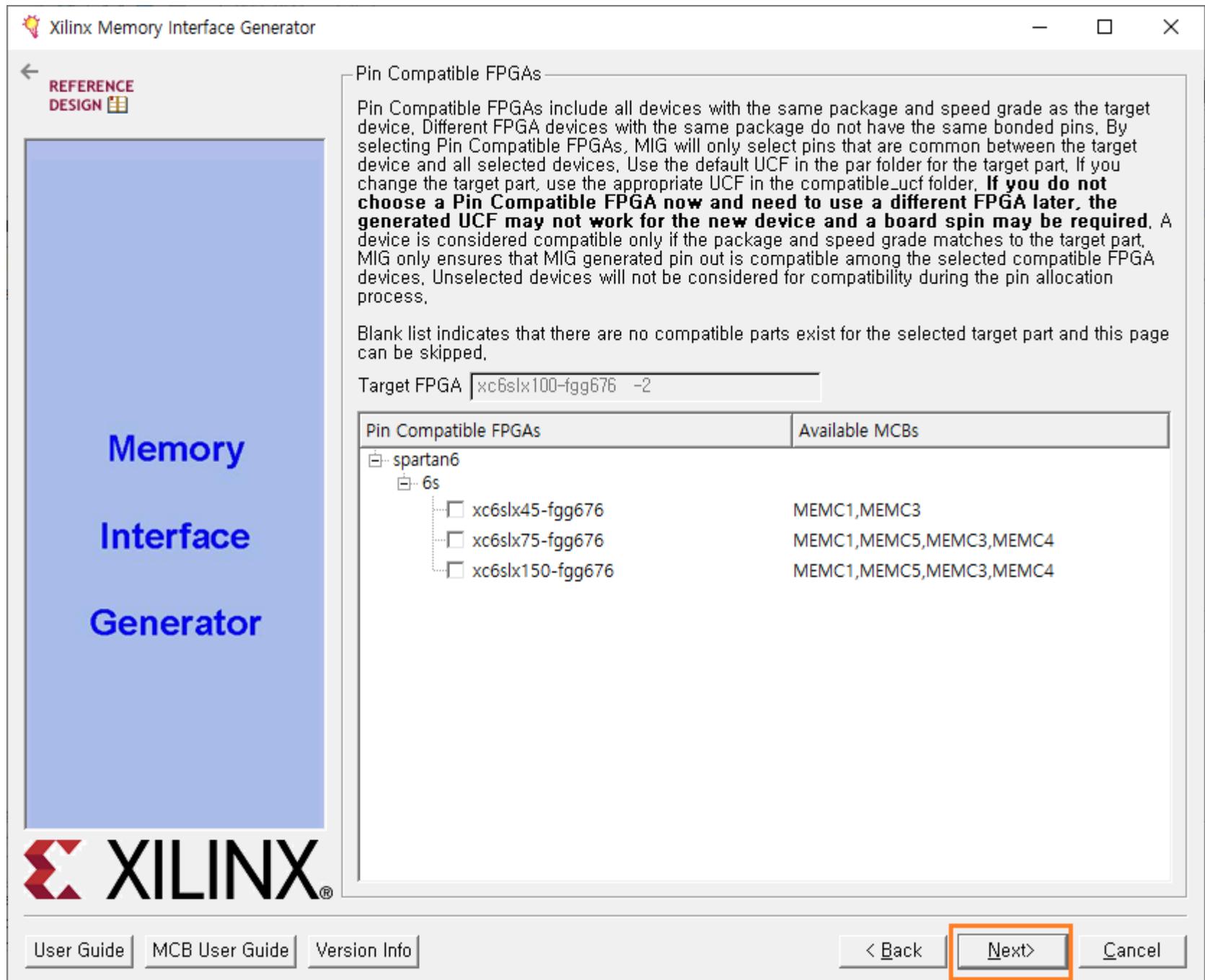
Next 클릭합니다.



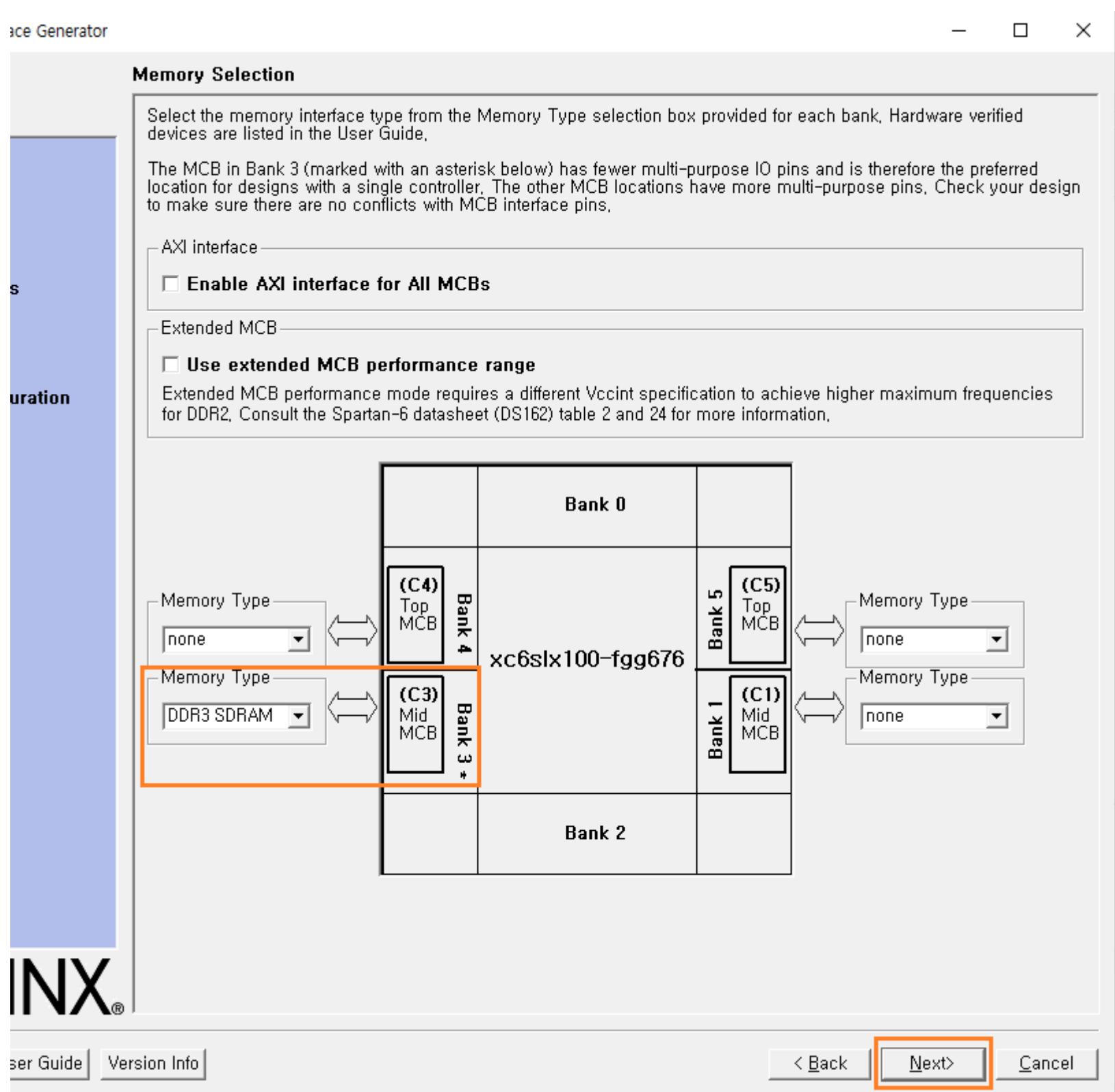
새로운 IP를 생성합니다. Create Design 선택하고 Next 클릭합니다.



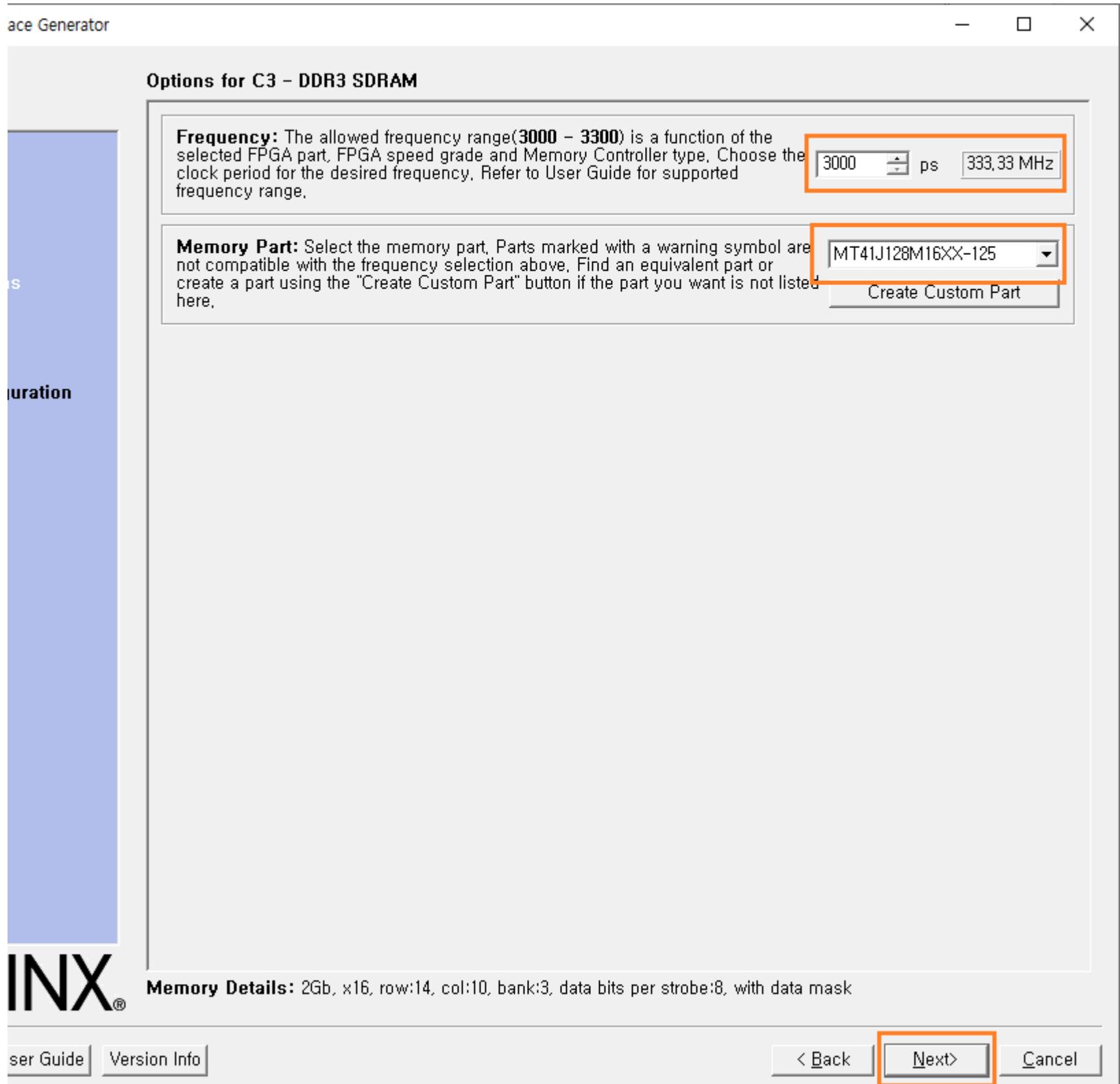
Next 클릭합니다.



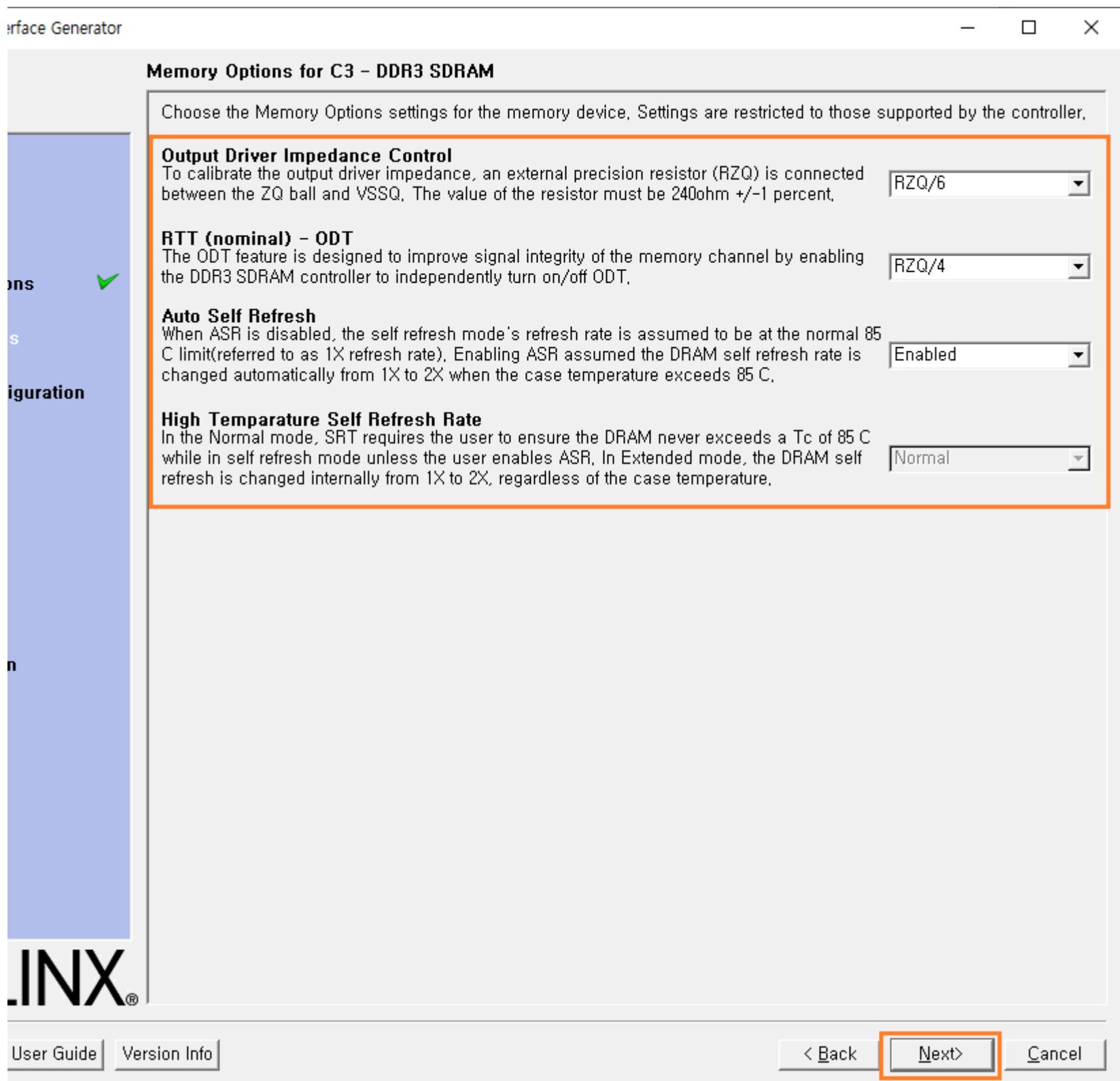
Memory를 연결할 Bank를 설정합니다. Bank3 으로 설정하겠습니다. (다른 Bank를 선택해도 됩니다) Next 클릭합니다.



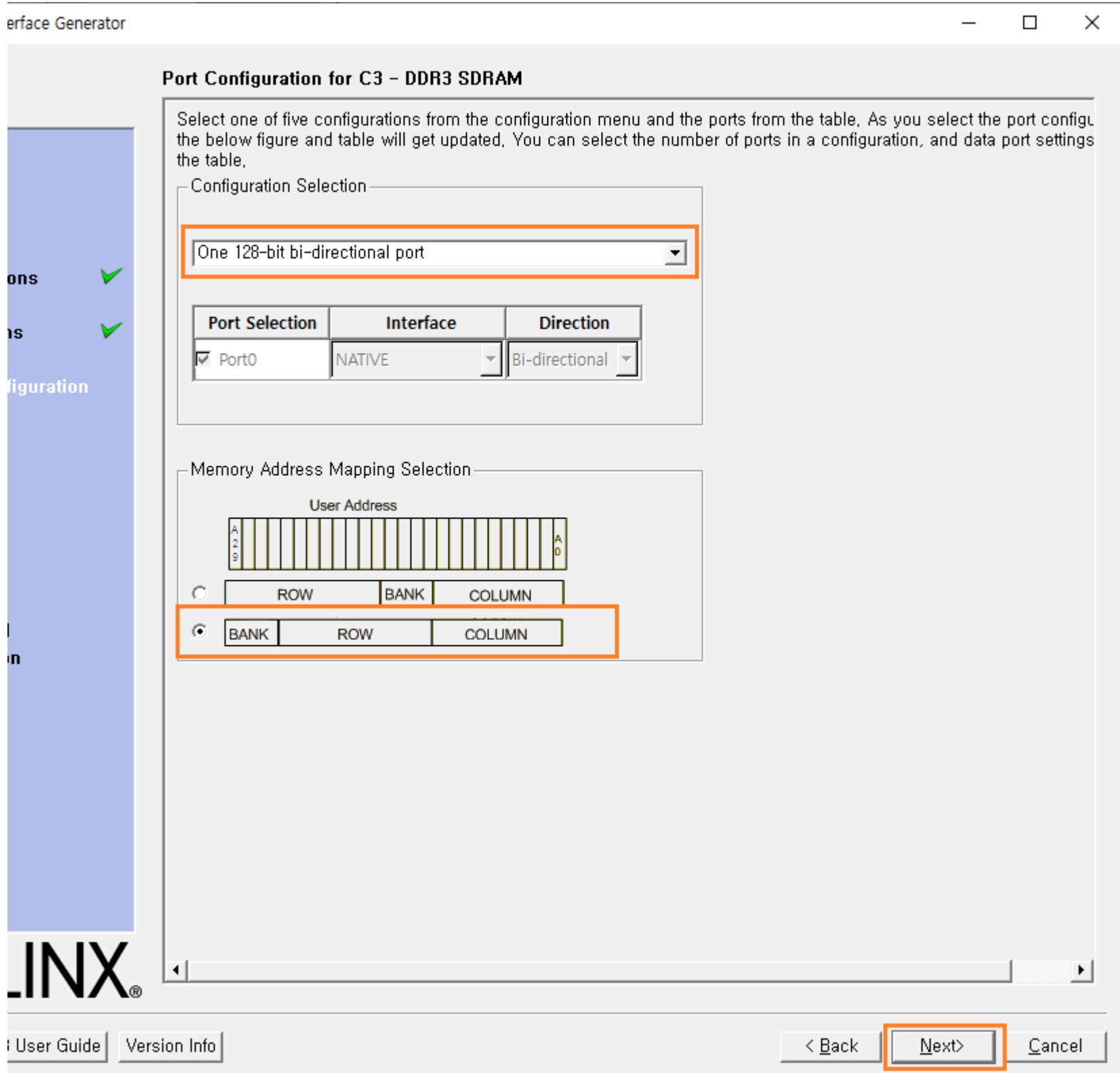
Frequency : 3000 ps (333.33 Mhz), Memory Part : MT41J128M16XX-125 을 설정합니다. 주파수는 기본값을 사용하고 Memory는 해당 메모리를 선택합니다. Next 클릭합니다.



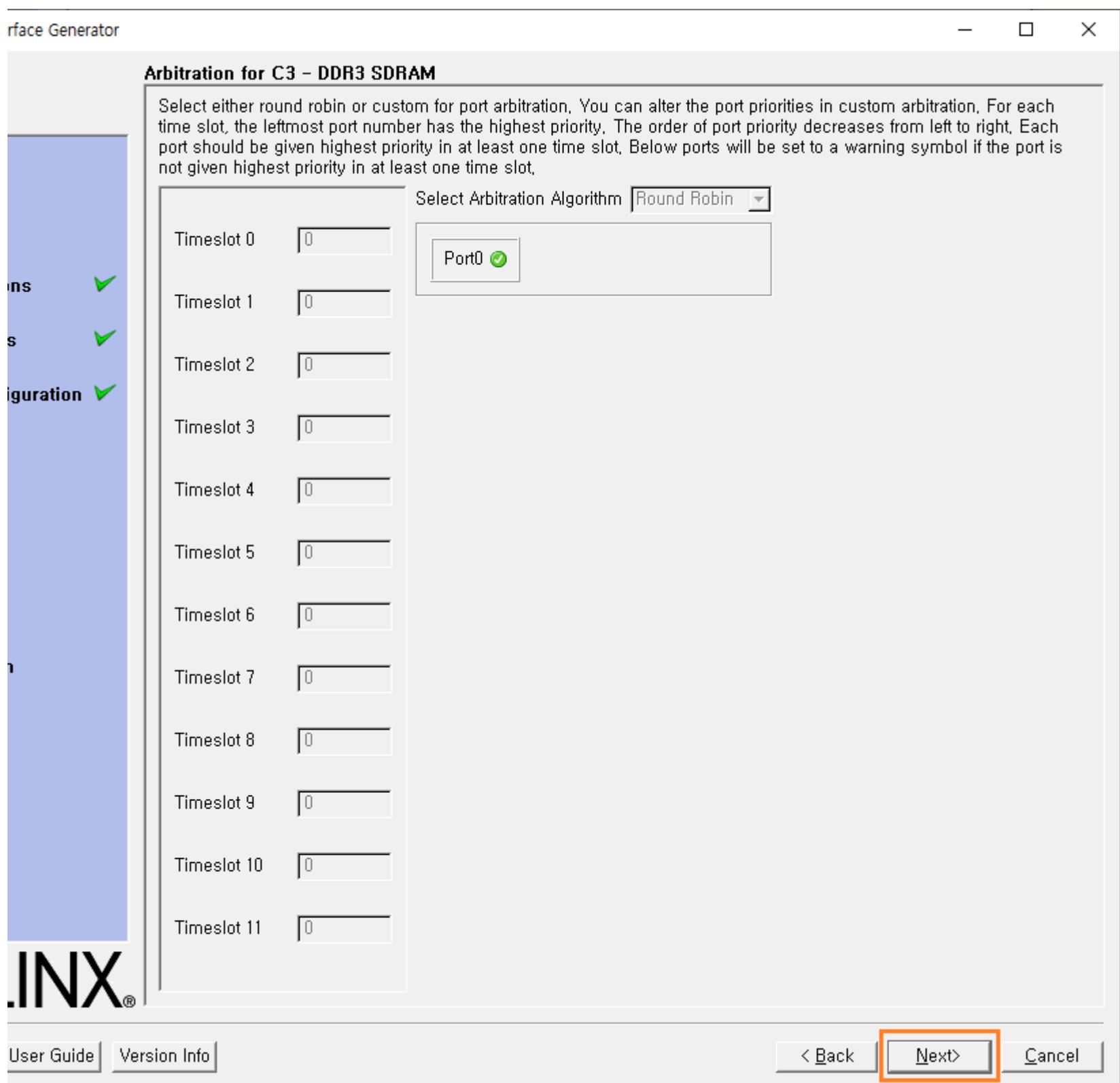
Output Driver Impedance, RTT 를 설정합니다. 기본값을 사용합니다. Next 클릭합니다.



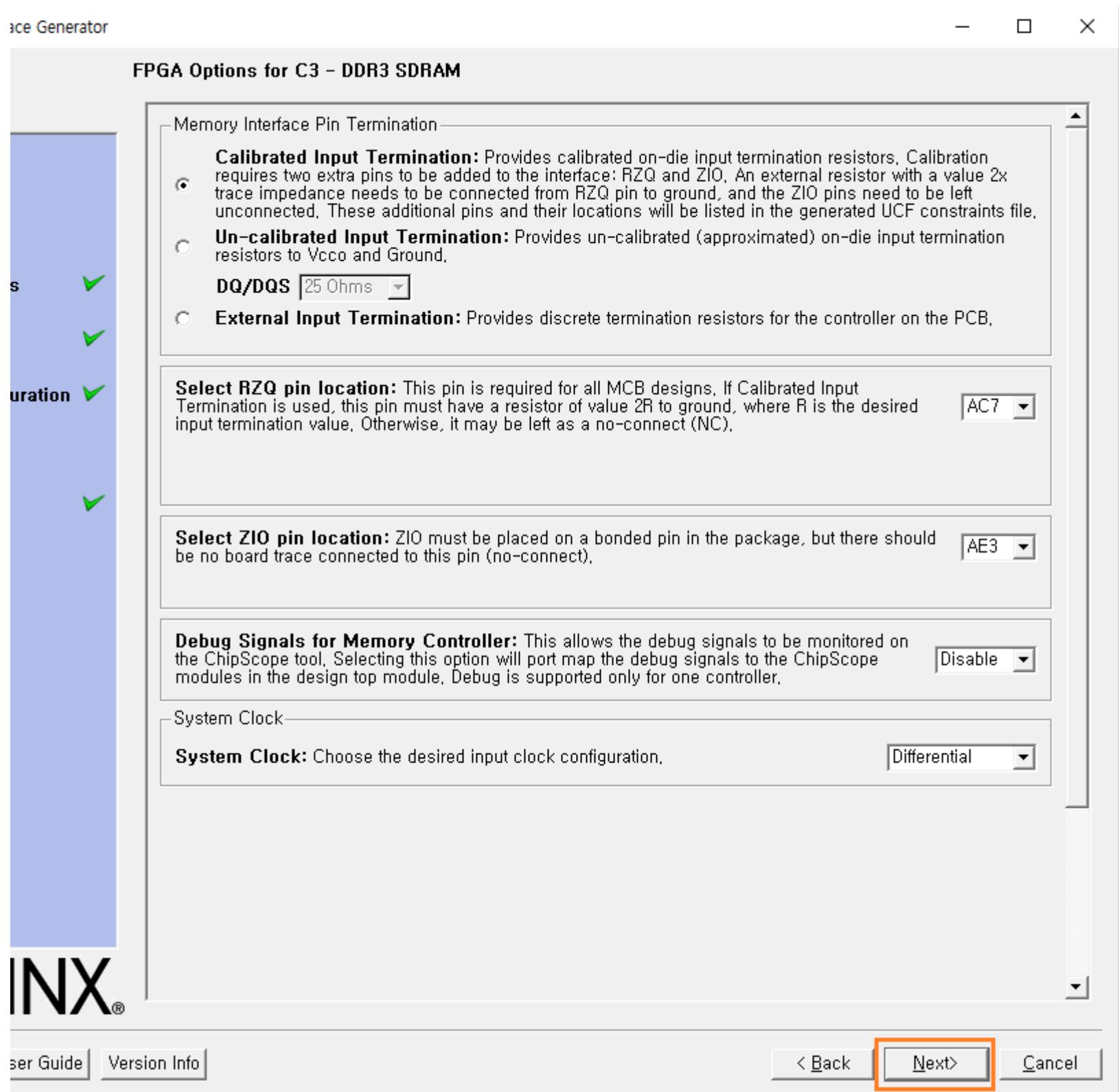
Port Configuration을 설정합니다. Memory에 Access 하는 것을 Port로 설정하고 최대 6개까지 설정가능하고, 각각 32bits로 설정할 수 있습니다. 가장 기본적인 1-Port, 128bits bidirectional port로 설정하겠습니다. Memory Address Mapping Selection은 Bank - Row - Column 으로 설정합니다. (나중에 다른 설정으로 구현해 보는 것도 도움이 됩니다) Next 클릭합니다.



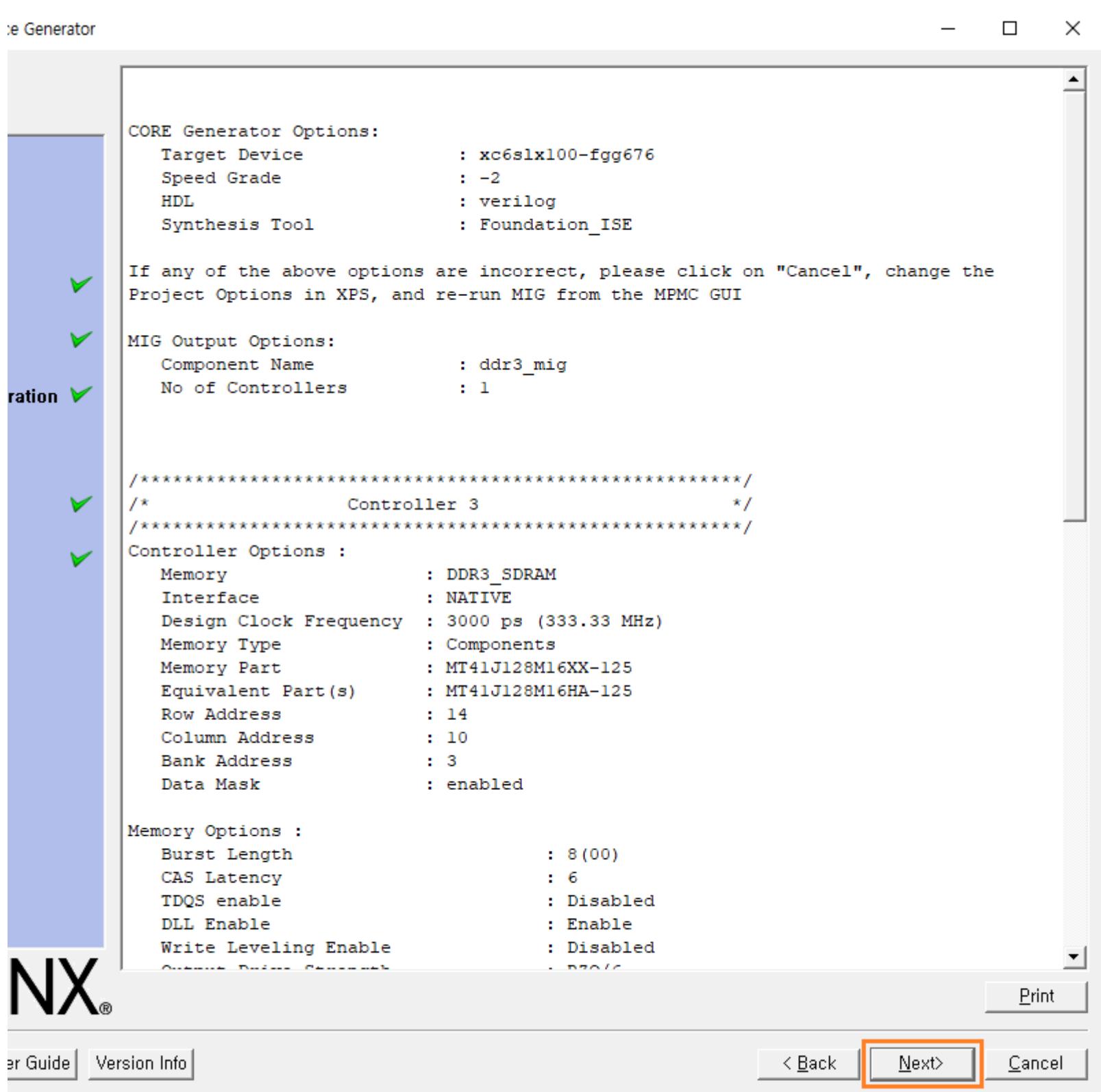
Arbitration 은 Round Robin 으로 설정됩니다. Next 클릭합니다.



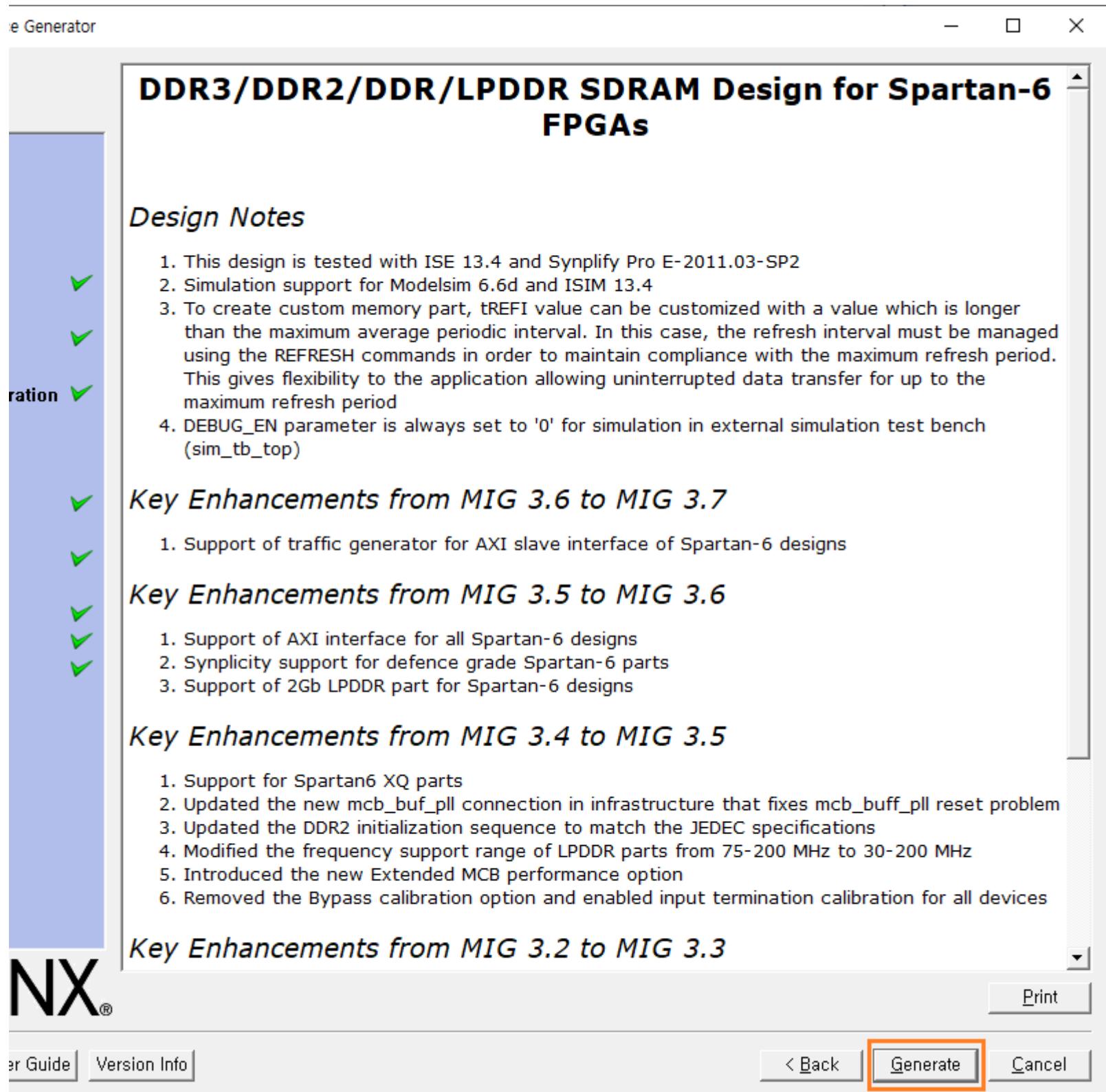
Memory 핀을 설정합니다. 기본값을 사용합니다. Next 클릭합니다.



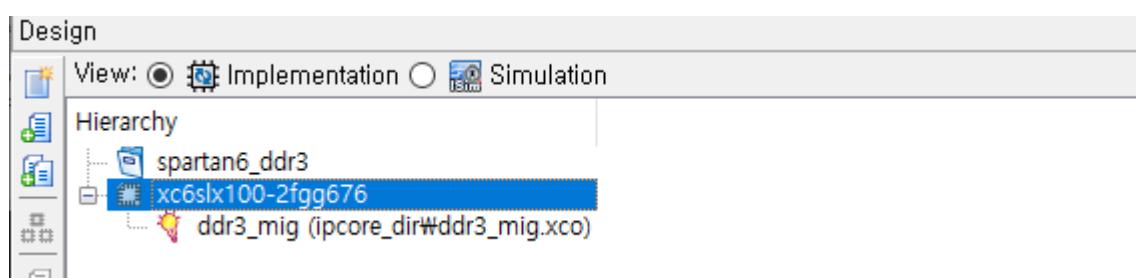
모든 설정이 완료되었습니다. Next 클릭합니다.



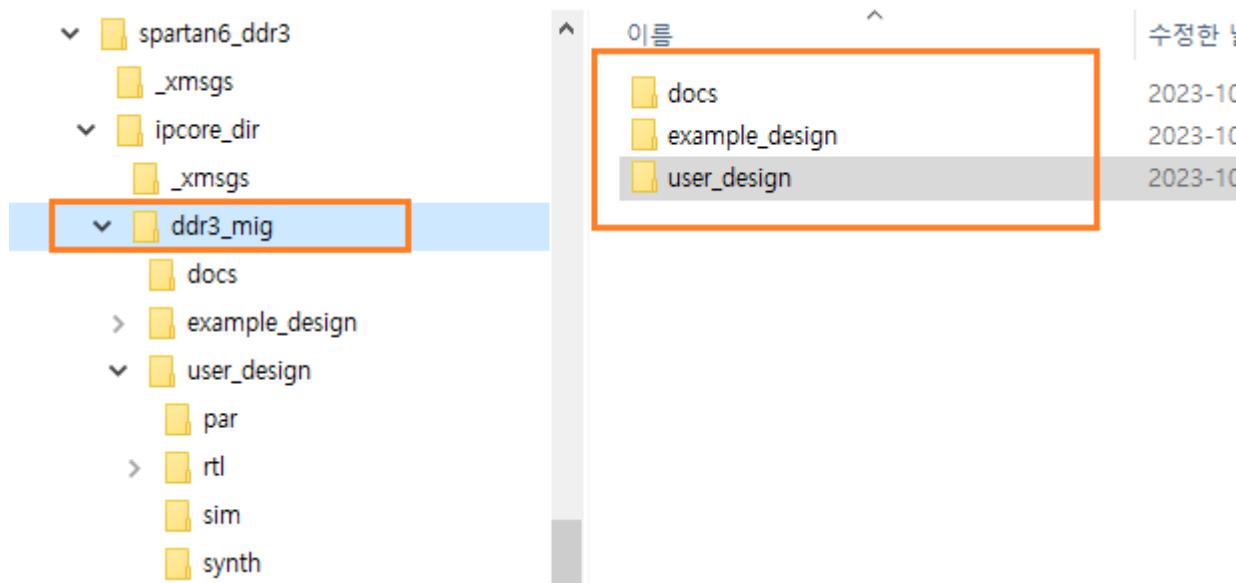
License 관련 내용이 나타납니다. Accept를 선택하고 Next 클릭합니다. 마지막을 Generate 클릭하면 IP가 생성됩니다.



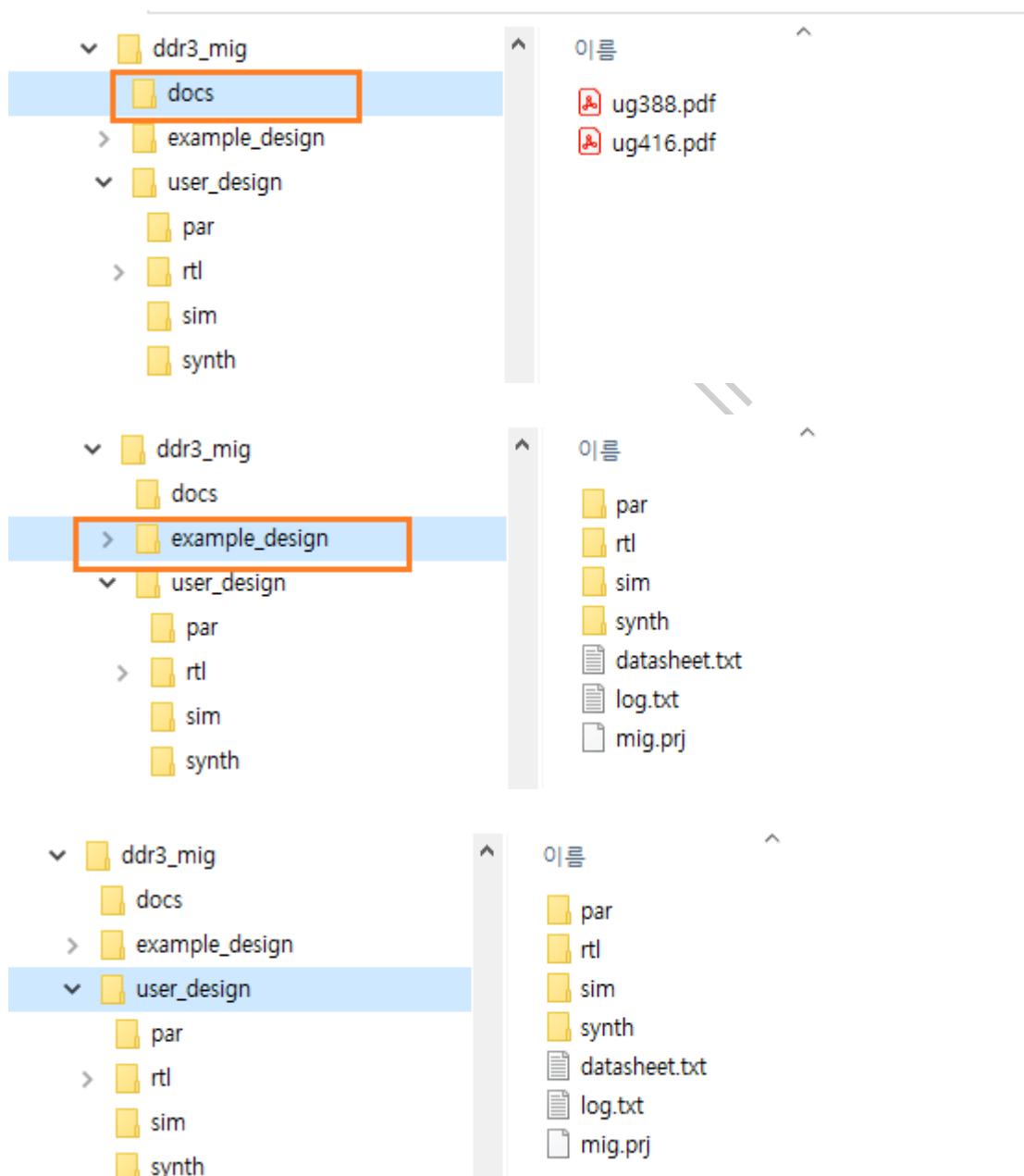
ddr3_mig IP가 추가되었습니다.



생성된 IP의 파일 구조를 살펴보겠습니다. 아래는 ddr3_mig 폴더를 보여줍니다. (ipcore_dir 폴더 안에 있습니다)



3개의 폴더가 있습니다. docs, example_design, user_design



“example_design”, “user_desgin” 폴더는 중복된 내용이 많습니다. 우리는 주로 “user_design” 폴더안의 파일들을 사용하도록 합니다.

9.3 mcb review

생성된 Memory Controller Block (MCB)의 구조 및 동작을 확인합니다. 본 장의 내용은 “Spartan-6 FPGA Memory Controller User Guide (UG388)”을 바탕으로 설명되었습니다.

아래 그림은 mcb의 구조를 보여줍니다.

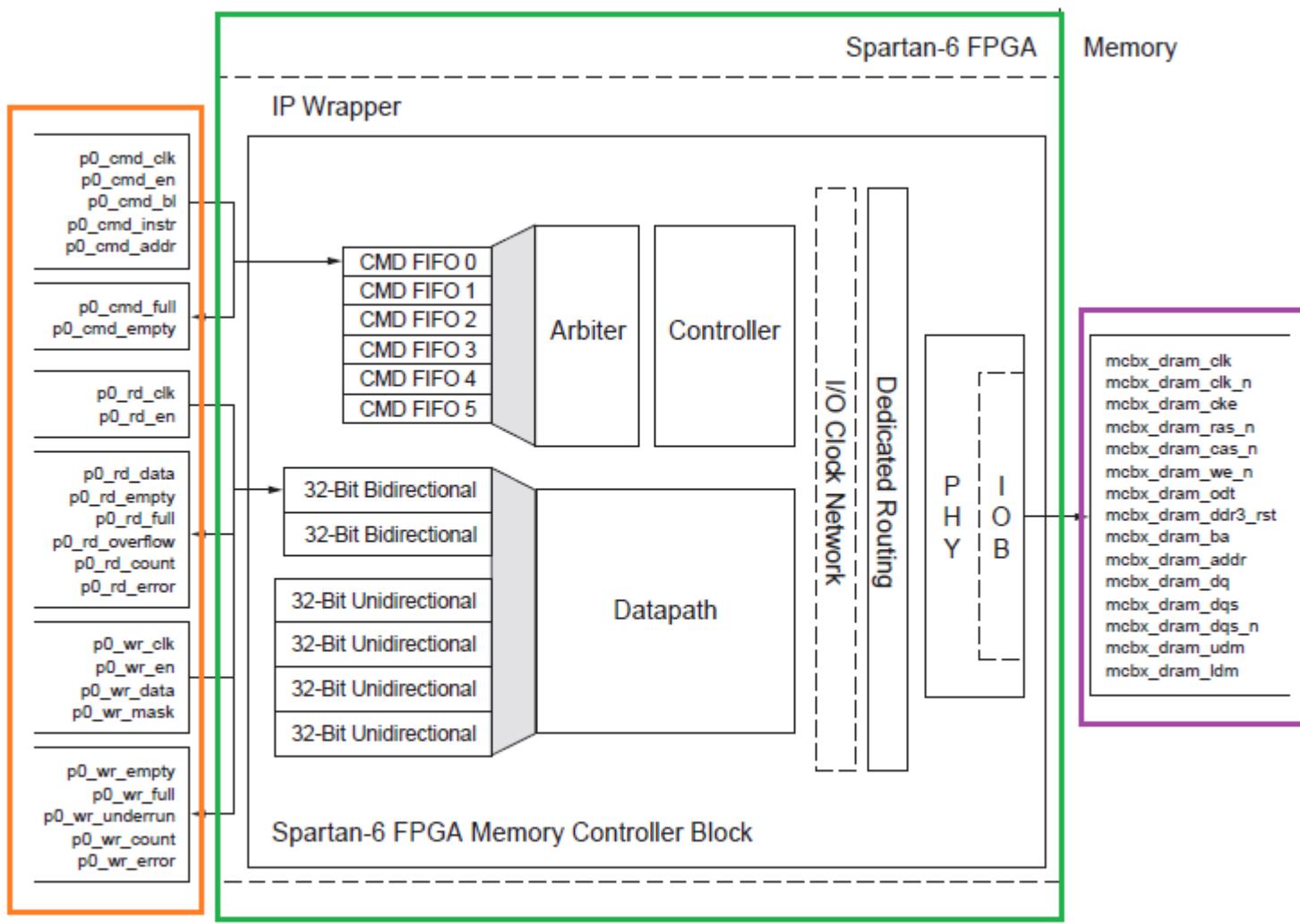


Figure 2-1: MCB Architecture with Major Internal and I/O Signals

중앙에 있는 것(IP Wrapper, ddr3_mig)은 이전 장에서 생성한 Memory Controller입니다. 오른쪽은 실제 Memory (ddr3)을 나타내고, 왼쪽은 User Interface Block입니다. 우리가 궁극적으로 Memory Controller를 구현한다는 것은 왼쪽의 User Interface를 구현한다는 것을 의미합니다.

User Interface Block은 크게 3개로 구성됩니다. Command를 위한 Command Block, ddr에 데이터를 저장하기 위한 (FIFO로 구성된) Write Block, 그리고 ddr에서 데이터를 읽어오는 Read Block으로 구성됩니다.

신호중에 p0는 Port 0을 의미합니다. 메모리 생성시 Port Configuration 항목에서 128bits, Port 0 만 사용하도록 설정하였습니다. 따라서 p0_ 신호만 사용하고 나머지 p1_ ~ p5_~ 신호는 사용하지 않습니다.

Command (Instruction)은 아래와 같이 5개로 구성됩니다. 이 중에서 user가 구현해야 할 것은 Write (000), Read (001)입니다. 나머지는 mcb 에서 자동으로 처리해 줍니다.

Instructions

Table 4-1 provides detailed descriptions for all memory instructions implemented by the MCB. To load an instruction into the Command FIFO of a User Interface port, the 3-bit code for the instruction is clocked into the pX_cmd_instr[2:0] inputs on the rising edge of pX_cmd_clk.

Table 4-1: Instructions Implemented by the MCB

| Instruction | Code [2:0] | Description |
|---------------------------|------------|--|
| Write | 000 | Memory Write. Writes the number of data words specified by pX_cmd.bl[5:0] to the memory device beginning at the byte address specified by pX.cmd.addr[29:0]. Prior to issuing this instruction, sufficient data must be loaded into the Write Data FIFO to complete the transaction. Otherwise a data “underrun” condition occurs. This instruction is valid for write only and bidirectional ports. |
| Read | 001 | Memory Read. Reads the number of data words specified by pX.cmd.bl[5:0] from the memory device beginning at the byte address specified by pX.cmd.addr[29:0]. Prior to issuing this instruction, the Read Data FIFO must have enough space to complete the transaction. Otherwise a data “overflow” condition occurs. This instruction is valid for read only and bidirectional ports. |
| Write with Auto Precharge | 010 | Memory Write with Auto Precharge. This instruction is the same as the Write instruction but with auto precharge appended after burst completion. Auto precharge closes the DRAM bank where the transaction ended. This can improve latency for applications with more random access patterns that tend to jump between rows in the same bank. Note: The MCB looks ahead at subsequent transactions. The auto precharge is skipped if the following transaction is to the same row accessed in the current transaction. |
| Read with Auto Precharge | 011 | Memory Read with Auto Precharge. This instruction is the same as the Read instruction but with auto precharge appended after burst completion. Auto precharge closes the DRAM bank where the transaction ended. This can improve latency for applications with more random access patterns that tend to jump between rows in the same bank. Note: The MCB looks ahead at subsequent transactions. The auto precharge is skipped if the following transaction is to the same row accessed in the current transaction. |
| Refresh | 1xx | Memory Refresh. Prompts the MCB to issue a refresh command to the memory device. Resets the tREFI counter allowing data to stream uninterrupted for a full refresh cycle. This instruction should only be used for highly customized dataflow structures. In general, the MCB automatically issues refresh commands on its own, which periodically results in increased latency for transactions. |

Address는 128bits를 Access 하기 위해서는 하위 4bits는 항상 0 입니다. 즉 Address 는 바이트 단위임을 알 수 있습니다.

Addressing

From the User Interface perspective, the MCB provides a simple and sequential byte addressing scheme into the physical DRAM. The fact that DRAMs store data in fixed segments is abstracted by this scheme, allowing for a simple SRAM-like address interface. For details on how the bank, row, and column address bits are mapped to the byte address, refer to [Byte Address to Memory Address Conversion, page 61](#).

[Table 4-2](#) shows how the byte address presented to the User Interface must be aligned to the port width. Depending on the number of bytes in the port width, a certain number of the low address bits must be set to 0 to ensure that consecutive addresses fall on data word boundaries. The write data mask inputs (pX_wr_mask) to the User Interface can be used to offset the starting address byte location. For example, to begin writing at byte address 0x01 when using a 32-bit (4-byte) User Interface, the byte address presented to the command port of the User Interface should be 0x00 to meet the requirements of [Table 4-2](#), but the least significant mask bit should be set to 1 such that only bytes at address 0x01 and higher are actually written.

Table 4-2: Address Requirements for Byte Address Alignment

| Port Width | Bytes per Data Word | Address Requirement |
|------------|---------------------|--------------------------------|
| 32 bits | 4 | $pX_cmd_addr[1:0] = 2'b00$ |
| 64 bits | 8 | $pX_cmd_addr[2:0] = 3'b000$ |
| 128 bits | 16 | $pX_cmd_addr[3:0] = 4'b0000$ |

아래 그림은 Command Path Timing을 보여줍니다.

Command Path Timing

The command path of the User Interface uses a simple 4-deep FIFO structure to hold pending commands. The instruction type, address, and burst length for the requested transaction are all loaded into this Command FIFO. The full flag (`pX_cmd_full`) signal from the command FIFO must be Low for a new command to be accepted into the FIFO when `pX_cmd_en` is asserted during the rising edge of `pX_cmd_clk`. Otherwise, the command is ignored. Figure 4-4 and Figure 4-5 demonstrate the protocol for loading a command into the FIFO.

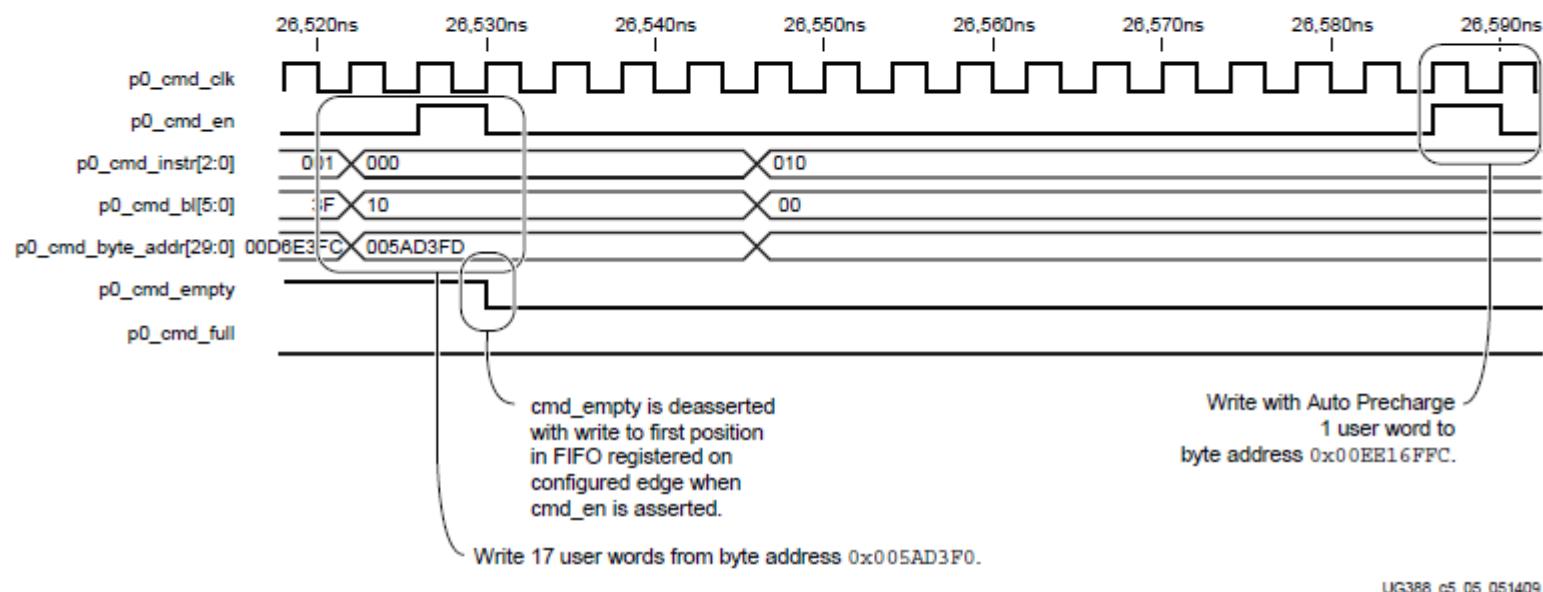


Figure 4-4: Command Path Timing (Write)

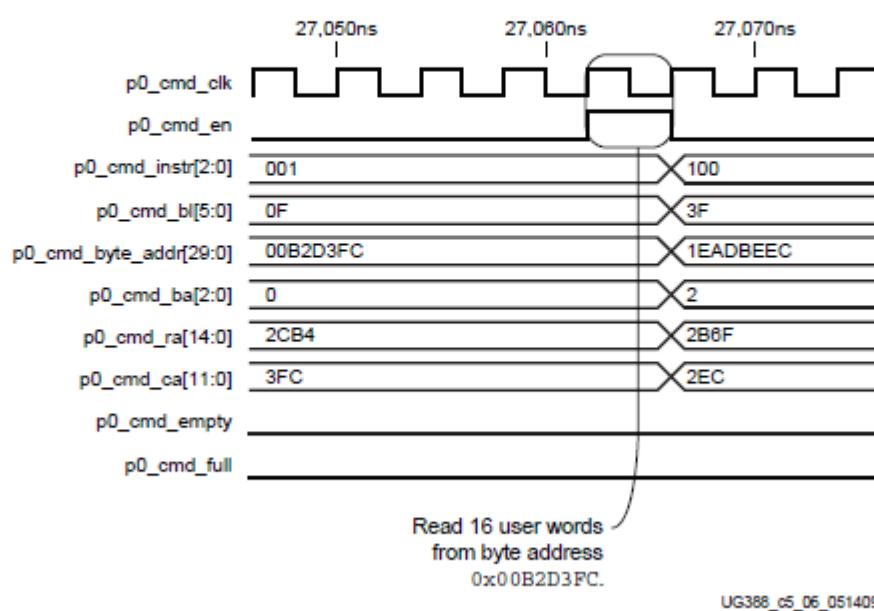


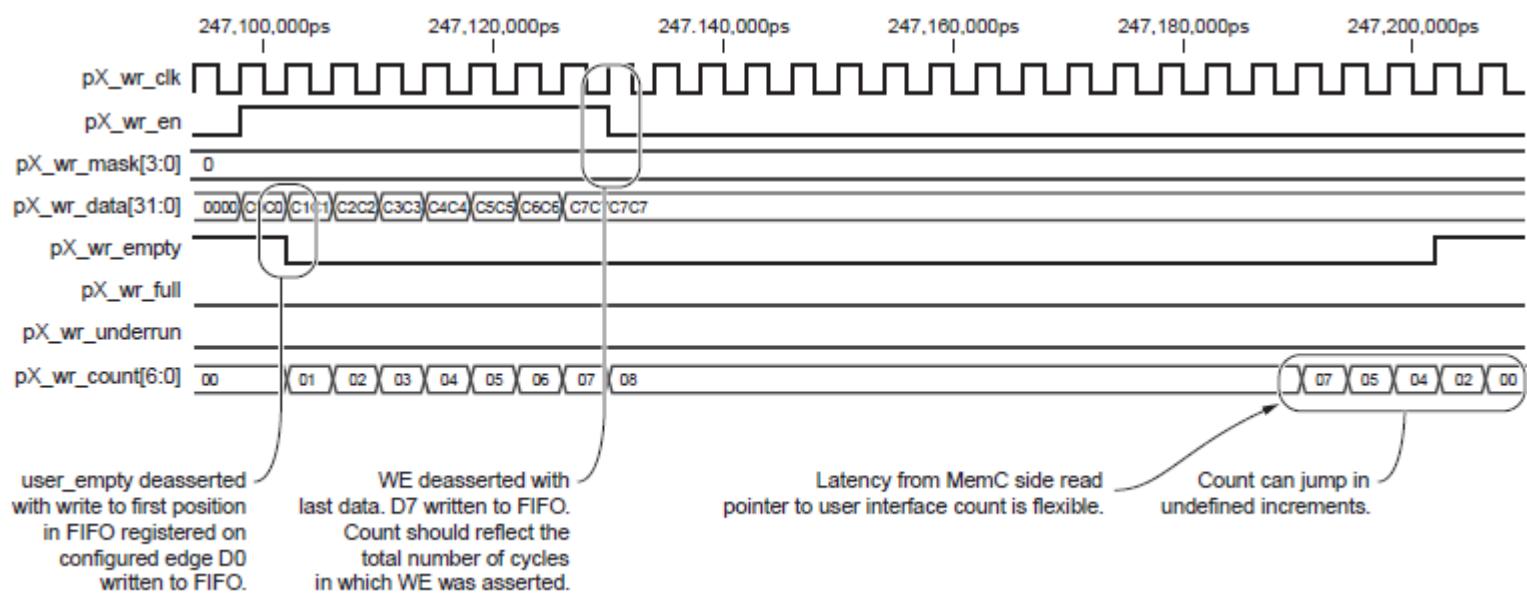
Figure 4-5: Command Path Timing (Read)

`cmd_instr[2:0]`(write : 000b, read : 001b), `cmd_bl[5:0]` (128bits 기준으로 몇개를 write 할지를 설정함. 0 : 128bits x 1, 63 : 128bits x 64), `cmd_byte_addr[29:0]`을 설정하고 `cmd_en`을 Active 합니다. command를 전송하기 전에 `cmd_full`이 0인지 check 해야 합니다. 만일 `cmd_full`이 1(Active)이면 command를 전송하면 안됩니다. `cmd_full`이 0가 될 때까지 기다려야 합니다. 모든 신호는 Positive edge에서 동작합니다. Figure 4-5에서 `cmd_ba`, `cmd_ra`, `cmd_ca` 신호는 사용하지 않습니다. `cmd_byte_addr`를 설정하면 내부에서 `ba`, `ra`, `ca`는 자동으로 계산됩니다.

아래 그림은 write path timing을 보여줍니다.

Write Path Timing

The write path of the User Interface uses a simple 64-deep FIFO structure to hold data in preparation for a Write transaction to memory. Similar to the Command FIFO, the full flag (`pX_wr_full`) from the Write Data FIFO must be Low for new data to be accepted into the FIFO when `pX_wr_en` is asserted during the rising edge of `pX_wr_clk`. Otherwise, the data is ignored. If the full flag is Low, the `pX_wr_data` bus is captured into the FIFO on the rising edge of `pX_wr_clk`. For every clock cycle that `pX_wr_en` is asserted, there must be valid data on the `pX_wr_data` bus. [Figure 4-6](#) demonstrates the protocol for loading data into the Write Data FIFO.



위 그림에서는 port width : 32bits를 기준으로 설명되어 있습니다. 우리가 생성한 128bits 기준으로 하면, `wr_mask[15:0]`, `wr_data[127:0]` 가 됩니다. `wr_mask`는 해당 바이트를 masking 처리함으로 write 하지 않음을 나타냅니다.

write 하기 위해서는 먼저 `wr_full` 인지를 확인해야 합니다. `wr_full` 이면, `wr_full` 이 아닐 때까지 기다립니다. write 하고자 하는 데이터 만큼 (128bits x N) 데이터를 `wr_data`에 쓰고, 동시에 `wr_en`을 Active로 만들어 줍니다. `wr_en`은 write하고자 하는 word(128bits) 수 만큼 Active 되어야 합니다. 한 번에 최대 write 할 수 있는 수는 64입니다. `cmd_b1[5:0]`가 write 하는 word 수를 나타냅니다 (write 수 - 1 값을 설정)

`wr_empty`는 `wr_en`이 active 되고 난 후, 2-clock 후에 0가 됩니다.

아래 그림은 read path timing을 보여줍니다.

Read Path Timing

The read path of the User Interface uses a simple 64-deep FIFO structure to hold data returning from a Read transaction. The empty flag (`pX_rd_empty`) from the Read Data FIFO can be used as a data valid indicator. Whenever `pX_rd_empty` is deasserted, there is valid data present on the `pX_rd_data` bus. To transfer data into the FPGA logic from the Read Data FIFO, the `pX_rd_en` signal must be asserted on the rising edge of `pX_rd_clk`. The `pX_rd_data` bus transitions on the rising edge of `pX_rd_clk`. The `pX_rd_en` signal can remain asserted at all times and the `pX_rd_empty` signal can be used as a data valid indicator, if desired. Figure 4-7 demonstrates the protocol for loading data out of the Read Data FIFO.

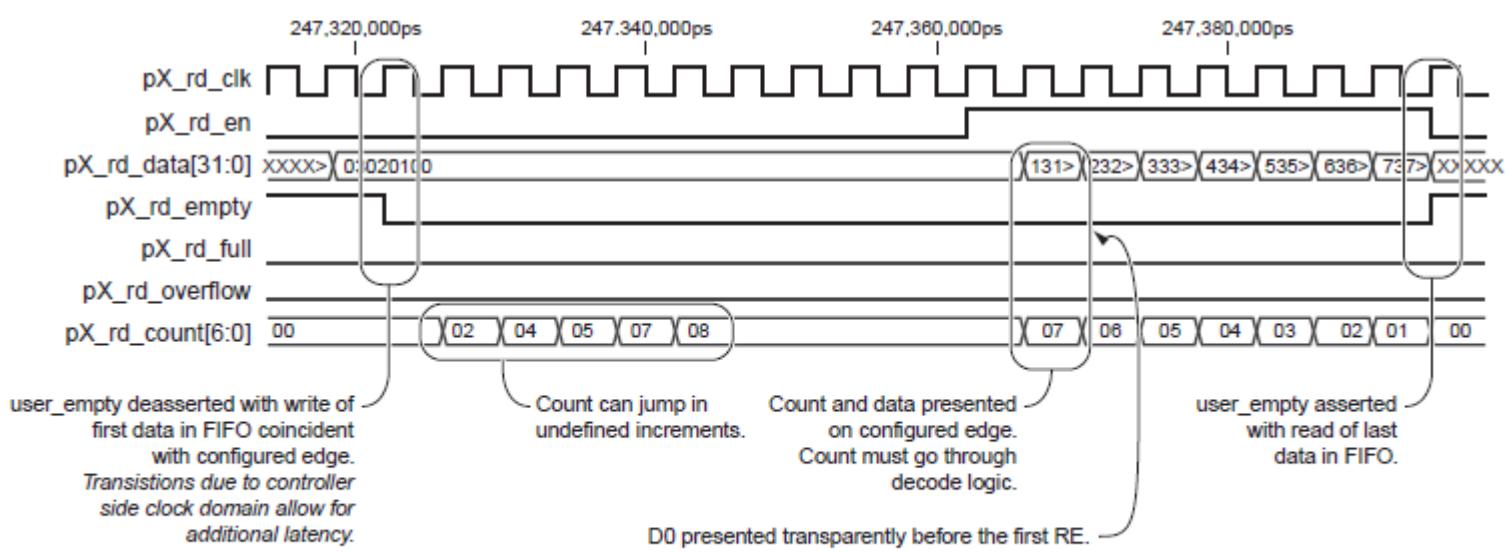


Figure 4-7: Read Path Timing

`rd_empty`가 0 이면 (read 가능한 데이터가 있음을 의미합니다) `rd_en`을 active 해서 `rd_data`를 읽습니다. `rd_empty` 가 1 이면, 읽을 수 있는 데이터가 없다는 것을 의미하기 때문에 `rd_empty`가 0가 될 때까지 기다려야 합니다.

아래는 write를 위한 전체 sequence입니다. write는 먼저 fifo에 데이터를 저장하고 write command를 전송합니다.

- 1) wr_full 신호가 0인지 확인합니다. (wr_full이 1이면 write_fifo가 full이기 때문에 데이터를 전송하면 안됩니다)
- 2) 데이터를 write path timing에 맞게 write_fifo에 저장합니다.
- 3) cmd_full 신호가 0인지 확인합니다. (0이 될 때까지 기다립니다)
- 4) write command를 전송합니다. 이때 write 할 word 수를 cmd_b1에, 주소를 cmd_addr에 설정합니다

아래는 read를 위한 전체 sequence입니다. read는 먼저 read command를 전송하고, fifo에 들어있는 데이터를 읽습니다.

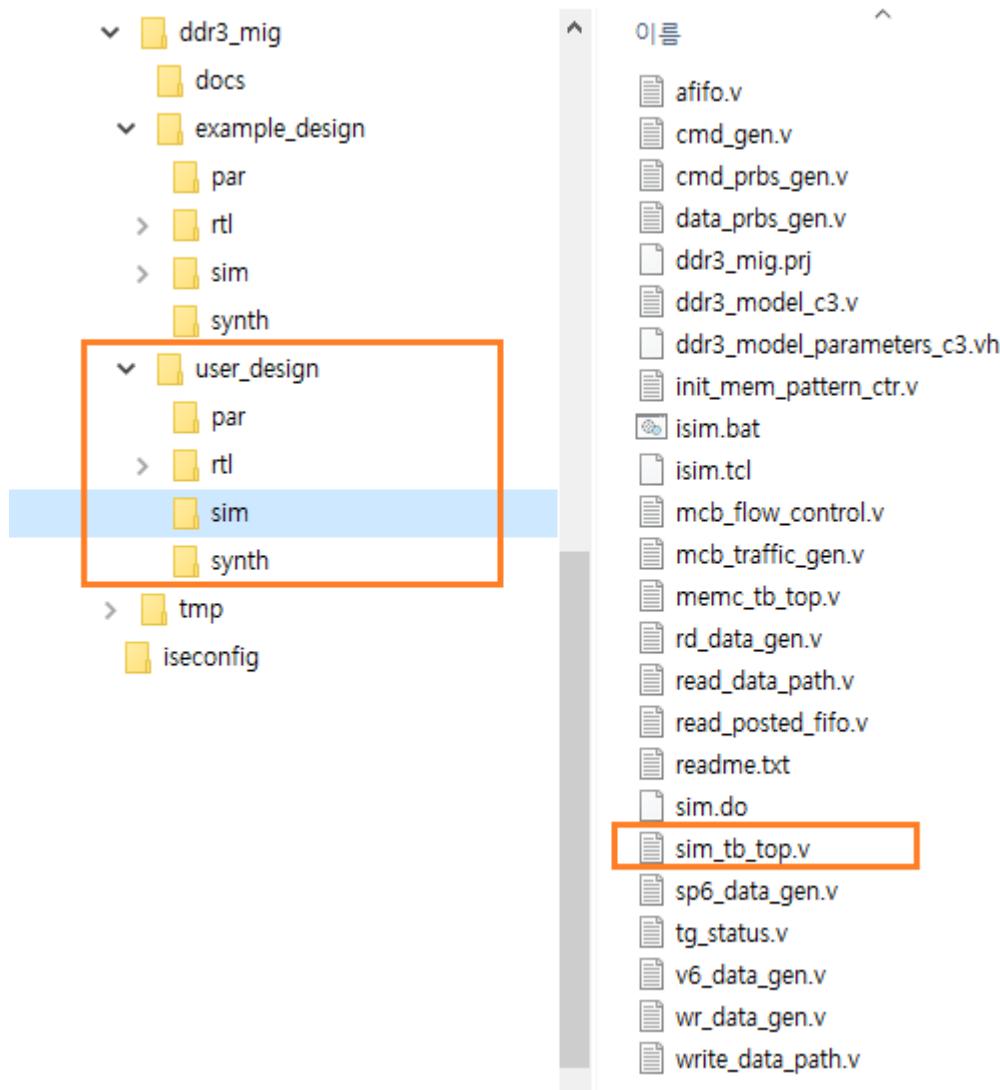
- 1) cmd_full 신호가 0인지 확인합니다. (0이 될 때까지 기다립니다)
- 2) read command를 전송합니다. 이때 read 할 word 수를 cmd_b1에, 주소를 cmd_addr에 설정합니다.
- 3) rd_empty가 0인지 확인합니다. (0이 될 때까지 기다립니다)
- 4) rd_en을 Active 해서 데이터를 읽습니다.

AIHIL

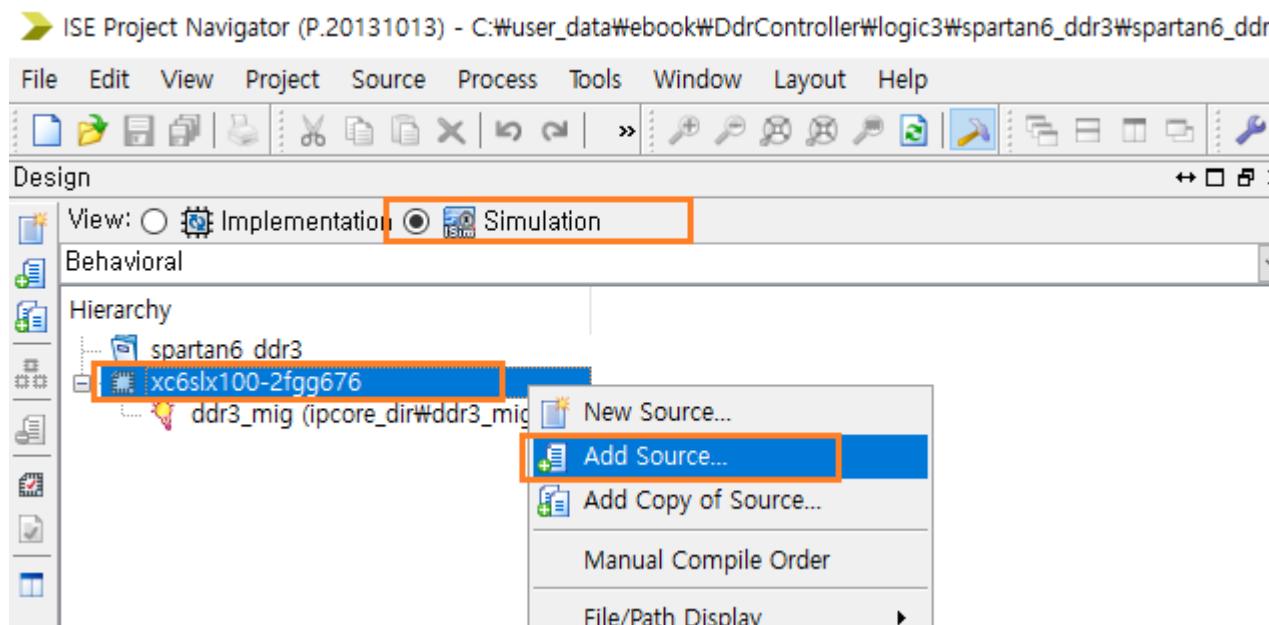
9.4 simulation을 통한 동작 확인

이번 장에서는 simulation을 통하여 memory controller의 동작을 확인합니다. 생성한 IP는 simulation을 위한 Test bench, memory model 등 필요한 파일들을 제공합니다. “example_design” 폴더내의 파일들을 사용할 수도 있지만, 이번장에서는 “user_design” 폴더내에 있는 파일들을 사용하여 simulation을 진행합니다.

아래는 “user_design\sim” 폴더에 있는 파일들을 보여줍니다. test bench 파일은 “sim_tb_top.v” 입니다.

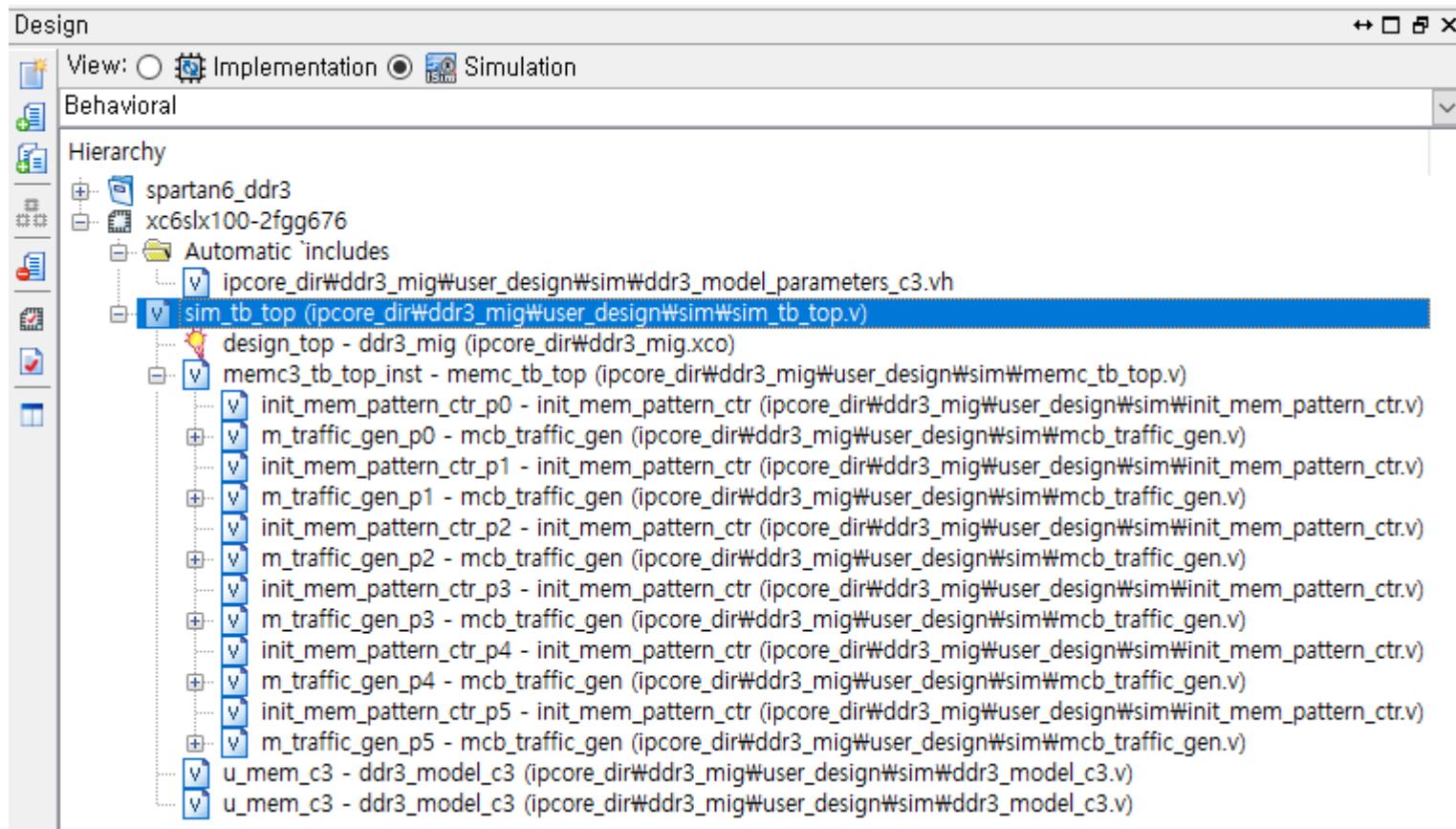


ISE에서 simulation을 선택하고, xc6slx100-2fgg676 - 우클릭 - Add Source... 클릭해서 sim_tb_top.v 파일과 그외 필요한 파일들을 추가합니다.

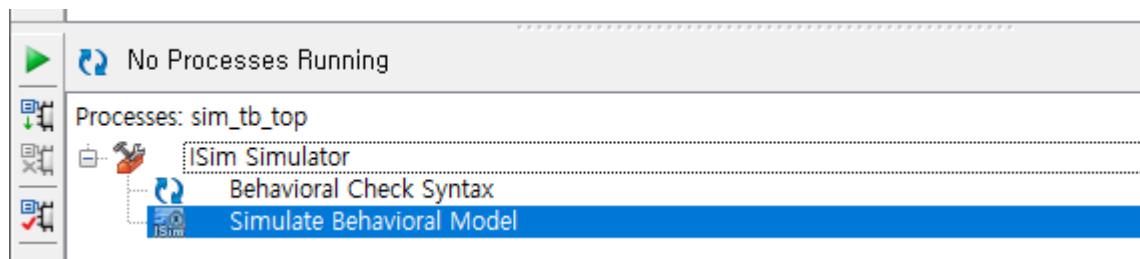


"user_desin\sim" 폴더내의 모든 파일을 추가합니다.

아래는 추가된 파일을 보여줍니다.



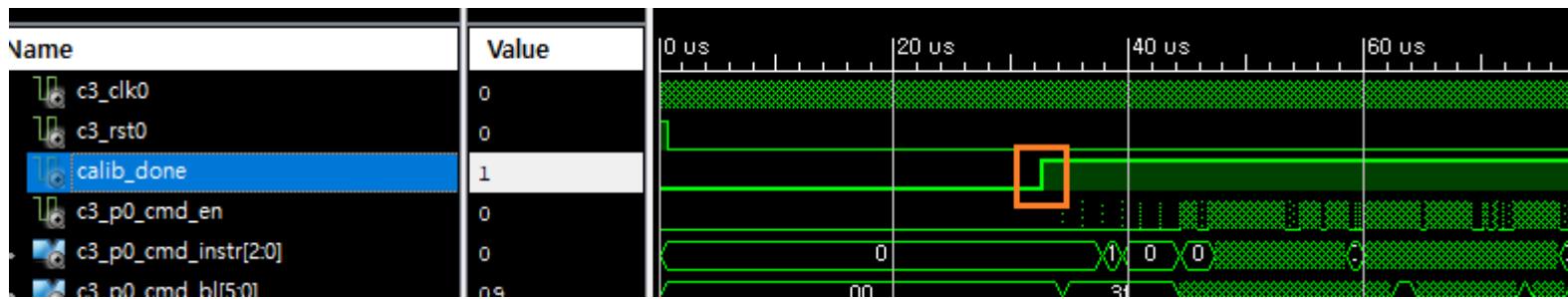
sim_tb_top을 선택하고, 하단의 ISim Simulator - Simulation Behavioral Model 을 더블 클릭해서 simulation을 진행합니다.



하단 console 윈도에 "run 0.1ms"을 입력 후 엔터키를 누르면 simulation이 진행됩니다.

9.4.1 calib_done

calibration 완료신호를 확인합니다.



32us 정도에서 active 되었습니다.

9.4.2 c3_clk0, c3_rst0

ddr3_mig (ddr controller)와 Interface를 하는 user logic에서 사용하도록 clock, reset 신호를 출력합니다. 내부 sync를 맞추기 위하여 ddr3_mig와 interface 하는 user logic은 제공되는 신호를 사용하는 것이 좋습니다. 우리는 c3_clk0 신호를 cmd_clk, wr_clk, rd_clk으로 사용합니다.

c3_clk0의 주기는 24ns, 주파수는 41.67Mhz입니다.



아래는 clock에 대한 내용입니다. cmd_clk, wr_clk, rd_clk을 동일 clock으로 사용하라는 것과, ddr3의 속도에 따라 clock 주파수를 설정하라는 내용입니다. ddr3_mig 가 제공하는 clock (c3_clk0)을 사용하면 됩니다.

The user clocks are completely asynchronous from the system and calibration clocks and therefore they can operate at any frequency dictated by the FPGA logic portion of the design. The FIFOs inside the MCB handle the necessary clock domain transfer. For best utilization of the available memory bandwidth, the user clocks should be set at or above the frequency determined by the ratio of the User Interface to the external Memory Device interface. For example:

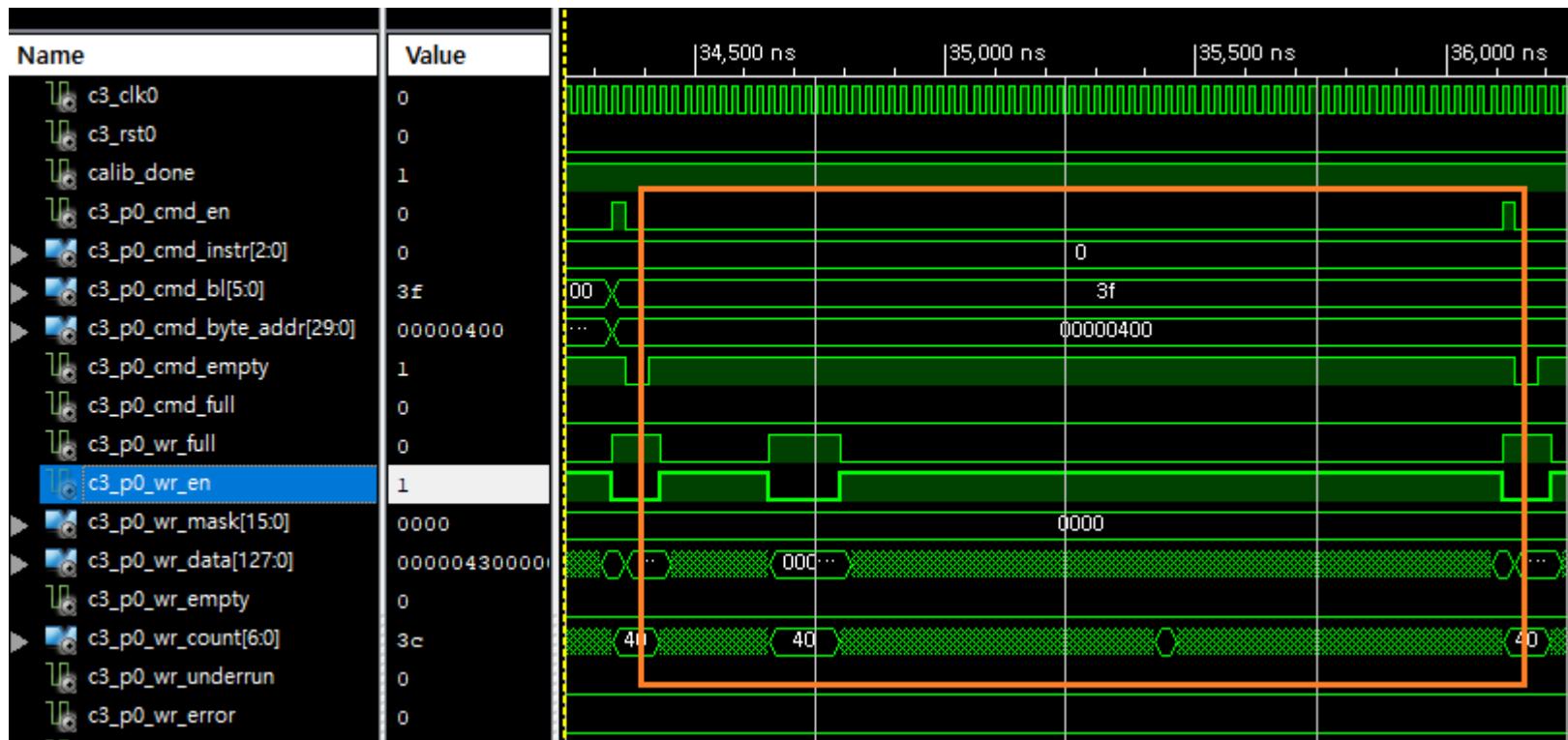
- For a DDR3 800 Mb/s interface with the memory clock = 400 MHz and a x8 bit memory device:
The result is 16 bits of data transfer per clock cycle (8 bits on each clock edge)
- For a x64 bit User interface:
The user clock should be set at or above $(16/64) * 400 \text{ MHz} = 100 \text{ MHz}$

While not technically required, it is also highly recommended that all three user clocks for a port (pX_cmd_clk, pX_wr_clk, and pX_rd_clk) be driven by the same clock source from the FPGA logic to avoid complex timing and synchronization issues in the user design.

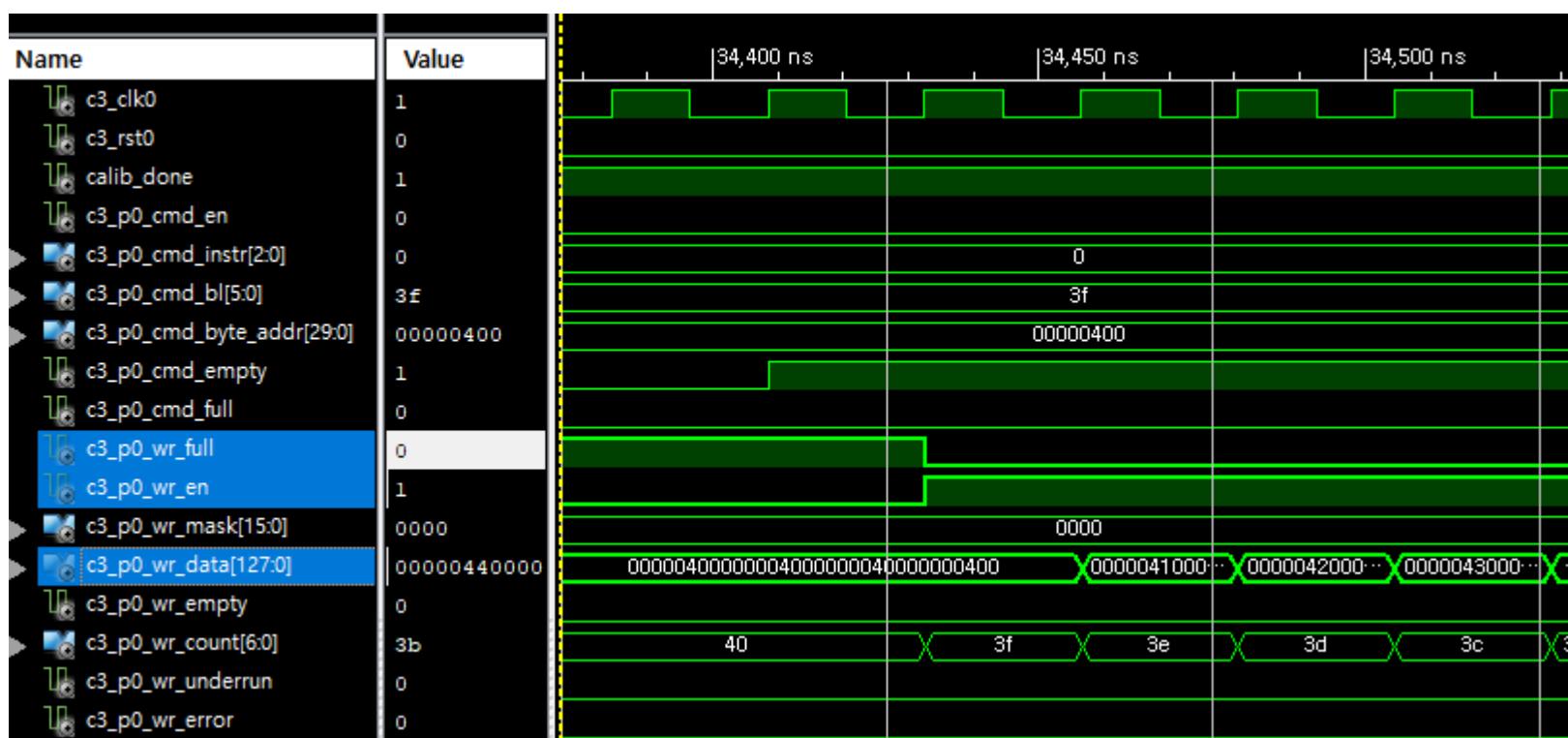
9.4.3 write timing

아래 그림은 write timing 을 보여줍니다. 0x400 번지에 64개의 데이터(128bits)를 write 합니다. 데이터는 다음과 같습니다. data : 0x00000400_00000400_00000400_00000400 ~ 0x000007f0_000007f0_000007f0_000007f0

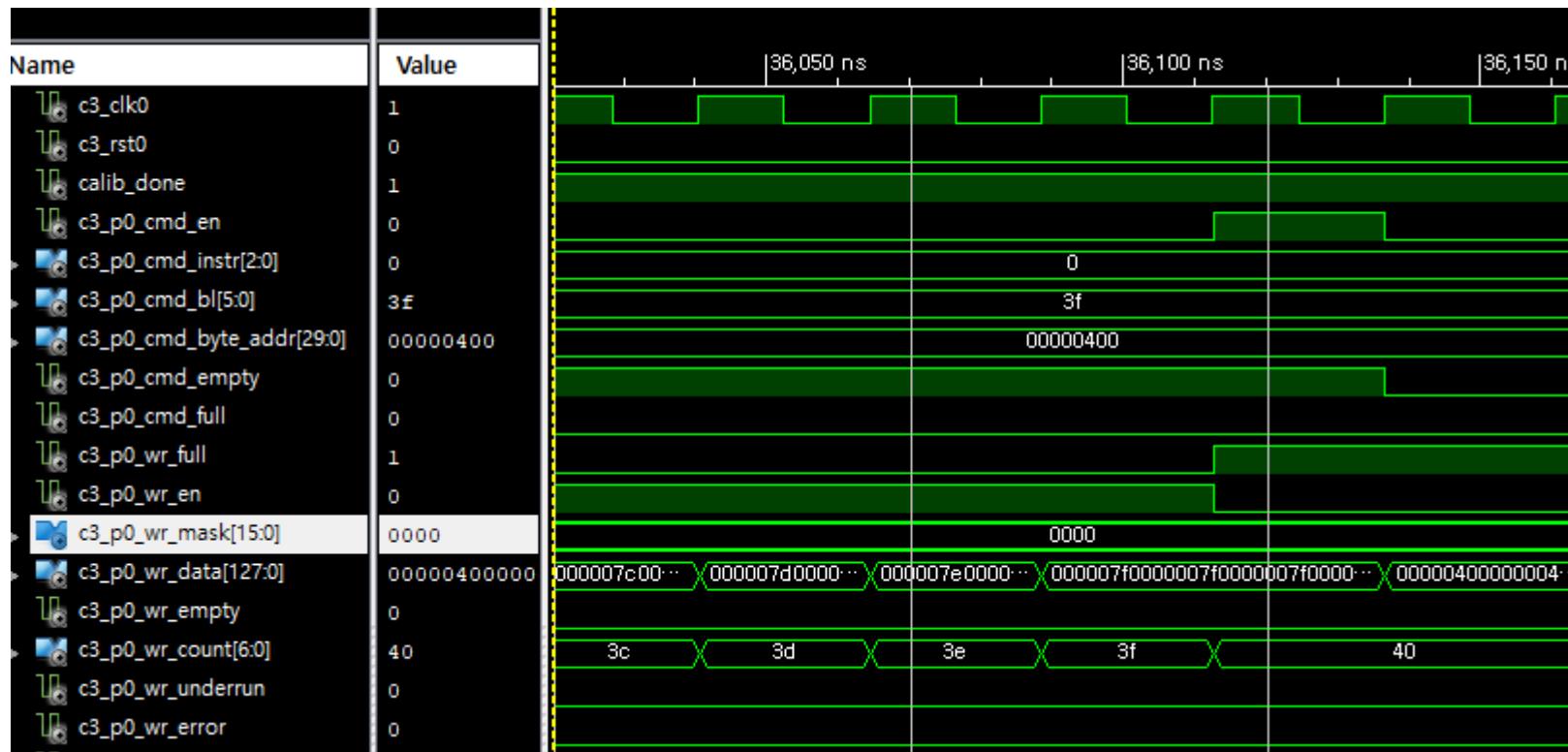
wr_full 이 0 일 때, wr_en 을 active 하고 wr_data 에 data를 write 합니다. 64개를 모두 write 한 후에, cmd_en을 active 합니다. 이 때 cmd_instr : 000b, cmd_byte_addr : 0x400, cmd_bl : 0x3f 입니다.



아래 그림은 처음 부분을 보여줍니다. wr_full 이 0 일 때, wr_en은 1이 되고, wr_data에 저장할 데이터를 넣어 줍니다.

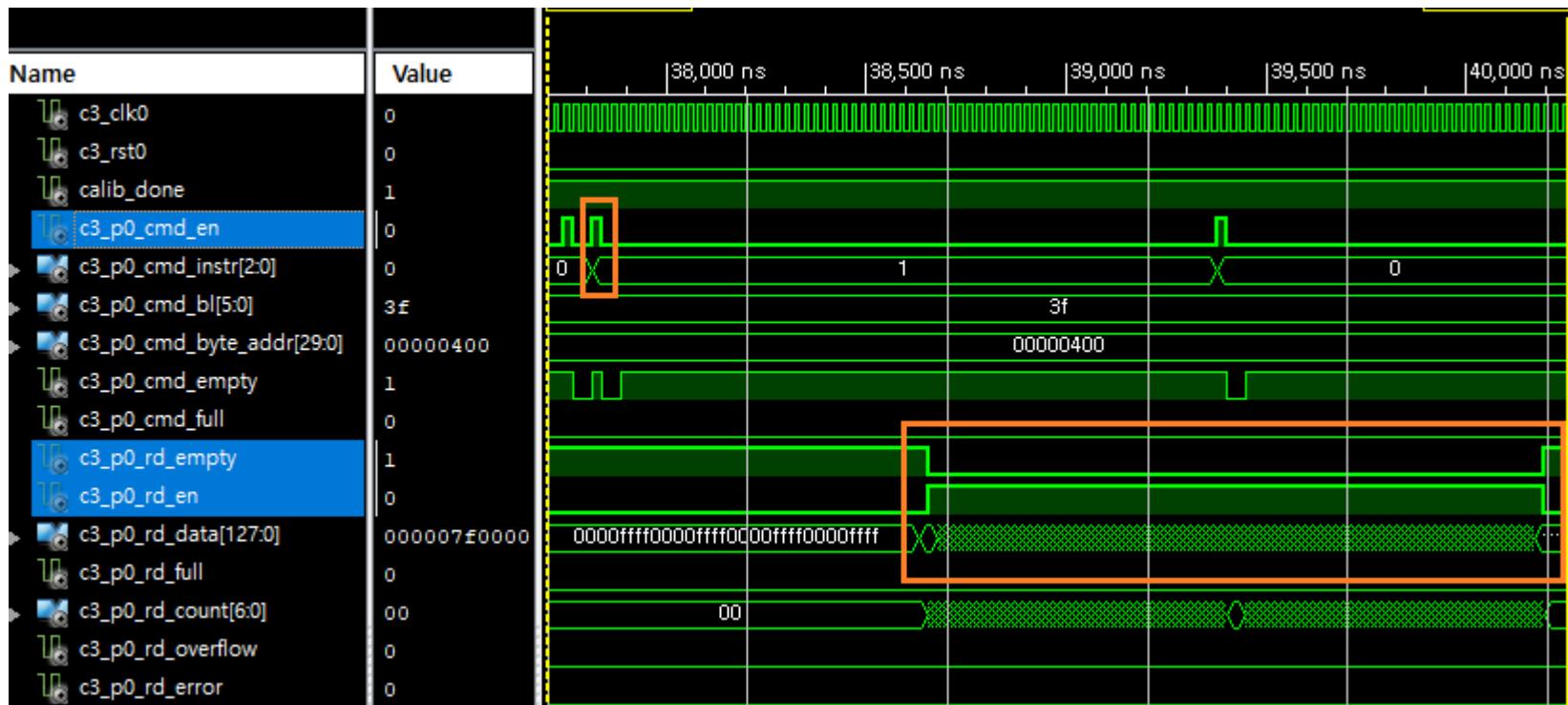


아래 그림은 64개의 마지막 부분을 보여줍니다. 0x7f0 까지 write 하고, cmd_en을 active 합니다. 이때 cmd_instr : 000b, cmd_bl : 0x3f, cmd_byte_addr : 0x400 입니다.

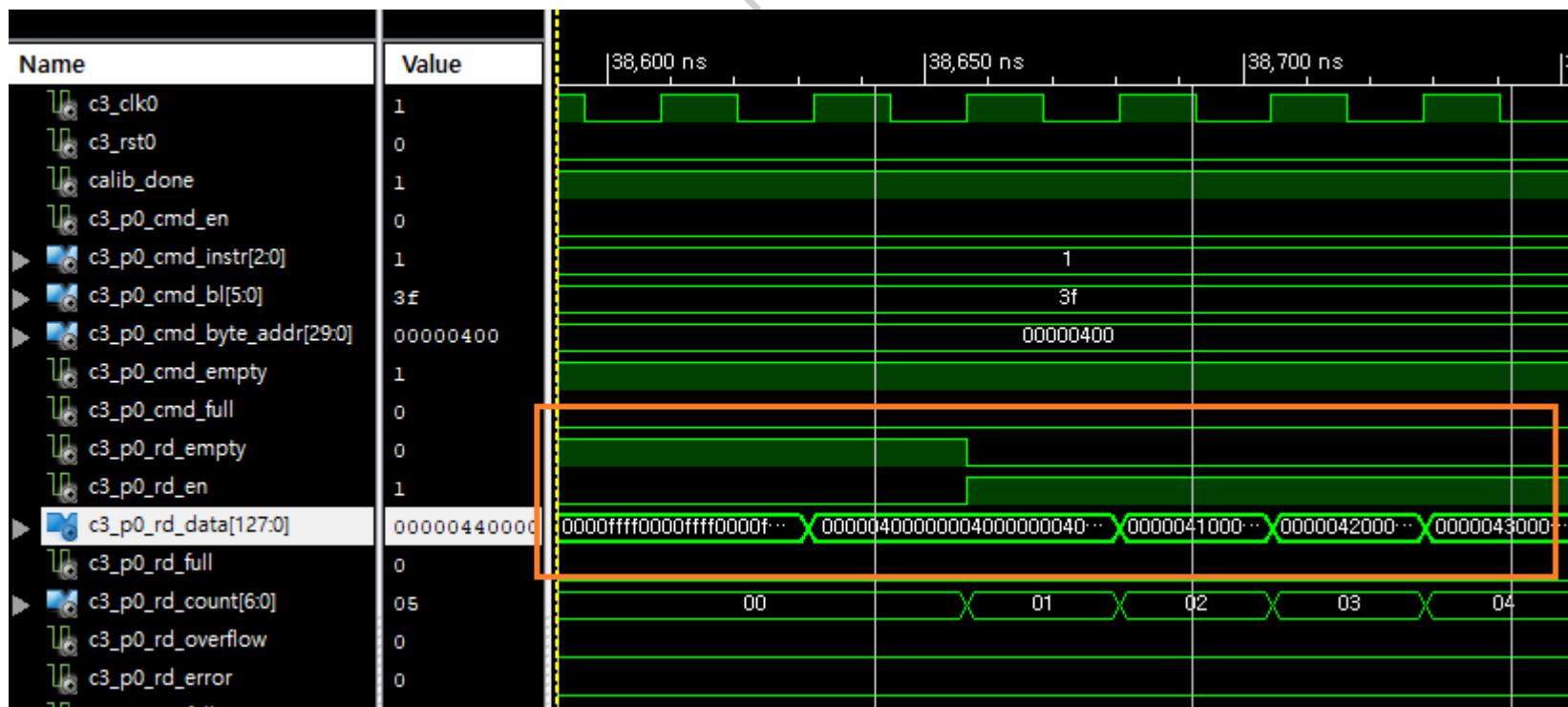


9.4.4 read timing

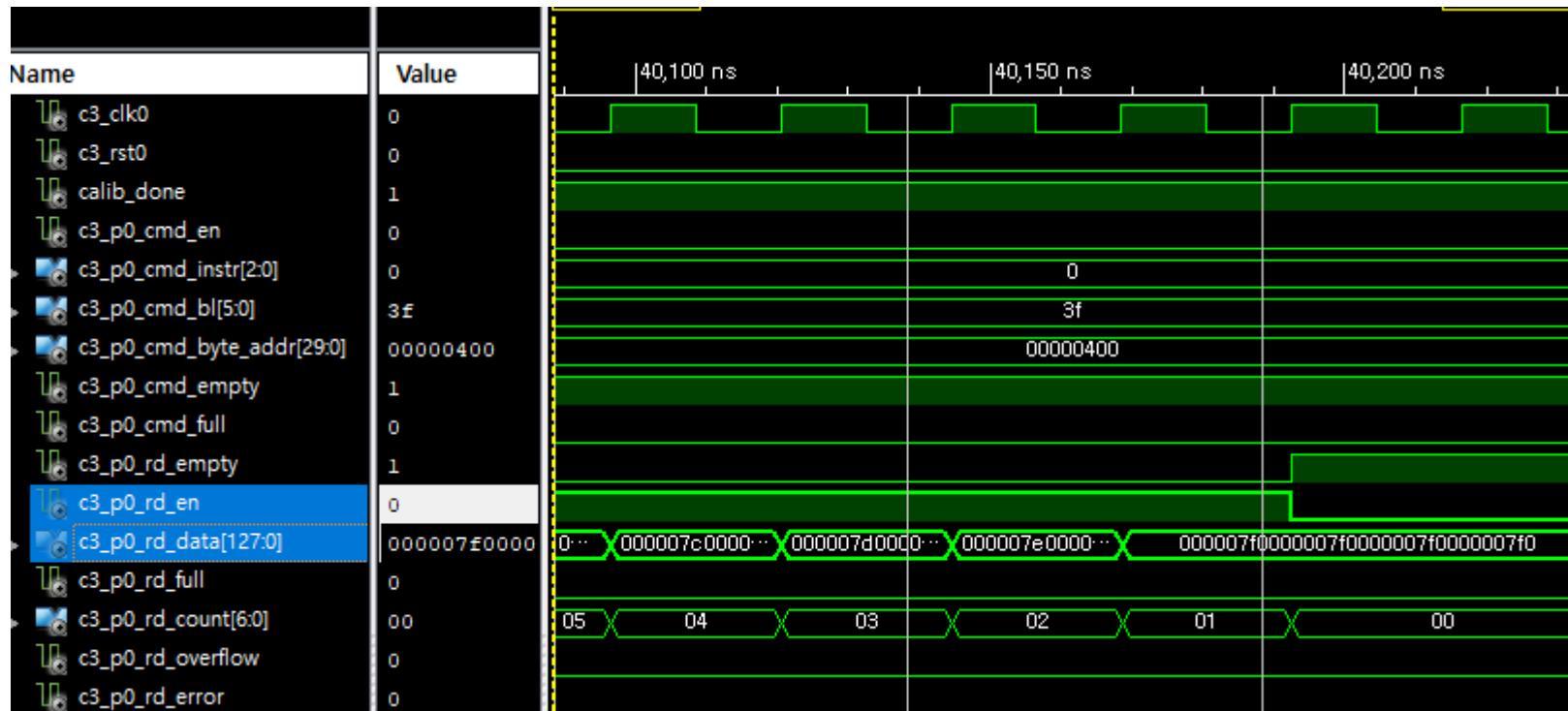
아래는 read timing을 보여줍니다. 0x400 번지에서 64개의 데이터(128bits)를 read 합니다. 먼저 read command 를 전송하고(cmd_instr : 001b, cmd_bl : 0x3f, cmd_byte_addr : 0x400), read 가 되면 (rd_empty 가 0 가 되면) rd_en을 active 로 하면, rd_data를 통하여 데이터를 출력됩니다.



아래는 read의 처음 부분을 보여줍니다. rd_en 0이 active 되면, rd_data에서 0x0000400~ 데이터가 출력됩니다.



아래 그림은 read 끝 부분을 보여줍니다. 0x000007f0 까지 데이터가 출력되는 것을 알 수 있습니다.



9.5 코드 구현

이번 장에서는 앞에서 분석한 내용을 바탕으로 코드를 구현합니다.

9.5.1 mcb_write_128x16

mcb_write_128x16은 128bits을 16개 write 하는 module입니다. 한 번에 최대 64개까지 write 할 수 있으므로 사용자가 원하는 만큼 정할 수 있습니다. 본 강의에서는 16개를 write 하는 module를 먼저 구현하고, 이를 이용해서 더 많은 데이터를 write 하는 모듈 (mcb_write)을 구현합니다.

port를 아래와 같이 정의합니다.

| signal | in/out | size | description |
|-----------|--------|---------|---|
| reset | in | [0] | reset, active low |
| clock | in | [0] | clock, c3_clk0을 사용합니다. 41.67Mhz |
| wstart | in | [0] | write start flag (user interface) |
| waddr | in | [29:0] | write address (user interface) |
| wdata | in | [127:0] | write data (user interface) |
| wready | out | [0] | write ready (user interface) |
| wdone | out | [0] | write done (user interface) |
| cmd_en | out | [0] | command enable (ddr3_mig interface, command) |
| cmd_instr | out | [2:0] | command instruction (ddr3_mig interface, command) |
| cmd_bl | out | [5:0] | command byte length (ddr3_mig interface, command) |
| cmd_addr | out | [29:0] | command address (ddr3_mig interface, command) |
| cmd_full | in | [0] | command full state flag (ddr3_mig interface, command) |
| wr_en | out | [0] | write enable (ddr3_mig interface, write) |
| wr_mask | out | [15:0] | write mask (ddr3_mig interface, write) |
| wr_data | out | [127:0] | write data (ddr3_mig interface, write) |
| wr_full | in | [0] | write full state flag (ddr3_mig interface, write) |

port는 크게 2분분으로 구성됩니다. user interface 와 ddr3_mig interface 입니다. user interface는 waddr 를 설정한 후, wstart를 active 해서 write 를 진행합니다. ddr3_mig 모듈에서 write ready (wread)가 active 되면 wdata를 전송합니다.

```

22  module mcb_write_128x16 (
23      reset      ,
24      clock      , // 41.6 Mhz
25
26      wstart     , // host
27      waddr      ,
28      wdata      ,
29      wready     ,
30      wdone      ,
31
32      cmd_en     , // ddr
33      cmd_instr  ,
34      cmd_bl     ,
35      cmd_addr   ,
36      cmd_full   ,
37
38      wr_en      , // ddr
39      wr_mask    ,
40      wr_data   ,
41      wr_full   ,
42 );
43 input      reset      ;
44 input      clock      ;
45 input      wstart     ;
46 input [29:0] waddr    ;
47 input [127:0] wdata   ;
48 output     wready    ;
49 output     wdone     ;
50
51 output     cmd_en    ;
52 output [2:0] cmd_instr ;
53 output [5:0] cmd_bl   ;
54 output [29:0] cmd_addr ;
55 input      cmd_full  ;
56
57 output     wr_en    ;
58 output [15:0] wr_mask ;
59 output [127:0] wr_data ;
60 input      wr_full  ;

```



- ✓ 라인 22 - 42 : mcb_write_128x16 module을 정의합니다.
- ✓ 라인 43 - 60 : in/out port를 정의합니다.

```

62 // State Parameter
63 parameter      M_IDLE  = 2'd0;
64 parameter      M_WFIFO = 2'd1;
65 parameter      M_WCMD  = 2'd2;
66 parameter      M_DONE  = 2'd3;
67
68
69 // -----
70 // State Control
71 reg   [1:0]  m_state;
72 wire   s_idle  = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
73 wire   s_wfifo = (m_state==M_WFIFO) ? 1'b1 : 1'b0;
74 wire   s_wcmd  = (m_state==M_WCMD ) ? 1'b1 : 1'b0;
75 wire   s_done  = (m_state==M_DONE ) ? 1'b1 : 1'b0;

```

- ✓ 라인 62 - 66 : state를 정의합니다. idle 상태에서 wstart 가 active 되면 wfifo 상태가 되고, 16개의 데이터를 write 하면 wcmd 상태가 되고, write command 전송 후 done 상태가 됩니다.

```

77  reg      [3:0]  wfifo_cnt;
78  always @(posedge clock or negedge reset)
79 begin
80      if(!reset) wfifo_cnt <= 4'b0;
81      else      wfifo_cnt <= ~s_wfifo ? 4'b0 : ~wr_full ? wfifo_cnt+1'bl : wfifo_cnt;
82 end
83
84 reg      [1:0]  wcmd_cnt;
85 always @(posedge clock or negedge reset)
86 begin
87      if(!reset) wcmd_cnt <= 2'b0;
88      else      wcmd_cnt <= ~s_wcmd ? 2'b0 : ~cmd_full ? wcmd_cnt+1'bl : wcmd_cnt;
89 end
90
91 wire      wready = (s_wfifo & ~wr_full) ? 1'bl : 1'b0;
92 wire      wdone  = s_done ;
93
94 wire      wr_en   = (s_wfifo & ~wr_full) ? 1'bl : 1'b0;
95 wire      [15:0] wr_mask = 16'b0;
96 wire      [127:0] wr_data = wdata;
97
98 wire      cmd_en   = (wcmd_cnt==2'd1) ? 1'bl : 1'b0;
99 wire      [2:0]  cmd_instr = 3'b0 ;
100 wire     [5:0]  cmd_bl   = 6'd15 ;
101 wire     [29:0] cmd_addr = waddr ;
102
103 always @(posedge clock or negedge reset)
104 begin
105 if(!reset) begin
106     m_state <= 2'b0;
107 end
108 else begin
109     m_state <= (s_idle & wstart) ? M_WFIFO :
110             (s_wfifo & (wfifo_cnt==4'd15)) ? M_WCMD :
111             (s_wcmd & (wcmd_cnt==2'd1)) ? M_DONE :
112             (s_done) ? M_IDLE :
113             m_state ;
114 end
115 end
116
117
118 endmodule

```

- ✓ 라인 77 - 82 : wr_fifo가 0 이면 데이터를 전송합니다.
- ✓ 라인 84 - 89 : cmd_full 이 0 이면 command를 전송합니다.
- ✓ 라인 91 : wready 신호를 생성합니다.
- ✓ 라인 90 : done 신호를 생성합니다.
- ✓ 라인 94 - 96 : wr_en, wr_mask, wr_data 신호를 생성합니다.
- ✓ 라인 98 - 101 : cmd_en, cmd_instr, cmd_bl, cmd_addr 을 생성합니다.
- ✓ 라인 103 - 115 : 상태 이동을 구현합니다. idle 상태에서 wstart가 active 되면 wfifo 가 됩니다. wfifo에서 data 를 16개를 write 하면, wcmd 상태가 됩니다. wcmd 상태에서 command를 전송하고 done 상태가 됩니다. done 상태에서는 다시 idle 상태가 됩니다.

9.5.2 simulation mcb_write_128x16

mcb_write_128x16을 simulation 합니다. test bench는 tb_mcb_write_128x16 입니다.

```
25 module tb_mcb_write_128x16();
26
27
28 reg reset, clock;
29 initial begin
30     reset = 0;
31     clock = 0;
32
33 #10000 reset = 1;
34 end
35
36 always #12 clock = ~clock; // 41.6Mhz
37
38
39 reg [19:0] cnt;
40 always @ (posedge clock or negedge reset)
41 begin
42     if(~reset) cnt <= 20'd0;
43     else cnt <= cnt + 1'bl;
44 end
```

- ✓ 라인 25 : tb_mcb_write_128x16 모듈 생성
- ✓ 라인 28 - 36 : reset, clock을 생성합니다. clock은 41.67 Mhz 입니다.
- ✓ 라인 39 - 44 : 신호 생성을 위하여 counter를 생성합니다.

```

46   reg          wstart;
47   always @ (posedge clock or negedge reset)
48   begin
49     if(~reset)      wstart <= 1'b0;
50     else           wstart <= (cnt==20'd1000) ? 1'b1 : (cnt==20'd1001) ? 1'b0 : wstart ;
51   end
52
53   reg      [29:0]  waddr;
54   always @ (posedge clock or negedge reset)
55   begin
56     if(~reset)      waddr <= 30'b0;
57     else           waddr <= (cnt==20'd1000) ? 30'h400 : waddr ;
58   end
59
60   wire          wready ;
61   reg      [127:0] wdata ;
62   always @ (posedge clock or negedge reset)
63   begin
64     if(~reset) wdata <= 128'b0;
65     else       wdata <= (cnt==20'd1000) ? 128'h0400_0400_0400_0400_0400_0400_0400_0400 :
66                               (wready) ? wdata + 128'h0010_0010_0010_0010_0010_0010_0010_0010 :
67                               wdata;
68   end
69
70   reg          cmd_full;
71   always @ (posedge clock or negedge reset)
72   begin
73     if(~reset)      cmd_full <= 1'b0;
74     else           cmd_full <= (cnt==20'd1000) ? 1'b0 :
75                               (cnt==20'd1014) ? 1'b1 :
76                               (cnt==20'd1040) ? 1'b0 : cmd_full ;
77   end
78
79   reg          wr_full;
80   always @ (posedge clock or negedge reset)
81   begin
82     if(~reset)      wr_full <= 1'b0;
83     else           wr_full <= (cnt==20'd1000) ? 1'b0 :
84                               (cnt==20'd1007) ? 1'b1 :
85                               (cnt==20'd1010) ? 1'b0 : wr_full ;
86   end

```

- ✓ 라인 46 - 51 : wstart 신호를 생성합니다.
- ✓ 라인 53 - 58 : waddr 신호를 생성합니다. wstart 가 active 되기 전에 설정되어야 합니다.
- ✓ 라인 60 - 68 : wdata를 생성합니다. 첫번째 wdata는 wready 신호가 생성되기 전에 미리 만들어져 있어야 합니다. wready 가 active 될 때마다 다음 데이터를 생성합니다.
- ✓ 라인 70 - 77 : test를 위하여 cmd_full 신호를 생성합니다. cmd_full 이 active 일 때에는 cmd가 전달되면 안됩니다.
- ✓ 라인 79 - 86 : test를 위하여 wr_full 신호를 생성합니다. wr_full 이 active 일 때에는 wr_en 신호가 0 가 되어야 합니다.

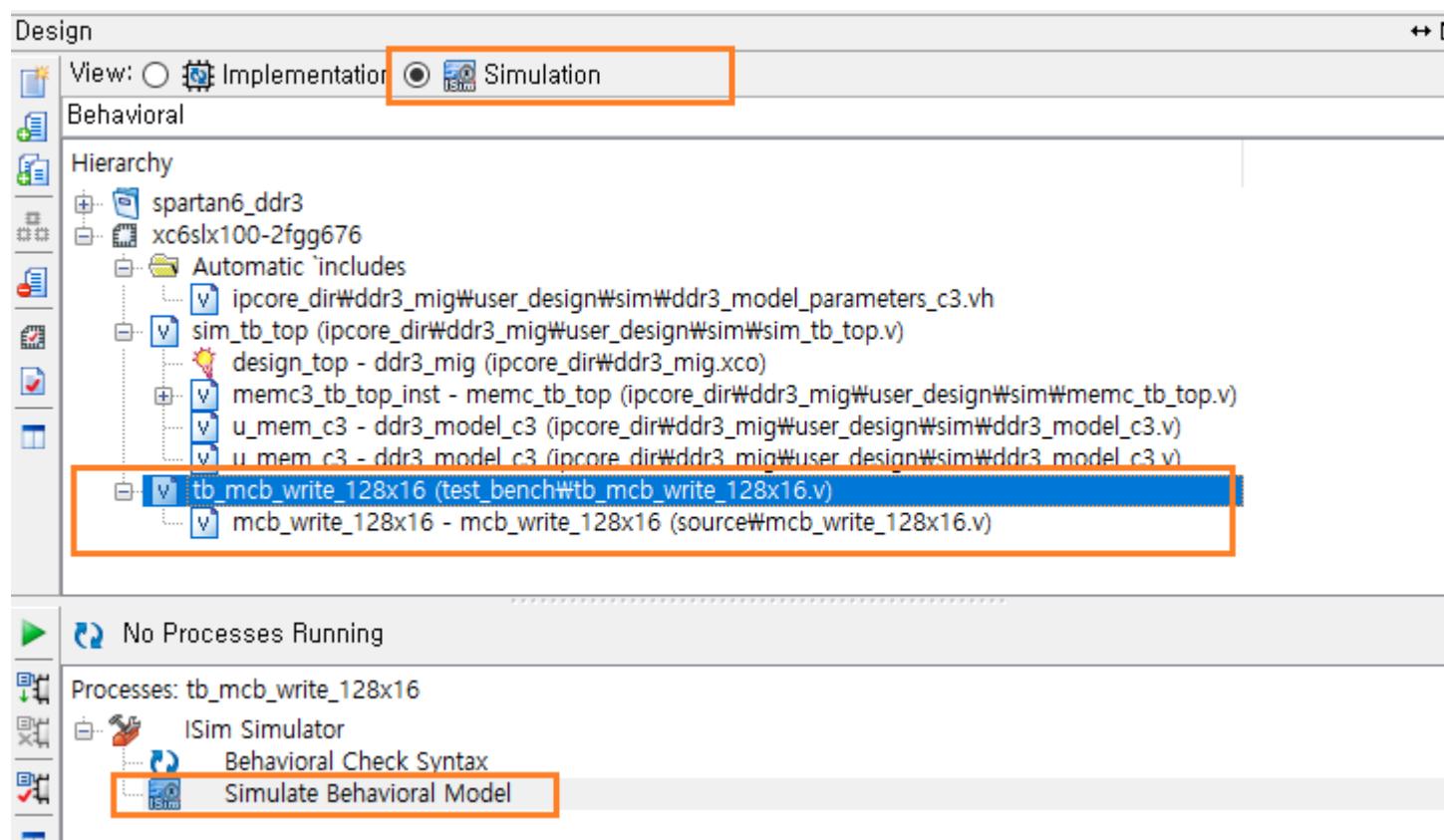
```

88   wire          wdone      ;
89   wire          cmd_en    ;
90   wire [2:0]    cmd_instr ;
91   wire [5:0]    cmd_bl    ;
92   wire [29:0]   cmd_addr  ;
93
94   wire          wr_en     ;
95   wire [15:0]   wr_mask   ;
96   wire [127:0]  wr_data   ;
97   mcb_write_128x16 mcb_write_128x16 (
98     .reset        (reset      ),
99     .clock        (clock      ),           // 41.6 Mhz
100
101    .wstart       (wstart      ),           // host
102    .waddr        (waddr      ),
103    .wdata        (wdata      ),
104    .wready       (wready      ),
105    .wdone        (wdone      ),
106
107    .cmd_en       (cmd_en      ),           // ddr
108    .cmd_instr    (cmd_instr   ),
109    .cmd_bl       (cmd_bl      ),
110    .cmd_addr    (cmd_addr   ),
111    .cmd_full    (cmd_full   ),
112
113    .wr_en       (wr_en      ),           // ddr
114    .wr_mask     (wr_mask    ),
115    .wr_data     (wr_data    ),
116    .wr_full    (wr_full   )
117  );
118
119 endmodule

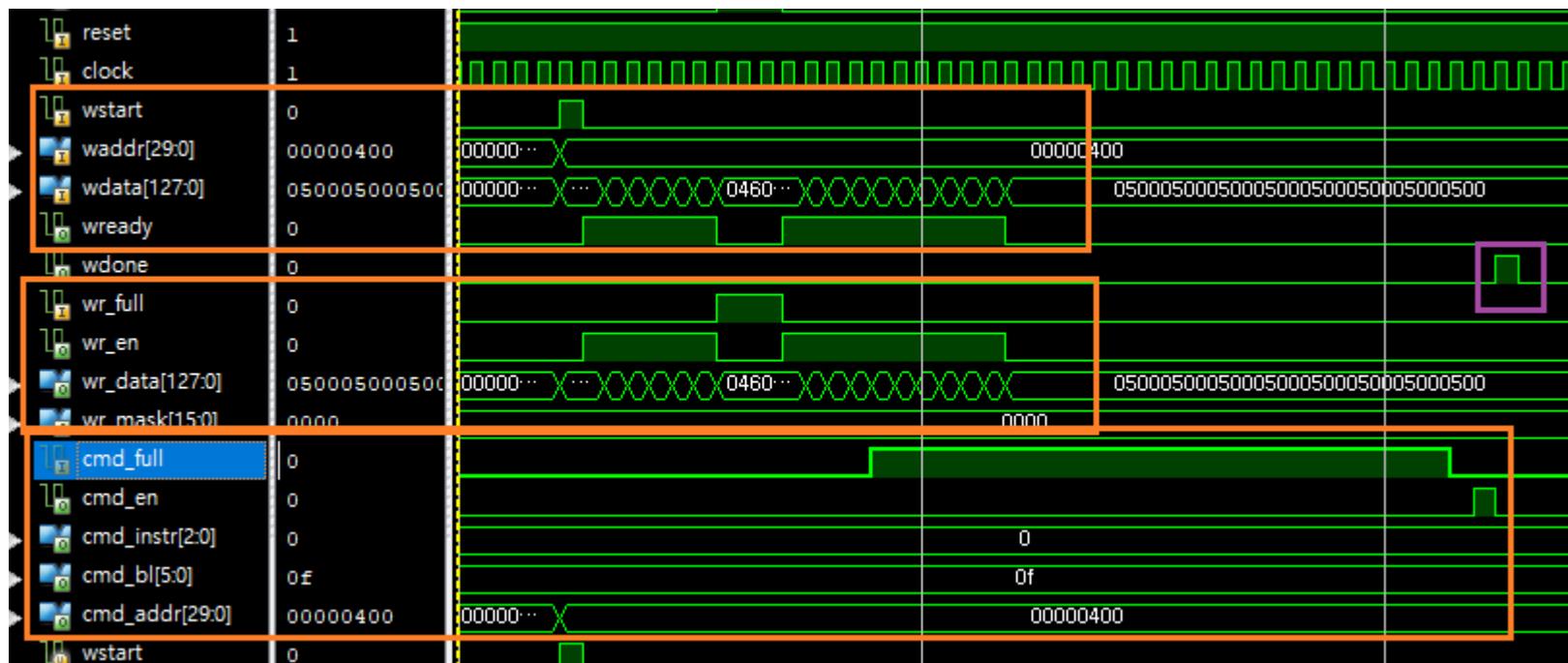
```

- ✓ 라인 88 - 117 : mcb_write_128x16 모듈을 추가합니다.

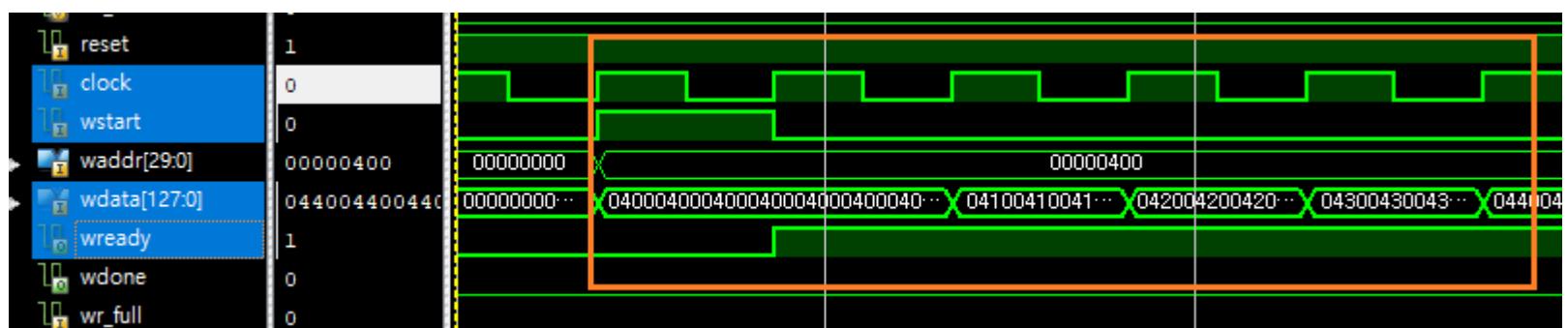
tb_mcb_write_128x16.v, mcb_write_128x16.v 파일을 simulation 파일에 추가합니다. tb_mcb_write_128x16을 선택하고, Simulate Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_write_128x16의 신호를 추가 (Add to Wave window)하고 “run 0.1ms” 입력해서 simulation을 진행합니다.



아래는 전체적인 파형을 보여줍니다. 첫번째는 waddr, 첫번째 wdata 를 생성하고, wstart 를 active 합니다. wready 신호가 active 되면 wdata를 전송합니다. 이 때 중요한 것은 wready 가 active 되기 전에 첫번째 wdata는 미리 준비가 되어야 합니다. 두번째는 wr_full 이 0 일 때, wr_en 을 active 로 만들고 wr_data를 전송합니다. 세번째는 16개 전송이 완료되면 command를 전송합니다. cmd_full 이 0 일 때까지 기다렸다가, cmd_full 이 0가 되면 cmd_en 신호를 active 합니다. 이 때 cmd_instr : 000b, cmd_bl : 0x0f, cmd_addr : 0x400 입니다.



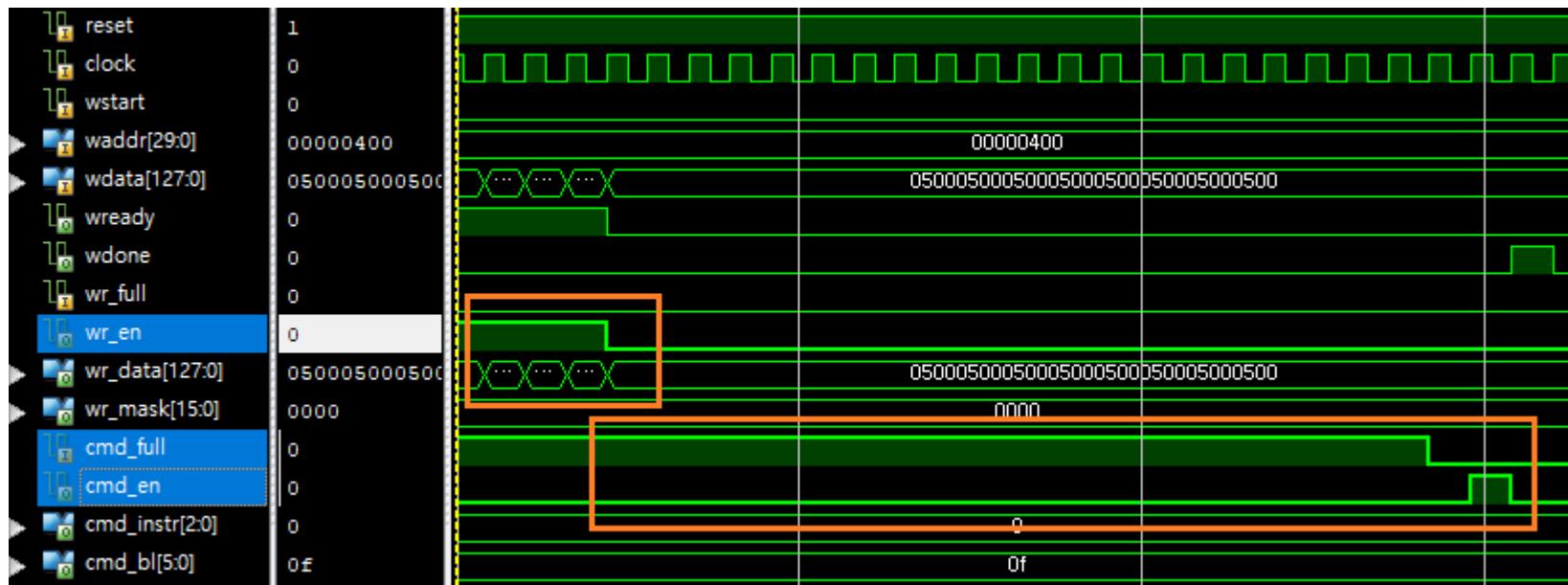
아래는 user interface (위의 그림에서 첫번째)의 시작부분을 확대해서 보여줍니다. user interface는 wready 가 active 되기 전에 미리 첫번째 write data를 준비하고 wstart를 active 해야 합니다. wready 가 active 될 때마다 다음 데이터를 전송 합니다. 아래 그림에서 400 데이터가 미리 준비 되어서 wready 가 active가 되면 바로 400 데이터가 전송되고, 그 다음에 410, 420 데이터가 순서대로 전달됩니다.



아래 그림은 wr_full 이 1일 때, wr_en은 0가 되고 wr_data는 변하지 않는 것을 보여줍니다.



아래 그림은 16개를 모두 write 한 후에, command를 전송할 때 cmd_full 이 1이므로, 0가 될 때까지 기다렸다가 0가 된 후에 cmd_en 0이 Active 되는 것을 보여줍니다.



9.5.3 mcb_write

mcb_write 모듈을 mcb_write_128x16 모듈을 사용하여 많은 양의 데이터를 write 합니다.

```

22  module mcb_write (
23      reset      ,
24      clock      , // 41.6 Mhz
25
26      mcb_wstart , // host
27      mcb_waddr  ,
28      mcb_wsiz   , // 128x16 unit
29      mcb_wdata  ,
30      mcb_wready ,
31      mcb_wdone  ,
32
33      cmd_en    , // ddr
34      cmd_instr ,
35      cmd_bl    ,
36      cmd_addr  ,
37      cmd_full  ,
38
39      wr_en    , // ddr
40      wr_mask  ,
41      wr_data  ,
42      wr_full  ,
43 );
44 input      reset      ;
45 input      clock      ;
46 input      mcb_wstart ;
47 input [29:0] mcb_waddr ;
48 input [9:0]  mcb_wsiz  ;
49 input [127:0] mcb_wdata ;
50 output     mcb_wready ;
51 output     mcb_wdone  ;
52
53 output     cmd_en    ;
54 output [2:0] cmd_instr ;
55 output [5:0]  cmd_bl   ;
56 output [29:0] cmd_addr  ;
57 input      cmd_full  ;
58
59 output     wr_en    ;
60 output [15:0] wr_mask  ;
61 output [127:0] wr_data  ;
62 input      wr_full  ;

```

- ✓ 라인 22 - 43 : mcb_write 모듈 선언
 - ✓ 라인 44 - 62 : in/out port 선언, mcb_wsiz 는 mcb_write_128x16 을 이용해서 몇 번 전송할지를 나타냅니다.
- 따라서 128x16 bits 단위입니다.

```

64 // State Parameter
65 parameter      M_IDLE  = 2'd0;
66 parameter      M_WRITE = 2'd1;
67 parameter      M_DONE  = 2'd2;
68
69
70 // -----
71 // State Control
72 reg   [1:0]  m_state;
73 wire   s_idle  = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
74 wire   s_write = (m_state==M_WRITE ) ? 1'b1 : 1'b0;
75 wire   s_done  = (m_state==M_DONE ) ? 1'b1 : 1'b0;

```

- ✓ 라인 64 - 75 : state를 정의합니다. idle, write, done 으로 구성됩니다.

```

78     wire          wdone;
79     reg           wdone_1d;
80     always @ (posedge clock or negedge reset)
81     begin
82         if (!reset)      wdone_1d <= 1'b0;
83         else            wdone_1d <= wdone ;
84     end
85
86     reg      [9:0]   block_cnt ;
87     always @ (posedge clock or negedge reset)
88     begin
89         if (!reset) block_cnt <= 10'b0;
90         else            block_cnt <= s_idle ? 10'b0 : wdone ? block_cnt + 1'b1 : block_cnt;
91     end
92
93     reg      [29:0]  waddr ;
94     always @ (posedge clock or negedge reset)
95     begin
96         if (!reset) waddr <= 30'b0;
97         else            waddr <= s_idle ? mcb_waddr : wdone_1d ? waddr + 9'h100 : waddr;
98     end
99
100    reg          wstart;
101   always @ (posedge clock or negedge reset)
102   begin
103       if (!reset) wstart <= 1'b0;
104       else            wstart <= (s_idle & mcb_wstart) ? 1'b1 :
105                           wdone_1d ? ((block_cnt==mcb_wsizE) ? 1'b0 : 1'b1) :
106                           1'b0 ;
107   end
108
109   always @ (posedge clock or negedge reset)
110   begin
111     if (!reset)      begin
112         m_state <= 2'b0;
113     end
114     else      begin
115         m_state <= (s_idle & mcb_wstart ) ? M_WRITE :
116                           (s_write & wdone_1d & (block_cnt==mcb_wsizE)) ? M_DONE :
117                           (s_done ) ? M_IDLE :
118                           m_state ;
119     end
120   end
121
122   reg          mcb_wdone;
123   always @ (posedge clock or negedge reset)
124   begin
125       if (!reset)      mcb_wdone <= 1'b0;
126       else            mcb_wdone <= s_done ;
127   end

```

- ✓ 라인 78 - 84 : wdone은 mcb_write_128x16 이 완료됨을 나타냅니다. delay를 맞추기 위하여 wdone_1d를 생성합니다.
- ✓ 라인 86 - 91 : wdone 신호마다 block_cnt를 증가합니다.
- ✓ 라인 93 - 98 : write address 를 생성합니다. 128x16 bits = 16x16 bytes 입니다. 따라서 0x100 만큼 address 가 증가합니다.
- ✓ 라인 100 - 107 : wstart 신호를 생성합니다. 처음에는 user interface에서 오는 mcb_wstart 신호를 사용하고, 그 다음부터는 wdone 신호를 사용합니다.
- ✓ 라인 109 - 120 : state 이동을 구현합니다. idle 상태에서는 mcb_wstart 신호가 active 되면 write 상태가 되고, write 상태에서는 원하는 block size (mcb_wsizE)만큼 write 되면 done 상태가 되고, done 상태에서는 다시 idle 상태가 됩니다.

- ✓ 라인 122 - 127 : mcb_rdone 신호를 생성합니다.

```
129     wire          mcb_wready ;
130     wire          cmd_en      ;
131     wire [2:0]    cmd_instr  ;
132     wire [5:0]    cmd_bl     ;
133     wire [29:0]   cmd_addr   ;
134     wire          wr_en      ;
135     wire [15:0]   wr_mask    ;
136     wire [127:0]  wr_data    ;
137     mcb_write_128x16 mcb_write_128x16 (
138         .reset      (reset      ),
139         .clock      (clock      ),           // 41.6 Mhz
140
141         .wstart     (wstart     ),           // host
142         .waddr      (waddr      ),
143         .wdata      (mcb_wdata  ),
144         .wready     (mcb_wready ),
145         .wdone      (wdone      ),
146
147         .cmd_en     (cmd_en     ),           // ddr
148         .cmd_instr  (cmd_instr  ),
149         .cmd_bl     (cmd_bl     ),
150         .cmd_addr   (cmd_addr   ),
151         .cmd_full   (cmd_full   ),
152
153         .wr_en      (wr_en      ),           // ddr
154         .wr_mask    (wr_mask    ),
155         .wr_data   (wr_data    ),
156         .wr_full   (wr_full   )
157     );
158
159
160 endmodule
```

- ✓ 라인 129 - 157 : mcb_write_128x16 모듈을 추가합니다.

9.5.4 simulation mcb_write

mcb_write 모듈을 simulation 합니다. test_bench는 tb_mcb_write 입니다. tb_mcb_write_128x16 과 거의 동일합니다.

```
25 module tb_mcb_write();
26
27
28 reg reset, clock;
29 initial begin
30     reset = 0;
31     clock = 0;
32
33 #10000 reset = 1;
34 end
35
36 always #22 clock = ~clock; // 41.67Mhz
37
38
39 reg [19:0] cnt;
40 always @ (posedge clock or negedge reset)
41 begin
42     if(~reset) cnt <= 20'd0;
43     else       cnt <= cnt + 1'b1;
44 end
```

- ✓ 라인 25 : tb_mcb_write 모듈 생성
- ✓ 라인 28 - 36 : reset, clock을 생성합니다. clock은 41.67 Mhz 입니다.
- ✓ 라인 39 - 44 : 신호 생성을 위하여 counter를 생성합니다

```

46 reg wstart;
47 always @ (posedge clock or negedge reset)
48 begin
49     if (~reset) wstart <= 1'b0;
50     else wstart <= (cnt==20'd1000) ? 1'bl : (cnt==20'd1001) ? 1'b0 : wstart ;
51 end
52
53 reg [29:0] waddr;
54 always @ (posedge clock or negedge reset)
55 begin
56     if (~reset) waddr <= 30'b0;
57     else waddr <= (cnt==20'd1000) ? 30'h400 : waddr ;
58 end
59
60
61 wire wready ;
62 reg [127:0] wdata;
63 always @ (posedge clock or negedge reset)
64 begin
65     if (~reset) wdata <= 128'b0;
66     else wdata <= (cnt==20'd1000) ? 128'h0400_0400_0400_0400_0400_0400_0400_0400 :
67                     (wready) ? wdata + 128'h0010_0010_0010_0010_0010_0010_0010_0010 :
68                     wdata;
69 end
70
71
72 reg cmd_full;
73 always @ (posedge clock or negedge reset)
74 begin
75     if (~reset) cmd_full <= 1'b0;
76     else cmd_full <= (cnt==20'd1000) ? 1'b0 :
77                     (cnt==20'd1014) ? 1'bl :
78                     (cnt==20'd1040) ? 1'b0 : cmd_full ;
79 end
80
81 reg wr_full;
82 always @ (posedge clock or negedge reset)
83 begin
84     if (~reset) wr_full <= 1'b0;
85     else wr_full <= (cnt==20'd1000) ? 1'b0 :
86                     (cnt==20'd1007) ? 1'bl :
87                     (cnt==20'd1010) ? 1'b0 : wr_full ;
88 end
89
90 wire [9:0] wsize = 10'd10;

```

- ✓ 라인 46 - 69 : wstart, waddr, wready 신호를 생성합니다.
- ✓ 라인 72 - 88 : cmd_full, wr_full 신호를 생성합니다.
- ✓ 라인 90 : wsize = 10으로 합니다.

```

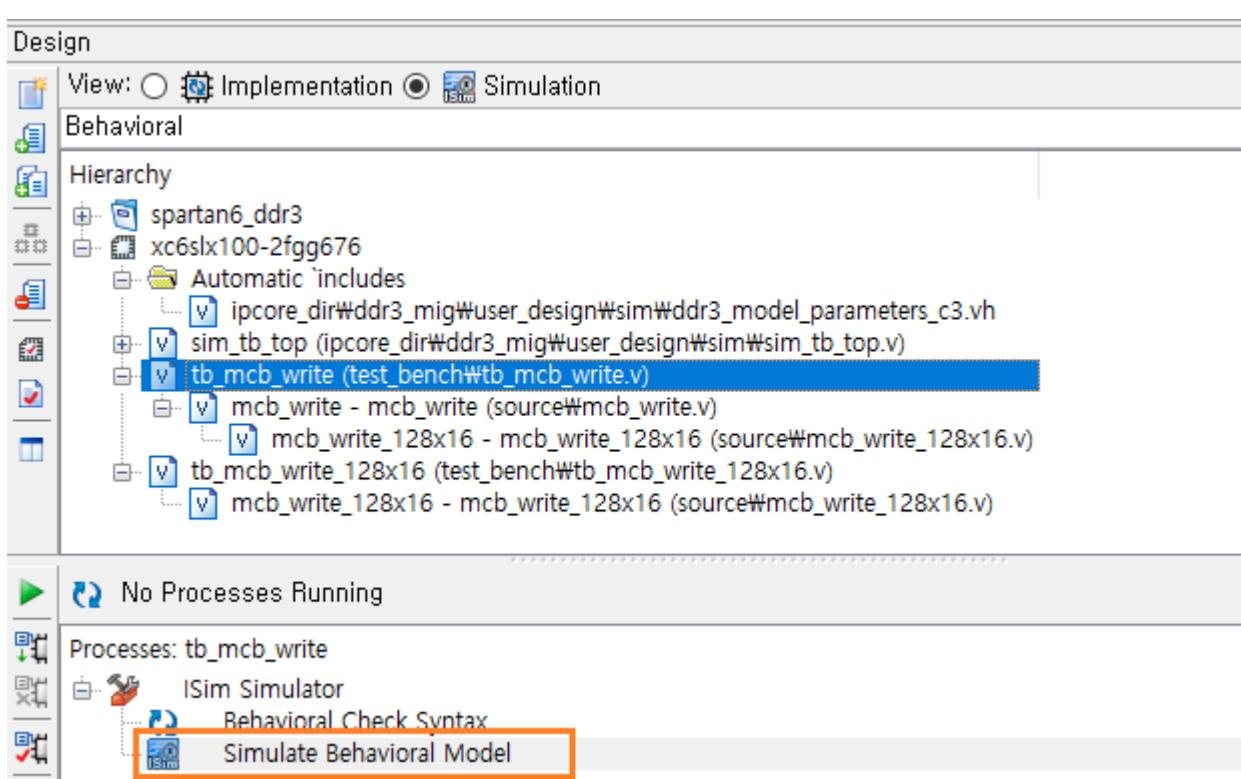
92   wire          wdone      ;
93   wire          cmd_en    ;
94   wire [2:0]    cmd_instr ;
95   wire [5:0]    cmd_bl    ;
96   wire [29:0]   cmd_addr  ;
97   wire          wr_en    ;
98   wire [15:0]   wr_mask  ;
99   wire [127:0]  wr_data  ;

100
101 mcb_write mcb_write (
102     .reset      (reset      ),
103     .clock      (clock      ),           // 41.6 Mhz
104
105     .mcb_wstart (wstart      ),           // host
106     .mcb_waddr  (waddr      ),
107     .mcb_wdata  (wdata      ),
108     .mcb_wsize  (wsize      ),
109     .mcb_wready (wready      ),
110     .mcb_wdone  (wdone      ),
111
112     .cmd_en    (cmd_en      ),           // ddr
113     .cmd_instr (cmd_instr  ),
114     .cmd_bl    (cmd_bl      ),
115     .cmd_addr  (cmd_addr  ),
116     .cmd_full  (cmd_full  ),
117
118     .wr_en    (wr_en      ),           // ddr
119     .wr_mask  (wr_mask  ),
120     .wr_data  (wr_data  ),
121     .wr_full  (wr_full  )
122 );
123
124 endmodule

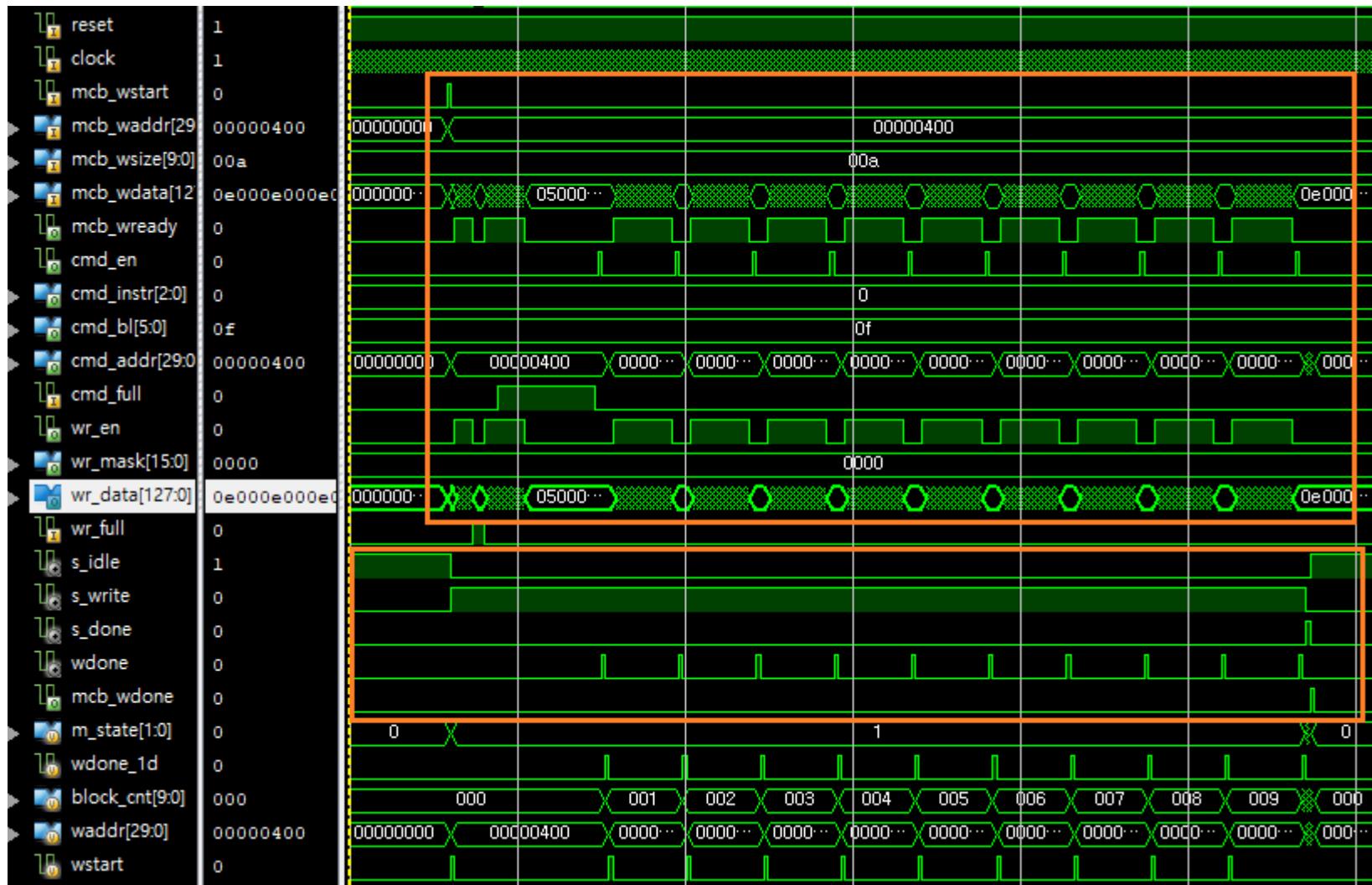
```

- ✓ 라인 92 - 122 : mcb_write 모듈을 추가합니다.

tb_mcb_write.v, mcb_write 파일을 simulation 파일에 추가합니다. tb_mcb_write 모듈을 선택하고, Simulate Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_write 신호를 추가하고 “run 0.1ms”입력해서 simulation을 진행합니다.



아래는 전체적인 파형을 보여줍니다.



mcb_wstart 가 active 되면 write를 시작합니다. state 가 idle, write, done, idle 로 이동됨을 확인할 수 있습니다. wsiz를 10으로 설정했기 때문에 mcb_write_128x160이 10번 동작합니다. wdone 신호가 총 10번 발생함을 알 수 있습니다. 자세한 사항은 직접 파형들을 확인해 보시길 바랍니다.

9.5.5 mcb_read_128x16

이번 장은 read를 구현합니다. 먼저 128x16 read를 구현합니다. port를 아래와 같이 정의합니다.

| signal | in/out | size | description |
|-----------|--------|---------|---|
| reset | in | [0] | reset, active low |
| clock | in | [0] | clock, c3_clk0을 사용합니다. 41.67Mhz |
| rstart | in | [0] | read start flag (user interface) |
| raddr | in | [29:0] | read address (user interface) |
| rdata | out | [127:0] | read data (user interface) |
| rvalid | out | [0] | read valid (user interface) |
| rdone | out | [0] | read done (user interface) |
| cmd_en | out | [0] | command enable (ddr3_mig interface, command) |
| cmd_instr | out | [2:0] | command instruction (ddr3_mig interface, command) |
| cmd_bl | out | [5:0] | command byte length (ddr3_mig interface, command) |
| cmd_addr | out | [29:0] | command address (ddr3_mig interface, command) |
| cmd_full | in | [0] | command full state flag (ddr3_mig interface, command) |
| rd_en | out | [0] | read enable (ddr3_mig interface, read) |
| rd_data | in | [127:0] | read data (ddr3_mig interface, read) |
| rd_empty | in | [0] | read empty state flag (ddr3_mig interface, read) |

port는 크게 2부분으로 구성됩니다. user interface와 ddr3_mig interface입니다. user interface는 raddr을 설정한 후, rstart를 Active 해서 read를 진행합니다. rvalid 신호가 active 되면 rdata를 read 합니다.

```

22  module mcb_read_128x16 (
23      reset      ,
24      clock      , // 41.6 Mhz
25
26      rstart     ,
27      raddr      ,
28      rdata      ,
29      rvalid     ,
30      rdone      ,
31
32      cmd_en    , // ddr
33      cmd_instr  ,
34      cmd_bl    ,
35      cmd_addr   ,
36      cmd_full   ,
37
38      rd_en     , // ddr
39      rd_data   ,
40      rd_empty   ,
41  );
42  input      reset      ;
43  input      clock      ;
44  input      rstart     ;
45  input [29:0] raddr     ;
46  output [127:0] rdata    ;
47  output      rvalid    ;
48  output      rdone     ;
49
50  output      cmd_en    ;
51  output [2:0] cmd_instr  ;
52  output [5:0] cmd_bl    ;
53  output [29:0] cmd_addr   ;
54  input       cmd_full   ;
55
56  output      rd_en     ;
57  input [127:0] rd_data   ;
58  input       rd_empty   ;

```

- ✓ 라인 22 - 41 : mcb_read_128x16 module을 정의합니다.
- ✓ 라인 42 - 58 : in/out port를 정의합니다.

```

60 // State Parameter
61 parameter      M_IDLE  = 2'd0;
62 parameter      M_RCMD = 2'd1;
63 parameter      M_RFIFO = 2'd2;
64 parameter      M_DONE  = 2'd3;
65
66
67 // -----
68 // State Control
69 reg   [1:0]  m_state;
70 wire      s_idle   = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
71 wire      s_rcmd   = (m_state==M_RCMD ) ? 1'b1 : 1'b0;
72 wire      s_rfifo  = (m_state==M_RFIFO ) ? 1'b1 : 1'b0;
73 wire      s_done   = (m_state==M_DONE ) ? 1'b1 : 1'b0;

```

- ✓ 라인 60 - 64 : state를 정의합니다. idle 상태에서 rstart가 active 되면 rcmd 상태가 되고, read command 전송 후에 rfifo 상태가 됩니다. read data를 모두 읽은 후에 done 상태가 됩니다.

```

76  reg      [1:0]   rcmd_cnt;
77  always @ (posedge clock or negedge reset)
78 begin
79      if (!reset) rcmd_cnt <= 2'b0;
80      else       rcmd_cnt <= ~s_rcmd ? 2'b0 : ~cmd_full ? rcmd_cnt+1'bl : rcmd_cnt;
81 end
82
83
84 reg      [3:0]   rfifo_cnt;
85 always @ (posedge clock or negedge reset)
86 begin
87     if (!reset) rfifo_cnt <= 4'b0;
88     else       rfifo_cnt <= ~s_rfifo ? 4'b0 : ~rd_empty ? rfifo_cnt+1'bl : rfifo_cnt;
89 end
90
91 wire      rvalid  = (s_rfifo & ~rd_empty) ? 1'bl : 1'b0;
92 wire      rdone   = s_done ;
93
94 wire      rd_en   = (s_rfifo & ~rd_empty) ? 1'bl : 1'b0;
95 wire      [127:0] rdata   = rd_data;
96
97 wire      cmd_en   = (rcmd_cnt==2'd1) ? 1'bl : 1'b0;
98 wire      [2:0]    cmd_instr = 3'b1 ;
99 wire      [5:0]    cmd_bl   = 6'd15 ;
100 wire     [29:0]   cmd_addr = raddr ;
101
102
103 always @ (posedge clock or negedge reset)
104 begin
105 if (!reset)
106     begin
107         m_state <= 2'b0;
108     end
109     else
110         begin
111             m_state <= (s_idle & restart) ? M_RCMD :
112                 (s_rcmd & (rcmd_cnt==2'd1)) ? M_RFIFO :
113                 (s_rfifo & (rfifo_cnt==4'd15)) ? M_DONE :
114                     (s_done) ? M_IDLE :
115                         m_state ;
116         end
117     end
118 endmodule

```

- ✓ 라인 76 - 81 : read command 전송을 위한 counter를 생성합니다. cmd_full이 0일 때에만 증가합니다. rcmd_cmt 값이 1일 때 read command를 전송합니다. (97 라인)
- ✓ 라인 84 - 89 : read data counter를 생성합니다. 16개의 data를 읽으면 완료됩니다. (111 라인)
- ✓ 라인 91 - 95 : rvalid, rdoen, rd_en, rdata 신호를 생성합니다. 특히 delay를 없애기 위하여 register를 사용하지 않고 wire로 처리합니다.
- ✓ 라인 97 - 100 : cmd_en, cmd_instr, cmd_bl, cmd_addr 값을 생성합니다.
- ✓ 라인 103 - 115 : state 이동을 구현합니다. idle 상태에서 restart가 active 되면 rcmd로 이동하고, rcmd에서 read command 전송후에 rfifo로 이동합니다. rfifo에서는 16개의 data를 read 한 후에 done 상태가 되고, done에서 다시 idle 상태가 됩니다.

9.5.6 simulation mcb_read_128x16

mcb_read_128x16을 simulation 합니다. test bench는 tb_mcb_read_128x16 입니다.

```
25 ⊜ module tb_mcb_read_128x16();
26
27
28     reg             reset, clock;
29 ⊜ initial         begin
30         reset = 0;
31         clock = 0;
32
33 #10000  reset = 1;
34 end
35
36 always #12      clock = ~clock;           // 41.67Mhz
37
38
39 reg      [19:0]  cnt;
40 always @ (posedge clock or negedge reset)
41 ⊜ begin
42     if(~reset)    cnt <= 20'd0;
43     else          cnt <= cnt + 1'bl;
44 end
```

- ✓ 라인 25 : tb_mcb_read_128x16 모듈 생성
- ✓ 라인 28 - 36 : reset, clock을 생성합니다. clock은 41.67 Mhz 입니다.
- ✓ 라인 39 - 44 : 신호 생성을 위하여 counter를 생성합니다



```

46  reg          rstart;
47  always @ (posedge clock or negedge reset)
48 begin
49      if(~reset) rstart <= 1'b0;
50      else      rstart <= (cnt==20'd1000) ? 1'bl : (cnt==20'd1001) ? 1'b0 : rstart ;
51 end
52
53 reg      [29:0] raddr;
54 always @ (posedge clock or negedge reset)
55 begin
56     if(~reset)      raddr <= 30'b0;
57     else           raddr <= (cnt==20'd1000) ? 30'h400 : raddr ;
58 end
59
60
61 reg      [127:0] rd_data;
62 always @ (posedge clock or negedge reset)
63 begin
64     if(~reset)      rd_data <= 128'b0;
65     else           rd_data <= rd_data + 1'bl;
66 end
67
68
69 reg          cmd_full;
70 always @ (posedge clock or negedge reset)
71 begin
72     if(~reset)      cmd_full <= 1'b0;
73     else           cmd_full <= (cnt==20'd1000) ? 1'b0 :
74                                     (cnt==20'd1014) ? 1'bl :
75                                     (cnt==20'd1040) ? 1'b0 : cmd_full ;
76 end
77
78 reg          rd_empty;
79 always @ (posedge clock or negedge reset)
80 begin
81     if(~reset)      rd_empty <= 1'bl;
82     else           rd_empty <= (cnt==20'd1000) ? 1'bl :
83                                     (cnt==20'd1100) ? 1'b0 :
84                                     (cnt==20'd1200) ? 1'bl : rd_empty ;
85 end

```

- ✓ 라인 46 - 51 : rstart 신호를 생성합니다.
- ✓ 라인 53 - 58 : raddr을 생성합니다.
- ✓ 라인 61 - 66 : test를 위하여 rd_data 를 임의의 값으로 생성합니다.
- ✓ 라인 69 - 85 : test를 위하여 cmd_full, rd_empty 를 임의로 생성합니다.

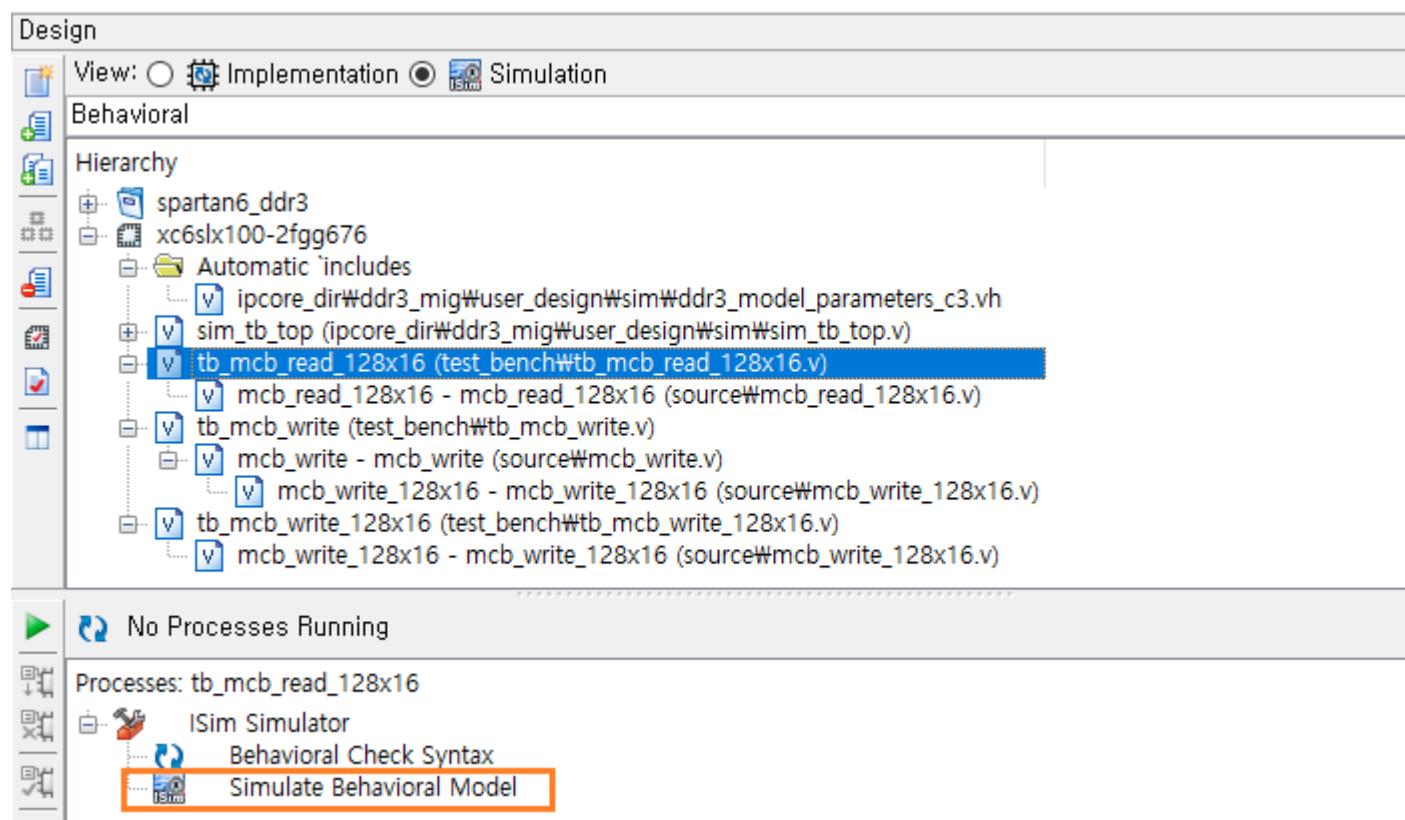
```

87   wire      cmd_en      ;
88   wire      [2:0]  cmd_instr  ;
89   wire      [5:0]  cmd_bl    ;
90   wire      [29:0] cmd_addr  ;
91
92   wire      [127:0] rdata      ;
93   wire      rvalid      ;
94   wire      rdone       ;
95   wire      rd_en       ;
96   mcb_read_128x16 mcb_read_128x16 (
97     .reset      (reset      ),
98     .clock      (clock      ),           // 41.6 Mhz
99
100    .rstart     (rstart     ),           // host
101    .raddr      (raddr      ),
102    .rdata      (rdata      ),
103    .rvalid     (rvalid     ),
104    .rdone      (rdone      ),
105
106    .cmd_en     (cmd_en     ),           // ddr
107    .cmd_instr  (cmd_instr  ),
108    .cmd_bl    (cmd_bl    ),
109    .cmd_addr  (cmd_addr  ),
110    .cmd_full  (cmd_full  ),
111
112    .rd_en     (rd_en     ),           // ddr
113    .rd_data   (rd_data   ),
114    .rd_empty  (rd_empty  )
115 );
116
117 endmodule

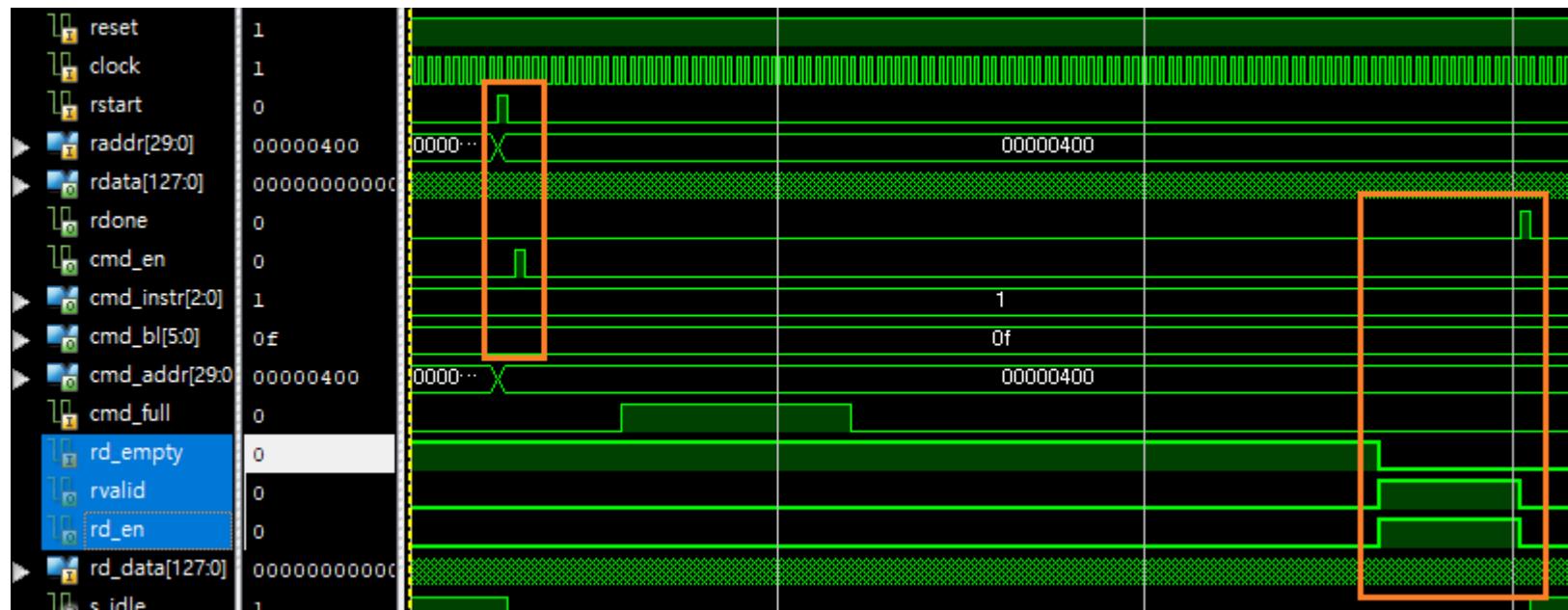
```

- ✓ 라인 87 - 115 : mcb_read_128x16 모듈을 추가합니다.

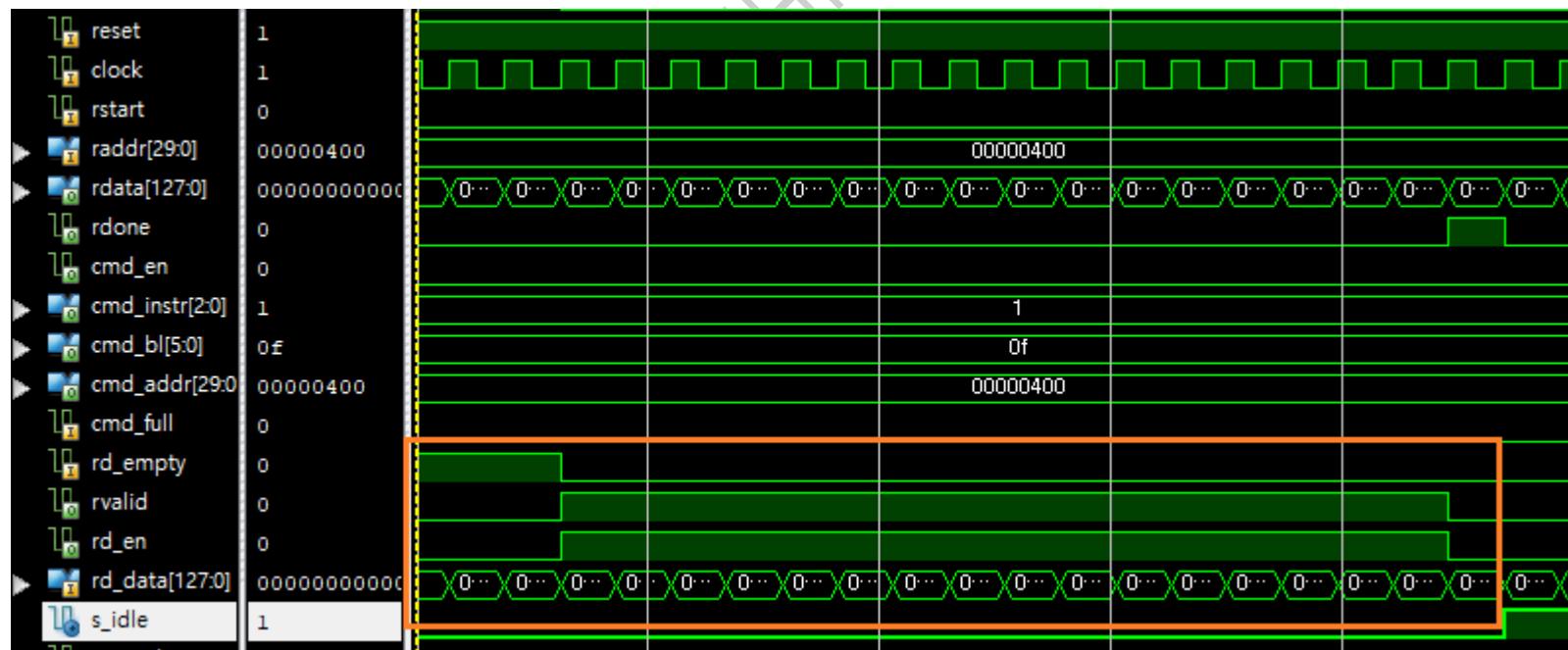
tb_mcb_read_128x16.v, mcb_read_128x16.v 파일을 simulation 파일에 추가합니다. tb_mcb_read_128x16을 선택하고, Simulation Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_read_128x16 신호를 추가하고, “run 0.1ms” 입력하여 simulation을 진행합니다.



아래는 전체적인 파형을 보여줍니다. 먼저 rstart가 active 되면 read command가 전송됩니다. cmd_en : 1, cmd_instr : 1, cmd_bl : 0x0f, cmd_addr : 0x400 입니다. 그리고 rd_empty 가 0가 되면 그 때부터 data를 read 합니다. user interface에게 data를 읽어가라는 rvalid 신호를 active로 만들어 주고, 동시에 rd_en 신호를 1로 만들어 주면서 data를 read 합니다. 16개의 data가 모두 read 되면, rdone 신호가 active 됩니다.



아래 그림은 data read 부분을 확대해서 보여줍니다. rd_empty가 0가 되면 바로 rvalid, rd_en 신호가 active 됩니다. 우리는 이 부분을 register로 처리하지 않고 wire로 처리함으로 불필요한 delay를 만들지 않습니다.



9.5.7 mcb_read

mcb_read 모듈은 mcb_read_128x16을 이용하여 많은 데이터를 read 합니다.

```

22 module mcb_read (
23   reset      ,
24   clock      // 41.6 Mhz
25
26   mcb_rstart , // host
27   mcb_raddr  ,
28   mcb_rsize  // 128x16 unit
29   mcb_rdata  ,
30   mcb_rvalid ,
31   mcb_rdone ,
32
33   cmd_en     // ddr
34   cmd_instr  ,
35   cmd_bl     ,
36   cmd_addr   ,
37   cmd_full   ,
38
39   rd_en      // ddr
40   rd_data   ,
41   rd_empty  ;
42 );
43 input      reset      ;
44 input      clock      ;
45 input      mcb_rstart ;
46 input [29:0] mcb_raddr ;
47 input [9:0]  mcb_rsize ;
48 output [127:0] mcb_rdata ;
49 output      mcb_rvalid ;
50 output      mcb_rdone ;
51
52 output      cmd_en   ;
53 output [2:0] cmd_instr ;
54 output [5:0] cmd_bl   ;
55 output [29:0] cmd_addr ;
56 input       cmd_full  ;
57
58 output      rd_en   ;
59 input [127:0] rd_data ;
60 input       rd_empty ;

```

- ✓ 라인 22 - 42 : mcb_read 모듈 선언
 - ✓ 라인 43 - 60 : in/out port 선언, mcb_rsize는 mcb_read_128x16을 이용하여 몇 번 read 할지를 나타냅니다.
- 따라서 128x16 bits 단위입니다.

```

62 // State Parameter
63 parameter      M_IDLE      = 2'd0;
64 parameter      M_READ     = 2'd1;
65 parameter      M_DONE     = 2'd2;
66
67
68 // -----
69 // State Control
70 reg   [1:0]  m_state;
71 wire    s_idle = (m_state==M_IDLE) ? 1'b1 : 1'b0;
72 wire    s_read = (m_state==M_READ) ? 1'b1 : 1'b0;
73 wire    s_done = (m_state==M_DONE) ? 1'b1 : 1'b0;

```

- ✓ 라인 62 - 65 : state를 정의합니다. idle, read, done 으로 구성됩니다.

```

75   wire          rdone;
76   reg           rdone_1d;
77   always @(posedge clock or negedge reset)
78   begin
79     if(!reset)    rdone_1d <= 1'b0;
80     else         rdone_1d <= rdone ;
81   end
82
83   reg [9:0]    block_cnt ;
84   always @(posedge clock or negedge reset)
85   begin
86     if(!reset)  block_cnt <= 10'b0;
87     else        block_cnt <= s_idle ? 10'b0 : rdone ? block_cnt + 1'b1 : block_cnt;
88   end
89
90   reg [29:0]   raddr ;
91   always @(posedge clock or negedge reset)
92   begin
93     if(!reset)  raddr <= 30'b0;
94     else        raddr <= s_idle ? mcb_raddr : rdone_1d ? raddr + 9'h100 : raddr ;
95   end
96
97   reg          rstart;
98   always @(posedge clock or negedge reset)
99   begin
100    if(!reset) rstart <= 1'b0;
101    else       rstart <= (s_idle & mcb_rstart ) ? 1'b1 :
102                  rdone_1d ? ((block_cnt==mcb_rsize) ? 1'b0 : 1'b1) : 1'b0 ;
103  end
104
105
106  always @(posedge clock or negedge reset)
107 begin
108   if(!reset)  begin
109     m_state <= 2'b0;
110   end
111   else  begin
112     m_state <= (s_idle & mcb_rstart ) ? M_READ : 
113                 (s_read & rdone_1d & (block_cnt==mcb_rsize)) ? M_DONE : 
114                 (s_done ) ? M_IDLE : m_state ;
115   end
116 end
117
118 reg          mcb_rdone;
119 always @(posedge clock or negedge reset)
120 begin
121   if(!reset)  mcb_rdone <= 1'b0;
122   else        mcb_rdone <= s_done ;
123 end

```

- ✓ 라인 75 - 81 : rdone은 mcb_read_128x16 이 완료됨을 나타냅니다. delay를 맞추기 위하여 rdone_1d를 생성 합니다.
- ✓ 라인 83 - 88 : rdone 신호마다 block_cnt를 증가합니다.
- ✓ 라인 90 - 95 : read address 를 생성합니다. 128x16 bits = 16x16 bytes 입니다. 따라서 0x100 만큼 address가 증가합니다.
- ✓ 라인 97 - 103 : rstart 신호를 생성합니다. 처음에는 user interface에서 오는 mcb_rstart 신호를 사용하고, 그 다음부터는 rdone 신호를 사용합니다.
- ✓ 라인 106 - 116 : state 이동을 구현합니다. idle 상태에서는 mcb_rstart 신호가 active 되면 read 상태가 되고, read 상태에서는 원하는 block size (mcb_rsize)만큼 read 되면 done 상태가 되고, done 상태에서는 다시 idle 상태가 됩니다.

- ✓ 라인 118 - 123 : mcb_rdone 신호를 생성합니다

```
125 wire      [127:0] mcb_rdata    ;
126 wire      mcb_rvalid   ;
127 wire      cmd_en      ;
128 wire      [2:0]  cmd_instr   ;
129 wire      [5:0]  cmd_bl     ;
130 wire      [29:0] cmd_addr   ;
131 wire      rd_en      ;
132 mcb_read_128x16 mcb_read_128x16 (
133     .reset      (reset      ),           // 41.6 Mhz
134     .clock      (clock      ),           //
135     .rstart      (rstart      ),           // host
136     .raddr      (raddr      ),           //
137     .rdata      (mcb_rdata  ),           //
138     .rvalid      (mcb_rvalid ),           //
139     .rdone      (rdone      ),           //
140
141     .cmd_en      (cmd_en      ),           // ddr
142     .cmd_instr   (cmd_instr   ),           //
143     .cmd_bl      (cmd_bl      ),           //
144     .cmd_addr   (cmd_addr   ),           //
145     .cmd_full   (cmd_full   ),           //
146
147     .rd_en      (rd_en      ),           // ddr
148     .rd_data    (rd_data    ),           //
149     .rd_empty   (rd_empty   )
150 );
151 );
152
153
154 endmodule
```

- ✓ 라인 125 - 151 : mcb_read_128x16 모듈을 추가합니다.

9.5.8 simulation mcb_read

mcb_read 모듈을 simulation 합니다. test_bench는 tb_mcb_read 입니다. 내용은 tb_mcb_read_128x16과 거의 동일 합니다.

```

25 module tb_mcb_read();
26
27 reg reset, clock;
28 initial begin
29     reset = 0;
30     clock = 0;
31
32 #10000 reset = 1;
33 end
34
35 always #12 clock = ~clock; // 41.67Mhz
36
37
38 reg [19:0] cnt;
39 always @ (posedge clock or negedge reset)
40 begin
41     if(~reset) cnt <= 20'd0;
42     else       cnt <= cnt + 1'bl;
43 end
44
45 reg rstart;
46 always @ (posedge clock or negedge reset)
47 begin
48     if(~reset) rstart <= 1'b0;
49     else       rstart <= (cnt==20'd1000) ? 1'bl : (cnt==20'd1001) ? 1'b0 : rstart ;
50 end
51
52 reg [29:0] raddr;
53 always @ (posedge clock or negedge reset)
54 begin
55     if(~reset) raddr <= 30'b0;
56     else       raddr <= (cnt==20'd1000) ? 30'h400 : raddr ;
57 end
58
59 reg [127:0] rd_data;
60 always @ (posedge clock or negedge reset)
61 begin
62     if(~reset) rd_data <= 128'b0;
63     else       rd_data <= rd_data + 1'bl;
64 end

```

- ✓ 라인 25 : tb_mcb_read 모듈 선언
- ✓ 라인 27 - 35 : reset, clock 생성
- ✓ 라인 38 - 43 : 신호 생성을 위한 counter 생성
- ✓ 라인 45 - 64 : rstart, raddr, rd_data를 생성합니다.

```

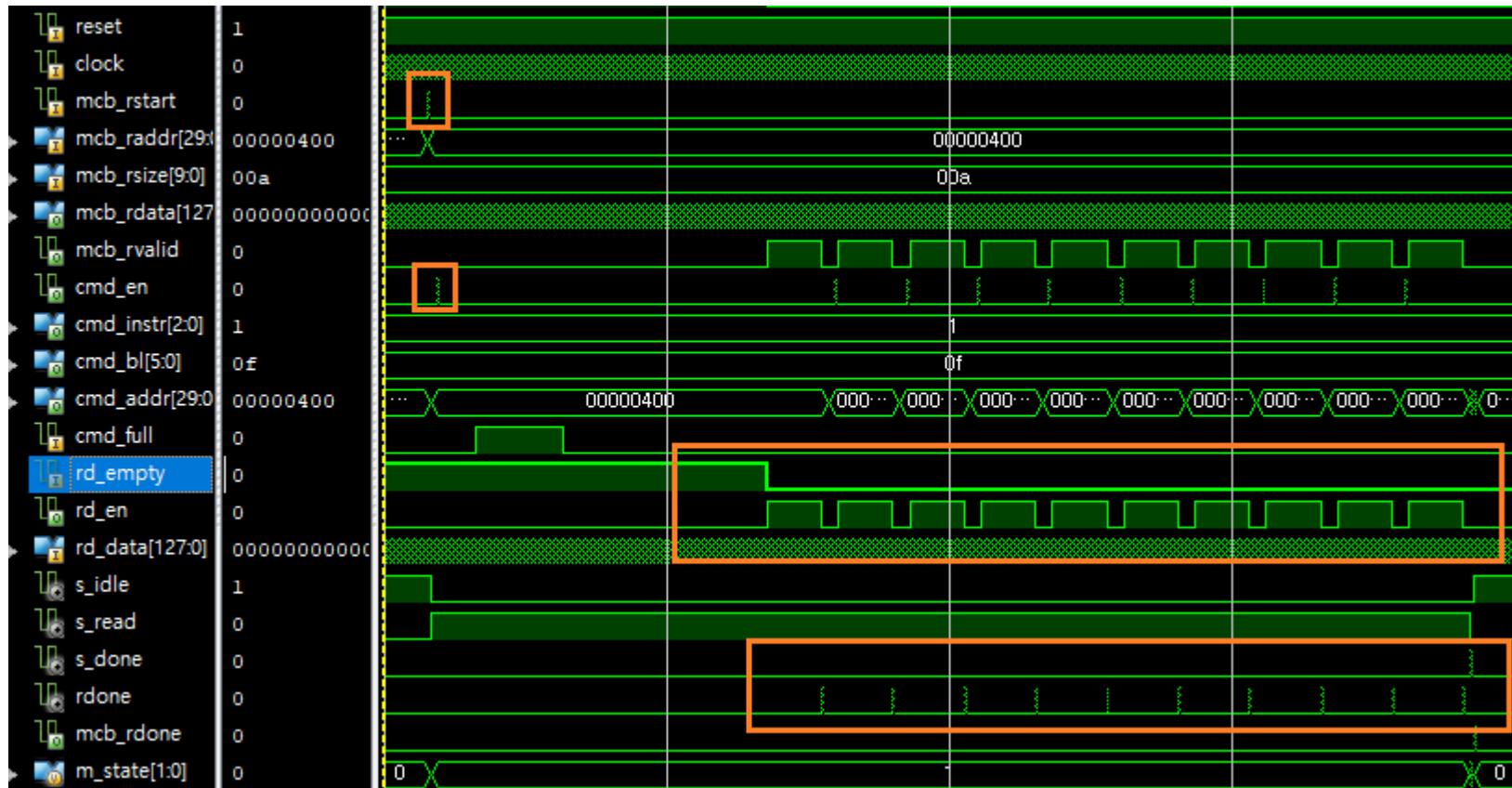
66   reg          cmd_full;
67   always @ (posedge clock or negedge reset)
68 begin
69     if (~reset)      cmd_full <= 1'b0;
70     else            cmd_full <= (cnt==20'd1000) ? 1'b0 :
71                               (cnt==20'd1014) ? 1'b1 :
72                               (cnt==20'd1040) ? 1'b0 : cmd_full ;
73 end
74
75 reg          rd_empty;
76 always @ (posedge clock or negedge reset)
77 begin
78   if (~reset)      rd_empty <= 1'b1;
79   else            rd_empty <= (cnt==20'd1000) ? 1'b1 :
80                               (cnt==20'd1100) ? 1'b0 :
81                               (cnt==20'd11200) ? 1'b1 : rd_empty ;
82 end
83
84 wire [9:0]    mcb_rsize = 10'd10;
85 wire          cmd_en    ;
86 wire [2:0]    cmd_instr ;
87 wire [5:0]    cmd_bl    ;
88 wire [29:0]   cmd_addr  ;
89
90 wire [127:0]  rdata    ;
91 wire          rvalid   ;
92 wire          rdone    ;
93 wire          rd_en    ;
94
95 mcb_read      mcb_read (
96   .reset        (reset      ),
97   .clock        (clock      ), // 41.6 Mhz
98
99   .mcb_rstart  (rstart    ), // host
100  .mcb_raddr   (raddr     ),
101  .mcb_rsize   (mcb_rsize ),
102  .mcb_rdata   (rdata     ),
103  .mcb_rvalid  (rvalid    ),
104  .mcb_rdone   (rdone     ),
105
106  .cmd_en      (cmd_en    ), // ddr
107  .cmd_instr   (cmd_instr ),
108  .cmd_bl      (cmd_bl    ),
109  .cmd_addr   (cmd_addr  ),
110  .cmd_full   (cmd_full  ),
111
112  .rd_en      (rd_en    ), // ddr
113  .rd_data   (rd_data  ),
114  .rd_empty   (rd_empty  )
115 );
116
117 endmodule

```

- ✓ 라인 66 - 82 : cmd_full, rd_empty 신호를 생성합니다.
- ✓ 라인 84 : rsize는 10입니다.
- ✓ 라인 95 - 115 : mcb_read 모듈을 추가합니다.

tb_mcb_read.v, mcb_read.v 파일을 simulation 파일에 추가합니다. tb_mcb_read 모듈을 선택하고, Simulate Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_read 신호를 추가하고 “run 0.1ms”입력해서 simulation을 진행합니다.

아래는 전체적인 파형을 보여줍니다.



mcb_rstart 신호가 active 되면, cmd_en 신호가 active 되어 read command가 전송됩니다. (cmd_instr : 1, cmd_bl : 15, cmd_addr : 0x400) rd_empty 신호가 0가 되면, mcb_rvalid, re_en 신호가 active 되면서 rd_data 값을 read 합니다. 총 10개의 block 이 read 되면, mcb_rdone 신호가 active 됩니다.

9.6 전 영역 read/write 구현

이번 장에서는 앞에서 구현한 코드를 바탕으로 ddr3 memory 전영역을 read/write 하는 코드를 구현합니다. 먼저 전영역을 write 하고, 그리고 전영역을 read 하면서 write data와 read data를 비교하여 verification까지 구현합니다.

9.6.1 mcb_test

mcb_test 모듈을 ddr3 memory 전영역을 write / read / verification 하는 코드입니다. 이번 장에서는 간단히 전체적인 sequence 를 구현하고 다음 장에서, 생성한 ddr3_mig 모듈과의 Interface를 구현하여 실제적인 검증을 진행합니다.

ddr3 memory는 2Gb 입니다. 2Gb 전영역을 access 하기 위해서는 아래와 같은 수식이 필요합니다.

- ✚ 128 x 16 x 256 (wsize/rsize) x 4096 (block size) = 2 Gb
- ✚ 128x16 은 mcb_write/read_128x16 module이 access 하는 사이즈입니다.
- ✚ 256은 mcb_write/read module의 wsize / rsize 입니다.
- ✚ 4096은 mb_write/red module의 필요한 개수 (block size) 입니다.

```

23 `define SIM_MODE
24 //`define RUN_MODE
25
26 // 128x16 x 256(wsize/rsize) x 4096(block_size) : 2Gb
27 module mcb_test (
28     reset ,
29     clock ,          // 41.6 Mhz
30     btn0 ,           // write test start
31     btn1 ,           // read test start
32     led_ok ,         // led
33     led_err ,
34
35     wstart ,          // ddr write
36     waddr ,
37     wsize ,
38     wdata ,
39     wready ,
40     wdone ,
41
42     rstart ,          // ddr read
43     raddr ,
44     rsize ,
45     rdata ,
46     rvalid ,
47     rdone ,
48
49     mcb_err_cnt ,
50     mcb_test_wdone ,
51     mcb_test_rdone
52 );

```

- ✓ 라인 23 - 24 : SIM_MODE : block size : 3, RUN_MODE : block_size : 4095
- ✓ 라인 27 - 52 : mcb_test module 선언

```

53  input      reset      ;
54  input      clock      ;
55  input      btn0       ;
56  input      btn1       ;
57  output     led_ok    ;
58  output     led_err   ;
59
60  output     wstart    ;
61  output [29:0] waddr   ;
62  output [9:0]  wsize   ;
63  output [127:0] wdata  ;
64  input      wready   ;
65  input      wdone    ;
66
67  output     rstart    ;
68  output [29:0] raddr   ;
69  output [9:0]  rsize   ;
70  input [127:0] rdata  ;
71  input      rvalid   ;
72  input      rdone    ;
73
74  output [24:0] mcb_err_cnt ;
75  output      mcb_test_wdone ;
76  output      mcb_test_rdone ;
77
78
79 `ifdef RUN_MODE      // 128x16 x 256 x 4096 = 2Gb
80     parameter BLOCK_SIZE = 10'd256;
81     parameter BLOCK_MAX  = 13'd4095;
82 `elsif SIM_MODE
83     parameter BLOCK_SIZE = 10'd256;
84     parameter BLOCK_MAX  = 13'd3;
85 `endif
86
87 parameter test_data1 = 128'h33333333_33333333_33333333_33333333 ;
88 parameter test_data2 = 128'hcccccccc_cccccccc_cccccccc_cccccccc ;
89 parameter test_data3 = 128'h55555555_55555555_55555555_55555555 ;
90 parameter test_data4 = 128'haaaaaaaa_aaaaaaaaaaaaaaaaaa ;

```

- ✓ 라인 53 - 76 : in/out port 정의
- ✓ 라인 79 - 85 : RUN_MODE - 전 영역 test, SIM_MODE - 4개의 block test
- ✓ 라인 87 - 90 : memory에 write 하는 데이터 정의, 4개의 데이터가 반복적으로 write 됩니다.

```

92 // State Parameter
93 parameter M_IDLE      = 3'd0;
94 parameter M_WRITE1    = 3'd1;
95 parameter M_WRITE2    = 3'd2;
96 parameter M_READ1     = 3'd3;
97 parameter M_READ2     = 3'd4;
98
99
100 // -----
101 // State Control
102 reg [2:0] m_state;
103 wire s_idle  = (m_state==M_IDLE ) ? 1'b1 : 1'b0;
104 wire s_write1 = (m_state==M_WRITE1) ? 1'b1 : 1'b0;
105 wire s_write2 = (m_state==M_WRITE2) ? 1'b1 : 1'b0;
106 wire s_read1 = (m_state==M_READ1 ) ? 1'b1 : 1'b0;
107 wire s_read2 = (m_state==M_READ2 ) ? 1'b1 : 1'b0;

```

- ✓ 라인 92 - 97 : state를 정의합니다. idle에서 btn0가 active 되면 write1로 이동하면서 write를 진행합니다. write1은 wstart flag를 생성하고 write2로 이동합니다. write2는 mcb_write가 완료되면 write1으로 이동합니다. 모든 block이 완료되면 idle로 이동합니다. idle에서 btn1가 active 되면 read1로 이동하면서 read를 진행합니다. read1은 rstart flag를 생성하고 read2로 이동합니다. read2는 mcb_read가 완료되면 read1으로 이동합니다.

모든 block이 완료되면 idle로 이동합니다.

```

109 // -----
110 // write task
111 reg [2:0] cnt_writel ;
112 always @ (posedge clock or negedge reset)
113 begin
114     if (!reset) cnt_writel <= 3'b0;
115     else      cnt_writel <= ~s_writel ? 3'b0 : cnt_writel + 1'bl;
116 end
117
118 wire [9:0] wsize = BLOCK_SIZE ;
119
120 reg wstart;
121 always @ (posedge clock or negedge reset)
122 begin
123     if (!reset) wstart <= 1'b0;
124     else      wstart <= (cnt_writel==3'd3) ? 1'bl : 1'b0;
125 end
126
127 reg [12:0] w_block;
128 always @ (posedge clock or negedge reset)
129 begin
130     if (!reset) w_block <= 13'b0;
131     else      w_block <= s_idle ? 13'b0 : wdone ? w_block+1'bl : w_block;
132 end
133
134 // 128 x 16 : 0x100
135 // 128 x 16 x 256 : 0x10000
136 reg [29:0] waddr;
137 always @ (posedge clock or negedge reset)
138 begin
139     if (!reset) waddr <= 30'b0;
140     else      waddr <= s_idle ? 30'b0 :
141                 (cnt_writel==3'd3) ? {1'b0, w_block, 16'b0} : waddr ;
142 end
143
144 reg [1:0] wready_cnt;
145 always @ (posedge clock or negedge reset)
146 begin
147     if (!reset) wready_cnt <= 2'b0;
148     else      wready_cnt <= s_idle ? 2'b0 :
149                 (cnt_writel==3'd3) ? 2'b0 :
150                 wready ? wready_cnt+1'bl : wready_cnt ;
151 end
152
153 wire [127:0] wdata = (wready_cnt==2'b00) ? test_datal :
154                     (wready_cnt==2'b01) ? test_data2 :
155                     (wready_cnt==2'b10) ? test_data3 : test_data4 ;

```

- ✓ 라인 111 - 116 : write1 상태를 위한 counter
- ✓ 라인 118 : mcb_write 모듈을 위한 wsize 설정
- ✓ 라인 120 - 125 : mcb_write 모듈을 위한 wstart 생성, cnt_write1 이 3일 때 active 됩니다.
- ✓ 라인 127 - 132 : write block counter, mcb_write module의 wdone 신호가 active 되면 1씩 증가합니다.
- ✓ 라인 134 - 142 : waddr 생성, wsize : 256 이므로 mcb_write의 waddr은 0x10000 만큼 증가합니다.
- ✓ 라인 144 - 155 : wdata를 생성하기 위하여 mcb_write의 wready 신호를 받아서 counter를 생성합니다.

```

158 // -----
159 // read task
160
161 reg [2:0] cnt_readl ;
162 always @(posedge clock or negedge reset)
163 begin
164     if(!reset) cnt_readl <= 3'b0;
165     else      cnt_readl <= ~s_readl ? 3'b0 : cnt_readl + 1'bl;
166 end
167
168 wire [9:0] rsize = BLOCK_SIZE;
169
170 reg          rstart;
171 always @(posedge clock or negedge reset)
172 begin
173     if(!reset) rstart <= 1'b0;
174     else      rstart <= (cnt_readl==3'd3) ? 1'bl : 1'b0;
175 end
176
177 reg [12:0] r_block;
178 always @(posedge clock or negedge reset)
179 begin
180     if(!reset) r_block <= 13'b0;
181     else      r_block <= s_idle ? 13'b0 : rdone ? r_block+1'bl : r_block;
182 end
183
184 reg [29:0] raddr;
185 always @(posedge clock or negedge reset)
186 begin
187     if(!reset) raddr <= 30'b0;
188     else      raddr <= s_idle ? 30'b0 :
189                     (cnt_readl==3'd3) ? {1'b0, r_block, 16'b0} : raddr ;
190 end

```

- ✓ 라인 161 - 166 : read1 상태를 위한 counter
- ✓ 라인 168 : mcb_read 모듈을 위한 rsize 설정
- ✓ 라인 170 - 175 : mcb_read 모듈을 위한 rstart 생성
- ✓ 라인 177 - 182 : read block counter, mcb_read 모듈의 rdone 신호가 active 되면 1씩 증가합니다.
- ✓ 라인 184 - 190 : raddr 생성

```

193 // -----
194 // verification
195 reg [1:0] rvalid_cnt;
196 always @(posedge clock or negedge reset)
197 begin
198     if(!reset) rvalid_cnt <= 2'b0;
199     else      rvalid_cnt <= s_idle ? 2'b0 :
200                     (cnt_readl==3'd3) ? 2'b0 :
201                     rvalid ? rvalid_cnt+1'bl : rvalid_cnt ;
202 end
203
204
205 reg [24:0] mcb_err_cnt ;
206 always @(posedge clock or negedge reset)
207 begin
208     if(!reset) mcb_err_cnt <= 25'b0;
209     else      mcb_err_cnt <= btn0 ? 25'b0 :
210                     (rvalid & (rvalid_cnt==2'b00) && (rdata!=test_data1)) ? mcb_err_cnt + 1'bl :
211                     (rvalid & (rvalid_cnt==2'b01) && (rdata!=test_data2)) ? mcb_err_cnt + 1'bl :
212                     (rvalid & (rvalid_cnt==2'b10) && (rdata!=test_data3)) ? mcb_err_cnt + 1'bl :
213                     (rvalid & (rvalid_cnt==2'b11) && (rdata!=test_data4)) ? mcb_err_cnt + 1'bl :
214                     mcb_err_cnt ;
215 end

```

- ✓ 라인 193 - 215 : write data와 read data를 비교하여 error counter 를 구합니다.

```

218  always @ (posedge clock or negedge reset)
219  begin
220      if (!reset)      begin
221          m_state <= 3'b0;
222      end
223      else   begin
224          m_state <= (s_idle    & btn0)  ?  M_WRITE1  :
225                      (s_idle    & btn1)  ?  M_READ1  :
226                      (s_write1 & (cnt_write1==3'd4)) ?  M_WRITE2  :
227                      (s_write2 & wdone ) ? ((w_block==BLOCK_MAX) ? M_IDLE : M_WRITE1) :
228                      (s_read1  & (cnt_read1==3'd4)) ?  M_READ2  :
229                      (s_read2  & rdone ) ? ((r_block==BLOCK_MAX) ? M_IDLE : M_READ1)  :
230                          m_state ;
231      end
232  end
233
234
235  reg           mcb_test_wdone;
236  always @ (posedge clock or negedge reset)
237  begin
238      if (!reset) mcb_test_wdone <= 1'b0;
239      else        mcb_test_wdone <= (s_write2 & wdone & (w_block==BLOCK_MAX)) ? 1'bl : 1'b0;
240  end
241
242  reg           mcb_test_rdone;
243  always @ (posedge clock or negedge reset)
244  begin
245      if (!reset) mcb_test_rdone <= 1'b0;
246      else        mcb_test_rdone <= (s_read2 & rdone & (r_block==BLOCK_MAX)) ? 1'bl : 1'b0;
247  end
248
249
250  reg           led_ok ;
251  always @ (posedge clock or negedge reset)
252  begin
253      if (!reset) led_ok <= 1'b0;
254      else        led_ok <= (s_idle & (mcb_err_cnt == 25'b0)) ? 1'bl : 1'b0;
255  end
256
257  reg           led_err ;
258  always @ (posedge clock or negedge reset)
259  begin
260      if (!reset) led_err <= 1'b0;
261      else        led_err <= (s_idle & (mcb_err_cnt != 25'b0)) ? 1'bl : 1'b0;
262  end
263
264
265 endmodule

```

- ✓ 라인 218 - 232 : state 이동을 구현합니다. idle 상태에서 btn0가 active 되면 write1으로 이동하고, btn1이 active 되면 read1로 이동합니다. write1에서는 wstart (waddr, wsize)신호를 active 하고 write2로 이동합니다. write2는 wdone신호가 active 되면, block 모두 write 하면 idle로 그렇지 않으면 write1으로 이동합니다. read1, read2도 동일합니다.
- ✓ 라인 235 - 247 : mcb_test_wdone, mcb_test_rdone 신호를 생성합니다.
- ✓ 라인 250 - 262 : error counter에 따라 led_on, led_err 를 on/off 합니다.

9.6.2 simulation mcb_test

mcb_test 모듈을 simulation 합니다. SIM_MODE로 test를 진행합니다.

```
--  
23 `define SIM_MODE  
24 //`define RUN_MODE
```

test_bench는 tb_mcb_test 입니다.

```
23 module tb_mcb_test();  
24  
25  
26 reg reset, clock;  
27  
28 initial begin  
29     reset = 0;  
30     clock = 0;  
31  
32 #10000 reset = 1;  
33  
34 end  
35  
36 parameter test_data1 = 128'h33333333_33333333_33333333_33333333 ;  
37 parameter test_data2 = 128'hcccccccc_cccccccc_cccccccc_cccccccc ;  
38 parameter test_data3 = 128'h55555555_55555555_55555555_55555555 ;  
39 parameter test_data4 = 128'haaaaaaaa_aaaaaaaaaaaaaaaaa ;  
40  
41 always #12 clock = ~clock; // 41.67Mhz  
42  
43 reg [19:0] cnt;  
44 always @ (posedge clock or negedge reset)  
45 begin  
46     if (!reset) cnt <= 20'b0;  
47     else cnt <= (cnt==20'd1020) ? 20'd1020 : cnt + 1'bl ;  
48 end
```

- ✓ 라인 23 : tb_mcb_test 모듈 선언
- ✓ 라인 26 - 34 : reset, clock 생성
- ✓ 라인 36 - 39 : test data 생성
- ✓ 라인 43 - 48 : 신호 생성을 위한 counter

```

52 // -----
53 // write test bench
54 reg btn0 ;
55 always @ (posedge clock or negedge reset)
56 begin
57     if (!reset)      btn0 <= 1'b0;
58     else           btn0 <= (cnt==20'd1000) ? 1'b1 :
59                           (cnt==20'd1001) ? 1'b0 : btn0 ;
60 end
61
62
63 reg [12:0] cnt2;
64 always @ (posedge clock or negedge reset)
65 begin
66     if (!reset) cnt2 <= 13'b0;
67     else         cnt2 <= (cnt==20'd1020) ? ((cnt2==13'd1060) ? 13'd0 : cnt2 + 1'b1) : 13'd0;
68 end
69
70 reg wready;
71 always @ (posedge clock or negedge reset)
72 begin
73     if (!reset) wready <= 1'b0;
74     else         wready <= (cnt2==13'd20) ? 1'b1 :
75                           (cnt2==13'd1044) ? 1'b0 : wready ;
76 end
77
78 reg wdone;
79 always @ (posedge clock or negedge reset)
80 begin
81     if (!reset) wdone <= 1'b0;
82     else         wdone <= (cnt2==13'd1050) ? 1'b1 :
83                           (cnt2==13'd1051) ? 1'b0 : wdone ;
84 end
85
86 wire btn1 = 1'b0;
87 wire rvalid = 1'b0;
88 wire rdone = 1'b0;
89 wire [127:0] rdata = 128'b0;

```

- ✓ 라인 52 - 89 : write를 위한 신호입니다. read 시에는 이 부분을 주석 처리합니다. btn0, wready, wdone 신호를 생성합니다.

```

92 /*
93 // -----
94 // read test bench
95 wire      btn0    = 1'b0;
96 wire      wready = 1'b0;
97 wire      wdone  = 1'b0;
98
99 reg       btn1 ;
100 always @(posedge clock or negedge reset)
101 begin
102     if(!reset)      btn1 <= 1'b0;
103     else           btn1 <= (cnt==20'd1000) ? 1'b1 :
104                           (cnt==20'd1001) ? 1'b0 : btn1 ;
105 end
106
107
108 reg      [12:0] cnt2;
109 always @(posedge clock or negedge reset)
110 begin
111     if(!reset) cnt2 <= 13'b0;
112     else       cnt2 <= (cnt==20'd1020) ? ((cnt2==13'd1060) ? 13'd0 : cnt2 + 1'b1) : 13'd0;
113 end
114
115 reg       rvalid;
116 always @(posedge clock or negedge reset)
117 begin
118     if(!reset)      rvalid <= 1'b0;
119     else           rvalid <= (cnt2==13'd20)   ? 1'b1 :
120                           (cnt2==13'd1044) ? 1'b0 : rvalid ;
121 end
122
123 reg       rdone;
124 always @(posedge clock or negedge reset)
125 begin
126     if(!reset)      rdone <= 1'b0;
127     else           rdone <= (cnt2==13'd1050) ? 1'b1 :
128                           (cnt2==13'd1051) ? 1'b0 : wdone ;
129 end
130
131
132 reg      [1:0] rvalid_cnt;
133 always @(posedge clock or negedge reset)
134 begin
135     if(!reset)      rvalid_cnt <= 2'b0;
136     else           rvalid_cnt <= ~rvalid ? 2'b0 : rvalid_cnt+1'b1 ;
137 end
138
139 wire      [127:0] rdata = (rvalid_cnt[1:0]==2'b00) ? test_data1 :
140                           (rvalid_cnt[1:0]==2'b01) ? test_data2 :
141                           (rvalid_cnt[1:0]==2'b10) ? test_data3 : test_data4 ;
142 */

```

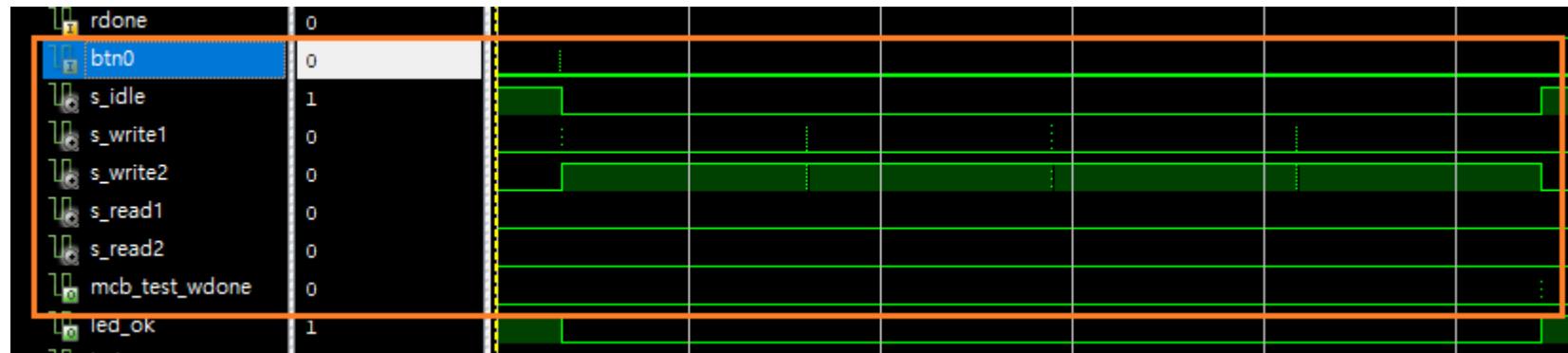
- ✓ 라인 92 - 142 : read를 위한 신호입니다. write 시에는 이 부분을 주석 처리합니다. btn1, rvalid, rdone, rdata 신호를 생성합니다.

```
145 wire led_ok ;  
146 wire led_err ;  
147 wire wstart ;  
148 wire [29:0] waddr ;  
149 wire [9:0] wsize ;  
150 wire [127:0] wdata ;  
151 wire rstart ;  
152 wire [29:0] raddr ;  
153 wire [9:0] rsize ;  
154 wire [24:0] mcb_err_cnt ;  
155 wire mcb_test_wdone ;  
156 wire mcb_test_rdone ;  
157 mcb_test mcb_test (  
158 .clock (clock ),  
159 .reset (reset ),  
160  
161 .btn0 (btn0 ),  
162 .btn1 (btn1 ),  
163  
164 .led_ok (led_ok ),  
165 .led_err (led_err ),  
166  
167 .wstart (wstart ),  
168 .waddr (waddr ),  
169 .wsize (wsize ),  
170  
171 .wdata (wdata ),  
172 .wready (wready ),  
173 .wdone (wdone ),  
174  
175 .rstart (rstart ),  
176 .raddr (raddr ),  
177 .rsize (rsize ),  
178  
179 .rdata (rdata ),  
180 .rvalid (rvalid ),  
181 .rdone (rdone ),  
182  
183 .mcb_err_cnt (mcb_err_cnt ),  
184 .mcb_test_wdone (mcb_test_wdone ),  
185 .mcb_test_rdone (mcb_test_rdone )  
186 );  
187  
188  
189 endmodule
```

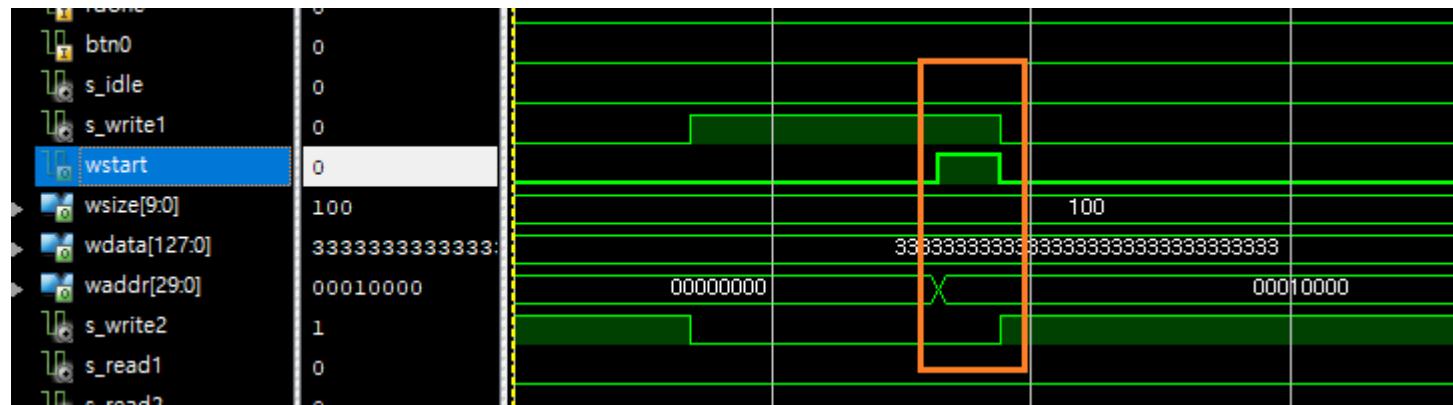
- ✓ 라인 145 - 185 : mcb_test 모듈을 추가합니다.

tb_mcb_test.v, mcb_test.v 파일을 simulation 파일에 추가합니다. 먼저 write를 simulation 하기 위하여 tb_mcb_test.v의 라인 93 - 141 을 주석 처리합니다. tb_mcb_test 모듈을 선택하고, Simulate Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_test 신호를 추가하고 “run 0.4ms”입력해서 simulation을 진행합니다.

아래 그림은 전체적인 파형을 보여줍니다. btn0가 active 되고, write1, write2 가 4번 발생합니다. 마지막으로 mcb_test_wdone 0이 active 되고, idle 상태가 됩니다.

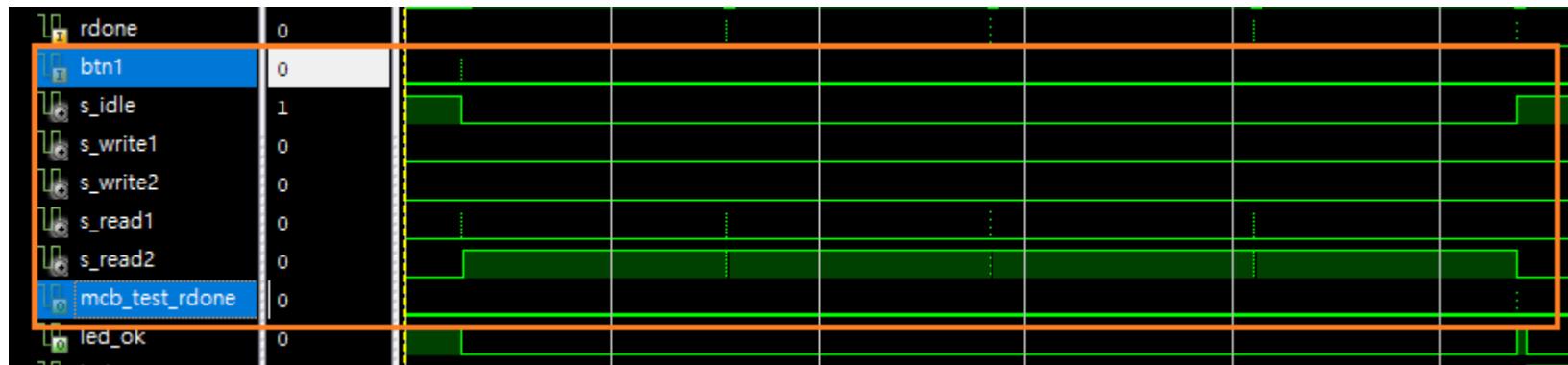


아래 그림은 두번째 s_write1 상태를 보여줍니다. wstart 가 active 될 때, wsiz : 0x100, wdata : 0x3333~ , waddr : 0x10000 가 정상적으로 생성됨을 알 수 있습니다. 첫번째 wdata는 wready 신호가 발생하기 전에 미리 준비되어야 합니다.

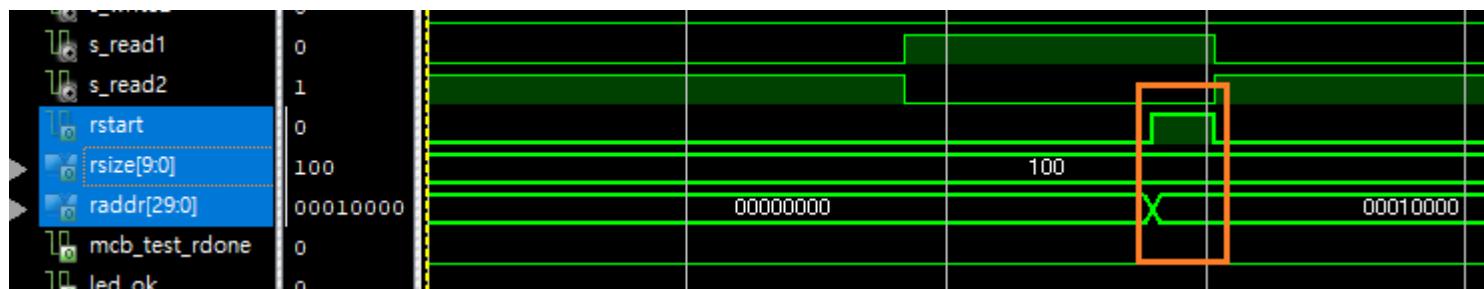


이번에는 read를 simulation 합니다. tb_mcb_test.v의 라인 93 - 141의 주석은 해제하고, 라인 52 - 89 을 주석 처리합니다. tb_mcb_test 모듈을 선택하고, Simulate Behavioral Model을 더블 클릭합니다. wave 윈도에 mcb_test 신호를 추가하고 “run 0.4ms”입력해서 simulation을 진행합니다.

아래 그림은 전체적인 파형을 보여줍니다. btn1가 active 되고, read1, read2 가 4번 발생합니다. 마지막으로 mcb_test_rdone이 active 되고, idle 상태가 됩니다.



아래 그림은 두번째 s_read1 상태를 보여줍니다. rstart 가 active 될 때, rsize : 0x100, raddr : 0x10000 가 정상적으로 생성됨을 알 수 있습니다.



9.6.3 tb_mcb_top

이번 장에서는 지금까지 구현했던 모든 것들을 이용하여 memory 전 영역을 write, read, verification 하는 것을 구현합니다.

tb_mcb_top 코드를 구현합니다. tb_mcb_top 코드 구현은, IP 생성하고 동작 확인을 위하여 simulation을 진행했던 sim_tb_top.v 코드를 이용하여 구현하도록 하겠습니다.

sim_tb_top.v는 크게 3부분으로 구성되어 있습니다.

- ✚ ddr3_mig module : 생성된 ddr controller 입니다. 라인 384 - 457
- ✚ memc_tb_top : simulation 을 위한 test pattern을 생성하는 모듈, 라인 464 - 668
- ✚ ddr3_model_c3 : ddr3 memory model, 라인 675 - 719

우리는 sim_tb_top.v 의 코드를 그대로 사용하면서, memc_tb_top 모듈 대신에 우리가 지금까지 구현했던 코드를 추가합니다. 필요 없는 신호들을 제거하도록 합니다.

아래는 코드를 보여줍니다.

```

67 `timescale 1ps/1ps
68 module tb_mcb_top;
69
70 // ===== //
71 // Parameters //
72 // ===== //
73 parameter DEBUG_EN          = 0;
74 parameter C3_MEMCLK_PERIOD = 3000;
75 parameter C3_RST_ACT_LOW   = 0;
76 parameter C3_INPUT_CLK_TYPE = "DIFFERENTIAL";
77 parameter C3_NUM_DQ_PINS   = 16;
78 parameter C3_MEM_ADDR_WIDTH = 14;
79 parameter C3_MEM_BANKADDR_WIDTH = 3;
80 parameter C3_MEM_ADDR_ORDER = "BANK_ROW_COLUMN";
81 parameter C3_PO_MASK_SIZE   = 16;
82 parameter C3_PO_DATA_PORT_SIZE = 128;
83 parameter C3_CALIB_SOFT_IP  = "TRUE";
84 parameter C3_SIMULATION     = "TRUE";

```

- ✓ 라인 67 : simulation은 1ps 단위로 진행됩니다.
- ✓ 라인 68 : tb_mcb_top 모듈 선언
- ✓ 라인 70 - 84 : IP 생성시 설정했던 값들을 parameter로 정의합니다.

```

86 // ===== //
87 // Signal Declarations // 
88 // ===== //
89 // Clocks
90 // Clocks
91 reg c3_sys_clk;
92 wire c3_sys_clk_p;
93 wire c3_sys_clk_n;
94 // System Reset
95 reg c3_sys_rst;
96 wire c3_sys_rst_i;
97
98 // Design-Top Port Map
99 wire c3_error;
100 wire c3_calib_done;
101 wire [31:0] c3_cmp_data;
102 wire c3_cmp_error;
103 wire [C3_MEM_ADDR_WIDTH-1:0] mcb3_dram_a;
104 wire [C3_MEM_BANKADDR_WIDTH-1:0] mcb3_dram_ba;
105 wire mcb3_dram_ck;
106 wire mcb3_dram_ck_n;
107 wire [C3_NUM_DQ_PINS-1:0] mcb3_dram_dq;
108 wire mcb3_dram_dqs;
109 wire mcb3_dram_dqs_n;
110 wire mcb3_dram_dm;
111 wire mcb3_dram_ras_n;
112 wire mcb3_dram_cas_n;
113 wire mcb3_dram_we_n;
114 wire mcb3_dram_cke;
115 wire mcb3_dram_odt;
116 wire mcb3_dram_reset_n;
117 wire mcb3_dram_udqs; // for X16 parts
118 wire mcb3_dram_udqs_n; // for X16 parts
119 wire mcb3_dram_udm; // for X16 parts
120
121 // User design Sim
122 wire c3_clk0;
123 wire c3_rst0;
124
125
126 // Error & Calib Signals
127 wire error;
128 wire calib_done;
129 wire rzq3;
130 wire zio3;

```

- ✓ 라인 86 - 130 : 내부에서 사용하는 신호입니다. mcb3_으로 시작하는 신호는 실제 외부 ddr3 memory와 연결됩니다. 여기서는 memory model과 연결됩니다.
- ✓ 라인 100 : calibration done 신호
- ✓ 라인 122-123 : user interface에서 사용하는 clock, reset

```
132 // ===== //  
133 // Clocks Generation //  
134 // ===== //  
135  
136     initial  
137         c3_sys_clk = 1'b0;  
138     always  
139         #(C3_MEMCLK_PERIOD/2) c3_sys_clk = ~c3_sys_clk;  
140  
141         assign c3_sys_clk_p = c3_sys_clk;  
142         assign c3_sys_clk_n = ~c3_sys_clk;  
143  
144 // ===== //  
145 // Reset Generation //  
146 // ===== //  
147  
148     initial begin  
149         c3_sys_rst = 1'b0;  
150         #20000;  
151         c3_sys_rst = 1'bl;  
152     end  
153         assign c3_sys_rst_i = C3_RST_ACT_LOW ? c3_sys_rst : ~c3_sys_rst;  
154  
155 // ===== //  
156 // Error Grouping //  
157 // ===== //  
158  
159         assign error = c3_error;  
160         assign calib_done = c3_calib_done;  
161  
162  
163  
164  
165 // The PULLDOWN component is connected to the ZIO signal primarily to avoid the  
166 // unknown state in simulation. In real hardware, ZIO should be a no connect(NC) pin.  
167         PULLDOWN zio_pulldown3 (.O(zio3));    PULLDOWN rzq_pulldown3 (.O(rzq3));
```

- ✓ 라인 132 - 153 : clock, reset을 생성합니다.

```

170 // ===== //
171 // User Interface //
172 // ===== //
173 wire c3_rstn = ~c3_rst0;
174
175 // State Parameter
176 parameter M_IDLE      = 2'd0;
177 parameter M_WRITE     = 2'd1;
178 parameter M_READ      = 2'd2;
179 parameter M_DONE      = 2'd3;
180
181 // -----
182 // State Control
183 reg [1:0] m_state;
184 wire s_idle   = (m_state==M_IDLE) ? 1'b1 : 1'b0;
185 wire s_write  = (m_state==M_WRITE) ? 1'b1 : 1'b0;
186 wire s_read   = (m_state==M_READ) ? 1'b1 : 1'b0;
187 wire s_done   = (m_state==M_DONE) ? 1'b1 : 1'b0;
188
189 reg [9:0] cnt_w ;
190 always @ (posedge c3_clk0 or negedge c3_rstn)
191 begin
192     if (!c3_rstn) cnt_w <= 10'b0;
193     else       cnt_w <= ~s_write ? 10'b0 : (cnt_w==10'd1023) ? 10'd1023 : cnt_w + 1'b1;
194 end
195
196 reg [9:0] cnt_r ;
197 always @ (posedge c3_clk0 or negedge c3_rstn)
198 begin
199     if (!c3_rstn) cnt_r <= 10'b0;
200     else       cnt_r <= ~s_read ? 10'b0 : (cnt_r==10'd1023) ? 10'd1023 : cnt_r + 1'b1;
201 end
202
203 reg btn0 ;
204 always @ (posedge c3_clk0 or negedge c3_rstn)
205 begin
206     if (!c3_rstn) btn0 <= 1'b0;
207     else       btn0 <= (cnt_w==10'd1000) ? 1'b1 :
208                  (cnt_w==10'd1001) ? 1'b0 : btn0 ;
209 end
210
211 reg btn1 ;
212 always @ (posedge c3_clk0 or negedge c3_rstn)
213 begin
214     if (!c3_rstn) btn1 <= 1'b0;
215     else       btn1 <= (cnt_r==10'd1000) ? 1'b1 :
216                  (cnt_r==10'd1001) ? 1'b0 : btn1 ;
217 end
218
219 wire mcb_test_wdone ;
220 wire mcb_test_rdone ;
221 always @ (posedge c3_clk0 or negedge c3_rstn)
222 begin
223     if (!c3_rstn) begin
224         m_state <= 2'b0;
225     end
226     else begin
227         m_state <= (s_idle & c3_calib_done) ? M_WRITE :
228                     (s_write & mcb_test_wdone) ? M_READ :
229                     (s_read & mcb_test_rdone) ? M_DONE :
230                         m_state ;
231     end
232 end

```

- ✓ 라인 170 - 232 : User Interface를 구현합니다.
- ✓ 라인 173 : reset은 c3_rst0의 반전을 사용합니다.
- ✓ 라인 175 - 179 : state를 정의합니다. idle 상태에서 calibration 이 완료되면 write로 이동합니다. write가 완료 되면 read로 이동합니다. read가 완료되면 done으로 이동합니다.

- ✓ 라인 189 - 217 : write 상태에서 1000 clock 후에 btn0 (ddr3 write)을 active 합니다. read 상태에서는 1000 clock 후에 btn1 (ddr3 read)을 active 합니다.
- ✓ 라인 219 - 232 : 상태 이동을 구현합니다. idle 상태에서는 calibration 이 완료되면 write 상태로, write 상태에서는 wdone 후에 read 상태로, read 상태에서는 rdone 후에 done 상태로 이동합니다.

```

235   wire      led_ok      ;
236   wire      led_err      ;
237
238   wire      mcb_wstart    ;
239   wire      [29:0] mcb_waddr    ;
240   wire      [9:0]  mcb_wsize    ;
241   wire      [127:0] mcb_wdata    ;
242   wire      mcb_wdone    ;
243   wire      mcb_wready    ;
244
245   wire      mcb_rstart    ;
246   wire      [29:0] mcb_raddr    ;
247   wire      [9:0]  mcb_rsize    ;
248
249   wire      [127:0] mcb_rdata    ;
250   wire      mcb_rvalid    ;
251   wire      mcb_rdone    ;
252   wire      [24:0] mcb_err_cnt    ;
253   mcb_test  mcb_test (
254     .reset      (c3_rstn      ),
255     .clock      (c3_clk0      ),
256
257     .btn0       (btn0        ),
258     .btn1       (btn1        ),
259
260     .led_ok     (led_ok      ),
261     .led_err    (led_err      ),
262
263     .wstart     (mcb_wstart    ),
264     .waddr      (mcb_waddr    ),
265     .wsize      (mcb_wsize    ),
266     .wdata      (mcb_wdata    ),
267     .wready     (mcb_wready    ),
268     .wdone      (mcb_wdone    ),
269
270     .rstart     (mcb_rstart    ),
271     .raddr      (mcb_raddr    ),
272     .rsize      (mcb_rsize    ),
273     .rdata      (mcb_rdata    ),
274     .rvalid     (mcb_rvalid    ),
275     .rdone      (mcb_rdone    ),
276
277     .mcb_err_cnt (mcb_err_cnt  ),
278     .mcb_test_wdone (mcb_test_wdone  ),
279     .mcb_test_rdone (mcb_test_rdone  )
280 );

```

- ✓ 라인 235 - 280 : mcb_test 모듈을 추가합니다.

```

282 wire      cmd_wr_en ;
283 wire [2:0] cmd_wr_instr ;
284 wire [5:0] cmd_wr_bl ;
285 wire [29:0] cmd_wr_addr ;
286 wire      c3_cmd_full ;
287 wire      c3_wr_en ;
288 wire [15:0] c3_wr_mask ;
289 wire [127:0] c3_wr_data ;
290 wire      c3_wr_full ;
291 mcb_write   mcb_write (
292     .reset      (c3_rstn      ),
293     .clock      (c3_clk0      ),
294
295     .mcb_wstart (mcb_wstart    ),
296     .mcb_waddr  (mcb_waddr    ),
297     .mcb_wsize  (mcb_wsize    ),
298     .mcb_wdata  (mcb_wdata    ),
299     .mcb_wready (mcb_wready    ),
300     .mcb_wdone  (mcb_wdone    ),
301
302     .cmd_en     (cmd_wr_en    ),           // ddr
303     .cmd_instr  (cmd_wr_instr  ),
304     .cmd_bl     (cmd_wr_bl    ),
305     .cmd_addr   (cmd_wr_addr  ),
306     .cmd_full   (c3_cmd_full  ),
307
308     .wr_en      (c3_wr_en    ),           // ddr
309     .wr_mask    (c3_wr_mask  ),
310     .wr_data   (c3_wr_data  ),
311     .wr_full   (c3_wr_full  )
312 );
313
314 wire      cmd_rd_en ;
315 wire [2:0] cmd_rd_instr ;
316 wire [5:0] cmd_rd_bl ;
317 wire [29:0] cmd_rd_addr ;
318 wire      c3_rd_en ;
319 wire [127:0] c3_rd_data ;
320 wire      c3_rd_empty ;
321 mcb_read   mcb_read (
322     .reset      (c3_rstn      ),
323     .clock      (c3_clk0      ),
324
325     .mcb_rstart (mcb_rstart    ),           // host
326     .mcb_raddr  (mcb_raddr    ),
327     .mcb_rsize  (mcb_rsize    ),
328     .mcb_rdata  (mcb_rdata    ),
329     .mcb_rvalid (mcb_rvalid    ),
330     .mcb_rdone  (mcb_rdone    ),
331
332     .cmd_en     (cmd_rd_en    ),           // ddr
333     .cmd_instr  (cmd_rd_instr  ),
334     .cmd_bl     (cmd_rd_bl    ),
335     .cmd_addr   (cmd_rd_addr  ),
336     .cmd_full   (c3_cmd_full  ),
337
338     .rd_en      (c3_rd_en    ),           // ddr
339     .rd_data   (c3_rd_data  ),
340     .rd_empty   (c3_rd_empty  )
341 );

```

- ✓ 라인 282 - 312 : mcb_write 모듈을 추가합니다.
- ✓ 라인 314 - 341 : mcb_read 모듈을 추가합니다.

```

344   reg          mcb_rw_flag ; // 0 : write, 1 : read
345   always @ (posedge c3_clk0 or negedge c3_rstn)
346   begin
347     if (!c3_rstn)    mcb_rw_flag <= 1'b0;
348     else            mcb_rw_flag <= mcb_wstart ? 1'b0 : mcb_rstart ? 1'bl : mcb_rw_flag ;
349   end
350
351   wire          c3_cmd_en    = ~mcb_rw_flag ? cmd_wr_en    : cmd_rd_en    ;
352   wire [2:0]    c3_cmd_instr = ~mcb_rw_flag ? cmd_wr_instr : cmd_rd_instr ;
353   wire [5:0]    c3_cmd_bl    = ~mcb_rw_flag ? cmd_wr_bl    : cmd_rd_bl    ;
354   wire [29:0]   c3_cmd_addr = ~mcb_rw_flag ? cmd_wr_addr : cmd_rd_addr ;
355
356
357   wire          c3_cmd_empty ;
358   wire          c3_wr_empty;
359   wire [6:0]    c3_wr_count;
360   wire          c3_wr_underrun;
361   wire          c3_wr_error;
362
363   wire          c3_rd_full;
364   wire [6:0]    c3_rd_count;
365   wire          c3_rd_overflow;
366   wire          c3_rd_error;

```

- ✓ 라인 344 - 354 : command 관련 신호들은 write, read 각각 있습니다. write, read 동작에 따라서 하나의 신호로 만들어 줍니다. 이 신호들은 ddr3_mig 신호에 연결됩니다.
- ✓ 라인 357 - 366 : ddr3_mig 신호들 중에 사용하지 않는 신호입니다.

```

373 ddr3_mig #(
374
375   .C3_P0_MASK_SIZE      (C3_P0_MASK_SIZE      ),
376   .C3_P0_DATA_PORT_SIZE (C3_P0_DATA_PORT_SIZE ),
377   .C3_MEMCLK_PERIOD     (C3_MEMCLK_PERIOD),
378   .C3_RST_ACT_LOW       (C3_RST_ACT_LOW),
379   .C3_INPUT_CLK_TYPE    (C3_INPUT_CLK_TYPE),
380   .DEBUG_EN              (DEBUG_EN),
381   .C3_MEM_ADDR_ORDER    (C3_MEM_ADDR_ORDER    ),
382   .C3_NUM_DQ_PINS       (C3_NUM_DQ_PINS       ),
383   .C3_MEM_ADDR_WIDTH    (C3_MEM_ADDR_WIDTH    ),
384   .C3_MEM_BANKADDR_WIDTH (C3_MEM_BANKADDR_WIDTH),
385
386   .C3_SIMULATION        (C3_SIMULATION),
387   .C3_CALIB_SOFT_IP     (C3_CALIB_SOFT_IP )
388 )
389 design_top (
390
391   .c3_sys_clk_p          (c3_sys_clk_p),
392   .c3_sys_clk_n          (c3_sys_clk_n),
393   .c3_sys_rst_i          (c3_sys_rst_i),
394
395   .mcb3_dram_dq          (mcb3_dram_dq),
396   .mcb3_dram_a           (mcb3_dram_a),
397   .mcb3_dram_ba          (mcb3_dram_ba),
398   .mcb3_dram_ras_n       (mcb3_dram_ras_n),
399   .mcb3_dram_cas_n       (mcb3_dram_cas_n),
400   .mcb3_dram_we_n        (mcb3_dram_we_n),
401   .mcb3_dram_odt         (mcb3_dram_odt),
402   .mcb3_dram_cke         (mcb3_dram_cke),
403   .mcb3_dram_ck          (mcb3_dram_ck),
404   .mcb3_dram_ck_n        (mcb3_dram_ck_n),
405   .mcb3_dram_dqs         (mcb3_dram_dqs),
406   .mcb3_dram_dqs_n       (mcb3_dram_dqs_n),
407   .mcb3_dram_udqs        (mcb3_dram_udqs), // for X16 parts
408   .mcb3_dram_udqs_n     (mcb3_dram_udqs_n), // for X16 parts
409   .mcb3_dram_udm         (mcb3_dram_udm), // for X16 parts
410   .mcb3_dram_dm          (mcb3_dram_dm),
411   .mcb3_dram_reset_n     (mcb3_dram_reset_n),
412   .c3_clk0               (c3_clk0),
413   .c3_rst0               (c3_rst0),
414
415   .c3_calib_done         (c3_calib_done),
416   .mcb3_rzq               (rzq3),
417   .mcb3_zio               (zio3),
418
419   .c3_p0_cmd_clk          (c3_clk0),
420   .c3_p0_cmd_en           (c3_cmd_en),
421   .c3_p0_cmd_instr         (c3_cmd_instr),
422   .c3_p0_cmd_bl            (c3_cmd_bl),
423   .c3_p0_cmd_byte_addr     (c3_cmd_addr),
424
425   .c3_p0_cmd_empty         (c3_cmd_empty),
426   .c3_p0_cmd_full          (c3_cmd_full),
427   .c3_p0_wr_clk            (c3_clk0),
428   .c3_p0_wr_en             (c3_wr_en),
429   .c3_p0_wr_mask           (c3_wr_mask),
430   .c3_p0_wr_data           (c3_wr_data),
431   .c3_p0_wr_full           (c3_wr_full),
432   .c3_p0_wr_empty          (c3_wr_empty),
433   .c3_p0_wr_count          (c3_wr_count),
434   .c3_p0_wr_underrun       (c3_wr_underrun),
435   .c3_p0_wr_error          (c3_wr_error),
436   .c3_p0_rd_clk            (c3_clk0),
437   .c3_p0_rd_en             (c3_rd_en),
438   .c3_p0_rd_data           (c3_rd_data),
439   .c3_p0_rd_full           (c3_rd_full),
440   .c3_p0_rd_empty          (c3_rd_empty),
441   .c3_p0_rd_count          (c3_rd_count),
442   .c3_p0_rd_overflow       (c3_rd_overflow),
443   .c3_p0_rd_error          (c3_rd_error)
444 );

```

- ✓ 라인 373 - 444 : ddr3_mig 모듈을 추가합니다.

```

447 // ===== //
448 // Memory model instances                                //
449 // ===== //
450
451 generate
452 if(C3_NUM_DQ_PINS == 16) begin : MEM_INST3
453   ddr3_model_c3 u_mem_c3(
454     .ck      (mcb3_dram_ck),
455     .ck_n    (mcb3_dram_ck_n),
456     .cke     (mcb3_dram_cke),
457     .cs_n    (1'b0),
458     .ras_n   (mcb3_dram_ras_n),
459     .cas_n   (mcb3_dram_cas_n),
460     .we_n    (mcb3_dram_we_n),
461     .dm_tdqs ({mcb3_dram_udm,mcb3_dram_dm}),
462     .ba      (mcb3_dram_ba),
463     .addr    (mcb3_dram_a),
464     .dq      (mcb3_dram_dq),
465     .dqs    ({mcb3_dram_udqs,mcb3_dram_dqs}),
466     .dqs_n   ({mcb3_dram_udqs_n,mcb3_dram_dqs_n}),
467     .tdqs_n  (),
468     .odt     (mcb3_dram_odt),
469     .rst_n   (mcb3_dram_reset_n)
470   );
471 end else begin
472   ddr3_model_c3 u_mem_c3(
473     .ck      (mcb3_dram_ck),
474     .ck_n    (mcb3_dram_ck_n),
475     .cke     (mcb3_dram_cke),
476     .cs_n    (1'b0),
477     .ras_n   (mcb3_dram_ras_n),
478     .cas_n   (mcb3_dram_cas_n),
479     .we_n    (mcb3_dram_we_n),
480     .dm_tdqs (mcb3_dram_dm),
481     .ba      (mcb3_dram_ba),
482     .addr    (mcb3_dram_a),
483     .dq      (mcb3_dram_dq),
484     .dqs    (mcb3_dram_dqs),
485     .dqs_n   (mcb3_dram_dqs_n),
486     .tdqs_n  (),
487     .odt     (mcb3_dram_odt),
488     .rst_n   (mcb3_dram_reset_n)
489   );
490 end
491 endgenerate

```

- ✓ 라인 447 - 491 : memory model을 생성합니다.

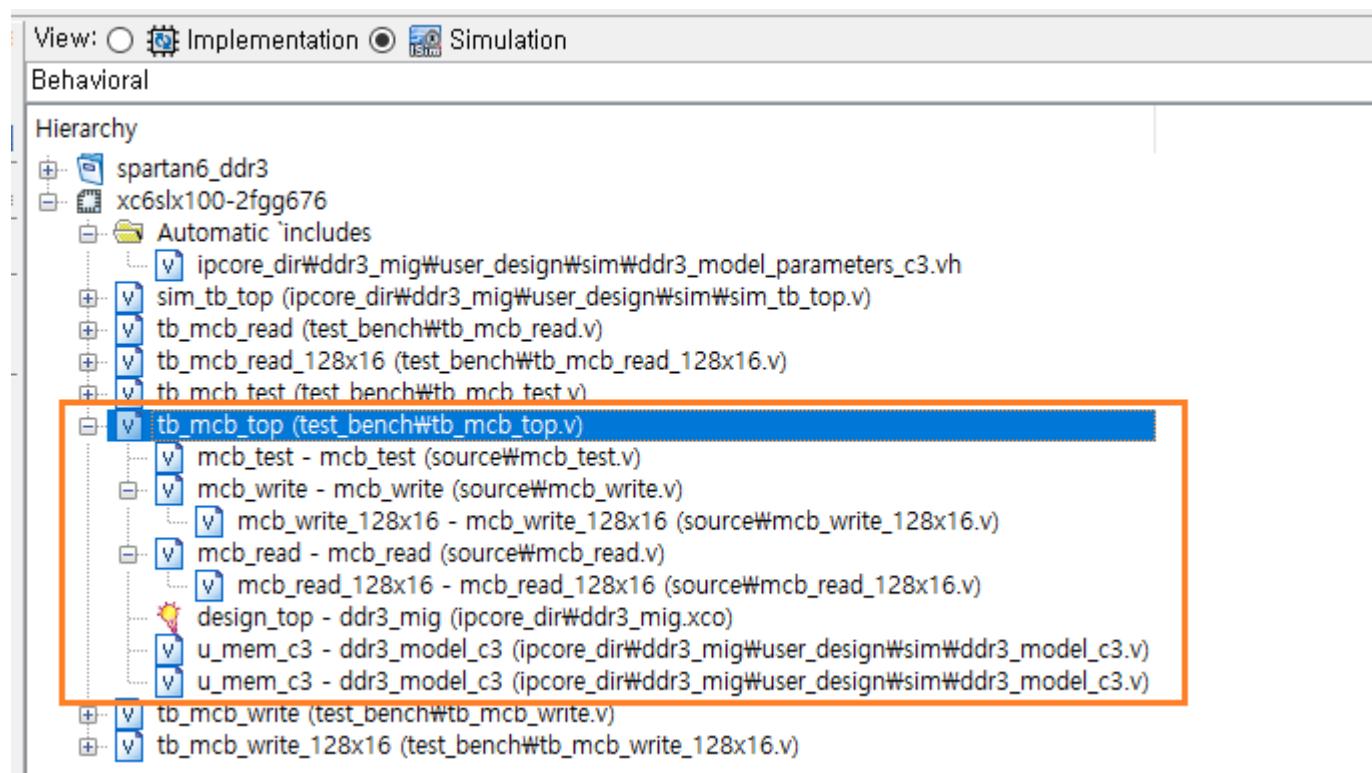
```

493 // ===== //
494 // Reporting the test case status
495 // ===== //
496 initial
497 begin : Logging
498 fork
499 begin : calibration_done
500     wait (calib_done);
501     $display("Calibration Done");
502     #50000000;
503 if (!error) begin
504     $display("TEST PASSED");
505 end
506 else begin
507     $display("TEST FAILED: DATA ERROR");
508 end
509 disable calib_not_done;
510 //$/finish;
511 end
512
513 begin : calib_not_done
514     #200000000;
515 if (!calib_done) begin
516     $display("TEST FAILED: INITIALIZATION DID NOT COMPLETE");
517 end
518 disable calibration_done;
519 $finish;
520 end
521 join
522 end
523
524 endmodule

```

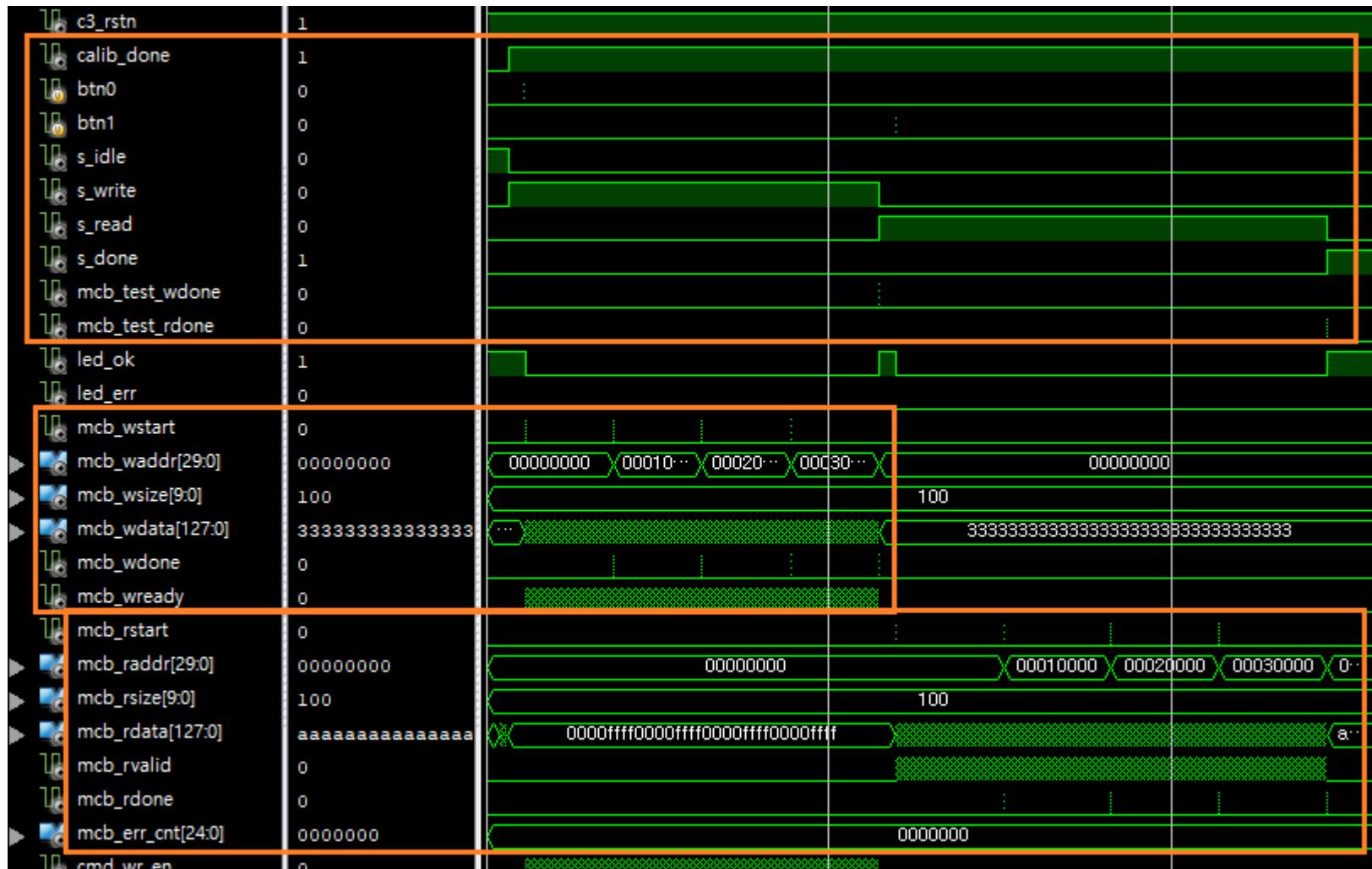
- ✓ 라인 493 - 522 : 상태 메시지를 출력합니다.
- ✓ 라인 510 : calibration 이 완료되면 simulation을 중지하기 때문에 본 라인은 주석 처리합니다.

tb_mcb_top.v 파일을 simulation 파일에 추가합니다. 아래는 파일 구조를 보여줍니다.



Simulation Behavioral Model을 더블 클릭해서 simulation을 진행합니다. “run 1.3ms”을 입력해서 simulation을 진행합니다.

아래는 전체 파형을 보여줍니다.



첫번째 block

- ✓ calib_done이 active 되면 btn0가 active 되면서 wirte 가 진행됩니다.
- ✓ write가 완료되면 mcb_test_wdone이 active 되고, read 가 진행됩니다.
- ✓ read가 완료되면 mcb_test_rdone이 active 되고, done 상태가 됩니다.

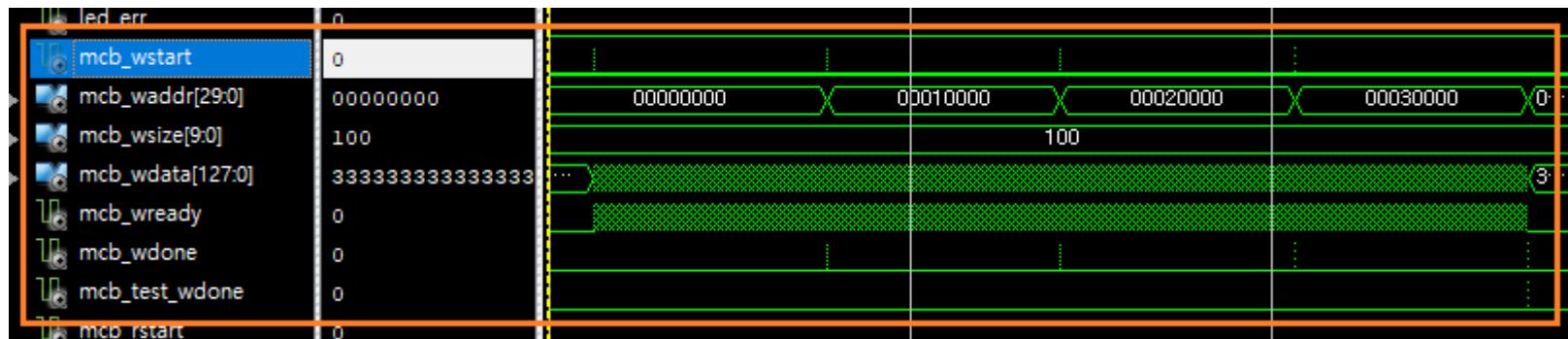
두번째 block

- ✓ mcb_wstart가 4번 active 됩니다.
- ✓ write 과정이 정상적으로 진행됩니다.

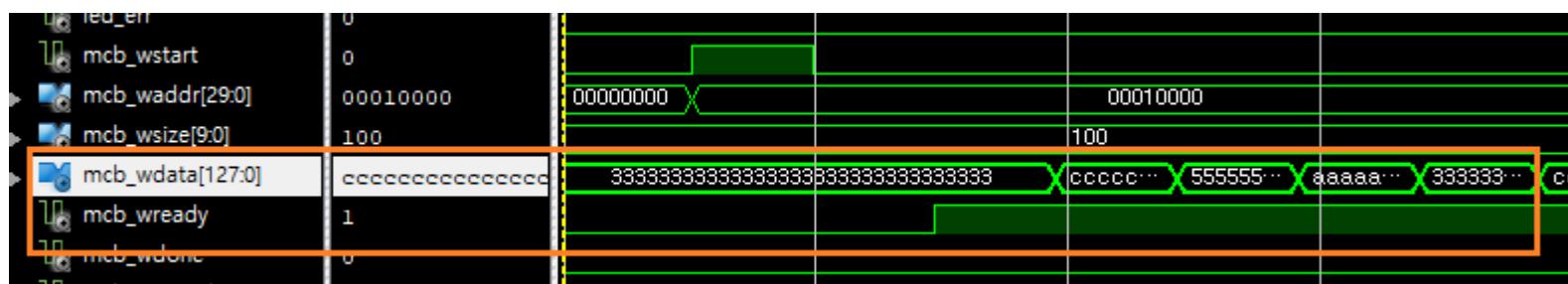
세번째 block

- ✓ mcb_rstart가 4번 active 됩니다.
- ✓ read 과정에서 write data와 read data를 비교하여 error를 count 합니다. mcb_err_cnt 값이 0 임을 알 수 있습니다.

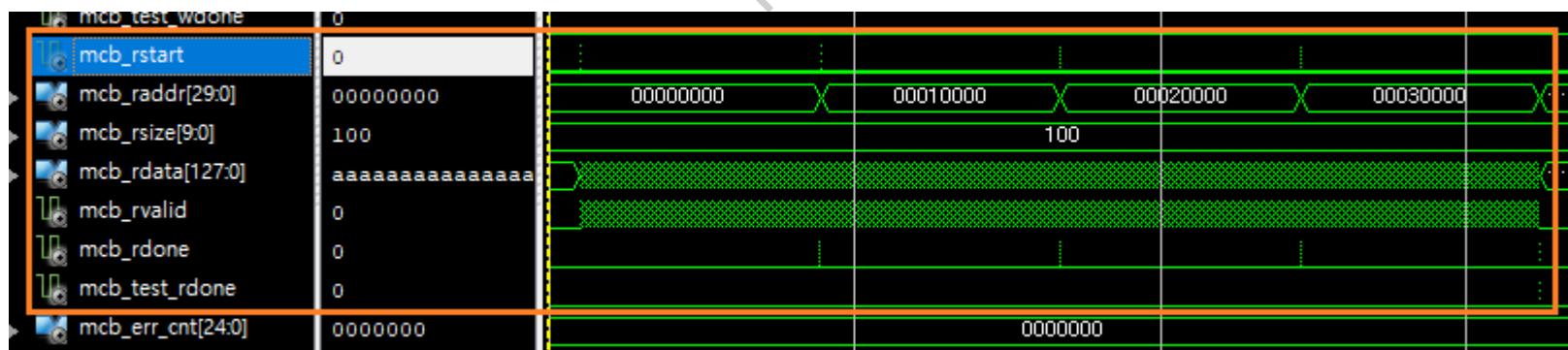
아래 그림은 write 를 확대해서 보여줍니다. mcb_wstart가 4번 발생하고, waddr, wszie, wdata 가 맞게 동작합니다. mcb_wdone, mcb_test_wdone 도 맞게 동작합니다.



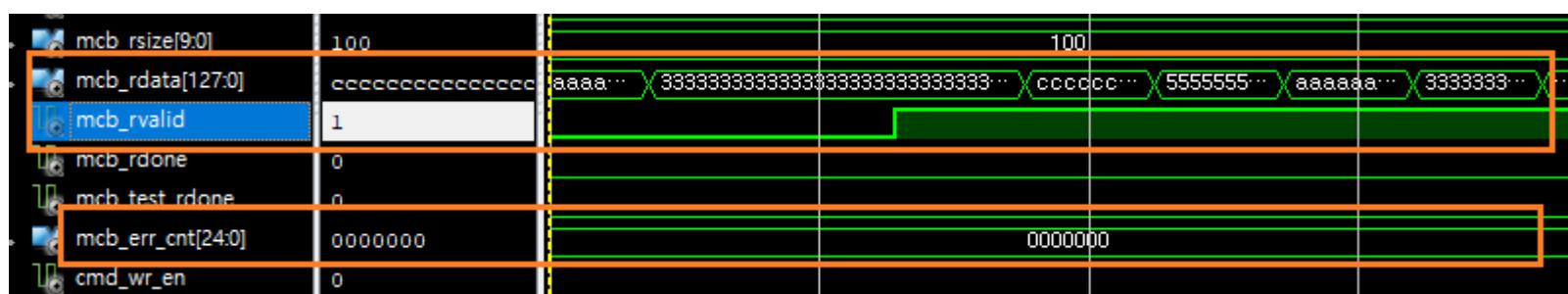
아래는 wdata를 보여줍니다. wready가 active 될 때, 0x333~ , 0xcccc~, 0x555~ , 0xaa~ 순서대로 입력되고 있습니다.



아래는 read를 확대해서 보여줍니다. mcb_rstart가 4번 발생하고, raddr, rsize, mcb_rdone, mcb_test_rdone 신호가 맞게 동작합니다.



아래는 rdata를 확대해서 보여줍니다. rvalid가 active 일 때, 0x333~ , 0xcccc~, 0x555~ , 0xaa~ 순서대로 data가 출력되고 있고, mcb_err_cnt 값이 0임을 보여줍니다.



9.7 결론

지금까지 ISE 14.7 버전에서 Spartan6을 위한 DDR Memory Controller를 구현하였습니다. 본 코드는 프로젝트를 진행하면서 보드에서 검증된 내용을 바탕으로 구성되었습니다.

본 강의의 내용을 모두 이해한다면 DDR Memory Controller는 어떤 환경에서나 구현할 수 있을 것으로 기대합니다.

AIHILL

10. DDR4 Controller

이번 장에서는 DDR4 Controller Interface 를 구현합니다. DDR4 Controller Interface도 DDR3 과 크게 다르지 않습니다. 5장에서 설계한 User Interface 코드를 그대로 사용해서 구현합니다. 5장에서 구현한 코드는 범용으로 사용 가능한 코드이기 때문에, 또한 Xilinx 사에서 제공하는 DDR3, DDR4 Memory Interface가 거의 비슷하기 때문에 그대로 사용할 수 있습니다.

이번 장에서는 Target Board에 DDR4 Memory가 없어서, 자사에서 진행했던 내용을 바탕으로 구현합니다. FPGA는 Artix Ultrascale+를 사용하고, Memory는 MT40A512M16LY-075를 사용합니다. 툴은 Vivado 2023.1 버전을 사용합니다.

10.1 프로젝트 생성

vivado 2023.1을 실행하고 프로젝트를 생성합니다. 프로젝트 name은 ddr4Top 으로 합니다.



Part Number 는 “xau15p-ffvb676-1-2”을 선택합니다.

| Part | I/O Pin Count | Available IOBs | LUT Elements | FlipFlops | Block RAMs | Ultra RAMs | DSPs | HI |
|----------------------|---------------|----------------|--------------|-----------|------------|------------|------|----|
| xau15p-ffvb676-2-e | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |
| xau15p-ffvb676-2-i | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |
| xau15p-ffvb676-1-e | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |
| xau15p-ffvb676-1-i | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |
| xau15p-ffvb676-1L-i | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |
| xau15p-ffvb676-1LV-i | 676 | 228 | 77760 | 155520 | 144 | 0 | 576 | |

10.2 IP 생성

IP Catalog - Memory & Storage Elements - External Memory Interface - DDR4 SDRAM (MIG)을 선택해서 IP를 생성합니다.

The screenshot shows the Xilinx IP Catalog interface. The 'Cores' tab is selected. In the search bar, 'AXI4' is entered. The table lists various IP cores under the 'External Memory Interface' category. The 'DDR4 SDRAM (MIG)' row is highlighted with a blue border, indicating it is selected. The columns in the table are Name, AXI4, Status, License, and VLNV.

| Name | AXI4 | Status | License | VLNV |
|-----------------------------|------|------------|----------|---------------------------|
| >Main Functions | | | | |
| Memories & Storage Elements | | | | |
| Algorithmic CAMs | | | | |
| ECC | | Production | Included | xilinx.com:ip:ecc:2.0 |
| External Memory Interface | | | | |
| DDR3 SDRAM (MIG) | AXI4 | Production | Included | xilinx.com:ip:ddr3:1.4 |
| DDR4 SDRAM (MIG) | AXI4 | Production | Included | xilinx.com:ip:ddr4:2.2 |
| LPDDR3 SDRAM (MIG) | | Production | Included | xilinx.com:ip:lpddr3:1.0 |
| QDRII+ SRAM (MIG) | | Production | Included | xilinx.com:ip:qdriiip:1.4 |
| QDRIV SRAM (MIG) | | Production | Included | xilinx.com:ip:qdriv:2.0 |
| RLDRAM3 (MIG) | | Production | Included | xilinx.com:ip:rld3:1.4 |

Customize IP 윈도가 생성됩니다. 아래와 같이 옵션을 설정합니다.

- ✓ Component Name : mig_ddr4
- ✓ Controller/PHY Mode : Controller and physical layer
- ✓ Memory Device Interface Speed (ps) : 1000 (1000 Mhz로 설정합니다. 938 - 1600 범위에서 설정하면 됩니다)
- ✓ Reference Input Clock Speed (ps) : 5000 (200Mhz)로 설정합니다. (외부에서 100Mhz 입력을 받아 Clock Generator로 200Mhz를 생성하여 입력하거나, 외부에서 Differential Clock 소자를 사용해도 됩니다. 본장에서는 Differential Clock 소자를 사용해서 구현합니다)
- ✓ Memory Part : MT40A512M16LY-075
- ✓ Data Width : 16

나머지는 Default 값을 사용합니다.

Component Name mig_ddr4

Basic Advanced Clocking Advanced Options I/O Planning and Design Checklist

Mode and Interface

Controller/PHY Mode Controller and physical layer AXI4 Interface

Clocking

Memory Device Interface Speed (ps) 1000

(1000 ps = 1000 MHz) Range:[938..1600]
The minimum supported time period for DCI CASCADE is 938 ps

PHY to controller clock frequency ratio 4:1

Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps) 5000 (200Mhz)

Controller Options

Enable Custom Parts Data File

Custom Parts Data File no_file_loaded

A complete list of valid values and sample CSV files can be found [here](#)

Configuration Components

Memory Part MT40A512M16LY-075

Memory Details: 8Gb, x16, Row=16, Column=10, Bank=2, Bank Group=1, Ranks=1, StackHeight=1

Slot Single

IO Memory Voltage 1.2V

Data Width 16

ECC

Data Mask and DBI DM NO DBI

Memory Address Map ROW COLUMN BANK

Ordering Normal

Force Read and Write commands to use AutoPrecharge when Column Address bit A3 is asserted high.

Advanced User Request Controller Options

Enable AutoPrecharge Input

Enable User Refresh and ZQCS Input

Advanced Clock 탭은 아래와 같이 설정합니다.

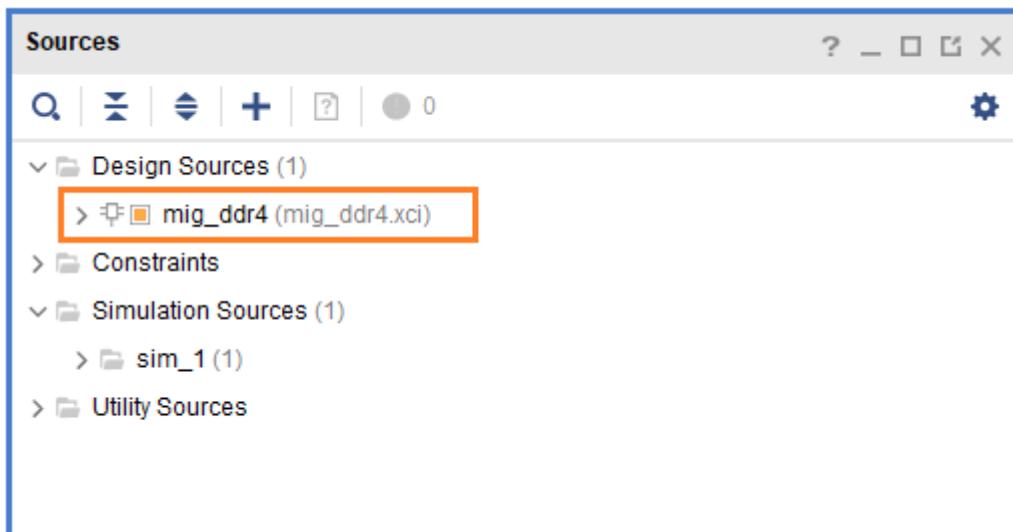
| | |
|----------------------------------|--------------------|
| CLKFBOUT_MULT (M) | 15 |
| DIVCLK_DIVIDE (D) | 2 |
| CLKOUT0_DIVIDE (D0) | 6 |
| VCO (MHz) | 1500.000 |
| Reference Input Clock Speed (ps) | 5000 (200.000 MHz) |
| PFD (MHz) | 100.000 |

- ✓ Reference Input Clock : Differential (외부에서 differential clock 을 사용합니다)
- ✓ Additional Clock Outputs : 총 4개의 Clock을 생성해서 다른 모듈에서 사용할 수 있습니다. 이 Clock은 DDR4 IP Block 에서 사용하는 Clock에서 생성하기 때문에 동기가 맞습니다. DDR4 Block과 동기를 맞추어야 하는 Block 에서 사용하기 위한 clock을 생성합니다. 본장에서는 100Mhz 1개만 사용합니다.

Advanced Options, I/O Planning and Design Checklist는 기본값을 사용합니다.

OK를 클릭해서 IP를 생성합니다

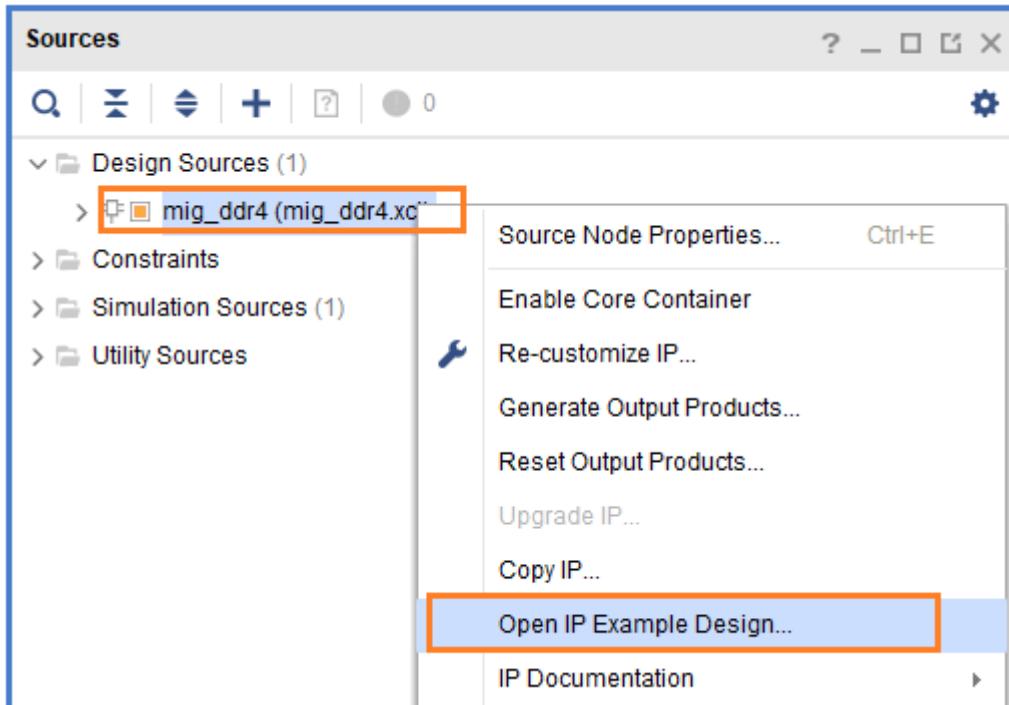
아래는 IP가 생성된 모습을 보여줍니다.



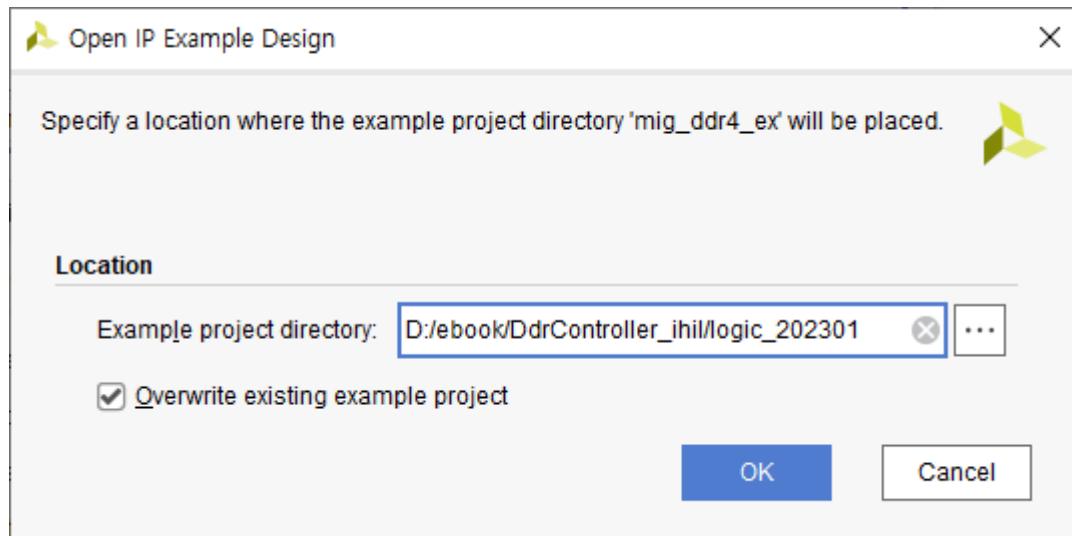
10.3 Simulation

생성된 IP의 동작을 확인하기 위하여 simulation을 진행합니다. 2023.1 버전은 툴이 업데이트 되어서 simulation 과정이 달립니다. 별도의 프로젝트 폴더를 생성해서 simulation을 진행합니다.

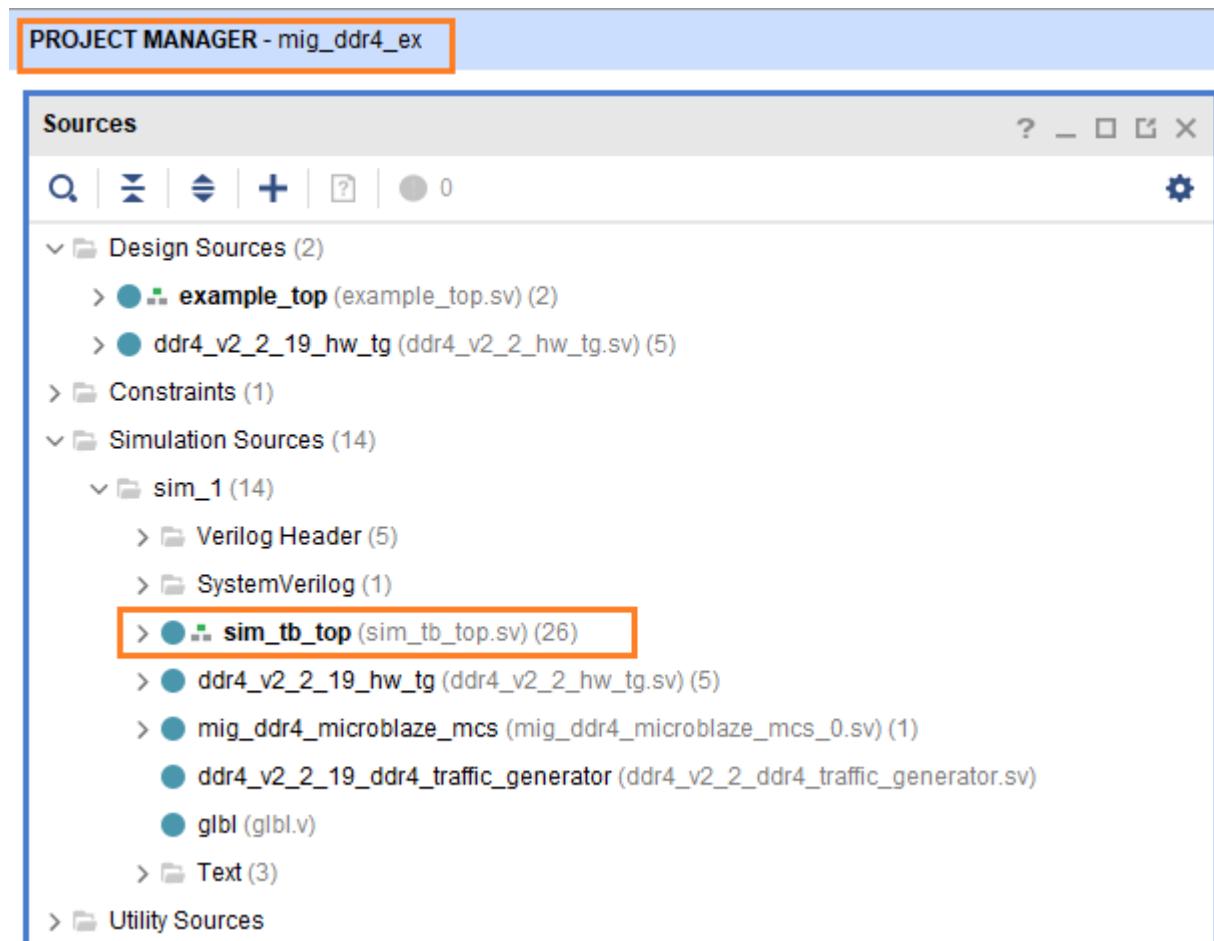
생성된 IP 선택 - 우클릭 - Open IP Example Design... 클릭합니다.



기존의 프로젝트에 “_ex”가 붙은 이름의 프로젝트가 생성됩니다. OK를 클릭합니다.



새로운 vivado 2023.1 이 실행되면서 example 용 프로젝트가 open 됩니다.



simulation Top 모듈은 sim_tb_top 입니다.

SIMULATION - Run Simulation - Run Behavioral simulation 클릭해서 simulation을 진행합니다.

우리가 관심이 있는 app_xxx 관련 신호를 wave 창에 추가합니다. sim_tb_top 아래에 u_example_top 모듈을 선택해서 “Add to Wave Window”를 클릭해서 신호를 추가합니다.

Tcl Console 창에 “run 0.1ms”을 입력하고 엔터키를 누르면 simulation을 진행합니다.

100 writes, 100 reads 후에 simulation이 종료됨을 알 수 있습니다. “TEST PASSED”로 정상적으로 동작함을 알 수 있습니다.

Tcl Console x Messages Log

INFO: [USF-XSim-97] XSim simulation ran for 1000ns

launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:20 . Memory (MB): peak = 2560.469 ; gain = 3.359

current_wave_config {Untitled 2}

Untitled 2

add_wave {{/sim_tb_top/u_example_top}}

run 0.1ms

sim_tb_top.#mem_model_x16.mem.memModelIs_Ri2[0].memModelI2[0].ddr4_model.always_diff_ck.if_diff_ck:Initialization complete @2956250
100 Writes and 100 Reads to the memory completed

TEST PASSED

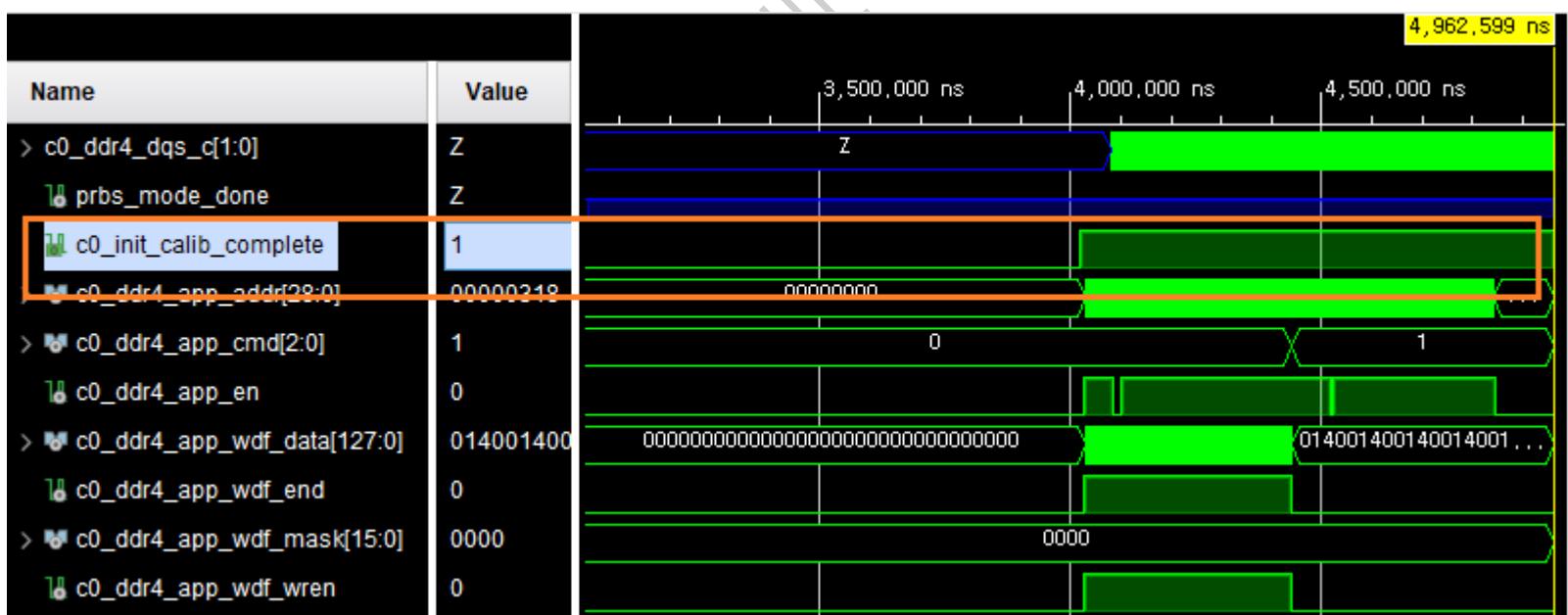
Test Completed Successfully

\$finish called at time : 4962599 ps : File "d:/ebook/DdrController_ihi1/logic_202301/mig_ddr4_ex/imports/example_tb.sv" Line 515

파형을 통하여 몇가지 동작을 확인합니다.

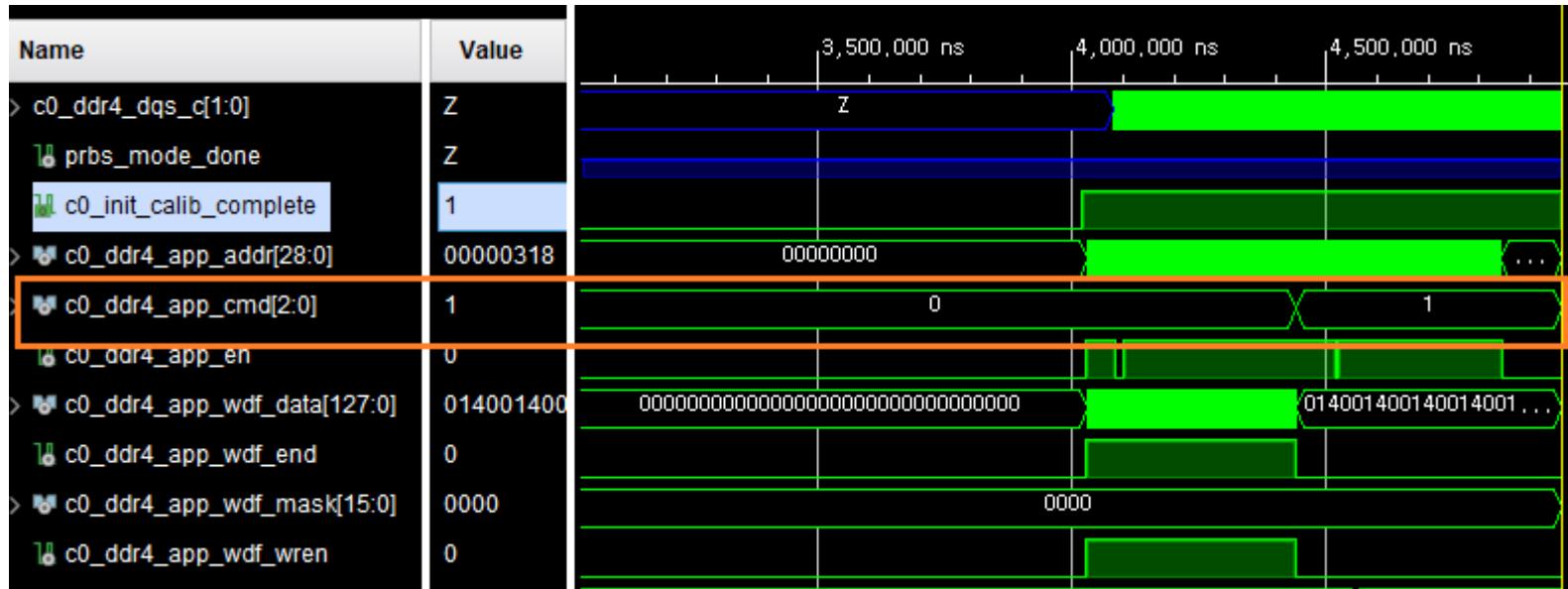
1) init_calib_complete

약 4us 정도에서 1이 됩니다. memory calibration 이 정상적으로 동작합니다.



2) write / read 동작

app_cmd[2:0] 값을 보면, 초기에 0 (write) 동작을 하고, 4.4us 정도에서 1 (read)로 동작함을 알 수 있습니다.



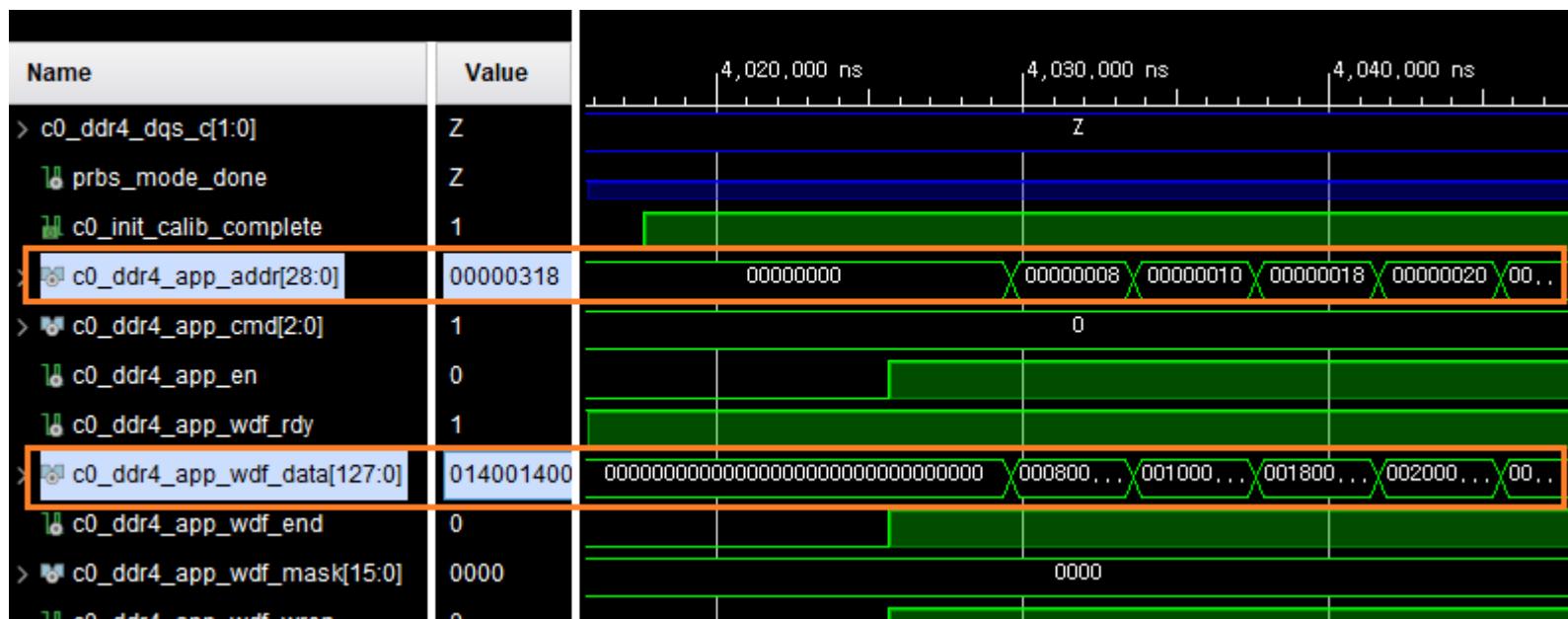
3) write 동작

write 동작은 address : 0x0000_0000 ~ 0x0000_0318 까지 100개를 write 합니다. address와 write data는 아래와 같습니다.

addr : 0x0000_0000 → write data : 0x0000_0000_0000_0000_0000_0000_0000

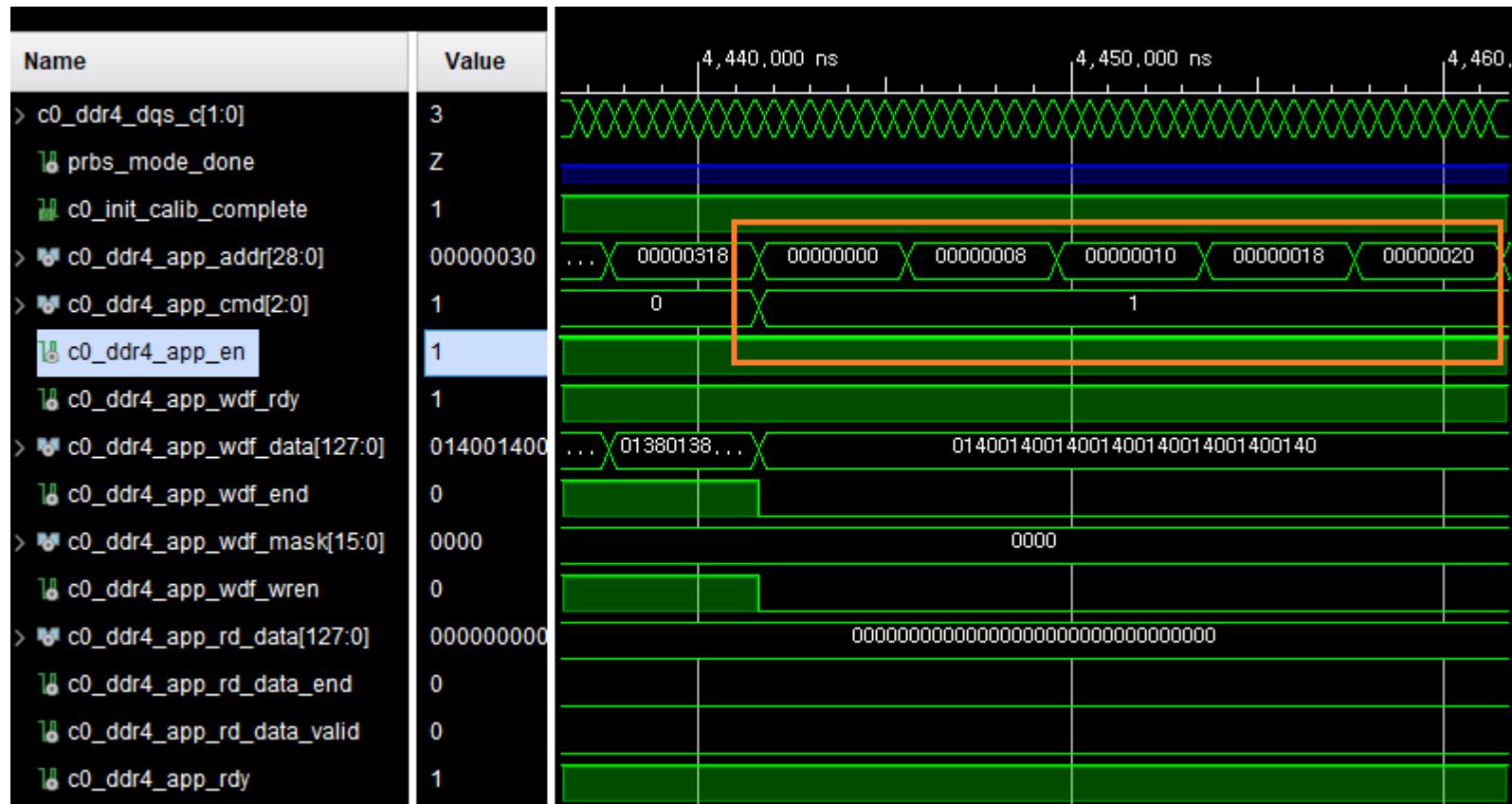
addr : 0x0000_0008 → write data : 0x0008_0008_0008_0008_0008_0008_0008

아래는 write 처음 부분을 보여줍니다.

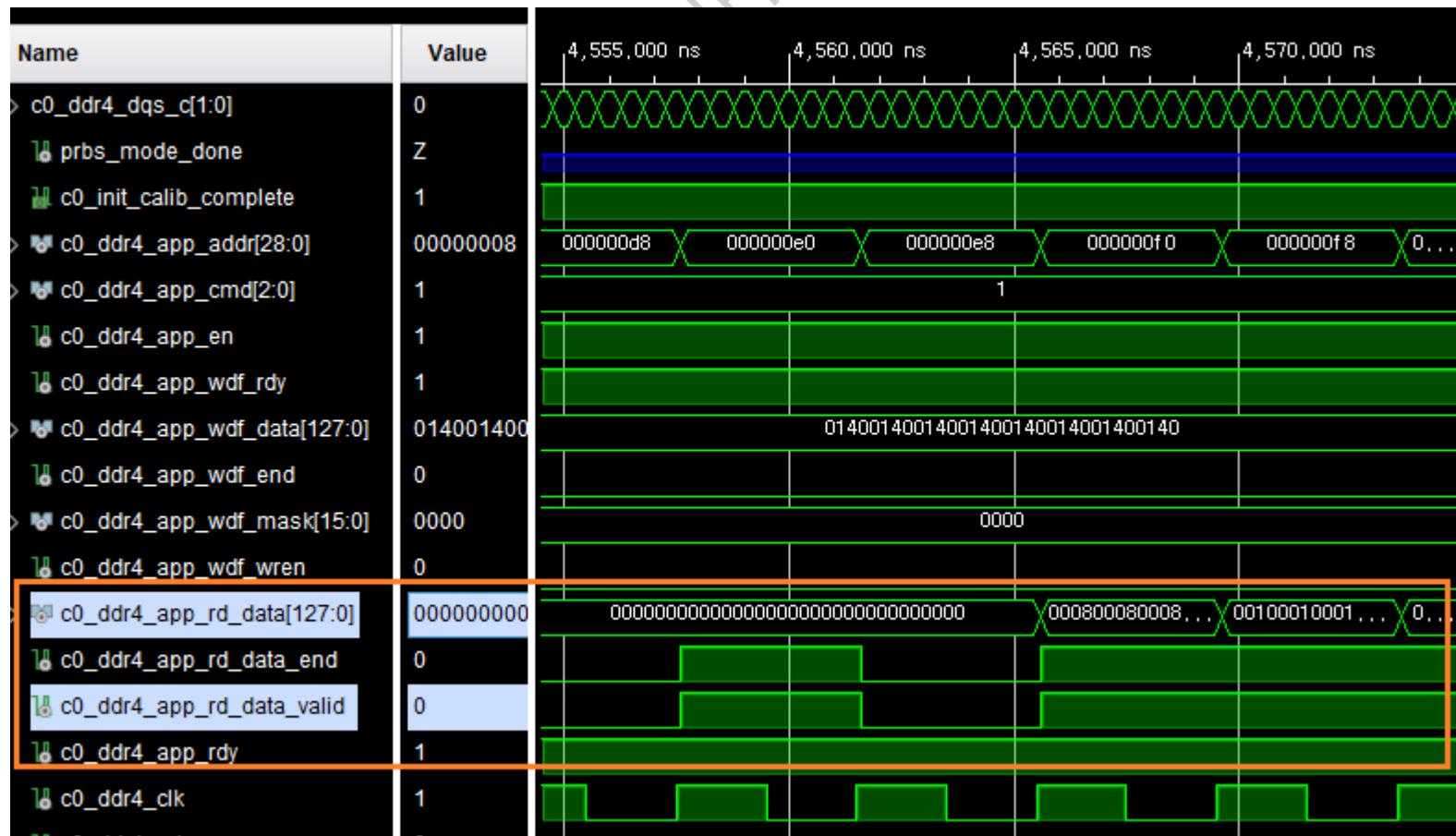


4) read 동작

read 동작은 write 한 데이터를 다시 read 합니다. address : 0x0000_0000 ~ 0x0000_0318까지 read 합니다. 아래는 read 처음 부분입니다. read를 위한 address가 설정됩니다. 0x0000_0000, 0x0000_0008 ~



아래는 read data를 보여줍니다. app_rd_data_valid 가 1인 구간에 app_rd_data가 나옵니다.



ddr memory read / write의 세부 동작은 “3.5장 - 3.6장, 4.3장”을 확인하시길 바랍니다. Xilinx 사에서 제공하는 IP의 구조가 DDR3이나 DDR4나 동일합니다. 따라서 DDR3에서 설계한 코드를 그대로 DDR4에서도 사용할 수 있습니다.

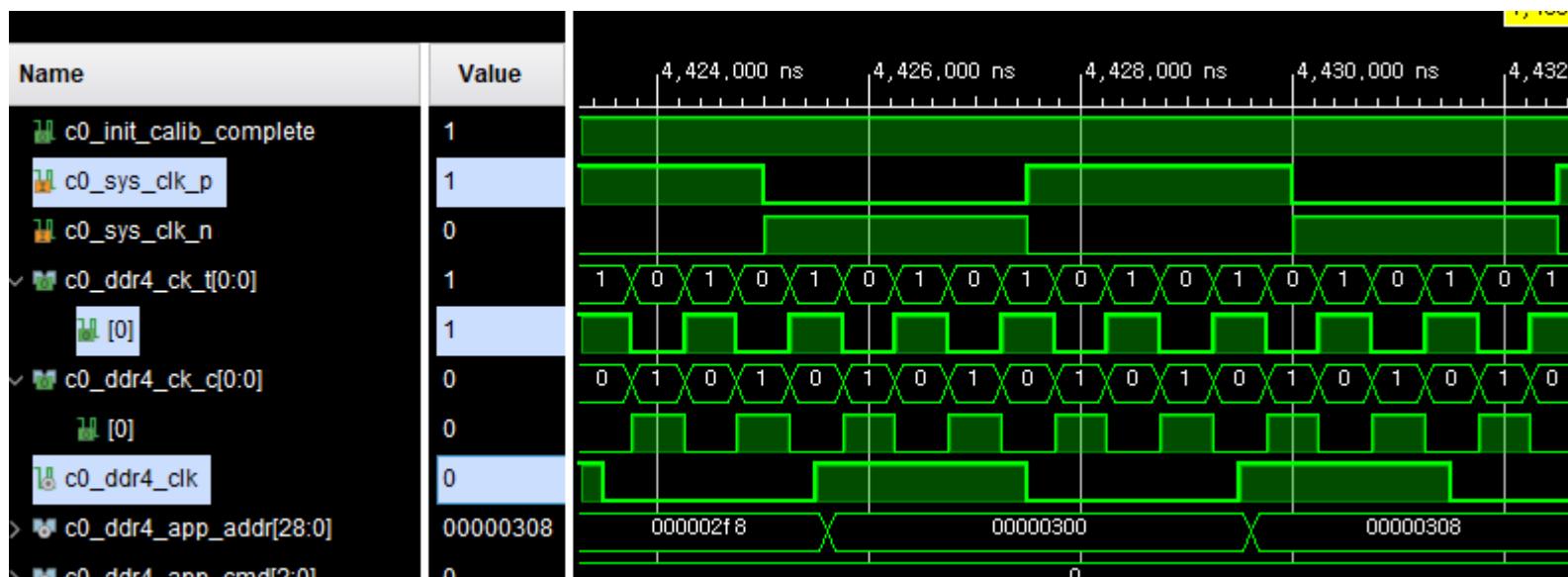
5) clock 구조 분석

Memory IP에서 사용되는 Clock 을 살펴봅니다.

| clock | frequency | description |
|-----------------|-----------|--|
| c0_sys_clk(p/n) | 200 Mhz | ddr4 reference clock, 외부에서 입력됨. differential clock 사용하도록 설정함. |
| c0_ddr4_ck(t/c) | 1000 Mhz | ddr4 memory clock Memory에 출력되는 clock, 1000 Mhz로 설정함. |
| c0_ddr4_clk | 250 Mhz | user interface clock. app_xxx 신호들은 이 clock에 동기되어 동작함. ddr4 memory interface와 연결된 block 들은 이 clock을 사용해서 동작함. |

DDR4 Memory는 1000Mhz, DDR mode, 16bits로 동작합니다. 따라서 User Interface (app_xxx) Block은 128bits로 250Mhz로 동작하게 됩니다. (128bits / 250Mhz \rightarrow 16bits / 2000Mhz)

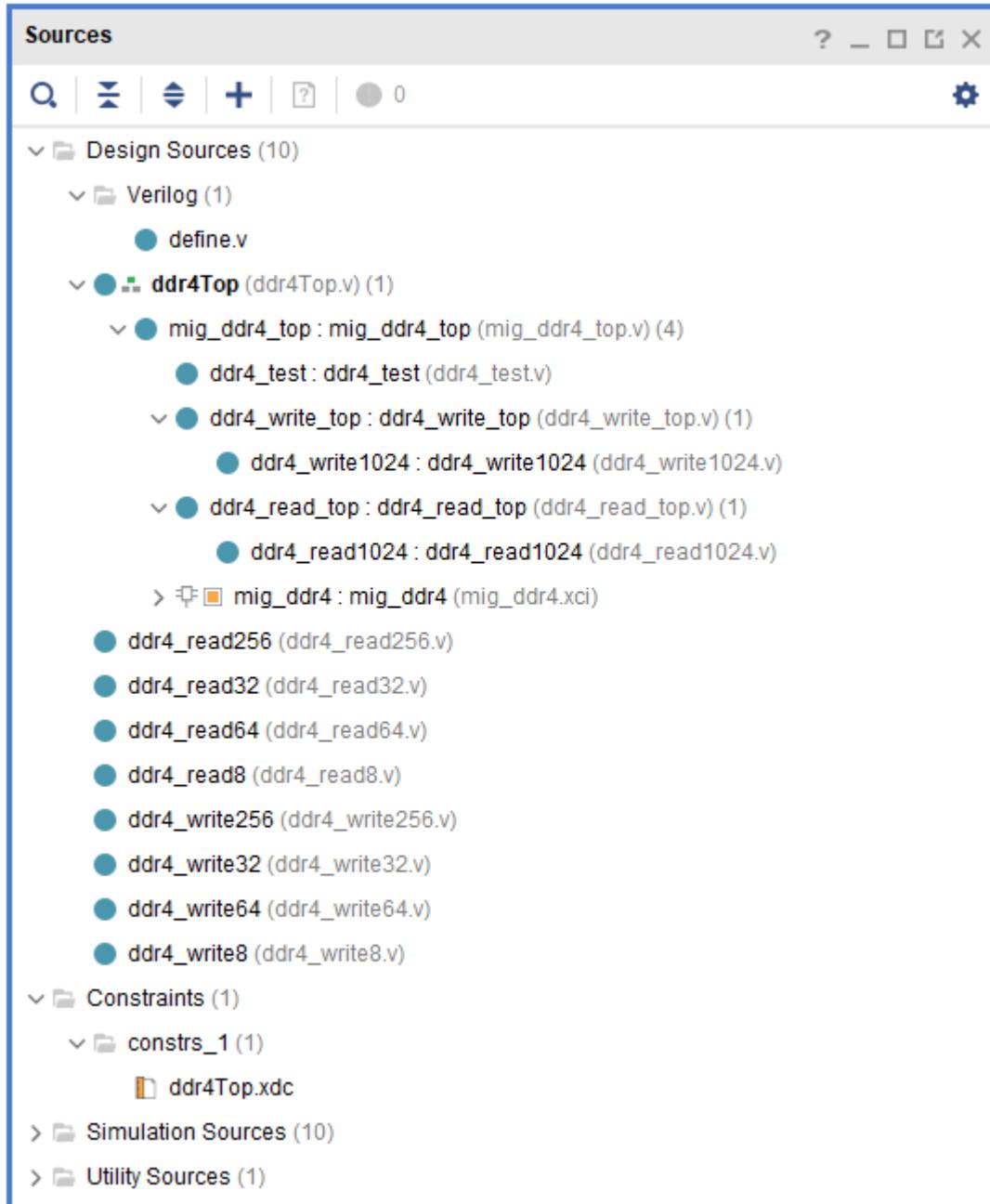
아래는 clock 파형을 보여줍니다.



10.4 코드 구현

이번 장에서는 코드를 구현합니다. 코드는 User Interface 관련 코드는 DDR3에서 구현한 것과 동일한 것을 사용하기 때문에 5-6장에서 구현한 코드를 거의 그대로 사용합니다. bits 수와 module name만 바꾸도록 합니다.

아래는 소스 구조를 보여줍니다.



ddr4Top 모듈이 Top Module입니다. mig_ddr4_top 모듈에 Memory 관련 코드가 들어 있습니다. ddr4_test 모듈은 전 영역 write, read 관련 코드가 있고, ddr4_write_top 모듈은 write 관련 코드가 있고, ddr4_read_top 모듈은 read 관련 코드가 있습니다. mig_ddr4는 Memory IP입니다.

1) ddr4Top module

```

23  module ddr4Top(
24      reset           ;                                // active high
25      sys_clk_p      ;
26      sys_clk_n      ;
27
28      ddr4_adr       ;
29      ddr4_ba        ;
30      ddr4_cke       ;
31      ddr4_cs_n      ;
32      ddr4_dm_dbi_n ;
33      ddr4_dq        ;
34      ddr4_dqs_c     ;
35      ddr4_dqs_t     ;
36      ddr4_odt       ;
37      ddr4_bg        ;
38      ddr4_reset_n   ;
39      ddr4_act_n     ;
40      ddr4_ck_c      ;
41      ddr4_ck_t      ;
42
43      ddr_start      ;
44      led_calib_done ;
45      tp_ddr_write   ;                                // status
46      tp_ddr_read    ;
47      led_ddr_done   ;
48      led_ddr_ok     ;
49      led_ddr_error  ;
50      led_onoffl    ;
51      led_onoff2    ;
52 );
53 input      reset      ;
54 input      sys_clk_p  ;
55 input      sys_clk_n  ;
56
57 output [16:0] ddr4_adr  ;
58 inout  [15:0] ddr4_dq   ;
59 output [1:0]  ddr4_ba   ;
60 output      ddr4_cke   ;
61 output      ddr4_cs_n  ;
62 inout  [1:0]  ddr4_dm_dbi_n ;
63 inout  [1:0]  ddr4_dqs_c  ;
64 inout  [1:0]  ddr4_dqs_t  ;
65 output      ddr4_odt   ;
66 output      ddr4_bg    ;
67 output      ddr4_reset_n ;
68 output      ddr4_act_n  ;
69 output      ddr4_ck_c  ;
70 output      ddr4_ck_t  ;
71
72 input      ddr_start  ;
73 output      led_calib_done ;
74 output      tp_ddr_write   ;                                // status
75 output      tp_ddr_read    ;
76 output      led_ddr_done   ;
77 output      led_ddr_ok     ;
78 output      led_ddr_error  ;
79 output      led_onoffl    ;
80 output      led_onoff2    ;

```

- ✓ 라인 53 - 55 : reset, clock 입니다. sys_clk_p/n은 외부에서 공급되는 200Mhz Differential Clock입니다. 100Mhz Single Ended Clock을 사용해서, Clock Generator를 이용하여 200 Mhz Single Ended Clock 생성 후 사용해도 됩니다. 이때에는 IP 생성할 때, “No Buffer”을 선택합니다.

System Clock Option

| | |
|-------------------------------------|-----------|
| Reference Input Clock Configuration | No Buffer |
|-------------------------------------|-----------|

- ✓ 라인 57 - 70 : ddr4 memory 와 연결되는 신호입니다.
- ✓ 라인 72 - 80 : ddr memory read / write 결과를 보여주는 신호입니다.
 - ddr_start : 외부 버튼에 연결됩니다. 이 버튼을 누르면 ddr4 memory 전 영역 write / read 를 시작합니다.
 - led_calib_done : calibration_complete 을 나타냅니다. calibration이 완료되면 연결된 led가 on 됩니다.
 - tp_ddr_write : 전영역 write 시간을 측정하기 위한 신호입니다. write 중에 enable (1) 됩니다. 스코프로 write 소요시간을 측정합니다.
 - tp_ddr_read : 전영역 write 시간을 측정하기 위한 신호입니다. write 중에 enable (1) 됩니다. 스코프로 read 소요시간을 측정합니다.
 - led_ddr_done : 전영역 write / read 후에 led가 on 됩니다.
- led_ddr_ok / error : 전영역 write / read 후에 verification을 진행합니다. 에러가 없으면 led_ddr_ok가 on 되고, 에러가 발생하면 led_ddr_error 가 on 됩니다.
- led_onoff1 : memory IP에서 생성한 100Mhz에 동기 되어 0.5초 주기로 on/off 됩니다. 보드가 동작하고 있는지 확인하는 용도입니다.
- led_onoff2 : ddr_ui_clk (250Mhz)에 동기 되어 0.5초 주기로 on/off 됩니다. 보드가 동작하고 있는지 확인하는 용도입니다.

```

97 wire      rst_du ;
98 wire      clk_du ;
99
100 wire      clk100      ;
101 wire      led_calib_done      ;
102 wire      tp_ddr_write      ;           // status
103 wire      tp_ddr_read      ;
104 wire      led_ddr_done      ;
105 wire      led_ddr_ok      ;
106 wire      led_ddr_error      ;
107 mig_ddr4_top mig_ddr4_top (
108     .reset      (reset      ),
109     .sys_clk_p      (sys_clk_p      ),
110     .sys_clk_n      (sys_clk_n      ),
111     .rst_du      (rst_du      ),           // user interface reset
112     .clk_du      (clk_du      ),           // user clock
113
114     .ddr4_adr      (ddr4_adr      ),
115     .ddr4_ba      (ddr4_ba      ),
116     .ddr4_cke      (ddr4_cke      ),
117     .ddr4_cs_n      (ddr4_cs_n      ),
118     .ddr4_dm_dbi_n      (ddr4_dm_dbi_n      ),
119     .ddr4_dq      (ddr4_dq      ),
120     .ddr4_dqs_c      (ddr4_dqs_c      ),
121     .ddr4_dqs_t      (ddr4_dqs_t      ),
122     .ddr4_odt      (ddr4_odt      ),
123     .ddr4_bg      (ddr4_bg      ),
124     .ddr4_reset_n      (ddr4_reset_n      ),
125     .ddr4_act_n      (ddr4_act_n      ),
126     .ddr4_ck_c      (ddr4_ck_c      ),
127     .ddr4_ck_t      (ddr4_ck_t      ),
128     .clk100      (clk100      ),
129
130     .ddr_start      (ddr_start      ),
131     .calib_done      (led_calib_done      ),
132     .ddr_sts_write      (tp_ddr_write      ),           // status
133     .ddr_sts_read      (tp_ddr_read      ),
134     .ddr_sts_done      (led_ddr_done      ),
135     .ddr_sts_ok      (led_ddr_ok      ),
136     .ddr_sts_error      (led_ddr_error      )
137 );

```

- ✓ 라인 97 - 137 : mig_ddr4_top모듈을 추가합니다.

```

140  reg      [25:0]  cntl ;
141  always @ (posedge clk100 or posedge reset)
142  begin
143      if(reset)      cntl <= 26'b0;
144      else          cntl <= (cntl==26'd50_000_000) ? 26'b0 : cntl+1'b1;
145  end
146
147
148  reg      led_onoff1 ;
149  always @ (posedge clk100 or posedge reset)
150  begin
151      if(reset)      led_onoff1 <= 1'b0;
152      else          led_onoff1 <= (cntl==26'd50_000_000) ? ~led_onoff1 : led_onoff1;
153  end
154
155  reg      [26:0]  cnt2 ;
156  always @ (posedge clk_du or posedge rst_du)
157  begin
158      if(rst_du)      cnt2 <= 27'b0;
159      else          cnt2 <= (cnt2==27'd125_000_000) ? 26'b0 : cnt2+1'b1;
160  end
161
162
163  reg      led_onoff2 ;
164  always @ (posedge clk_du or posedge rst_du)
165  begin
166      if(rst_du)      led_onoff2 <= 1'b0;
167      else          led_onoff2 <= (cnt2==27'd125_000_000) ? ~led_onoff2 : led_onoff2;
168  end
169
170
171 endmodule

```

- ✓ 라인 140 - 153 : 100Mhz clock으로 led_onof1을 0.5초 간격으로 on/off 합니다.
- ✓ 라인 155 - 168 : 250Mhz clock으로 led_onof2를 0.5초 간격으로 on/off 합니다.

2) mig_ddr4_top module

ddr4_test, ddr4_write_top, ddr4_read_top, mig_ddr4 모듈을 추가합니다. mig_ddr4 모듈은 아래의 파일을 참조합니다.

| ddr4Top > ddr4Top.gen > sources_1 > ip > mig_ddr4 | | | |
|---|---------------------|--------------------|--|
| 이름 | 수정한 날짜 | 유형 | |
| bd_0 | 2025-01-02 오후 9:43 | 파일 폴더 | |
| doc | 2025-01-02 오후 9:43 | 파일 폴더 | |
| ip_0 | 2025-01-02 오후 9:43 | 파일 폴더 | |
| ip_1 | 2025-01-02 오후 9:43 | 파일 폴더 | |
| par | 2025-01-04 오후 12:09 | 파일 폴더 | |
| rtl | 2025-01-02 오후 9:43 | 파일 폴더 | |
| sim | 2025-01-02 오후 9:43 | 파일 폴더 | |
| sim_tlm | 2025-01-02 오후 9:43 | 파일 폴더 | |
| sw | 2025-01-02 오후 9:43 | 파일 폴더 | |
| tb | 2025-01-04 오후 12:09 | 파일 폴더 | |
| mig_ddr4.dcp | 2025-01-02 오후 9:49 | Vivado Checkpoi... | |
| mig_ddr4.veo | 2025-01-02 오후 9:43 | VEO 파일 | |
| mig_ddr4.vho | 2025-01-02 오후 9:43 | VHO 파일 | |

3) ddr4_test module

```

25  module ddr4_test (
26      reset      ,           // ui_clk_sync_rst from ddr controller
27      mclk       ,           // ui_clk from ddr controller
28      calib_done ,
29      ddr_start   ,
30
31      wstart      ,           // ddr write ...
32      waddr       ,
33      wsize       ,
34      wdata       ,
35      wready      ,
36      wdone       ,
37
38      rstart      ,           // ddr read ...
39      raddr       ,
40      rsize       ,
41      rdata       ,
42      rdata_valid ,
43      rdone       ,
44
45      ddr_sts_write ,        // status
46      ddr_sts_read  ,
47      ddr_sts_done  ,
48      ddr_sts_ok   ,
49      ddr_sts_error
50  );

```

- ✓ 라인 25 - 50 : module의 in/out을 정의합니다.

```

111 parameter      test_data1 = 128'h33333333_33333333_33333333_33333333 ;
112 parameter      test_data2 = 128'hcccccccc_cccccccc_cccccccc_cccccccc ;
113 parameter      test_data3 = 128'h55555555_55555555_55555555_55555555 ;
114 parameter      test_data4 = 128'haaaaaaaa_aaaaaaaaa_aaaaaaaaa_aaaaaaaaa ;
115
116 // State Parameter
117 parameter      IDLE     = 3'd0;
118 parameter      W_READY  = 3'd1;
119 parameter      WRITE1   = 3'd2;
120 parameter      WRITE2   = 3'd3;
121 parameter      R_READY  = 3'd4;
122 parameter      READ1   = 3'd5;
123 parameter      READ2   = 3'd6;
124 parameter      DONE    = 3'd7;

```

- ✓ 라인 111 - 114 : test data는 4개의 데이터를 순차적으로 write 합니다. write 가 완료되면 read 하면서 write한 데이터와 read한 데이터가 같은지를 check 합니다.
- ✓ 라인 117 - 124 : state machine 을 나타냅니다.

```

305   always @ (posedge mclk or posedge reset)
306   begin
307     if(reset)      begin
308       ddr_state <= 3'b0;
309     end
310     else begin
311       ddr_state <= (s_idle & calib_done ) ? W_READY :
312           (s_wready & (cnt_wready==10'd1000)) ? WRITE1 :
313           (s_write1 & (cnt_writel==3'd4) ) ? WRITE2 :
314           (s_write2 & wdone ) ? ((w_block==BLOCK_MAX) ? R_READY : WRITE1) :
315           (s_rready & (cnt_rready==10'd1000)) ? READ1 :
316           (s_read1 & (cnt_readl==3'd4) ) ? READ2 :
317           (s_read2 & rdone ) ? ((r_block==BLOCK_MAX) ? DONE : READ1) :
318           (s_done & ddr_start ) ? IDLE :
319           ddr_state ;
320     end
321   end

```

- ✓ 라인 305 - 321 : state 이동을 보여줍니다. idle 상태에서 calibration이 완료되면 wready로 이동합니다. 1000 clock 후에 write1으로 이동합니다. wrtie1 에서는 write 위한 신호를 생성합니다. write2 상태로 이동해서 순차적으로 data를 write 합니다. 전영역 write가 완료되면 rready 상태로 이동합니다. 1000 clock 후에 read1 상태로 이동합니다. read1에서 read를 위한 신호를 생성하고 read2 상태로 이동해서 순차적으로 data를 read 하면서 에러를 확인합니다. 자세한 사항들은 “5 User Interface Logic 구현”을 참조하시길 바랍니다.

4) ddr4_write_top

memory write를 진행합니다. 자세한 사항은 “5 User Interface Logic 구현”을 참조하시길 바랍니다.



5) ddr4_read_top

memory read를 진행합니다. 자세한 사항은 “5 User Interface Logic 구현”을 참조하시길 바랍니다.

10.5 Address 관련

Memory의 Address를 좀 더 분석해 보도록 하겠습니다. 아래는 사용하는 메모리의 스펙을 보여줍니다.

| DDR4 SDRAM PART | | | | | |
|----------------------------|----------------|------------------------|------------------|-------------------|----------------------------------|
| MT40A512M16LY-075:E | | | | | |
| Specifications | | | | | |
| Component Density | Speed | MT/s | I/O Voltage | Operating Temp | Bus Width |
| 8Gb | 1333MHz | 2666MTPS | 1.2 VOLTS | OC to +95C | x16 |
| CAS Latency | Pin Count | Part Status Code | Component Config | Dry Pack Qty | Package Dimension (W x L x H) mm |
| CL = 19 | 96-ball | Contact Factory | 512M x16 | 1080 | 7.50 x 13.50 x 1.20 |
| Tape & Reel Qty | Package Type | Number of Components | Package | Family | Technology |
| 2000 | GREEN | 1 | TFBGA | DRAM | DDR4 SDRAM |

(참조 : <https://www.micron.com/products/memory/dram-components/ddr4-sdram/part-catalog/part-detail/mt40a512m16ly-075-e>)

총 8Gb, 512M x 16 bits 입니다. address (app_addr[28:0]) 범위는 0x0000_0000 ~ 0x1FFF_FFFF 입니다. user interface는 data를 128bits씩 access 합니다. 128bits씩 access 할 때, address는 128/16 = 8 씩 증가하게 됩니다.

ddr4_write1024 (or ddr4_read1024) 모듈은 128bits x 1024 x wsize (or rsize) 만큼 access 합니다.

wsize (or rsize, BLOCK_SIZE) = 2로 설정되어 있기 때문에 ddr4_write1024 (or ddr4_read1024)을 실행할 때마다, 16364 (8 x 1024 x 2) address 만큼 실행하게 됩니다.

전영역을 write / read 하기 위해서는 ddr4_write1024 (or ddr4_read1024)를 총 512M / 16K = 32768 번 실행해야 합니다.

아래 코드는 ddr_test.v 의 내용입니다.

```

78  `ifdef RUN_MODE
79    parameter BLOCK_MAX = 15'd32767;
80
81  `ifndef DDR4_STEP_128x8
82    parameter BLOCK_SIZE = 10'd256;
83  `elsif DDR4_STEP_128x32
84    parameter BLOCK_SIZE = 10'd64;
85  `elsif DDR4_STEP_128x64
86    parameter BLOCK_SIZE = 10'd32;
87  `elsif DDR4_STEP_128x256
88    parameter BLOCK_SIZE = 10'd8;
89  `elsif DDR4_STEP_128x1024
90    parameter BLOCK_SIZE = 10'd2;
91  `endif

```

DDR4_STEP_128x1024 를 사용할 때, BLOCK_SIZE (wsize / rsize)는 2로 설정되어 있고, ddr4_write1024 (or ddr4_read1024)를 총 BLOCK_MAX (32767) 번 실행하면 전영역을 write / read 하게 됩니다.

아래 코드는 ddr4_write1024 (or ddr4_read1024)을 실행할 때, 처음 address를 설정하는 waddr, raddr을 보여줍니다. ddr4_write1024 (or ddr4_read1024)을 실행할 때마다, 16384씩 증가함을 알 수 있습니다.

```

168  reg [14:0] w_block;
169  always @ (posedge mclk or posedge reset)
170  begin
171    if(reset)      w_block <= 15'b0;
172    else          w_block <= s_idle ? 14'b0 : wdone ? w_block+1'bl : w_block;
173  end
174
175  reg [28:0] waddr;
176  always @ (posedge mclk or posedge reset)
177  begin
178    if(reset)      waddr <= 29'b0;
179    else          waddr <= s_idle ? 29'b0 : (cnt_writel==3'd3) ? {w_block, 14'b0} : waddr ;
180  end
181
182  reg [14:0] r_block;
183  always @ (posedge mclk or posedge reset)
184  begin
185    if(reset)      r_block <= 15'b0;
186    else          r_block <= s_idle ? 15'b0 : rdone ? r_block+1'bl : r_block;
187  end
188
189  reg [28:0] raddr;
190  always @ (posedge mclk or posedge reset)
191  begin
192    if(reset)      raddr <= 29'b0;
193    else          raddr <= s_idle ? 29'b0 : (cnt_readl==3'd3) ? {r_block, 14'b0} : raddr ;
194  end

```

10.6 xdc 생성

이번 장에서는 xdc을 생성합니다.

```

2 ######
3 # BANK_64, HP
4 set_property -dict {PACKAGE_PIN AD20 IOSTANDARD DIFF_SSTL12} [get_ports sys_clk_p]
5 set_property -dict {PACKAGE_PIN AE20 IOSTANDARD DIFF_SSTL12} [get_ports sys_clk_n]
6
7 #####
8 #####
9 # BANK_85, HD
10 set_property PACKAGE_PIN A10 [get_ports reset]
11 set_property IOSTANDARD LVCMOS33 [get_ports reset]
12 set_property PULLDOWN true [get_ports reset]
13
14 set_property -dict {PACKAGE_PIN D9 IOSTANDARD LVCMOS33} [get_ports {ddr_start}]
15 set_property -dict {PACKAGE_PIN D10 IOSTANDARD LVCMOS33} [get_ports {led_calib_done}]
16 set_property -dict {PACKAGE_PIN D11 IOSTANDARD LVCMOS33} [get_ports {tp_ddr_write}]
17 set_property -dict {PACKAGE_PIN E10 IOSTANDARD LVCMOS33} [get_ports {tp_ddr_read}]
18 set_property -dict {PACKAGE_PIN E11 IOSTANDARD LVCMOS33} [get_ports {led_ddr_done}]
19 set_property -dict {PACKAGE_PIN F9 IOSTANDARD LVCMOS33} [get_ports {led_ddr_ok}]
20 set_property -dict {PACKAGE_PIN F10 IOSTANDARD LVCMOS33} [get_ports {led_ddr_error}]
21 set_property -dict {PACKAGE_PIN G9 IOSTANDARD LVCMOS33} [get_ports {led_onoff1}]
22 set_property -dict {PACKAGE_PIN G10 IOSTANDARD LVCMOS33} [get_ports {led_onoff2}]
23
24 #####
25 #####
26 # BANK_64, HP VCCO - VCC1V2
27 #####
28
29 set_property -dict {PACKAGE_PIN AF24 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[0]}]
30 set_property -dict {PACKAGE_PIN AB25 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[1]}]
31 set_property -dict {PACKAGE_PIN AB26 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[2]}]
32 set_property -dict {PACKAGE_PIN AC24 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[3]}]
33 set_property -dict {PACKAGE_PIN AF25 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[4]}]
34 set_property -dict {PACKAGE_PIN AB24 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[5]}]
35 set_property -dict {PACKAGE_PIN AD24 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[6]}]
36 set_property -dict {PACKAGE_PIN AD25 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[7]}]
37 set_property -dict {PACKAGE_PIN AB21 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[8]}]
38 set_property -dict {PACKAGE_PIN AE21 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[9]}]
39 set_property -dict {PACKAGE_PIN AE23 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[10]}]
40 set_property -dict {PACKAGE_PIN AD23 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[11]}]
41 set_property -dict {PACKAGE_PIN AC23 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[12]}]
42 set_property -dict {PACKAGE_PIN AD21 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[13]}]
43 set_property -dict {PACKAGE_PIN AC22 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[14]}]
44 set_property -dict {PACKAGE_PIN AC21 IOSTANDARD POD12_DCI} [get_ports {ddr4_dq[15]}]
```

Bank 64는 sys_clock (200M, differential), ddr 관련 신호가 연결되었습니다. VCCO는 1.2V를 사용합니다.

그 외의 신호들은 3.3V, Bank 85를 사용하였습니다.

10.7 결과 확인

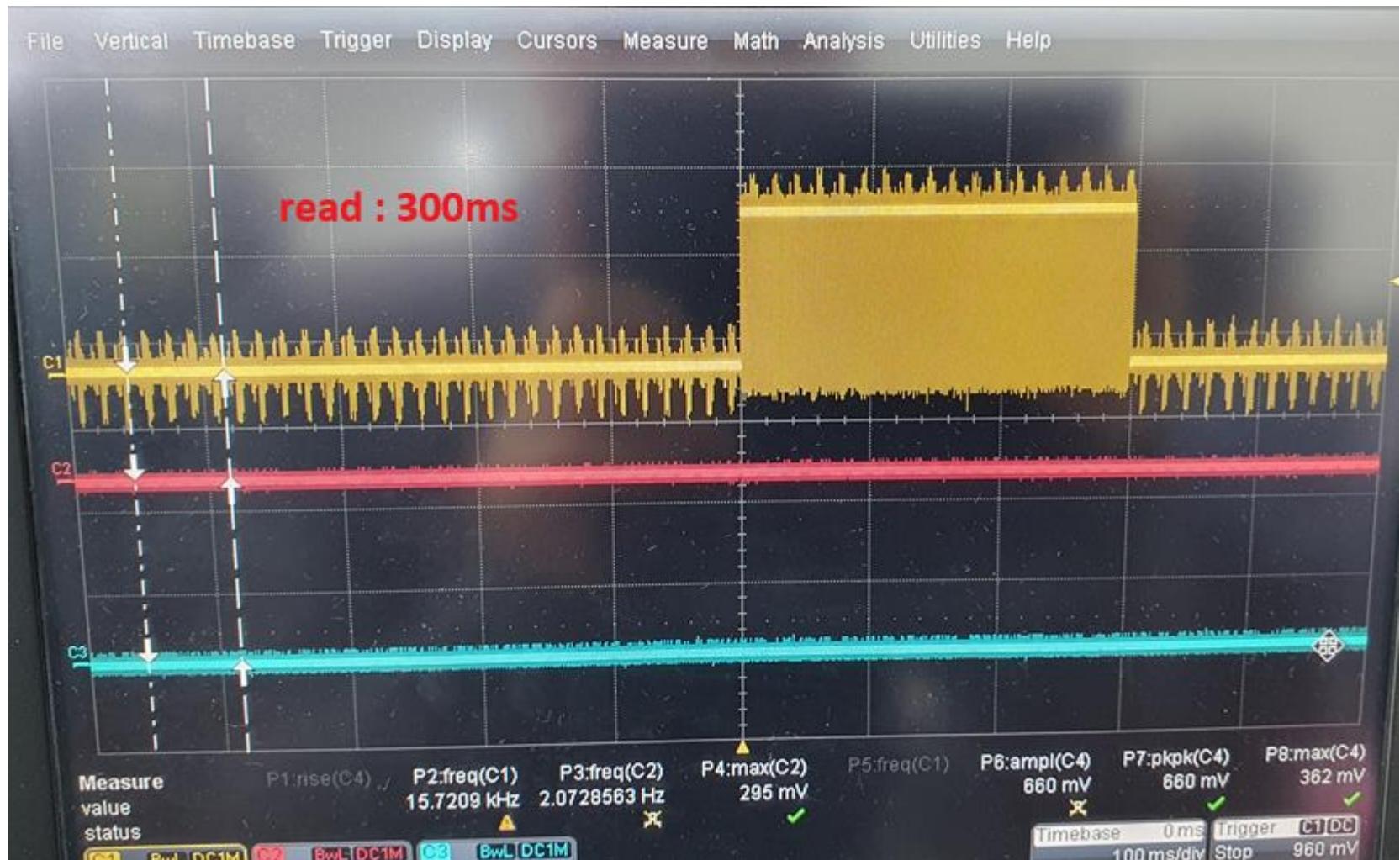
이번장에서는 자사에서 진행한 결과를 보여줍니다. 스코프로 전영역 write / read 하는데 소요되는 시간을 측정하였습니다.

아래는 tp_ddr_write 신호를 측정한 것을 보여줍니다.



전영역을 write 하는데 소요되는 시간이 약 340ms로 측정되었습니다.

아래는 tp_ddr_read 신호를 측정한 것을 보여줍니다.



전역 read하는데 약 300ms 정도 소요되었습니다.

이론적으로 계산해 보면,

$$0x2000_0000 \times (1 / (1000M \times 2)) = 0.268s = 268ms$$

입니다. (0x2000_0000 : 총 address, 1000M = Memory Clock, 2 : ddr)

340ms, 300ms 정도면 최대 소요시간 (268ms)와 거의 비슷한 속도로 write / read가 구현된 것을 알 수 있습니다.

10.8 결론

이번 장에서는 DDR4 Memory Interface를 구현하였습니다. Xilinx사에서 제공하는 Memory Interface IP의 구성이 DDR3이나 DDR4나 거의 동일합니다. 따라서 DDR3에서 구현하였던 User Interface 코드를 그대로 사용할 수 있었습니다. 특별히 Artix UltraScale+ 와 MT40A51216LY에서 코드를 구현하고 결과를 확인하였습니다.

DDR3, DDR4 Memory Interface는 본 강의에서 설명한 내용을 이해하면 어떠한 과제도 충분히 해결할 수 있을 것으로 기대합니다. 저도 본 강의에서 사용된 코드를 실무에서 그대로 사용해서 프로젝트를 진행하고 있습니다. DDR에 연관된 모든 프로젝트는 본 강의에서 사용된 코드를 그대로 사용하고 있습니다.

본 강의를 통하여 DDR Memory Interface를 이해하고 실무에 활용할 수 있길 기대합니다.

/HIL

11. 참고 자료

- 1) Zynq-7000 Soc and 7 Series Devices Memory Interface Solution v4.2 (Xilinx)
- 2) MT41K128M16 DDR3L SDRAM Datasheet (Micron)
- 3) Arty A7 Schematic (Digilent)
- 4) Spartan-6 FPGA Memory Controller (UG388, v2.3)

<https://cafe.naver.com/worshippt> 카페의 전자문서 - DDR Controller 방에 질문을 남겨주시면 답변해 드리도록 하겠습니다.

감사합니다~

/HIL

12. Revision History

| Date | Version | Description of Revisions |
|--------------|---------|------------------------------------|
| 2022. 05. 30 | 1.0 | Initial Release |
| 2022. 11. 30 | 2.0 | 전면 개정 |
| 2023. 06. 05 | 2.1 | 2. HW 구성 |
| 2023. 10. 25 | 2.2 | 7. Spartan6 DDR Controller 구현 추가됨. |
| 2024. 05. 01 | 2.3 | 8. DDR3 Memory Access 속도 추가됨 |
| 2024. 07. 06 | 2.4 | 9. 32Bits Interface 구현 |
| 2024. 09. 22 | 2.6 | 목차 수정 |
| 2025. 01. 04 | 2.7 | 10장 DDR4 Controller 추가 |

IHL

[저작권 관련 내용]

이 자료는 대한민국 저작권법의 보호를 받습니다. 작성된 모든 내용의 권리는 작성자에게 있으며, 작성자의 동의 없는 사용이 금지됩니다.

본 자료의 일부 혹은 전체 내용을 무단으로 복제/배포하거나 2차적 저작물로 재편집하는 경우, 5년 이하의 징역 또는 5천만원 이하의 벌금과 민사상 손해배상을 청구합니다.

※ 저작권법 제 30조(사적이용을 위한 복제)

공표된 저작물을 영리를 목적으로 하지 아니하고, 개인적으로 이용하거나 가정 및 이에 준하는 한정된 범위 안에서 이용하는 경우에는 그 이용자는 이를 복제할 수 있다. 다만, 공중의 사용에 제공하기 위하여 설치된 복사기기에 의한 복제는 그러지 아니하다.

※ 저작권법 제 136조(벌칙)의 ① 다음 각 호의 어느 하나에 해당하는 자는 5년 이하의 징역 또는 5천만원 이하의 벌금에 처하거나 이를 병과할 수 있다.

지적재산권 및 이 법에 따라 보호되는 재산적 권리(제93조에 따른 권리는 제외한다)를 복제, 공연, 공중 송신, 전시, 배포, 대여, 2차적 저작물 작성의 방법으로 침해한 자

※ 민법 제750조(불법행위의 내용)

고의 또는 과실로 인한 위법행위로 타인에게 손해를 가한 자는 그 손해를 배상할 책임이 있다.