TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-05-29　(63 회차)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 정유경
ucong@naver.com

< 환경 구성하기 >

1. 리눅스환경에서 Vivado 를 설치한다.

1. 2017.1 버전 다운로드 (All OS Installer Single File)

http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

2. Home > Xilinx_Vivado 폴더 생성하여 여기에 1 번 파일의 압축을 푼다

tar -zxvf [파일명]

3. 압축 푼 경로로 들어가서 다음을 실행한다.

sudo dpkg-reconfigure dash → No

sudo ./xsetup

4. 인스톨 화면에서 다음과 같이 설정한다.

Next → Agree Agree Agree → Next → **Vivado HL WebPack 선택**

→다음을 모두 체크한다

Software Development Kit(SDK) , DocNav , Production Devices, Engineering Sample Devices(Zynq 전부 체크), Installation Options

→ 설치 디렉토리를 다음과 같이 설정한다

Home > yukyoung > Xilinx_Vivado

5.vlm 을 실행하여 라이센스를 얻는다

cd /home/yukyoung/vivado/2017.1/bin 들어가서 ./vlm

6. Obtain License 해서 메일로 라이센스를 받는다

Get free ISE Webpack → Connect → 사용기한 None 표시된 2 개 체크

7. Copy License

8. Vivado 를 실행한다.

cd /home/yukyoung/vivado/2017.1/bin 들어가서 ./vivado

2. 바탕화면에 바로가기 만들기

(1) vi SDK.desktop

```
#!/usr/bin/env xdg-open
[Desktop Entry]
Name=Xilinx SDK
Type=Application
Exec=/home/yukyoung/Xilinx/SDK/2017.1/bin/xsdk
Terminal=false
Icon=/home/yukyoung/Xilinx/SDK/2017.1/data/sdk/images/sdk_logo.ico
Comment=Xilinx SDK Program
NoDisplay=false
Categories=Development;IDE;
Name[en]=SDK
```

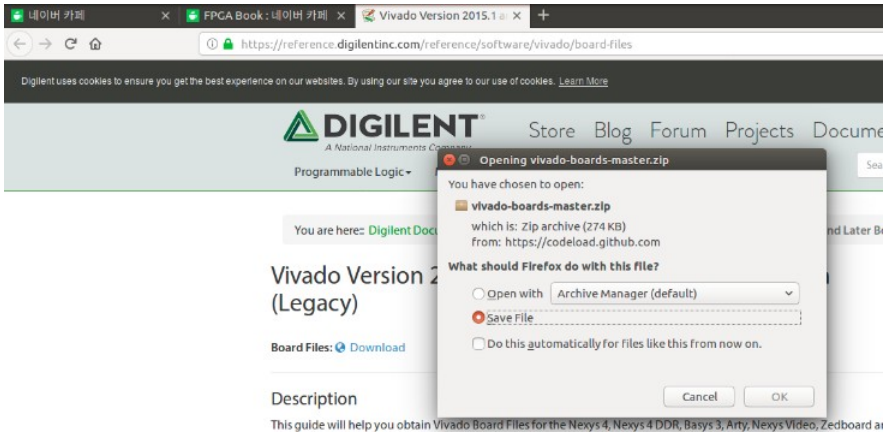
(2) vi vivado.desktop

```
#!/usr/bin/env xdg-open
[Desktop Entry]
Name=Xilinx Vivado
Type=Application
Exec=/home/yukyoung/Xilinx/Vivado/2017.1/bin/vivado
Terminal=false
Icon=/home/yukyoung/Xilinx/Vivado/2017.1/doc/images/vivado_logo.ico
Commnet=Xilinx Vivado Program
NoDisplay=false
Categories=Development;IDE;
Name[en]=Vivado
```

(3) 생성된 두 아이콘에 대해 다음을 실행한다.

바탕화면에서 아이콘 우클릭 > 속성 > Allow Executing File as a Program 체크

3. vivado Board Mater 를 다운로드 한다

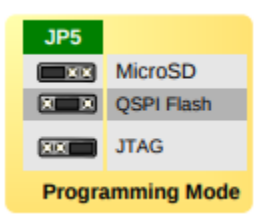


다운받은 파일의 압축을 푼 후

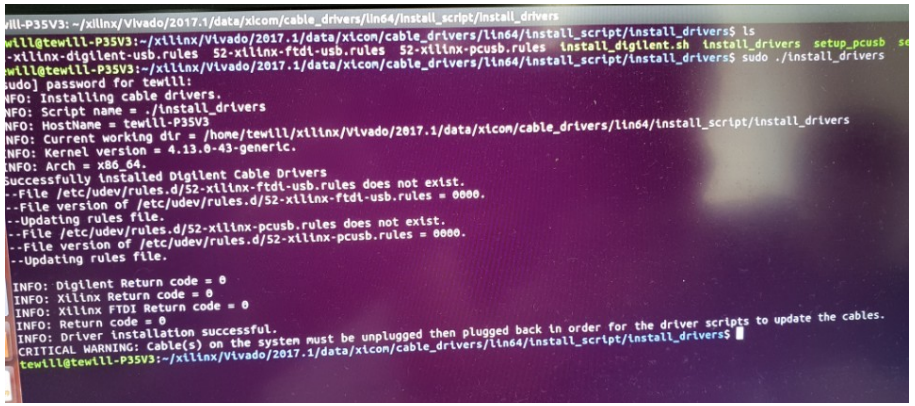
Board_Files 폴더에 sudo 명령으로 다운받은 파일을 옮겨준다

4. Programming Mode 를 JTAG 로 설정한다

보드 상에서 점퍼를 직접 옮길것



5. Auto Connect 안되는 경우 드라이버를 설치한다.



1. AND Gate 생성 (by VHDL)

1. 프로젝트 생성

Target Language : Verilog

Default Part: xc7z010clg400-1

2. 소스파일 생성

Add Sources > Add or Create Design Sources 선택

Create File 에서 VHDL 선택하고 and_gate.vhd 생성

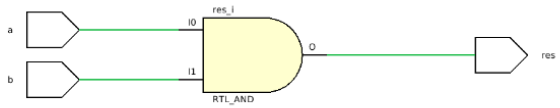
3. and_gate.vhd 에 소스코드 작성

```
entity ander is
    port (
        a,b : in std_logic;
        res : out std_logic ; 붙이지 않는다
    );
end ander;
architecture Behavioral of ander is // ander 의 기능을 정의한다
begin
    res <= a and b;
end Behavioral;
```

And Gate 2 개 생성할 경우

```
entity ander is
    port (
        a,b,c,d: in std_logic;
        res1,res2 : out std_logic ; 붙이지 않는다
    );
end ander;
architecture Behavioral of ander is // ander 의 기능을 정의한다
begin
    res <= a and b;
    res <= c and d;
end Behavioral;
```

4. Open Elaborated(정교한) Design 클릭해서 Schematics 생성



5. Schematics 에서 I/O Ports 선택

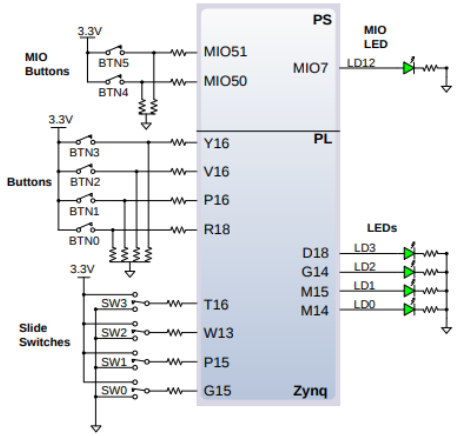


Figure 14. ZYBO GPIO.

*. ZYBO FPGA Board Reference Manual 을 참고하여 핀 설정한다.

https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf

- I/O Standard: LVCMOS33 으로 변경

(입출력 포트의 전압을 3.3V 로 설정)

- P15, G15 스위치를 input 으로 M15 LED 를 출력으로 설정한다
(and_gate 2 개 설정할 경우 T16, W13 in / M14 out 을 추가한다)

6. Run Synthesis -> Run Implementation -> Generate Bitstream 을 순서대로 실행한다

7. Open Target >

7. 비트파일까지 생성한 이후에 Open Hardware Manager 를 선택한다.

8. Zynq 보드를 연결하고 Open Target

9. 앞에서 생성한 Bit Stream File 을 링크하고 Program 한다.

10. 스위치를 직접 제어하여 LED On/Off

2. GPIO Port 제어 (by Verilog)

[1] Vivado

1. 프로젝트 생성 (RTL Project)

2. Create Block Design

+ ZYNQ7

→ Run Block Automation → OK

+ AXI_GPIO

→ 두번 클릭 → GPIO Width: 1 (LED 1 개 제어) → Select Board Part Interface : Custom

3. Create Constraint 하고 다음을 입력한다

(Constraint 에서 Add Sources 클릭 하고 Add or Create Constraint)

```
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [ get_ports {gpio_rtl_tri_io}];
```

4. Design Sources 에서 우클릭하여 Create HDL Wrapper 실행

(Let vivado manage Wrapper 선택)

5. Generate Bitstream

(비트파일까지 생성한 이후에 Open Block Design 을 선택한다. Block Design 을 open 하지 않으면 SDK 연동이 잘 안되는 경우가 발생하기 때문)

6. File > Export Hardware

(include Bitstream 선택)

7. File > Lanch SDK

[2] SDK

1. New Project 생성

File > Application Project

(Hardware Platform: [생성한 Wrapper] 확인, Processor: [ps7_cortexa9_0] 확인)

Next → Empty Application 선택

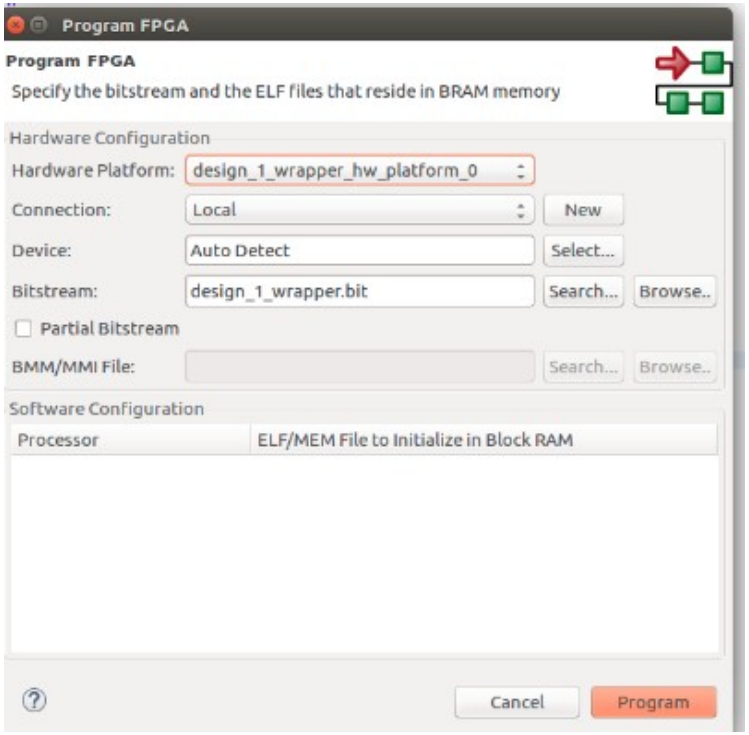
2. 생성한 프로젝트의 src 폴더내에 main.c 파일 생성

```
#include <stdio.h>
#include <xgpio.h>
#include "xparameters.h"
#include "sleep.h"

int main(void)
{
  XGpio in, out;
  XGpio_Initialize(&out, XPAR_AXI_GPIO_0_DEVICE_ID);
  XGpio_SetDataDirection(&out, 1, 0x0);

  while(1)
  {
      Xil_Out32(0x41200000, 0xFFFFFFFF);
      sleep(1);
      Xil_Out32(0x41200000, 0x0);
      sleep(1);
  }
  return 0;
}
```

3. 저장(Compile)→ Build → Program FPGA →Run

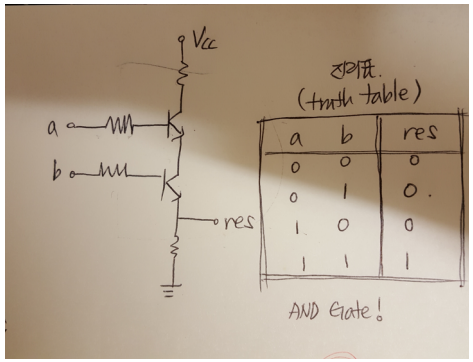


4. LED Toggle 되는지 확인한다

*. xpr : fpga 확장자

vhd : VHDL source

*. TR 2 개로 제어하는 AND 회로



*. 32bit eax 레지스터를 하나 만드는데 TR 이 100 개 필요하다

→ FPGA 는 각종 트랜지스터를 논리적으로 프로그래밍 한것

(디지털 칩도 결국 내부는 아날로그, 복잡해지면 사람이 할수 없다. 그러므로 논리를 만들어서 단순화시키고, 논리대로 부품을 배치하는 과정이 필요하다)

*. CPU 칩내에 트랜지스터 수십억개 들어있다

FPGA 를 이용하여 이러한 트랜지스터 수십억개를 반도체에 집어넣을 수 있다

*. Zynq = PL + PS

programmable logic = VHDL, Verilog

processing system = Linux

*. Xilinx 사에서는 FPGA 를 설계하기 위한 tool 로 Vivado 를 제공한다

일반적으로 Vivado 를 이용해 FPGA 를 설계하는 과정은 다음과 같다.

(1)코드 작업을 한다. 이때 사용되는 언어의 종류를 HDL(hardware description language)라 부르며 주로 VHDL 이나 Verilog 가 사용된다. 여기서는 VHDL 을 사용한다고 가정하자.

다음으로 작성한 HDL 을 (2)RTL(register-transfer level) 시뮬레이션한다. 이를 쉽게 말하면, HDL 코드를 실제 FPGA 내부에서 디지털 회로로 합성하여 시뮬레이션 하는 것은 시간이 오래 걸리기 때문에, 단순히 코드가 잘 되는지 빠르게 시뮬레이션하는 것이다. 부가 설명을 하면 일반적으로 HDL 코드를 RTL 코드 또는 RTL 형태로 서술된 코드라 한다. Register 란 보통 flip-flop 으로 구성되며 clock 신호가 들어가므로 회로를 동기화(synchronize)하는 역할을 한다. 또한 마치 메모리처럼 데이터(bit)을 읽고 쓸 수 있다. HDL 코드에서는 이러한 register 들을 마치 변수처럼 선언해서 사용한다. HDL 코드만 봐서는 실제 디지털 회로가 어떻게 구현될지는 알 수 없지만 register 레벨에서는 대략 알 수 있기 때문에 이렇게 불리는 듯 하다. RTL 과 비교되는 용어로는 Gate level 이 있다.

그 후 (3)HDL 코드를 synthesis(합성)해서 netlist 를 만든다. FPGA 에서 netlist 란 synthesis 를 해야 생성된다. HDL(RTL)이었던 것을 실제 디지털 회로(게이트 등)로 구현하는 것이다. synthesis 가 끝나면 (4)implementation 을 하게 된다. 이때 실제로 target FPGA 에 디지털 회로를 배치하고 배선하게 되며 이를 place & route 라고 한다. 이 두 과정에서 시간이 좀 소요된다. 마지막으로 (5)timing 시뮬레이션을 거친 후 (6)bitstream(.bit 파일)을 생성해 FPGA 에 입히게 된다.