

- Subject : Lab1 Use Vivado to build an Embedded System
- Created date : 2017. 06. 01
- Recently modified date : -
- Creator : Choi Jun-Ho
- Contact : peluza6332@gmail.com
- Worked dates to make working of subject : 2017. 06. 01
- Documentation status : **finished**
- Summary
: Use Vivado to build an Embedded System 개인 한글화 문서

- Contents
 - Introduction
 - Objectives
 - Procedure
 - Design Description
 - General Flow for this Lab
 - 1. Create a Vivado Project
 - 2. Creating the System Using the IP Integrator
 - 3. Generate Top-Level and Export to SDK
 - 4. Generate Memory TestApp in SDK
 - 5. Test in Hardware
 - Conclusion

- Introduction

This lab guides you through the process of using Vivado to create a simple ARM Cortex-A9 based processor design targeting the ZedBoard or Zybo board. Where the instructions refer to both boards, choose the board you are using. **You will use Vivado to create the hardware system and SDK (Software Development Kit) to create an example application to verify the hardware functionality.**

- Objectives

After completing this lab, you will be able to:

- Create a Vivado project for a **Zynq system**(PS + PL)
- Use the IP Integrator to create a hardware system(**make for PL**)
- Use SDK to create a standard memory test project
- Run the test application on the board

- Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow.

이 랩은 세부적인 지침에 대한 정보를 제공하는 일반적인 개요 문으로 구성된 단계로 구분됩니다.

Follow these detailed instructions to progress through the lab.

This lab comprises 5 primary steps:

You will create a top-level project using Vivado,
create the processor system(PS) using the Vivado IP Integrator,
generate the top-level HDL(hardware description language) and
export the design to SDK.
create a Memory Test application in SDK,
and finally, test in hardware.

PS를 만들고 HDL로 따로 만든다는 뜻인가?

IPI로 PS와 필요한 PL 부분을 설계하고 HDL로 Generate 한다는 뜻인듯?

- Design Description

The purpose of the lab exercises is to walk you through a complete hardware and software processor system design. Each lab will build upon the previous lab.**(이 큰 랩에 대해 각 세부 랩들이 이전 랩들에 이어서 한다는 것)**

The following diagram represents the completed design (Figure 1).

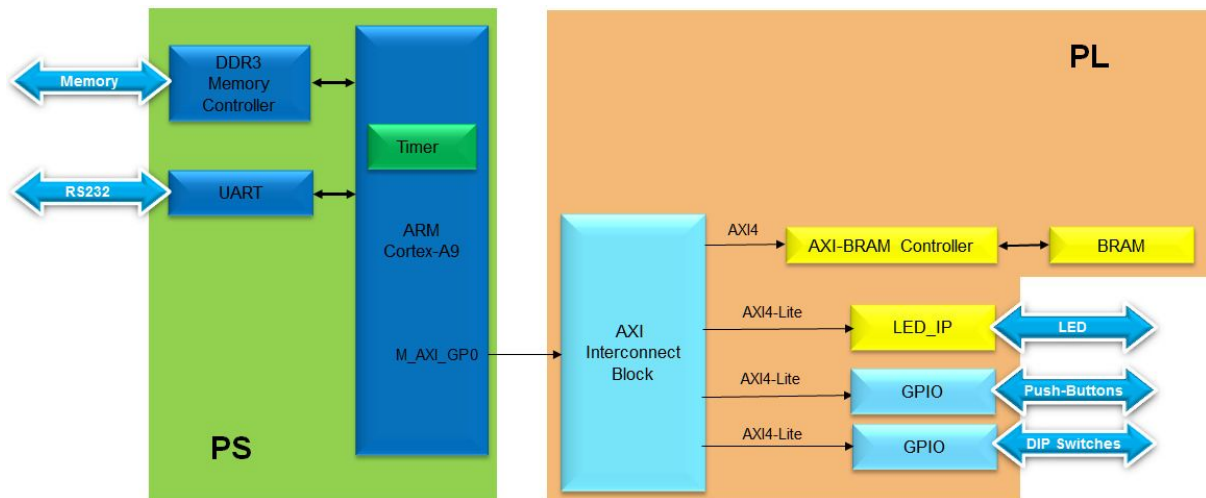


Figure 1. Completed Design

Memory Controller와 UART를 사용한다.

Memory Controller는 아마도 위에서 Memory Test가 있으므로 그것 때문에 있는 것이고 UART는 이런 Memory Test 등의 결과를 통신하여 콘솔 창에서 보여주기 위한 것.

M_AXI_GP0 이것이 AXI 인터페이스의 버스인 것 같은데 M이나 GP0이 의미하는 바는 아직 모르겠다.

마찬가지로 아직 Timer를 어떻게 하면 정확히 쓸 수 있는지도 모른다.

AXI Interconnect Block은 PS에 있는 것 아닌가?

AXI-BRAM : DRAM도 아니고 BRAM?

AXI4-Lite는 보통 수준의 또는 저속의 통신 속도이며 이게 있다는 건 고속 통신이 가능한 모드가 있다는 것이지. 또 현재 이것을 사용하는 IP들이 굳이 고속으로 통신할 필요가 없거나 이 정도 속도로도 충분하다는 것을 의미하고.

LED_IP는 읽혀진 값을 통해 불을 밝히기 때문에 이렇게 돼 있고, BTN, SWT는 입력을 받아야 하기 때문에 GPIO로

In this lab, you will use IP Integrator to create a processing system based design consisting of the following (Figure 2):

- ARM Cortex A9 core (PS)
- UART for serial communication
- DDR3 controller for external DDR3_SDRAM memory

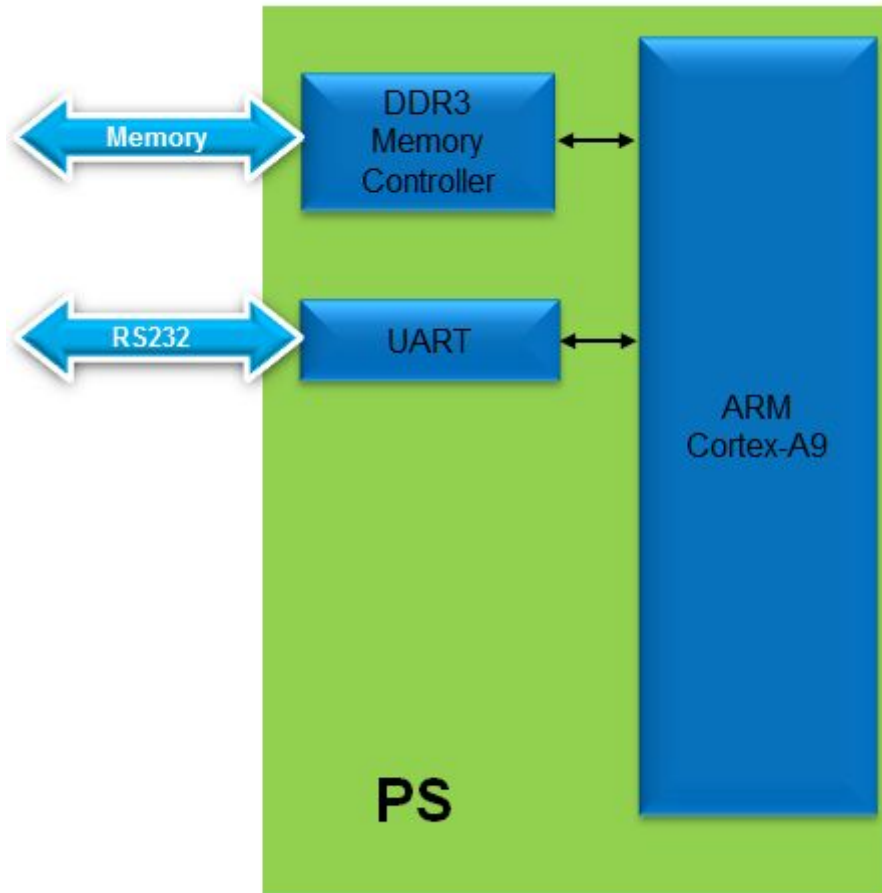


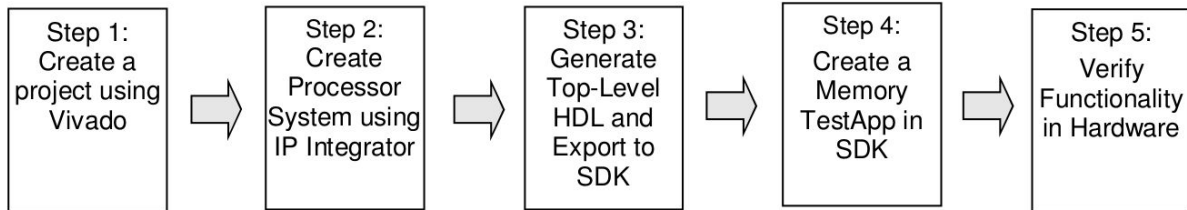
Figure 2. Processor Design of this Lab

이 랩에서, IPI로 만든 PS는 위(Figure 2)와 같은 형태로 되어 있다.

DDR Controller, UART, ARM(PS)를 사용.

아니 그런데 PS를 만드는 데 위 세 가지를 쓴다고 되어 있는데 ARM core에도 괄호 치고 PS가 쓰여있네?

- General Flow for this Lab



In the instructions below;

{sources} refers to: C:\xup\embedded\2015_2_zynq_sources

{labs} refers to : C:\xup\embedded\2015_2_zynq_labs

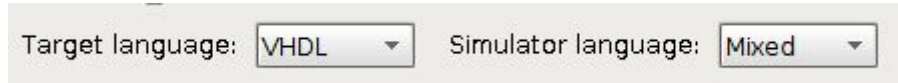
Board support for the Zybo is not included in Vivado 2015.2 by default. The relevant files “zybo.zip” need to be extracted and saved to: {Vivado installation}\data\boards\board_files\zynq

These files can be downloaded from either from the Digilent, Inc. webpage (<http://www.digilentinc.com/>) or the XUP webpage (<http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-embedded-design-flow-zynq.html>) where this material is also hosted.

<http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-embedded-design-flow-zynq.html>

- 1. Create a Vivado Project

1-1. Launch Vivado and create an empty project targeting ~~the ZedBoard~~ or the Zybo, using the VHDL language.



- 2. Creating the System Using the IP Integrator
 - Use the IP Integrator to create a new Block Design, add the ZYNQ processing system block, and import the provided xml file for the board.

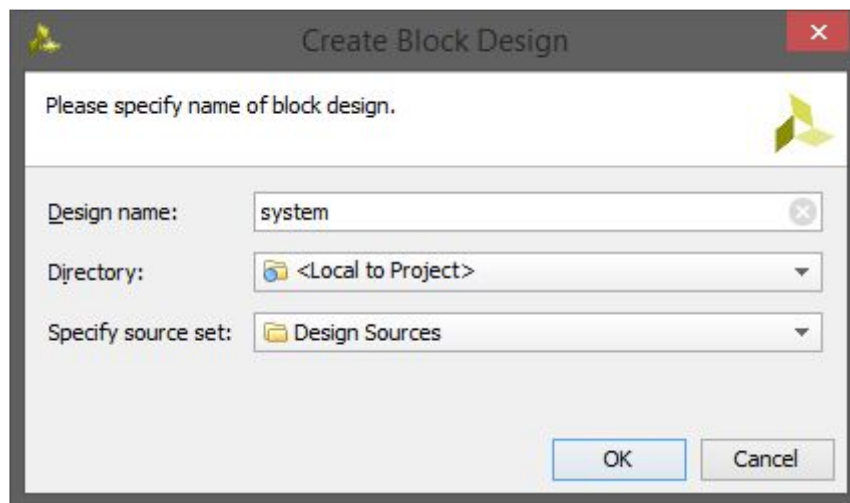


Figure 3. Project Name Entry

보통 이렇게 Block Design에서 Design의 이름은 system이다.

IP from the catalog can be added in different ways. Click the Add IP icon in the block diagram side bar, press Ctrl + I, click the Add IP icon in the empty Diagram workspace, or right-click anywhere in the Diagram workspace and select Add IP.

Add IP를 하는 세 가지 방법들

Notice the message at the top of the Diagram window that Designer Assistance available. Click Run Block Automation and select /processing_system7_0

IP를 추가하면 상단 Designer Assistance가 끈다. Run Block Automation 한다.

In the Run Block Automation window, leave the default settings, including Apply Board Preset checked, and click OK

default로 OK

Once Block Automation has been complete, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible.

Block 자동 완성이 완료되면 DDR가 Fixed IO를 위해 몇 가지 포트들이 자동으로 추가된다.

Fixed IO가 뭘까?

그리고 그 외에도 몇 가지 추가적인 포트들을 볼 수 있다.

TTC0_WAVE0_OUT 등이 뭔지 모르지만 WAVE는 어떤 뿔다 내렸다 하는 이산적인 파형과 관련된 것 같고 TTC는 Triple Timer Counter인 것 같으니까 타이머 관련된 포트인 것 같다.

FCLK_CLK0, FCLK_RESET0_N은 Clock 관련된 것 같은데 둘다 확실히 아는 건 아니다. 다만 RESET0은 Clock 초기화와 관련됐을 것 같고

Double-click on the added block to open its Customization window.

IP Block을 더블 클릭해서 수동 조작한다.

Notice now the Customization window **shows selected peripherals (with tick marks)**. This is the default configuration for the board applied **by the block automation**.

이 IP Block에 대한 설정들은 block automation에 의해 기본 설정돼 있다.

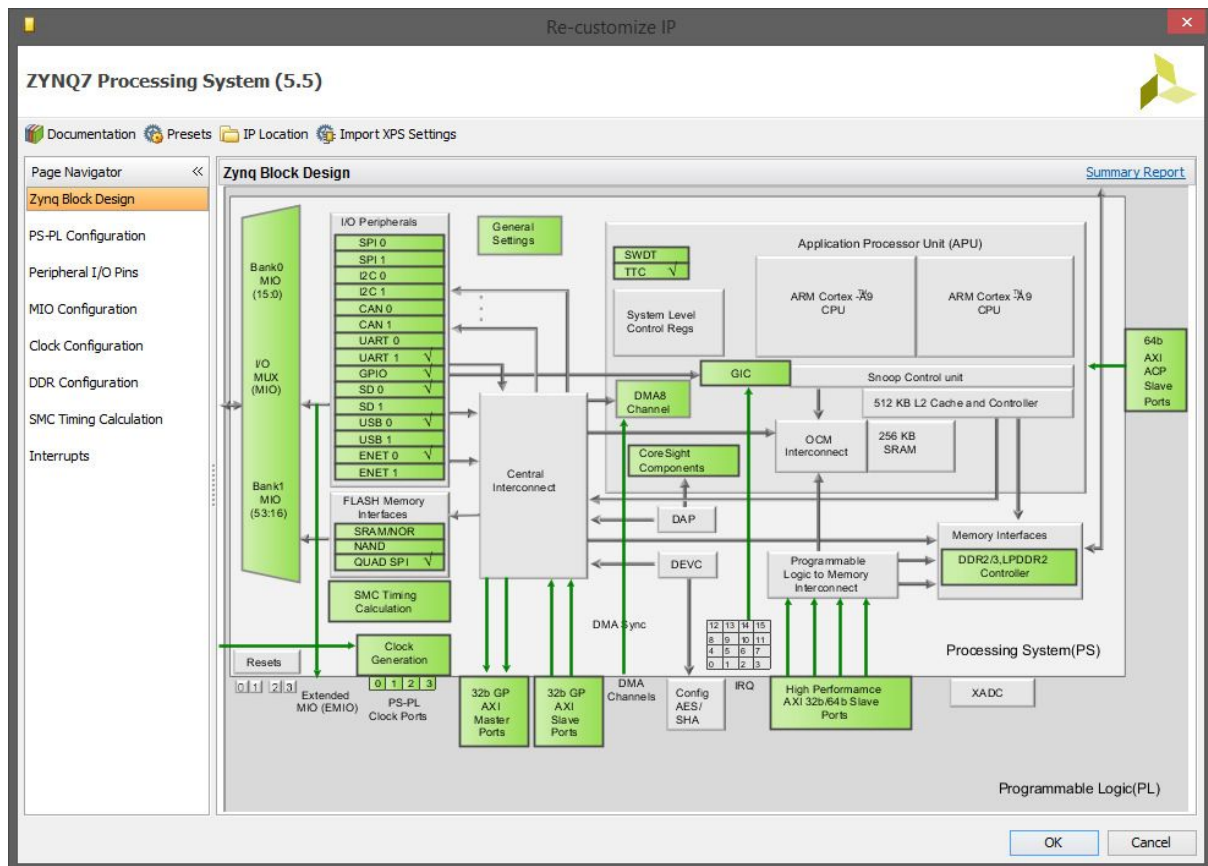


Figure 13. Imported peripherals settings

- **Configure the processing block with just UART 1 peripheral enabled.**

A block diagram of the Zynq should now be open again, showing various configurable blocks of the Processing System.

At this stage, the designer **can click** on various configurable blocks (**highlighted in green**) and **change the system configuration**.

Only the UART is required for this lab, **so all other peripherals will be deselected**.

Click on one of the peripherals (in green) in the IOP Peripherals block, or select the MIO Configuration tab on the left to open the configuration form

녹색의 IOP Peripherals block에서 클릭하거나 MIO Configuration tab을 선택한다.

Expand I/O peripherals if necessary, and ensure all the following I/O peripherals are deselected except UART 1.

i.e. Remove:

ENET 0

USB 0

SD 0

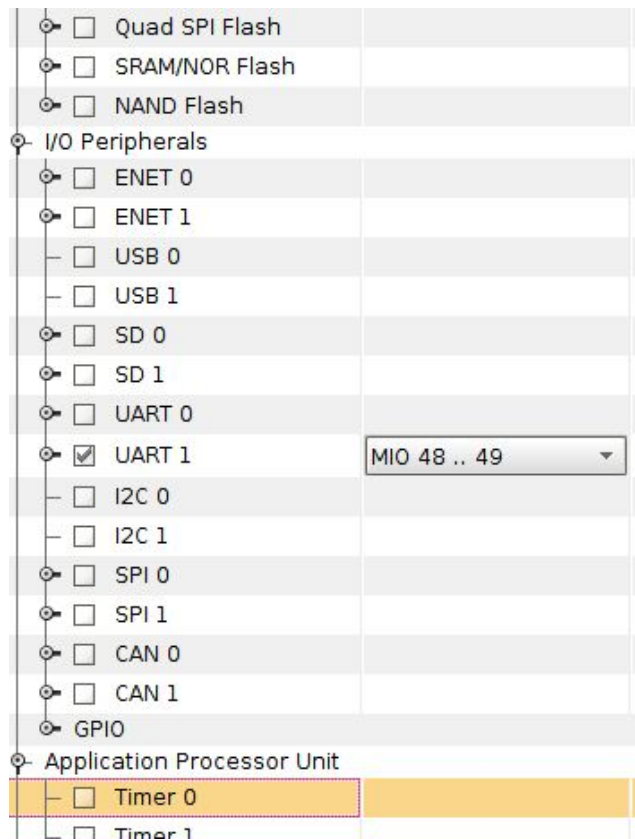
Expand GPIO to deselect GPIO MIO

Expand Memory Interfaces to deselect Quad SPI Flash

Expand Application Processor Unit to disable Timer 0.

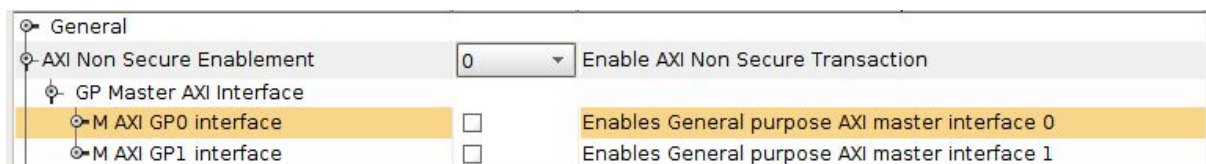
<input checked="" type="checkbox"/>	Quad SPI Flash	MIO 1 .. 6
<input type="checkbox"/>	SRAM/NOR Flash	
<input type="checkbox"/>	NAND Flash	
I/O Peripherals		
<input checked="" type="checkbox"/>	ENET 0	MIO 16 .. 27
<input type="checkbox"/>	ENET 1	
<input checked="" type="checkbox"/>	USB 0	MIO 28 .. 39
<input type="checkbox"/>	USB 1	
<input checked="" type="checkbox"/>	SD 0	MIO 40 .. 45
<input type="checkbox"/>	SD 1	
<input type="checkbox"/>	UART 0	
<input checked="" type="checkbox"/>	UART 1	MIO 48 .. 49
<input type="checkbox"/>	I2C 0	
<input type="checkbox"/>	I2C 1	
<input type="checkbox"/>	SPI 0	
<input type="checkbox"/>	SPI 1	
<input type="checkbox"/>	CAN 0	
<input type="checkbox"/>	CAN 1	
<input type="checkbox"/>	GPIO	
Application Processor Unit		
<input checked="" type="checkbox"/>	Timer 0	EMIO

에서

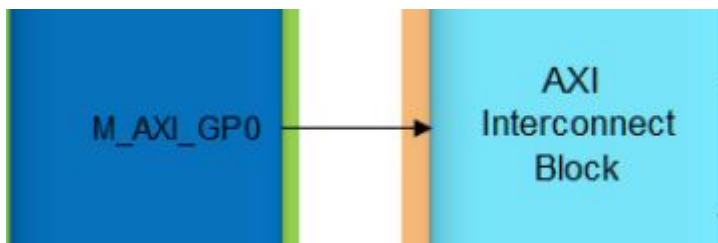


Select the PS-PL Configuration tab on the left.

Expand AXI Non Secure Enablement > GP Master AXI interface and deselect M AXI GP0 interface.



위에 AXI_GP0을 사용하는 것 같은데 왜 꺼버릴까?



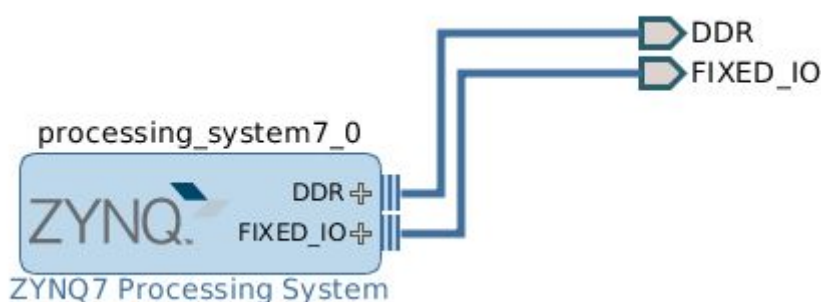
Expand General > Enable Clock Resets and deselect the FCLK_RESET0_N option.

General		
UART0 Baud Rate	115200	Baud rate is generated with internally fixed UART Ref Clock Fr...
UART1 Baud Rate	115200	Baud rate is generated with internally fixed UART Ref Clock Fr...
PL AXI idle Port	<input type="checkbox"/>	Enables idle AXI signal to the PS used to indicate that there ...
DDR ARB bypass Port	<input type="checkbox"/>	Enables DDR urgent/arb signal used to signal a critical memo...
PS-PL Debug interface	<input type="checkbox"/>	Enables PL debug signals to PS and vice-versa
FTM Trace data interface	<input type="checkbox"/>	Enables FTM Trace AXI stream interface used to capture dat...
FTM Trace buffer	0	Generates a FIFO to hold trace data
FTM Data edge detector	0	Stores trace data in the FIFO when the data changes as mar...
FTM Trace buffer FIFO size	128	FTM Trace buffer FIFO size
FTM Trace buffer clock delay	12	Number of clock cycles interval for a trace data output from ...
Include ACP transaction checker	<input type="checkbox"/>	Enables ACP transaction checker.
Trace data/control signal pipeline wi...	8	Enables configurable number of pipeline stages on the TRAC...
Power-on reset(POR) 4k timer	<input type="checkbox"/>	Enables power-on reset(POR) 4k timer. By default, 64k timer i...
Processor event interface	<input type="checkbox"/>	Enables event bus which provides a low-latency and direct m...
Address Editor		
Enable Clock Triggers		
Enable Clock Resets		
FCLK_RESET0_N	<input type="checkbox"/>	Enables general purpose reset signal 0 for PL logic
FCLK_RESET1_N	<input type="checkbox"/>	Enables general purpose reset signal 1 for PL logic
FCLK_RESET2_N	<input type="checkbox"/>	Enables general purpose reset signal 2 for PL logic
FCLK_RESET3_N	<input type="checkbox"/>	Enables general purpose reset signal 3 for PL logic

Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and deselect the **FCLK_CLK0** option and click **OK**.

Processor/Memory Clocks					
IO Peripheral Clocks					
PL Fabric Clocks					
<input type="checkbox"/> FCLK_CLK0	IO PLL	100	10.000000	0.100000 : 250.0000...	
<input type="checkbox"/> FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.0000...	
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.0000...	
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.0000...	
System Debug Clocks					
Timers					

Click on the  (Regenerate Layout) button and see the following block diagram.



에서

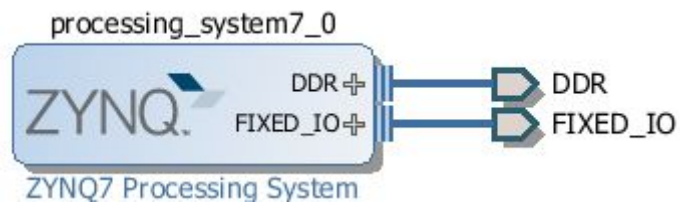



Figure 15. Updated Zynq Block

Click on the  (Validate Design) button and make sure that there are no errors.

- 3. Generate Top-Level and Export to SDK.
 - **Generate IP Integrator Outputs, the top-level HDL, and start SDK by exporting the hardware.**

In the sources panel, right-click on system.bd, and **select Generate Output Products ...** and click Generate to generate the Implementation, Simulation and Synthesis files for the design.


(You can also click on Generate Block Design in the Flow Navigator pane to do the same)

system.bd(block design file) 오른쪽 클릭 후 Generate Output Products와 Flow Navigator의 Generate Block Design이랑 같은 것.

Right-click again on system.bd, and select **Create HDL Wrapper...** to generate the top-level VHDL model.

Leave the **Let Vivado manager wrapper and auto-update option selected**, and click OK.

The system_wrapper.vhd file will be created and added to the project. **Double-click on the file to see the content in the Auxiliary pane.**

Notice that the VHDL file is already Set As the Top module in the design, indicated by the icon 

--Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.

```
-----
--Tool Version: Vivado v.2016.4 (lin64) Build 1756540 Mon Jan 23 19:11:19 MST 2017
--Date      : Thu Jun  1 16:03:44 2017
--Host      : peluza-B85H3-M7 running 64-bit Ubuntu 16.04.2 LTS
--Command   : generate_target system_wrapper.bd
--Design    : system_wrapper
--Purpose   : IP block netlist
-----
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity system_wrapper is
  port (
    DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
    DDR_ba   : inout STD_LOGIC_VECTOR ( 2 downto 0 );
    DDR_cas_n : inout STD_LOGIC;
    DDR_ck_n : inout STD_LOGIC;
    DDR_ck_p : inout STD_LOGIC;
    DDR_cke   : inout STD_LOGIC;
    DDR_cs_n : inout STD_LOGIC;
    DDR_dm    : inout STD_LOGIC_VECTOR ( 3 downto 0 );
```

```

DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
DDR_odt : inout STD_LOGIC;
DDR_ras_n : inout STD_LOGIC;
DDR_reset_n : inout STD_LOGIC;
DDR_we_n : inout STD_LOGIC;
FIXED_IO_ddr_vrn : inout STD_LOGIC;
FIXED_IO_ddr_vrp : inout STD_LOGIC;
FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
FIXED_IO_ps_clk : inout STD_LOGIC;
FIXED_IO_ps_porb : inout STD_LOGIC;
FIXED_IO_ps_srstb : inout STD_LOGIC
);
end system_wrapper;

```

architecture STRUCTURE of system_wrapper is

component system is

```

port (
  DDR_cas_n : inout STD_LOGIC;
  DDR_cke : inout STD_LOGIC;
  DDR_ck_n : inout STD_LOGIC;
  DDR_ck_p : inout STD_LOGIC;
  DDR_cs_n : inout STD_LOGIC;
  DDR_reset_n : inout STD_LOGIC;
  DDR_odt : inout STD_LOGIC;
  DDR_ras_n : inout STD_LOGIC;
  DDR_we_n : inout STD_LOGIC;
  DDR_ba : inout STD_LOGIC_VECTOR ( 2 downto 0 );
  DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
  DDR_dm : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
  DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
  FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
  FIXED_IO_ddr_vrn : inout STD_LOGIC;
  FIXED_IO_ddr_vrp : inout STD_LOGIC;
  FIXED_IO_ps_srstb : inout STD_LOGIC;
  FIXED_IO_ps_clk : inout STD_LOGIC;
  FIXED_IO_ps_porb : inout STD_LOGIC
);
end component system;
begin
system_i: component system
  port map (
    DDR_addr(14 downto 0) => DDR_addr(14 downto 0),

```

```

DDR_ba(2 downto 0) => DDR_ba(2 downto 0),
DDR_cas_n => DDR_cas_n,
DDR_ck_n => DDR_ck_n,
DDR_ck_p => DDR_ck_p,
DDR_cke => DDR_cke,
DDR_cs_n => DDR_cs_n,
DDR_dm(3 downto 0) => DDR_dm(3 downto 0),
DDR_dq(31 downto 0) => DDR_dq(31 downto 0),
DDR_dqs_n(3 downto 0) => DDR_dqs_n(3 downto 0),
DDR_dqs_p(3 downto 0) => DDR_dqs_p(3 downto 0),
DDR_odt => DDR_odt,
DDR_ras_n => DDR_ras_n,
DDR_reset_n => DDR_reset_n,
DDR_we_n => DDR_we_n,
FIXED_IO_ddr_vrn => FIXED_IO_ddr_vrn,
FIXED_IO_ddr_vrp => FIXED_IO_ddr_vrp,
FIXED_IO_mio(53 downto 0) => FIXED_IO_mio(53 downto 0),
FIXED_IO_ps_clk => FIXED_IO_ps_clk,
FIXED_IO_ps_porb => FIXED_IO_ps_porb,
FIXED_IO_ps_srstb => FIXED_IO_ps_srstb
);
end STRUCTURE;

```

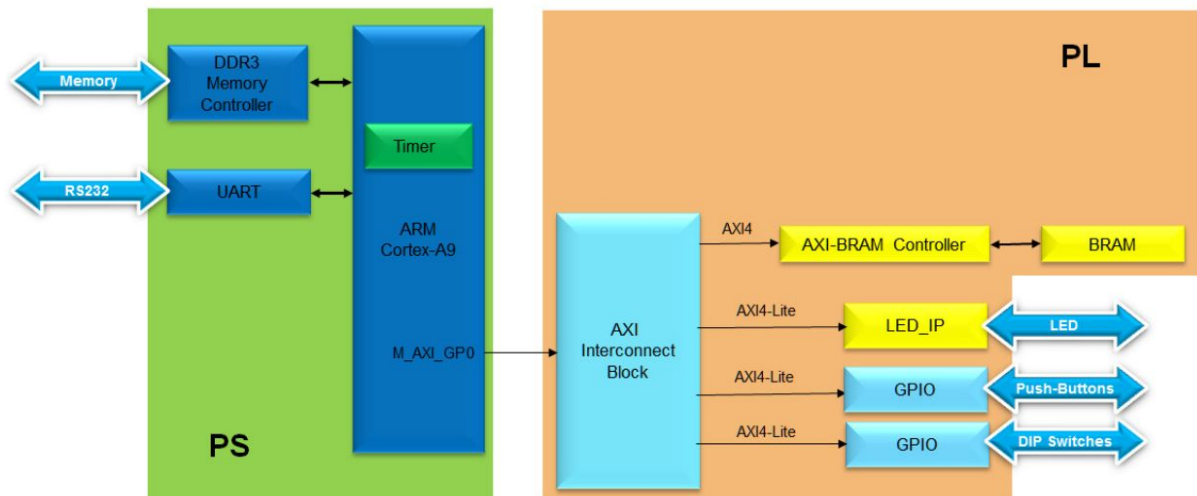
잘은 몰라도 위에 설정이 DDR Memory와 Fixed_IO만 설정 했기 때문에 위와 같이 그에 관련된 것들만 코딩 돼 있다.

Select **File > Export > Export** hardware and click **OK**. (Save the project if prompted)

Note: Since we do not have any hardware in Programmable Logic (PL) there is no bitstream to generate, hence the Include bitstream option is not necessary at this time.

드디어 bitstream에 대한 이해가 조금 더 올라갔다.

그런데



이와 같이 PL을 사용하는데도 위 Note와 같이 PL이 없다고 나온다.
 이것은 아마도 현재 세부화된 Lab인 이 Lab에서 PL을 사용하지 않는다는 것이기 때문.
 위 Figure 1은 현재 Lab이 모두 진행된 최종 모습인 것 같다.

이렇게 되면 위에서 AXI를 Disabled한 것도 납득이 간다. 현재 이 Lab에서는 사용하지 않기 때문.

Select **File > Launch SDK** leaving the default settings, and click OK

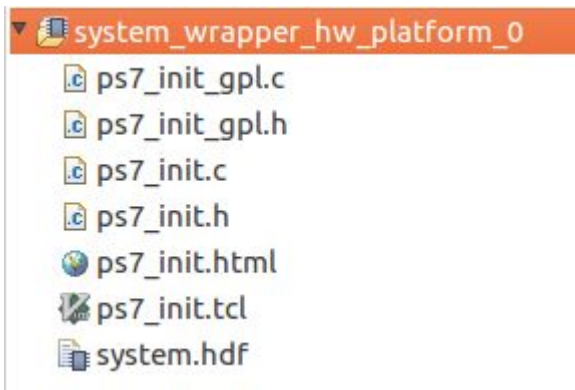
SDK should now be open.

If only the Welcome panel is visible, close or minimize this panel to view the Project Explorer and Preview panel

(Welcome panel은 꺼도꺼도 항상 나온다.)

A Hardware platform project should have been automatically created,

Hardware platform project가



이걸 말하는 듯. 이걸 Vivado에서 export hardware하면 만들어지는 것으로 알고 있다.

and the system_wrapper_hw_platform_0 folder should exist in the Project Explorer panel.

영 Hardware platform project랑 system_wrapper_hw_platform_0이랑 다른 건가 보다.
아마도 위에서 말하는 HPP는 래퍼 하드웨어 플랫폼을 감싸는 전체 프로젝트를 말하는 듯. 즉
이 SDK가 갖고 있는 프로젝트 그 자체.

The system.hdf file (Hardware Description File) for the Hardware platform
should open in the preview pane.

Double click system.hdf to open it if it is not.

Basic information about the hardware configuration of the project can be found in the .hdf
file, along with the Address maps for the PS systems, and driver information.

프로젝트의 하드웨어 설정에 대한 기본 정보는 .hdf 파일과 PS 시스템의 주소 맵 및 드라이버
정보에서 찾을 수 있다.

The .hdf file is used in the software environment to determine the peripherals available in the
system, and their location in the address map.

.hdf 파일은 소프트웨어 환경에서 시스템에서 사용할 수 있는 주변 장치와 주소 맵에서의 해당
위치를 결정하는 데 사용된다.

- 4. Generate Memory TestApp in SDK

**Generate memory test application using one of the standard projects
template.**

In SDK, select File > New > Application Project

Name the project **mem_test**, and in the Board Support Package section, leave Create New
selected and leave the **default name mem_test_bsp** and click **Next**. *(Note that this
application will run on ps7_cortexa9_0 i.e. core 0 of the two processor cores available.)*

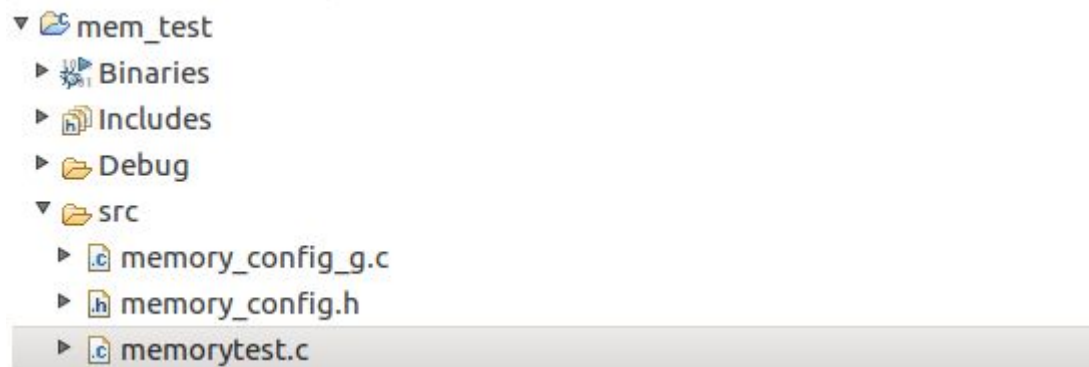
Select **Memory Tests** from the Available Templates window, and click **Finish**.

The mem_test project and the board support project mem_test_bsp **will be created and will
be visible** in the Project Explorer window of SDK, **and the two projects will be
automatically built**. *You can monitor the progress in the Console panel.*

Expand folders in the Project Explorer view, and observe that **there are three projects –
system_wrapper_hw_platform_0, mem_test_bsp, and mem_test**. The mem_test
project is the application that we will use to verify the
functionality of the design. The hw_platform includes the
ps7_init function which initializes the PS as part of the first
stage bootloader, and mem_test_bsp is the board support
package.

Open the **memorytest.c** file in the mem_test project (under src), and examine the contents.

This file calls the **functions to test the memory**.



```
/******  
*  
* Copyright (C) 2009 - 2014 Xilinx, Inc. All rights reserved.  
*  
* Permission is hereby granted, free of charge, to any person obtaining a copy  
* of this software and associated documentation files (the "Software"), to deal  
* in the Software without restriction, including without limitation the rights  
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
* copies of the Software, and to permit persons to whom the Software is  
* furnished to do so, subject to the following conditions:  
*  
* The above copyright notice and this permission notice shall be included in  
* all copies or substantial portions of the Software.  
*  
* Use of the Software is limited solely to applications:  
* (a) running on a Xilinx device, or  
* (b) that interact with a Xilinx device through a bus or interconnect.  
*  
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR  
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY,  
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT  
SHALL  
* XILINX BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,  
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
OUT OF  
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  
THE  
* SOFTWARE.  
*  
* Except as contained in this notice, the name of the Xilinx shall not be used  
* in advertising or otherwise to promote the sale, use or other dealings in
```

* this Software without prior written authorization from Xilinx.

*

*****/

```
#include <stdio.h>
#include "xparameters.h"
#include "xil_types.h"
#include "xstatus.h"
#include "xil_testmem.h"
```

```
#include "platform.h"
#include "memory_config.h"
#include "xil_printf.h"
```

```
/*
 * memory_test.c: Test memory ranges present in the Hardware Design.
 *
 * This application runs with D-Caches disabled. As a result cacheline requests
 * will not be generated.
 *
 * For MicroBlaze/PowerPC, the BSP doesn't enable caches and this application
 * enables only I-Caches. For ARM, the BSP enables caches by default, so this
 * application disables D-Caches before running memory tests.
 */
```

```
void putnum(unsigned int num);
```

```
void test_memory_range(struct memory_range_s *range) {
    XStatus status;
```

```
/* This application uses print statements instead of xil_printf/printf
 * to reduce the text size.
 *
 * The default linker script generated for this application does not have
 * heap memory allocated. This implies that this program cannot use any
 * routines that allocate memory on heap (printf is one such function).
 * If you'd like to add such functions, then please generate a linker script
 * that does allocate sufficient heap memory.
 */
```

```
print("Testing memory region: "); print(range->name); print("\n\r");
print("  Memory Controller: "); print(range->ip); print("\n\r");
#ifdef __MICROBLAZE__
    print("      Base Address: 0x"); putnum(range->base); print("\n\r");
    print("      Size: 0x"); putnum(range->size); print (" bytes \n\r");
#else
```

```

        xil_printf("        Base Address: 0x%x\n",range->base);
        xil_printf("        Size: 0x%x bytes\n",range->size);
    #endif

    status = Xil_TestMem32((u32*)range->base, 1024, 0xAAAA5555,
XIL_TESTMEM_ALLMEMTESTS);
    print("        32-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
    print("\n\n");

    status = Xil_TestMem16((u16*)range->base, 2048, 0xAA55,
XIL_TESTMEM_ALLMEMTESTS);
    print("        16-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
    print("\n\n");

    status = Xil_TestMem8((u8*)range->base, 4096, 0xA5,
XIL_TESTMEM_ALLMEMTESTS);
    print("        8-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
    print("\n\n");

}

int main()
{
    int i;

    init_platform();

    print("--Starting Memory Test Application--\n\n");
    print("NOTE: This application runs with D-Cache disabled.");
    print("As a result, cacheline requests will not be generated\n\n");

    for (i = 0; i < n_memory_ranges; i++) {
        test_memory_range(&memory_ranges[i]);
    }

    print("--Memory Test Application Complete--\n\n");

    cleanup_platform();
    return 0;
}

```


- 5. Test in Hardware

Zybo: Make sure that the JP7 is set to select USB power and JP5 is set to JTAG mode. Connect the board with a micro-usb cable and power it ON.


Establish the serial communication using SDK's Terminal tab.

(이거 안되는 리눅스 사용자 분들은 Minicom 등 다른 직렬 통신 프로그램 사용하기를)

Zybo: Make sure that the JP7 is set to select USB power, and JP5 is set to JTAG. Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

Select the  **Terminal** tab. If it is not visible then select Window > Show view > Terminal.

(역시나 이게 안되면 minicom)

~~Click on  and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown.~~

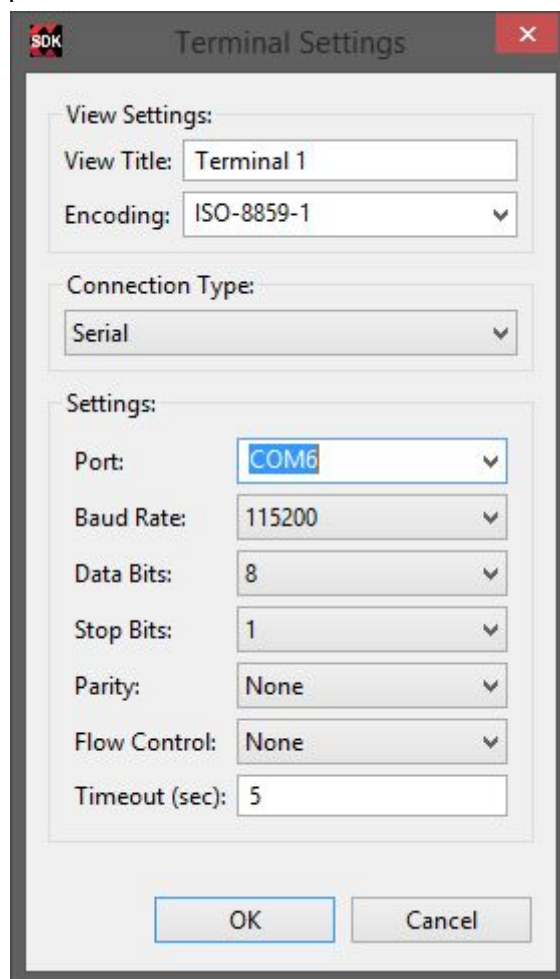


Figure 23. SDK Terminal Settings

You can find the COM port from the Windows Device Manager, in this case, COM6:

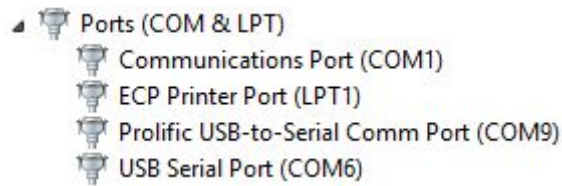


Figure 24. COM ports in Windows control panel

Run the mem_test application and verify the functionality.

In SDK, select the mem_test project in Project Explorer, **right-click and select Run As > Launch on Hardware (GDB)** to download the application, and will execute ps7_init, and then execute **mem_test.elf** (user application).

You should see the following output on the Terminal tab.

```
peluza@peluza-B85H3-M7: ~
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will not be generated
Testing memory region: ps7_ddr_0
Memory Controller: ps7_ddr_0
Base Address: 0x100000
Size: 0x1FF00000 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
Testing memory region: ps7_ram_1
Memory Controller: ps7_ram_1
Base Address: 0xFFFF0000
Size: 0xFE00 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
--Memory Test Application Complete--
```

나 같은 경우 .elf 타겟이 없다는 오류가 나면서 안됐었다.

이 문제는 해당 링크의

<https://forums.xilinx.com/t5/Embedded-Development-Tools/No-Elf-file-associated-with-target-Vivado-2016-4/td-p/740496>

XLINX sadanan
Xilinx Employee



Posts: 88
Registered: 10-21-2010

Re: No Elf file associated with target - Vivado 2016.4 [Edited]

Options ▾

01-03-2017 08:23 PM - edited 01-11-2017 09:04 PM

Hi,
mwr command in xmd doesn't support -force option, hence the error, but the error message appears to be bogus. You can remove the -force option from the script or use system debugger (right-click on the application, select Debug As -> Launch on Hardware (System Debugger)). We'll get Vivado export to remove -force option from ps7_init scripts.

Update: AR to get past the issue
<http://www.xilinx.com/support/answers/68503.html>

[View solution in original post](#)

Message 5 of 10 (3,588 Views)

1 Kudo Reply

이 부분을 읽어서 해결했다.

Debug 모드의 System Debugger로 구동하면 위와 같이 잘 작동하는 것을 알 수 있다.

Close SDK and Vivado by selecting File > Exit in each program.

- Conclusion

Vivado and the IP Integrator allow **base embedded processor systems and applications** to be generated very quickly. **After the system has been defined, the hardware can be exported and SDK can be invoked from Vivado.** Software development is done in SDK which provides several application templates **including memory tests.** You verified the operation of the hardware **by downloading a test application, executing on the processor, and observing the output in the serial terminal window.**