

# How to control RC servo motor with PWM

이대로

skseofhek@daum.net

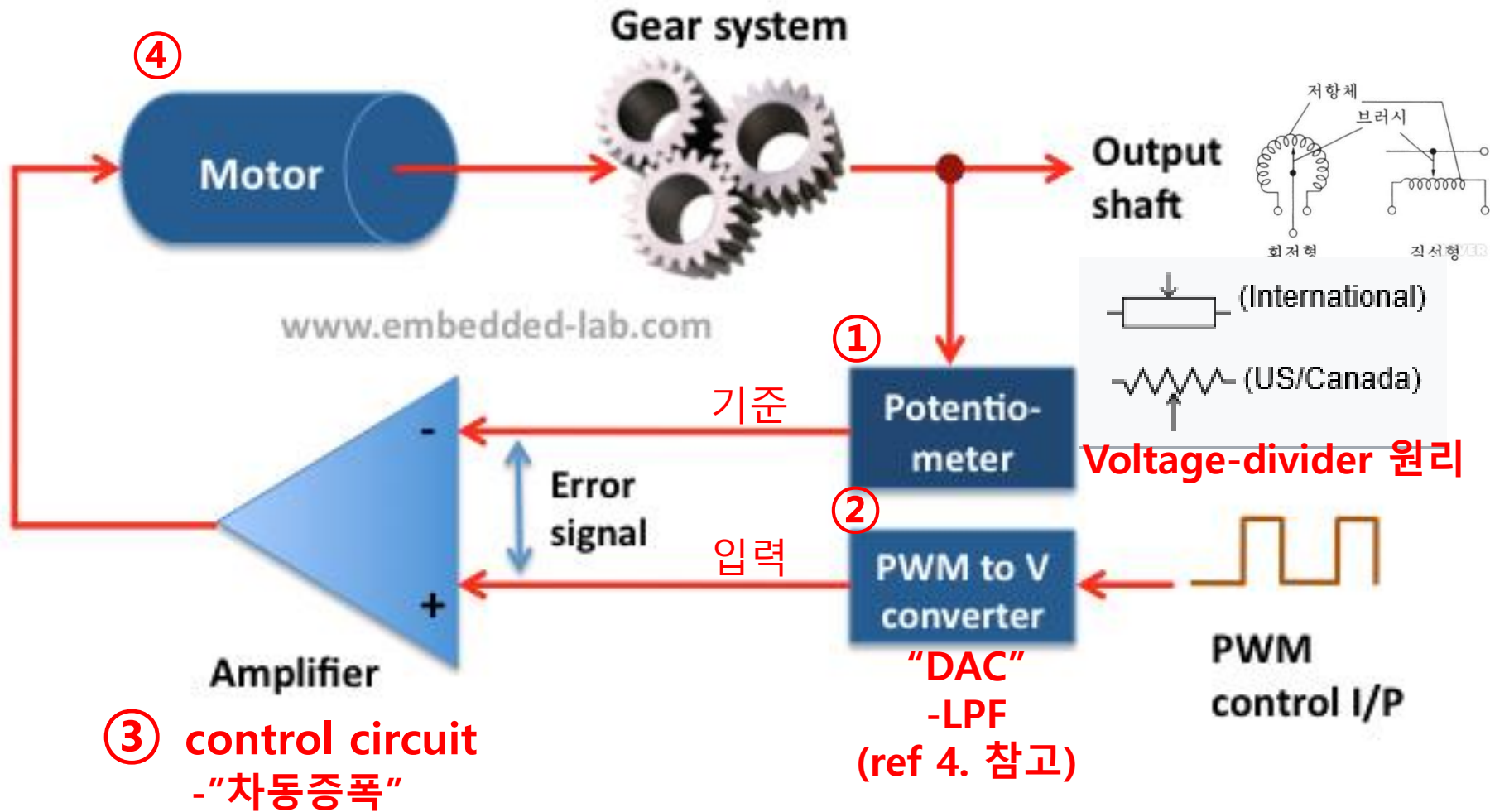
## 서보 모터란?

- 특정 위치로 이동하거나, 특정한 수치(속도 등)만큼 가동시킬 때 사용
- 모터로부터의 피드백을 통해 정확하게 제어할 수 있는 구조 : closed-loop system
- 모터와 기어박스 그리고 제어회로로 구성
- 자동화 생산 시스템, 로봇, 장난감, 가전제품 등 광범위하게 쓰인다.

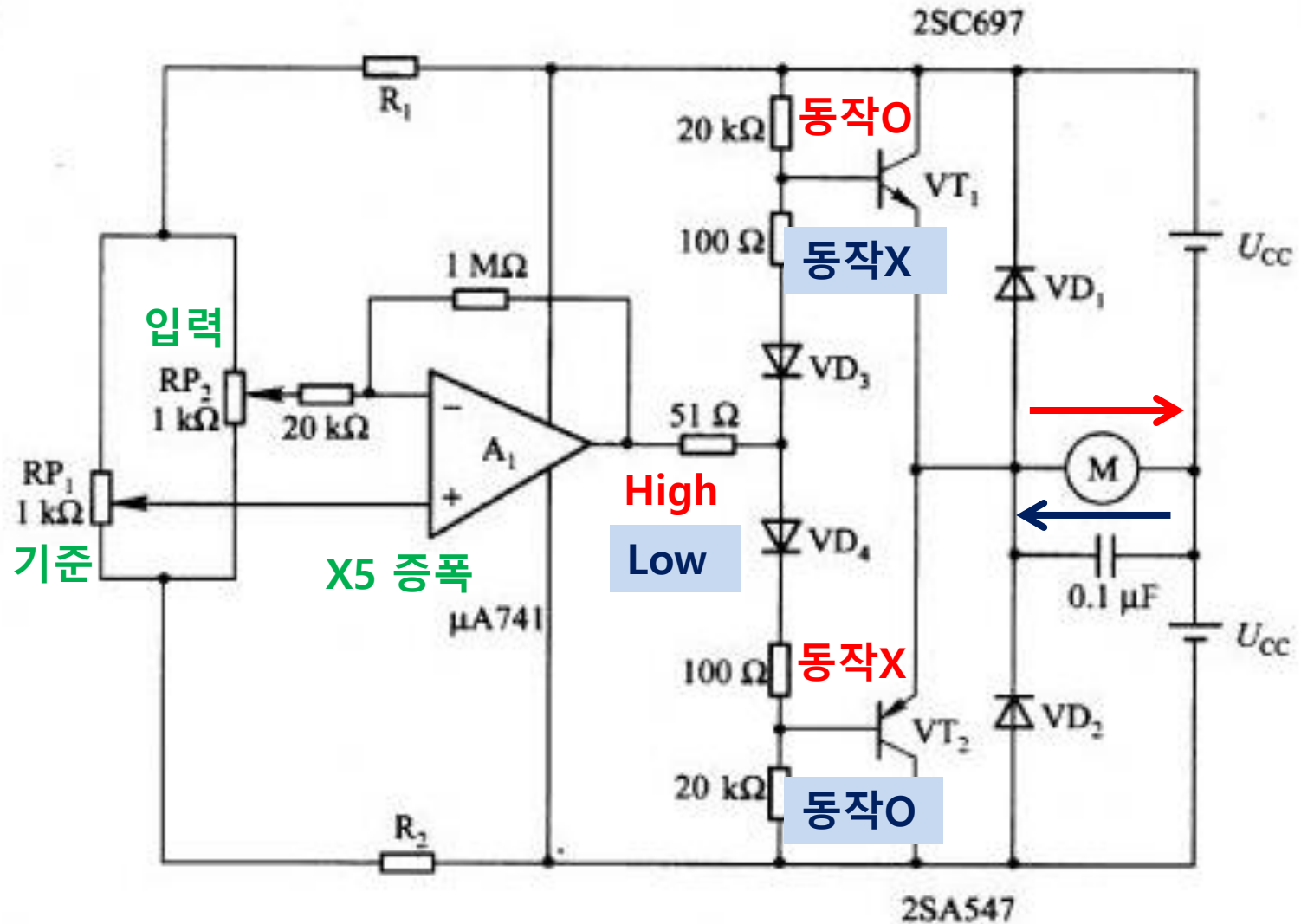
산업용 vs RC용



# 서보 모터 구조 및 작동원리



### ③ Control circuit 예시



# PWM 신호 발생

## SG90 9 g Micro Servo



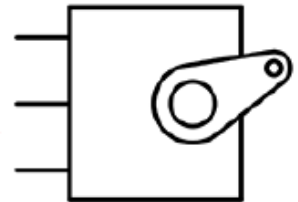
## Specifications

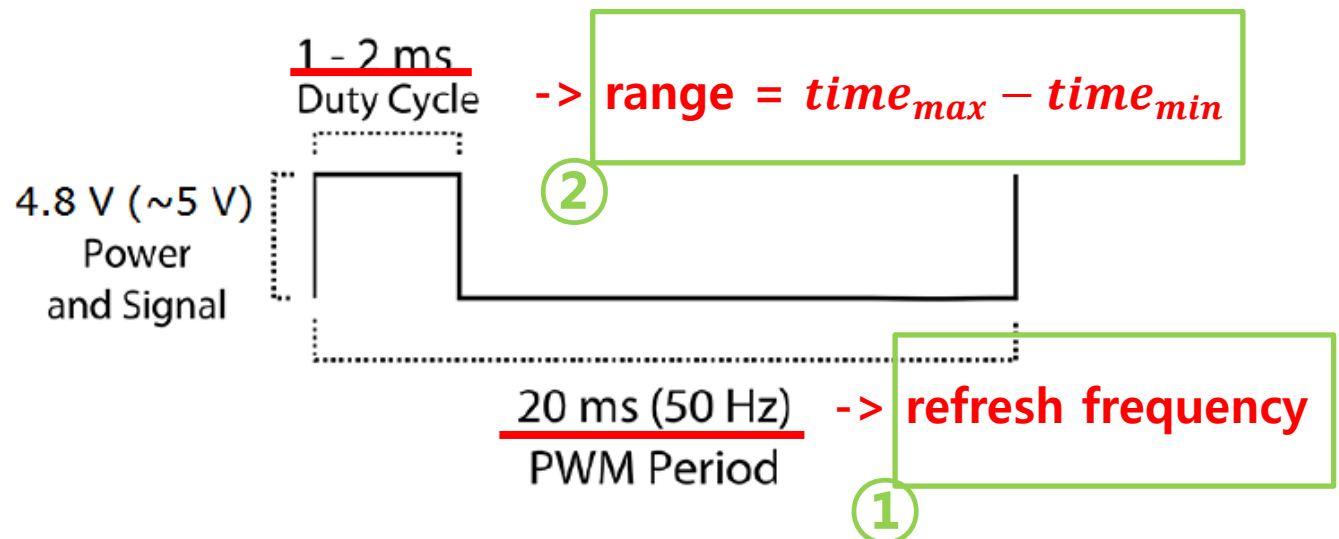
- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10  $\mu$ s
- Temperature range: 0 °C – 55 °C

PWM=Orange (  )

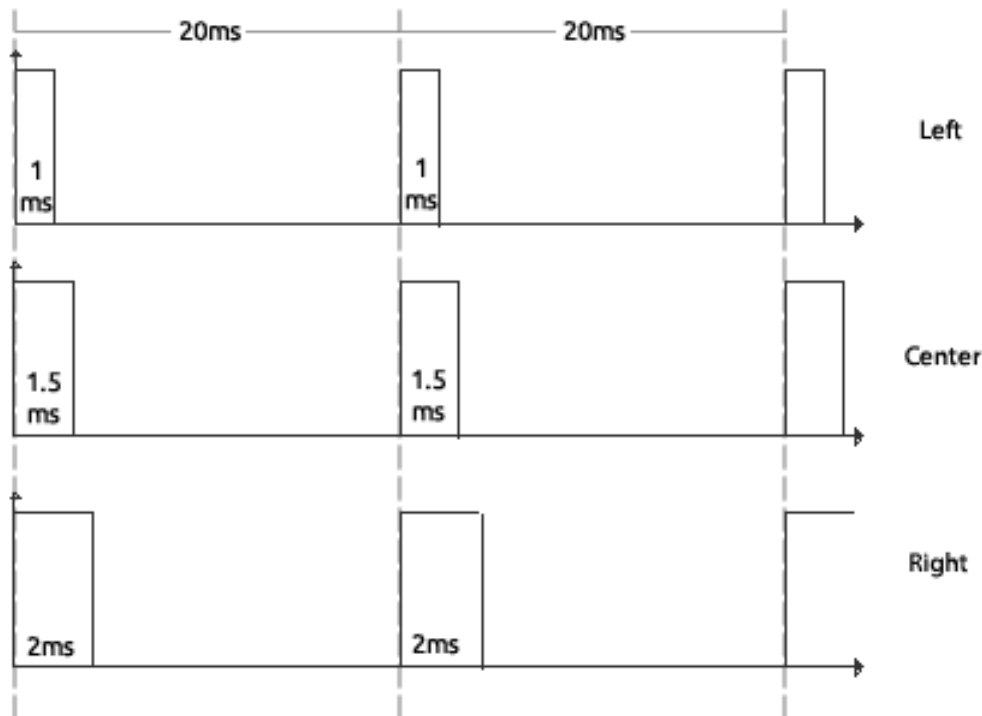
Vcc = Red ( + )

Ground=Brown ( - )





Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.



$-90^\circ = 1\text{ms}$

$0^\circ = 1.5\text{ms}$

$90^\circ = 2\text{ms}$

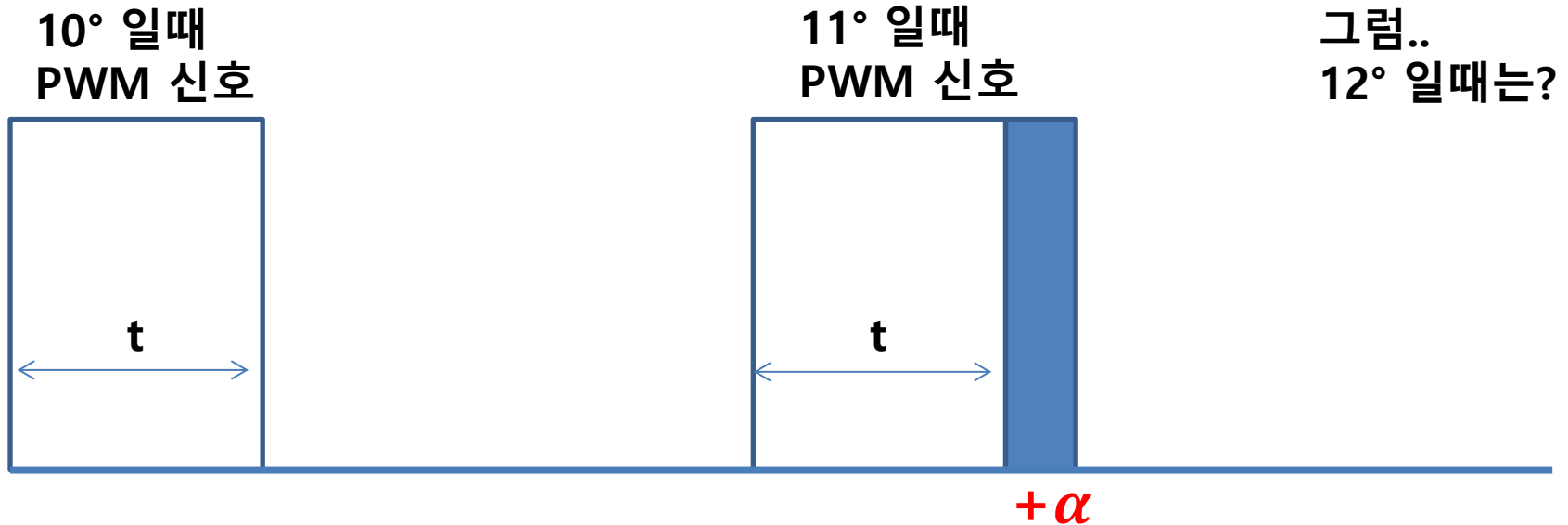
각 duty cycle 마다 각도가 정해져 있음!

③

$-90^\circ \sim 90^\circ$  제어 가능  
( $180^\circ$ )

Q) “1°단위로 제어하고 싶다”

-> 10°와 11°를 구분



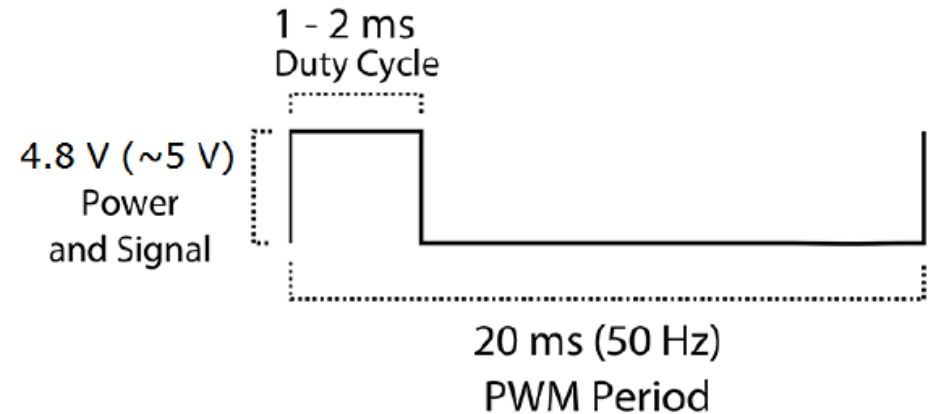
+α 를 기준 clock -> “counter”

## 기준 clock은 얼마로 해야 하나?

1)  $refresh\ frequency = 20ms$

2)  $range = time_{max} - time_{min}$   
 $= 2ms - 1ms = 1ms$

3)  $f_{needed} = \left(\frac{range}{resolution}\right)^{-1}$



1°단위로 제어 -> resolution = 180

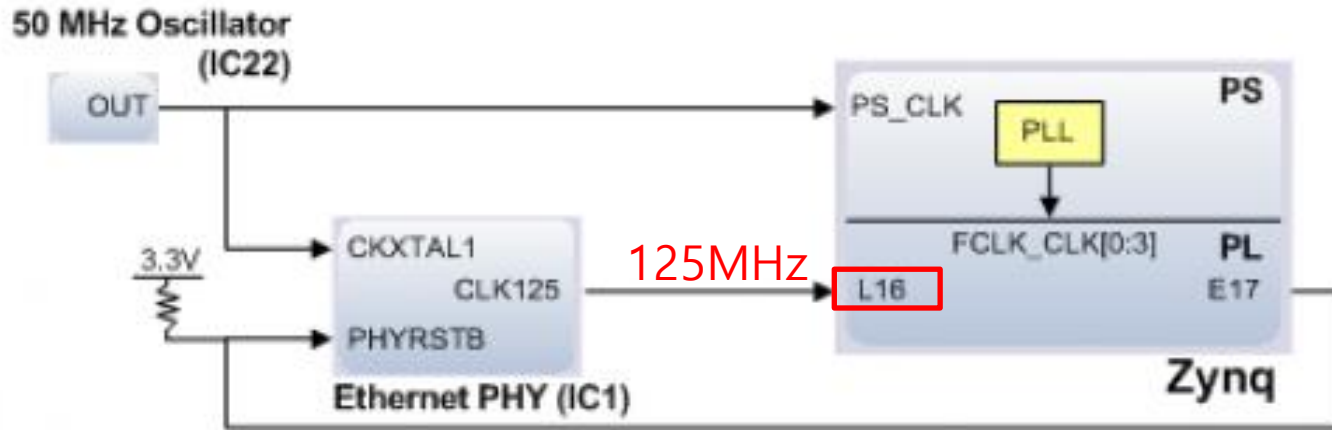
//그럼.. 10°단위로 제어하고 싶을 땐? -> resolution = 18

$f_{needed} = \left(\frac{1ms}{180}\right)^{-1} = 180kHz$  -> 이 clock 신호를 만들어 줘야 됨..



frequency\_divider : clk\_180kHz

ZYBO 외부 125MHz 클럭(L16핀)이용 -> 180kHz 클럭 생성

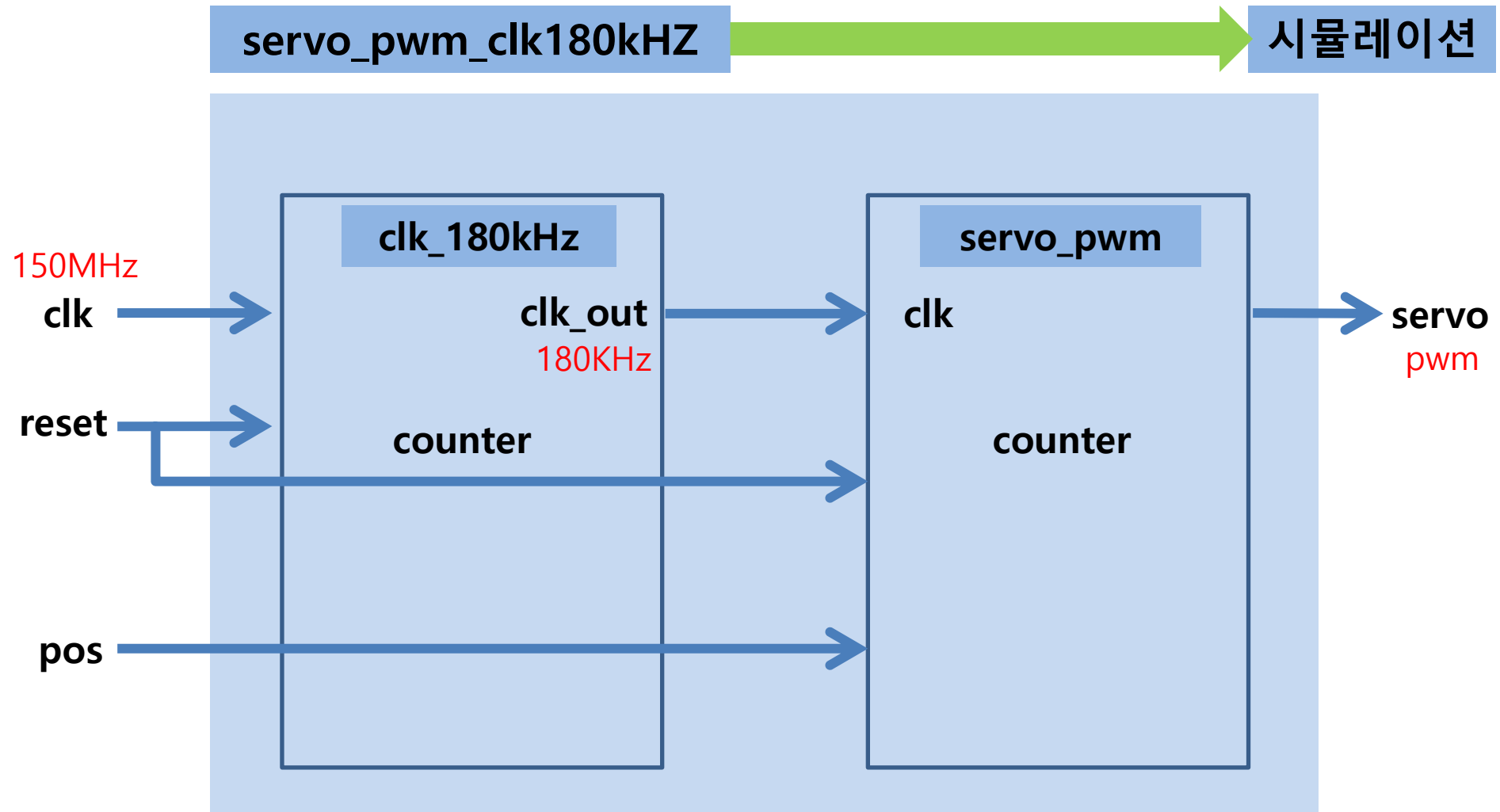


125MHz로 180kHz 만들기

$$\begin{aligned} \text{Scale} &= \frac{f_{in}}{f_{out}} \\ &= \frac{125\text{MHz}}{180\text{kHz}} = 694.4444 \dots \approx 694 \end{aligned}$$

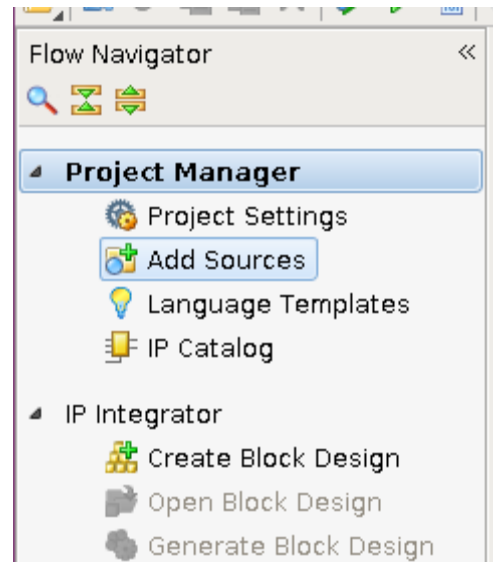
-> 150MHz를 694번 씩 세면(=>counter) 180kHz가 만들어짐

# PWM 생성 flow

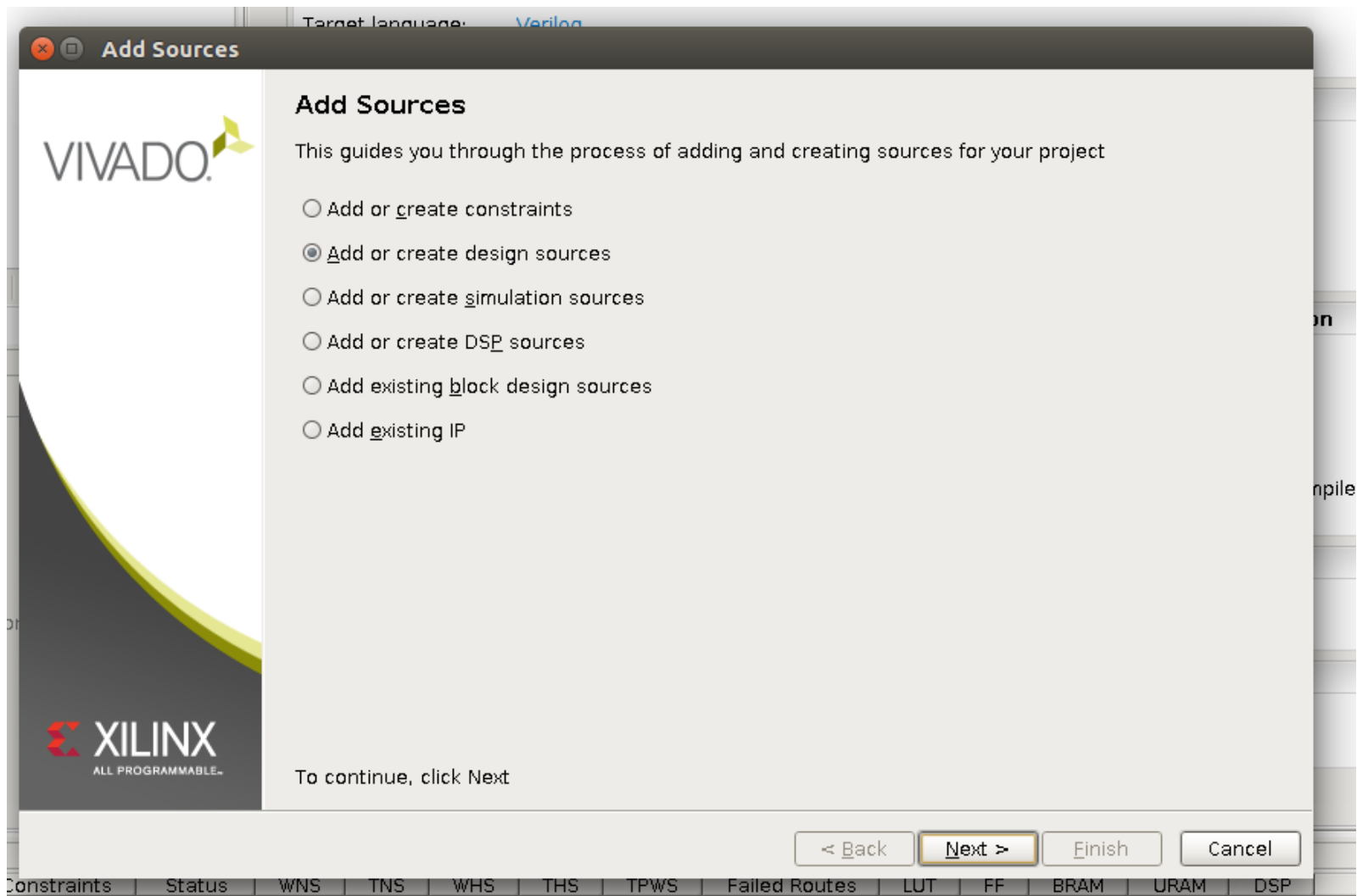


# Vivado 실습

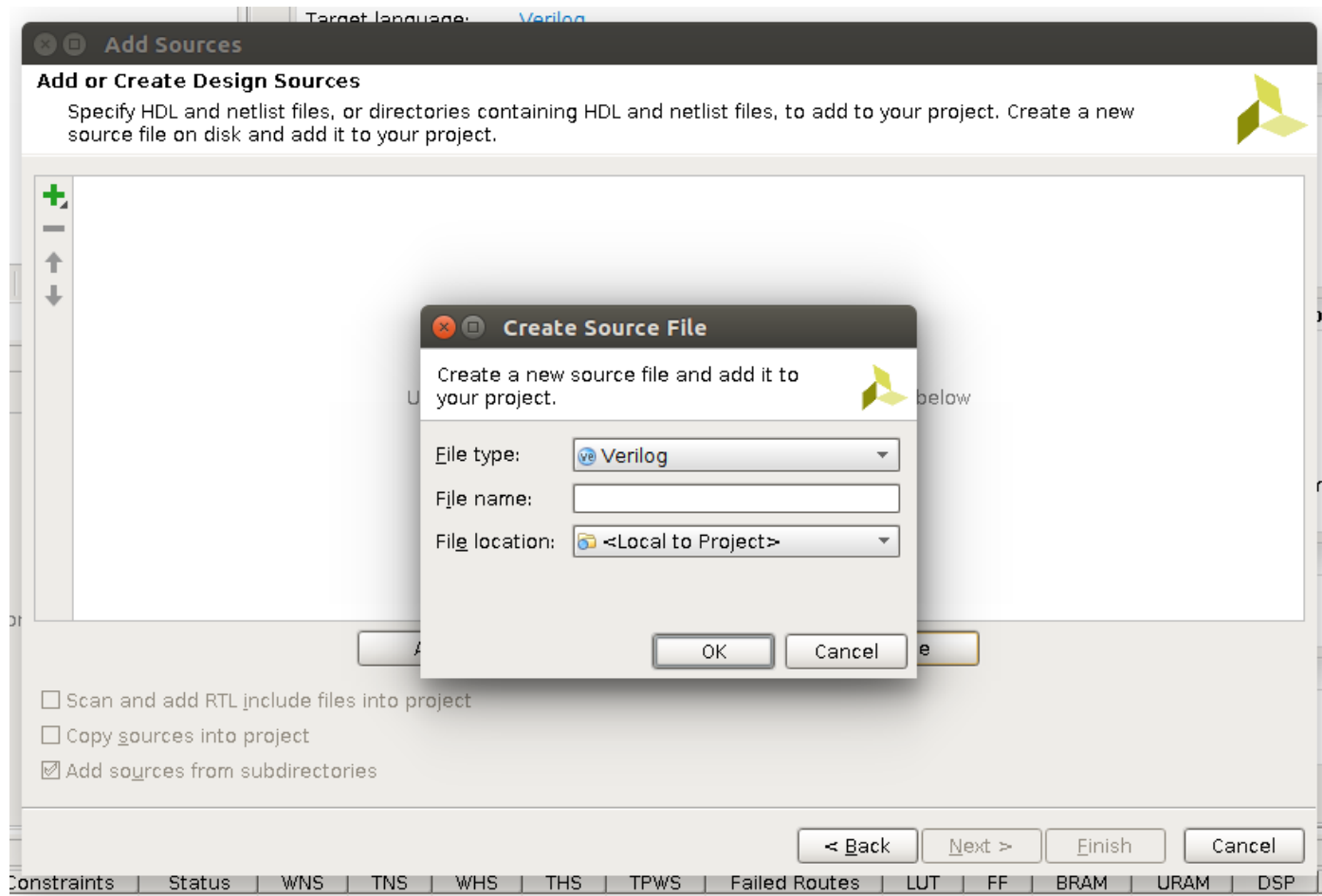
1. 새 project를 생성
  - Target language와 Simulator language는 VHDL 로 하고  
board는 zybo로 설정
2. 왼쪽에 있는 Flow Navigator -> Project Manager -> Add Sources클릭



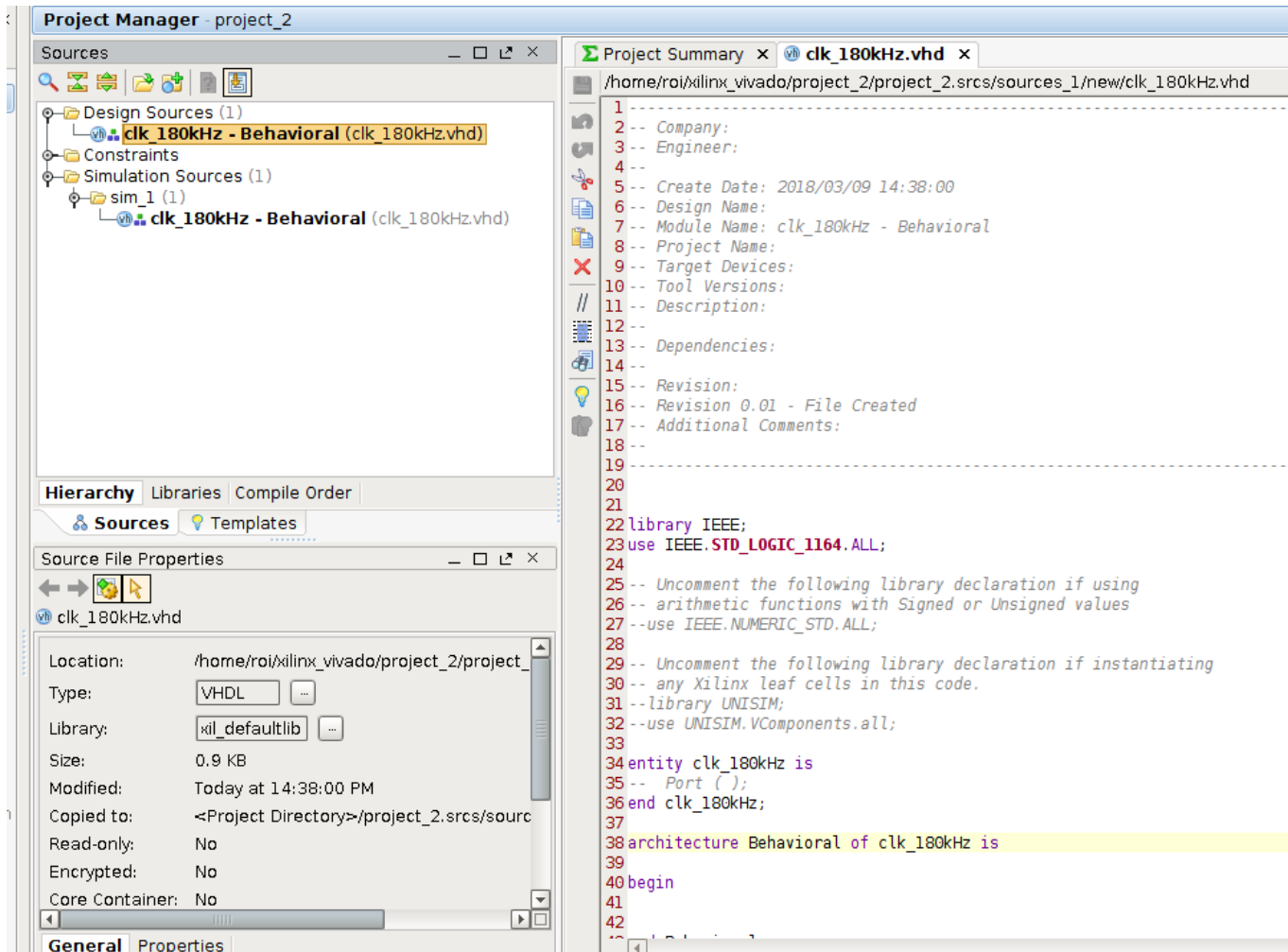
### 3. 'Add or create design sources' 체크 후 Next> 버튼



4. 'Create File'누르고 'Create Source File'창 뜨면 File name에 'clk\_180kHz' 입력, 'Ok'버튼 누르고 나와서 'Finish'클릭



5. 'Define Module' 창이 뜨면 그냥 'Ok'클릭 -> 'Yes'클릭
6. Design Sources' -> clk\_180kHz 더블 클릭해서 코드 입력.



## clk\_180kHz VHDL 코드

```
Project Summary x vh clk_180kHz.vhd * x
/home/roi/xilinx_vivado/project_2/project_2.srscs/sources_1/new/clk_180kHz.vhd
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;                                --라이브러리 및 패키지 선언
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity clk_180kHz is                          --entity : 모듈 이름, 입/출력 port 설정
26 -- Port ( );
27     port(
28         clk_in : in std_logic;
29         reset : in std_logic;
30         clk_out : out std_logic
31     );
32
33 end clk_180kHz;
```

```

34
35 architecture Behavioral of clk_180KHz is
36     signal temporal : std_logic;
37     signal counter : integer range 0 to 346 :=0;
38 begin
39     frequency_divider : process (reset, clk_in)
40     begin
41         if(reset ='1') then
42             temporal <= '0';
43             counter <= 0;
44         elsif rising_edge(clk_in) then
45             if (counter = 346) then
46                 temporal <= NOT(temporal);
47                 counter <= 0;
48             else
49                 counter <= counter +1;
50             end if;
51         end if;
52     end process;
53
54     clk_out <= temporal;
55
56 end Behavioral;
57

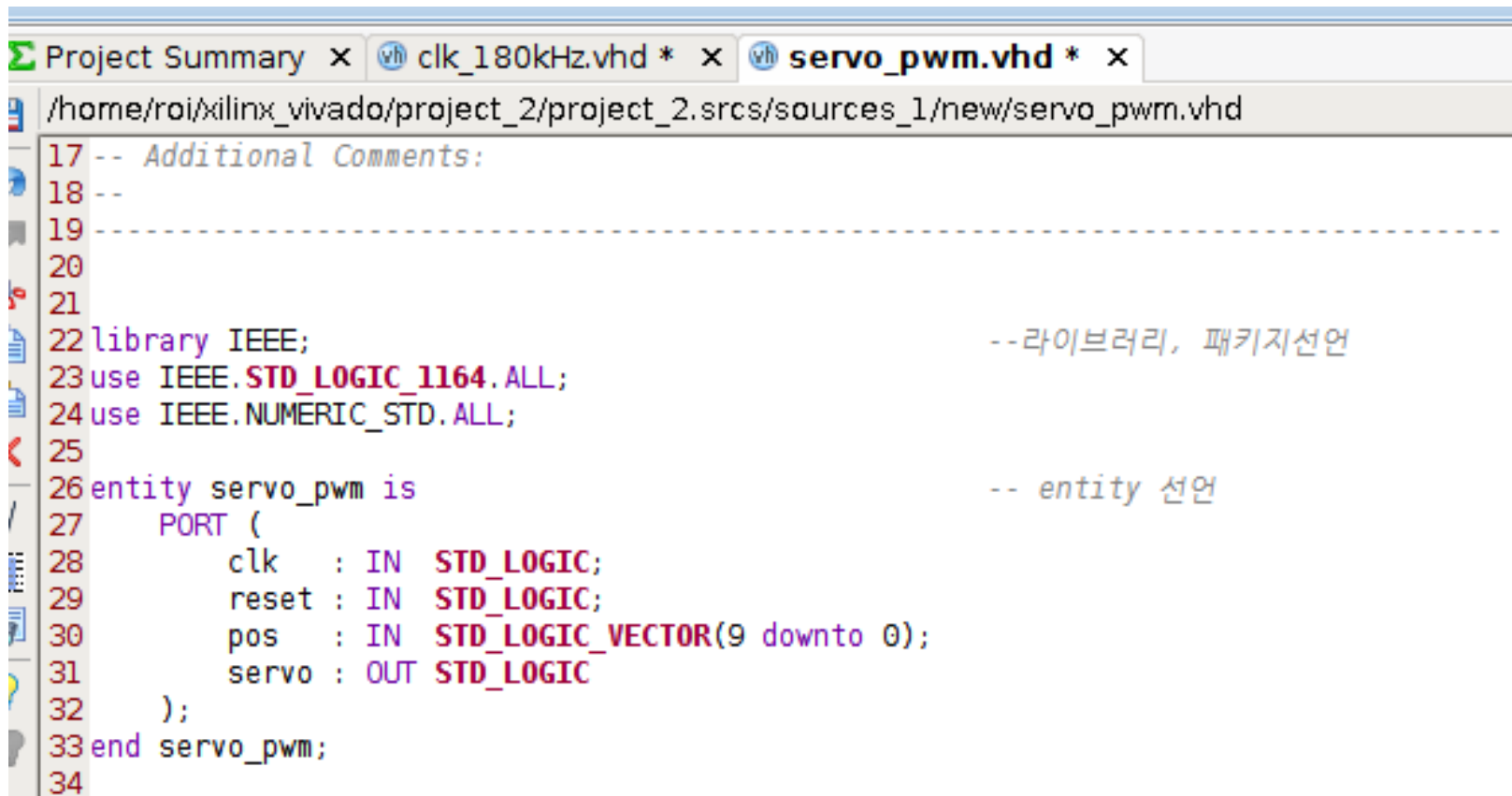
```

--architecture  
--선언부 : 데이터 타입, 신호 및 컴포넌트 등 선언  
-- 구현부 : 회로 구현  
-- reset 설정  
-- counter = 346일때 까지 clk의 rising edge를 세겠다!  
-- counter = 346이면 temporal을 반전  
-- counter < 346일 때, 1씩 증가  
-- temporal 신호를 clk\_out으로!

위에서 150MHz를 694번 세어 준다고 했는데,,  
Duty 비가 50%인 clock을 만들어 주기 위해  
절반인 347(0부터 시작하니까 346이 되어야 함)  
이 되면 temporal를 (0에서 1로, 1에서 0으로)  
반전시켜 준다.



7. 2-6 반복해서 File name01 servo\_pwm인 source 생성 후 코드 입력.



```
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;                                -- 라이브러리, 패키지선언
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.NUMERIC_STD.ALL;
25
26 entity servo_pwm is                          -- entity 선언
27     PORT (
28         clk    : IN  STD_LOGIC;
29         reset  : IN  STD_LOGIC;
30         pos    : IN  STD_LOGIC_VECTOR(9 downto 0);
31         servo  : OUT STD_LOGIC
32     );
33 end servo_pwm;
34
```

```

34
35 architecture Behavioral of servo_pwm is
36
37     signal cnt : unsigned(11 downto 0);
38
39     signal pwmi: unsigned(9 downto 0);
40 begin
41
42     pwmi <= unsigned(pos) + 180;
43
44     counter: process (reset, clk) begin
45         if (reset = '1') then
46             cnt <= (others => '0');
47         elsif rising_edge(clk) then
48             if (cnt = 3599) then
49                 cnt <= (others => '0');
50             else
51                 cnt <= cnt + 1;
52             end if;
53         end if;
54     end process;
55
56     servo <= '1' when (cnt < pwmi) else '0';
57 end Behavioral;
58

```

-- Counter, from 0 to 3599. 20ms를 만들기 위해서 180x20 = 3600이 필요

-- Temporal signal used to generate the PWM pulse.

-- 최소 duty cycle이 1ms가 되어야 하므로 180을 더해줌.

-- Counter process, from 0 to 3599.

-- reset 설정

-- cnt=3599에 도달하면 cnt를 0으로!

-- cnt < 3599 이면 +1

-- pwm 출력

# cnt를 3600(0부터 3599)번 세어주는 이유

- refresh frequency 에 해당하는 20ms를 만들어주기 위함
- 180kHz -> 1s : 180k 개를 세면 1s
- 180 -> 1ms : 180개를 세면 1ms
- => 20ms를 만들어주기 위해서는 180x20 = 3600개를 세어야 함.

## 56줄에 보면 cnt < pwmi 일 때 servo로 출력이 '1'나감

- pwmi = 최소 duty cycle인 1ms(180개 세면 1ms이므로 180)  
+ 원하는 각도(-90일 때를 0으로 생각)의 이진 표현

8. 2-6 반복해서 File name이 servo\_pwm\_180kHz인 source 생성 후 코드 입력.

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity servo_pwm_clk180kHz is                                -- entity
26     PORT(
27         clk : IN  STD_LOGIC;
28         reset: IN  STD_LOGIC;
29         pos  : IN  STD_LOGIC_VECTOR(9 downto 0);
30         servo: OUT STD_LOGIC
31     );
32 end servo_pwm_clk180kHz;
33
34 architecture Behavioral of servo_pwm_clk180kHz is            --architecture
35     COMPONENT clk_180kHz                                       --clk_180kHz 의 인스턴스 생성
36     PORT(
37         clk    : in  STD_LOGIC;
38         reset  : in  STD_LOGIC;
39         clk_out: out STD_LOGIC
40     );
41     END COMPONENT;
42
```

```

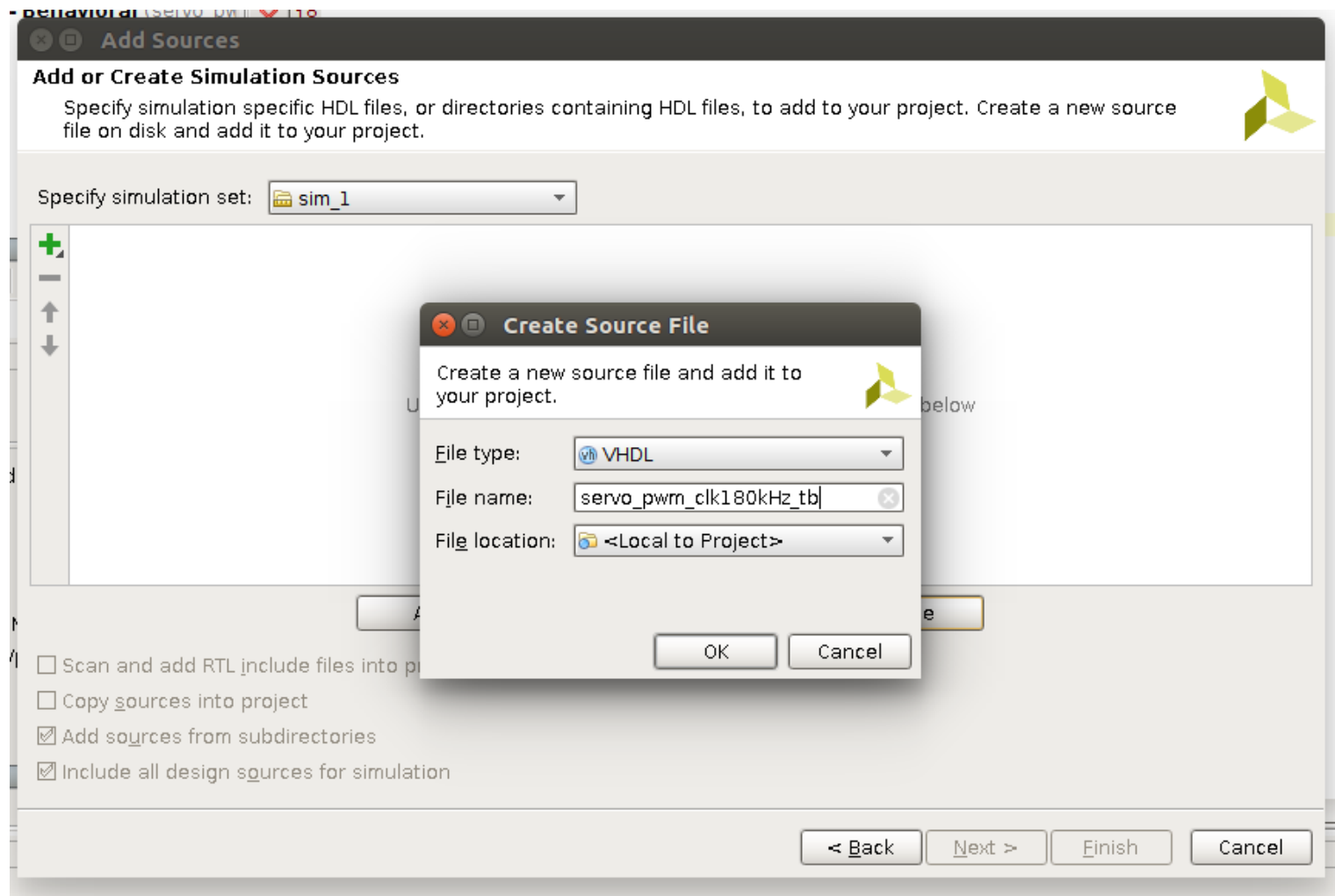
42
43 COMPONENT servo_pwm                                --servo_pwm의 인스턴스 생성
44     PORT (
45         clk    : IN  STD_LOGIC;
46         reset  : IN  STD_LOGIC;
47         pos    : IN  STD_LOGIC_VECTOR(9 downto 0);
48         servo  : OUT STD_LOGIC
49     );
50 END COMPONENT;
51
52 signal clk_out : STD_LOGIC := '0';
53 begin
54     clk180kHz_map: clk_180kHz PORT MAP(              --인스턴스 이름 : 하위레벨 entity 이름
55         clk=>clk, reset=>reset, clk_out=>clk_out      --포트 연결
56     );
57
58     servo_pwm_map: servo_pwm PORT MAP(               --인스턴스 이름 : 하위레벨 entity 이름
59         clk=>clk_out, reset=>reset, pos=>pos, servo=>servo --포트 연결
60     );
61 end Behavioral;
62

```

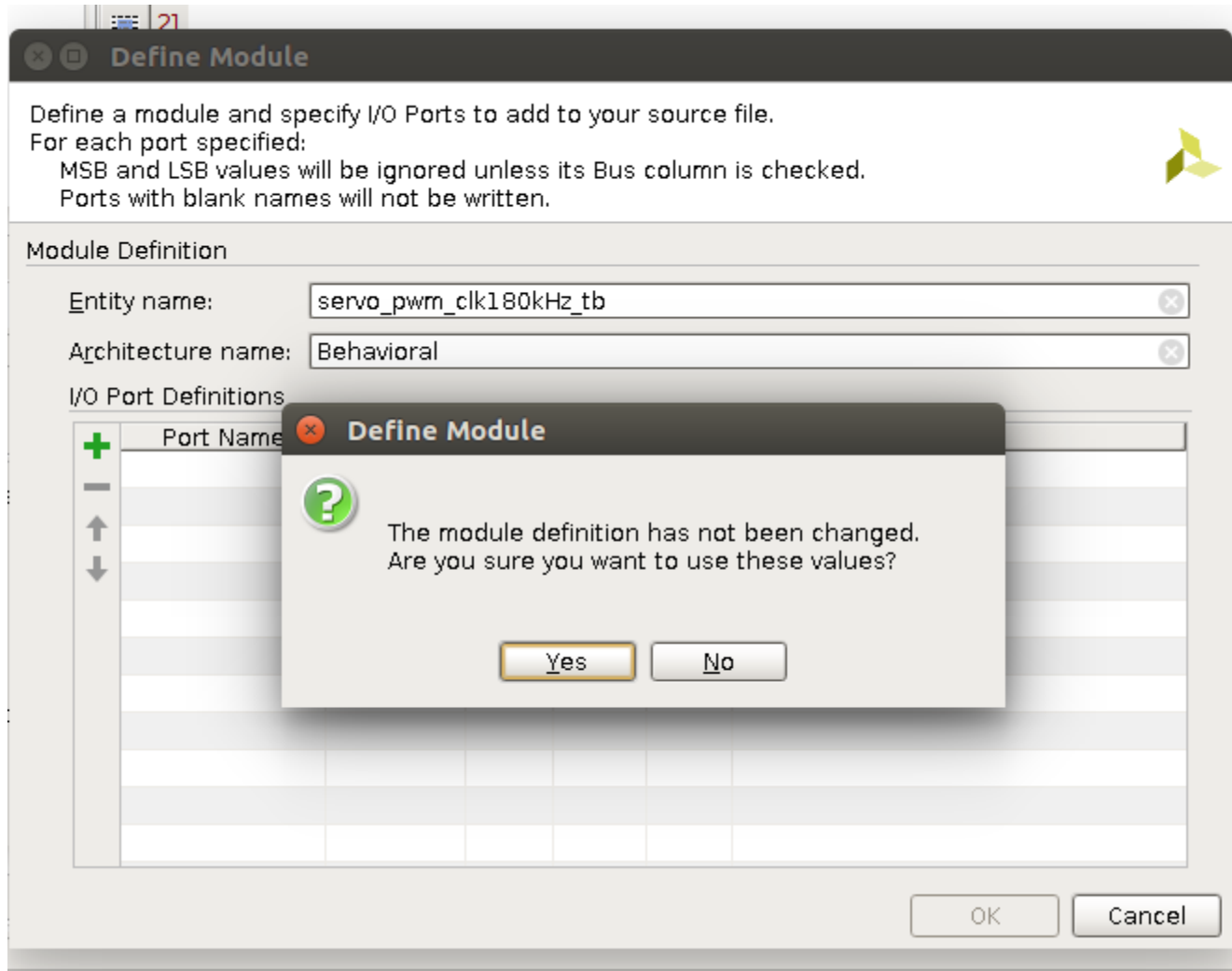
9. Flow Navigator > Project Manager> Add Sources 클릭

10. 'Add or create simulation sources'체크 후 Next>버튼

11. 'Create Source File'창에서 File name에 servo\_pwm\_clk180kHz\_tb입력 후  
'Ok'클릭 > 'Finish' 클릭



## 12. 'Define Module'창 에서 'Ok'클릭 > 'Yes' 클릭



### 13. 'Simulation Sources'>sim\_1>servo\_pwm\_clk180kHz\_tb 더블 클릭 후 코드 입력

```
rvo_pwm_clk180kHz_tb.vhd*
/home/roi/xilinx_vivado/project_2/project_2.srscs/sim_1/new/servo_pwm_clk180kHz_tb.vhd
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  -----
20
21
22  LIBRARY ieee;                                -- 라이브러리 및 패키지 선언
23  USE ieee.std_logic_1164.ALL;
24
25  ENTITY servo_pwm_clk180kHz_tb IS              -- 테스트 벤치는 port 선언이 필요 없음
26  END servo_pwm_clk180kHz_tb;
27
28  ARCHITECTURE behavior OF servo_pwm_clk180kHz_tb IS
29    -- Unit under test.
30    COMPONENT servo_pwm_clk180kHz              -- 검증하고자 하는 모듈을 인스턴스로 생성하기 위해 component 선언
31    PORT(
32        clk    : IN  std_logic;
33        reset  : IN  std_logic;
34        pos    : IN  std_logic_vector(9 downto 0);
35        servo  : OUT std_logic
36    );
37  END COMPONENT;
38
```

```

38
39 -- Inputs.
40 signal clk : std_logic := '0';
41 signal reset: std_logic := '0';
42 signal pos : std_logic_vector(9 downto 0) := (others => '0');
43 -- Outputs.
44 signal servo : std_logic;
45 -- Clock definition.
46 constant clk_period : time := 8 ns;
47 BEGIN
48 -- Instance of the unit under test.
49 uut: servo_pwm_clk180kHz PORT MAP (
50     clk => clk,
51     reset => reset,
52     pos => pos,
53     servo => servo
54 );
55
56
57 clk_process :process begin
58     clk <= '0';
59     wait for clk_period/2;
60     clk <= '1';
61     wait for clk_period/2;
62 end process;
63
64
65 stimuli: process begin
66     reset <= '1';
67     wait for 50ns;
68     reset <= '0';
69     wait for 50ns;
70     pos <= B"00_0000_0000";
71     wait for 20ms;
72     pos <= B"00_0011_1100";
73     wait for 20ms;
74     pos <= B"00_0111_1000";
75     wait for 20ms;
76     pos <= B"00_1011_0100";
77     wait;
78 end process;
79 END;

```

-- 생성한 인스턴스와 포트 연결할 시그널 선언

-- 외부 125MHz의 클럭 주기  $1/125\text{MHz} = 8\text{ns}$

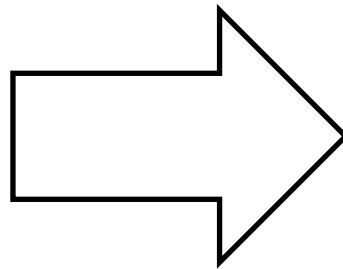
-- 인스턴스 생성

-- clock 프로세스 정의

-- Stimuli process : 검증하고자 하는 모듈의

-- 입력포트에 인가할 입력 신호들을 만들어 줌

1.5ms pwm 신호를 0도라 할 때



-90도 => 00\_0000\_0000

-30도 => 00\_0011\_1100

+30도 => 00\_0111\_1000

+90도 => 00\_1011\_0100



14. Ctrl+S 눌러 저장

15. Flow Manager>Simulation>Simulation Settings>Simulation 에서 runtime을 100ms로 설정 >Apply 클릭 > Ok 클릭

The screenshot displays the Xilinx Vivado Flow Navigator interface. On the left, the 'Project Manager' tree shows the project structure, including 'servo\_pwm\_clk180kHz - Behavioral' and 'sim\_1'. The 'Simulation' section is highlighted in the left sidebar. The main window shows the 'Project Settings' dialog box with the 'Simulation' tab selected. The 'Target simulator' is set to 'Vivado Simulator', 'Simulator language' is 'VHDL', and 'Simulation set' is 'sim\_1'. The 'Simulation top module name' is 'servo\_pwm\_clk180kHz\_tb'. The 'Clean up simulation files' checkbox is checked. The 'Compilation' tab is selected, and the 'runtime' is set to '100ms'. The 'Advanced' tab is also visible. The background shows a code editor with VHDL code for 'servo\_pwm\_clk180kHz\_tb.vhd'.

**Project Manager - project\_2**

Sources

- Design Sources (1)
  - servo\_pwm\_clk180kHz - Behavioral (servo\_pwm)
  - clk180kHz\_map - clk\_180kHz - Behavioral (clk\_180k)
  - servo\_pwm\_map - servo\_pwm - Behavioral (servo\_p)
- Constraints
- Simulation Sources (1)
  - sim\_1 (1)
    - servo\_pwm\_clk180kHz\_tb - behavi

**Project Settings**

**Simulation**

Target simulator: Vivado Simulator

Simulator language: VHDL

Simulation set: sim\_1

Simulation top module name: servo\_pwm\_clk180kHz\_tb

☒ Clean up simulation files

☐ Generate scripts only

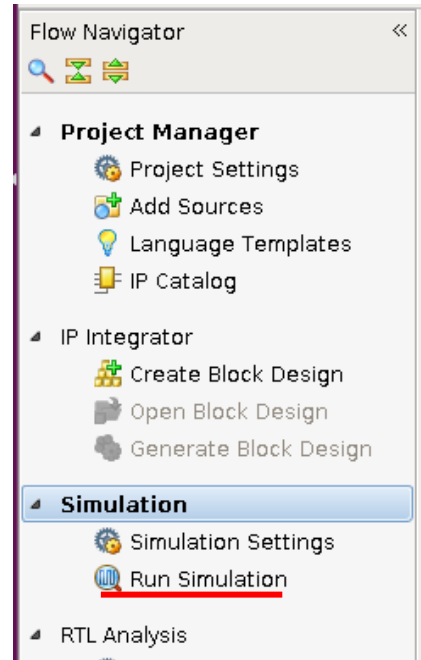
**Compilation** **Elaboration** **Simulation** **Netlist** **Advanced**

Property	Value
xsim.simulate.runtime*	100ms
xsim.simulate.uut	
xsim.simulate.wdb	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	<input type="checkbox"/>
xsim.simulate.xsim.more_options	

**xsim.simulate.runtime\***  
Specify simulation run time

OK Cancel Apply

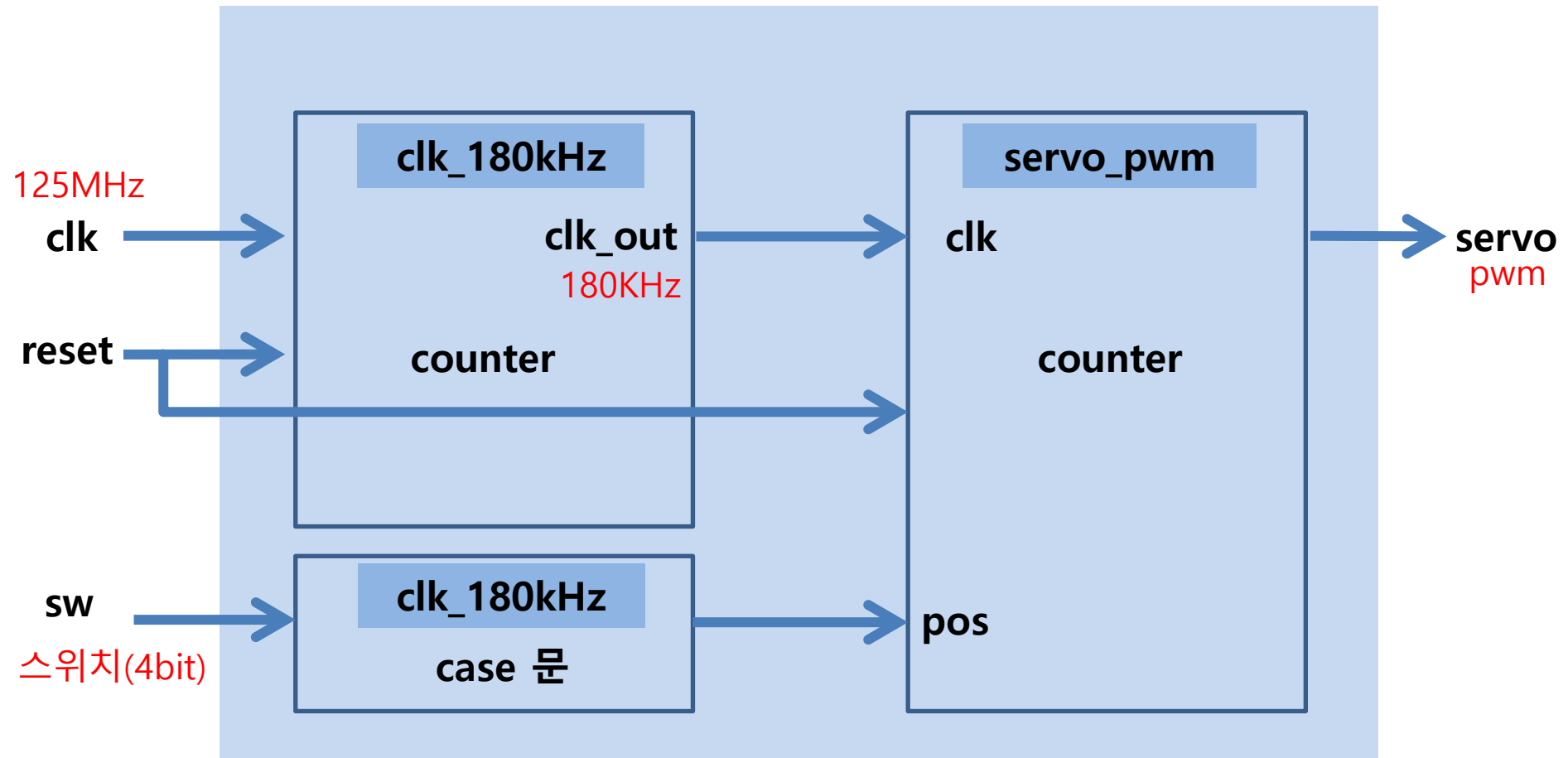
## 16. Flow Manager>Simulation>Run Simulation > Run Behavioral Simulation 클릭하여 시뮬레이션 시작



# PWM 생성 flow – 수정

- switch(4bit)로 pos(8bit) 입력 후 실제 서보 모터 제어

**servo\_pwm\_clk180kHz**



## 17. 2-6 반복해서 File name이 swt인 source 생성 후 코드 입력

```
Project Summary x swt.vhd * x
/home/roi/xilinx_vivado/project_2/project_2.srcs/sources_1/new/swt.v
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity swt is
26 -- Port ( );
27 port(
28     sw : in std_logic_vector(3 downto 0);
29     pos : out std_logic_vector(7 downto 0)
30 );
31
32 end swt;
```

버튼(4bit)이 1씩 증가할 때  
마다 10°씩 증가

```
34 architecture Behavioral of swt is
35
36 begin
37     P1: process
38
39         begin
40             swt : case sw is
41                 when "0000" => pos <= X"00";
42                 when "0001" => pos <= X"10";
43                 when "0010" => pos <= X"20";
44                 when "0011" => pos <= X"30";
45                 when "0100" => pos <= X"40";
46                 when "0101" => pos <= X"50";
47                 when "0110" => pos <= X"60";
48                 when "0111" => pos <= X"70";
49                 when "1000" => pos <= X"80";
50
51                 when "1001" => pos <= X"90";
52                 when "1010" => pos <= X"a0";
53                 when "1011" => pos <= X"b0";
54                 when "1100" => pos <= X"c0";
55                 when "1101" => pos <= X"d0";
56                 when "1110" => pos <= X"e0";
57                 when "1111" => pos <= X"f0";
58                 when others => pos <= X"00";
59             end case swt;
60         end process;
61
62
63 end Behavioral;
```

case구문 사용

## 18. servo\_pwm, servo\_pwm\_clk180kHz 코드 수정

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.NUMERIC_STD.ALL;
25
26 entity servo_pwm is
27     PORT (
28         clk : IN STD_LOGIC;
29         reset : IN STD_LOGIC;
30         pos : IN STD_LOGIC_VECTOR(7 downto 0);
31         servo : OUT STD_LOGIC
32     );
33 end servo_pwm;
34
35 architecture Behavioral of servo_pwm is
36
37     signal cnt : unsigned(11 downto 0);
38
39     signal pwmi : unsigned(9 downto 0);
40 begin
41
42     pwmi <= "00" & unsigned(pos) + 180;
43
44     counter: process (reset, clk) begin
45         if (reset = '1') then
46             cnt <= (others => '0');
47         elsif rising_edge(clk) then
48             if (cnt = 3599) then
49                 cnt <= (others => '0');
50             else
51                 cnt <= cnt + 1;
52             end if;
53         end if;
54     end process;
55
56     servo <= '1' when (cnt < pwmi) else '0';
57 end Behavioral;
58
```

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity servo_pwm_clk180kHz is
26     PORT(
27         clk : IN STD_LOGIC;
28         reset: IN STD_LOGIC;
29         sw : in std_logic_vector(3 downto 0);
30
31         servo: OUT STD_LOGIC
32     );
33 end servo_pwm_clk180kHz;
34
35 architecture Behavioral of servo_pwm_clk180kHz is
36     component swt
37         Port (
38             sw : in std_logic_vector(3 downto 0);
39             pos : out std_logic_vector(7 downto 0)
40         );
41     end component;
42
43 COMPONENT clk_180kHz
44     PORT(
45         clk : in STD_LOGIC;
46         reset : in STD_LOGIC;
47         clk_out: out STD_LOGIC
48     );
49 END COMPONENT;
50
51 COMPONENT servo_pwm
52     PORT (
53         clk : IN STD_LOGIC;
54         reset : IN STD_LOGIC;
55         pos : IN STD_LOGIC_VECTOR(7 downto 0);
56         servo : OUT STD_LOGIC
57     );
58 END COMPONENT;
59
60 signal clk_out : STD_LOGIC := '0';
61 signal pos : STD_LOGIC_VECTOR(7 downto 0);
62
63
```

```

38 port(
39     sw : in std_logic_vector(3 downto 0);
40     pos : out std_logic_vector(7 downto 0)
41 );
42
43 end component;
44
45 COMPONENT clk_180kHz --cl
46     PORT(
47         clk      : in  STD_LOGIC;
48         reset    : in  STD_LOGIC;
49         clk_out   : out STD_LOGIC
50     );
51 END COMPONENT;
52
53 COMPONENT servo_pwm --se
54     PORT (
55         clk      : IN  STD_LOGIC;
56         reset    : IN  STD_LOGIC;
57         pos      : IN  STD_LOGIC_VECTOR(7 downto 0);
58         servo    : OUT STD_LOGIC
59     );
60 END COMPONENT;
61
62 signal clk_out : STD_LOGIC := '0';
63 signal pos     : STD_LOGIC_VECTOR(7 downto 0);
64 begin
65     swt_in : swt port map(
66         sw=>sw, pos => pos
67     );
68


```


## 18-1. constraints 재설정


- 구글에 zybo xdc 치고 xilinx github에서 올려져 있는 ZYBO 핀 중에서 사용할 핀만 Ctrl+ C, V 해서 constraint 파일을 만들어 줌

[Code](#) [Issues 3](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Insights](#)

Branch: master [ZYBO / Resources / XDC / ZYBO\\_Master.xdc](#) [Find file](#) [Copy path](#)

 jpeyron zybo: xdc: fixed issue 9643c8e on Sep 30 2016

3 contributors 

147 lines (116 sloc) 11.4 KB [Raw](#) [Blame](#) [History](#) 

```
1  ## This file is a general .xdc for the ZYBO Rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used signals according to the project
5
6
7  ##Clock signal
8  #set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
9  #create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];
10
11
12  ##Switches
13  #set_property -dict { PACKAGE_PIN G15    IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
14  #set_property -dict { PACKAGE_PIN P15    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L24P_T3_34 Sch=SW1
15  #set_property -dict { PACKAGE_PIN W13    IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
16  #set_property -dict { PACKAGE_PIN T16    IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3
17
18
19  ##Buttons
```

- Constraint 폴더에서 우클릭 한 다음 add sources>Add or create constraints  
체크 > Next 클릭 > Create File > File name : zybo 인 파일 만들기
- L16번 : clock, R18번 : reset button,
- G15번 : sw[0], P15번 : sw[1], W13번 : sw[2], T16번 : sw[3]
- T20번 : servo 로 설정한다.

```

6
7 #Clock signal
8 set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
9 create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];
10
11 ##Buttons
12 set_property -dict { PACKAGE_PIN R18    IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L20N_T3_34 Sch=BTN0
13
14 #Switches
15 set_property -dict { PACKAGE_PIN G15    IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
16 set_property -dict { PACKAGE_PIN P15    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L24P_T3_34 Sch=SW1
17 set_property -dict { PACKAGE_PIN W13    IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
18 set_property -dict { PACKAGE_PIN T16    IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3
19
20 #Pmod Header JB
21 set_property -dict { PACKAGE_PIN T20    IOSTANDARD LVCMOS33 } [get_ports { servo }]; #IO_L15P_T2_DQS_34 Sch=JB1_p
22
23

```



## 19. ZYBO 보드의 JP5를 JTAG에 두고 ZYBO와 컴퓨터 usb 연결



## 20. ZYBO reference manual 16. Pmod Ports 부분을 참고해 서보 모터의 pwm선 (주황색)과 ZYBO (JB T20핀) 연결, 서보 모터 전원(5v), ground 연결

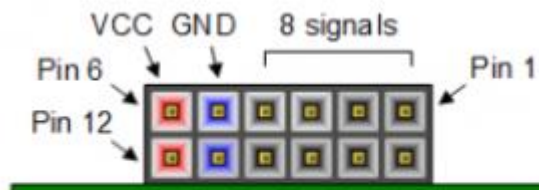
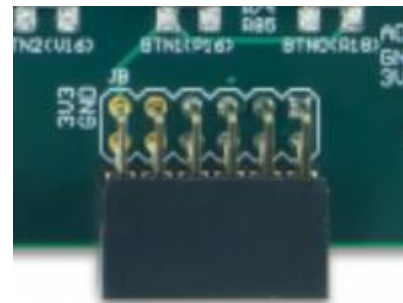
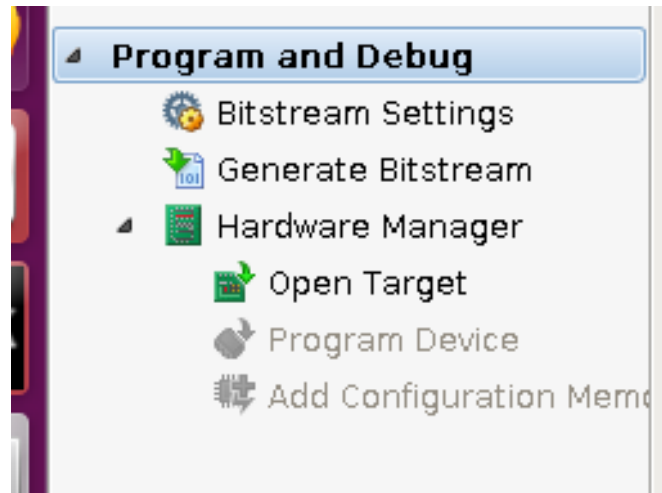


Figure 16. Pmod diagram.

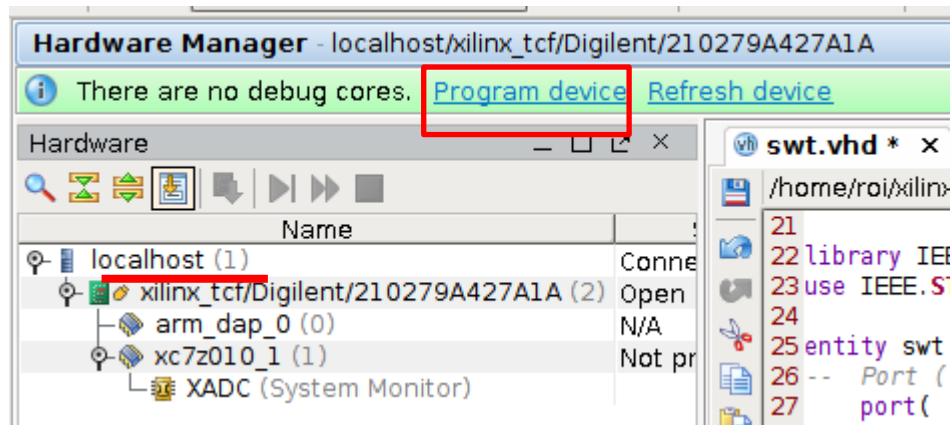
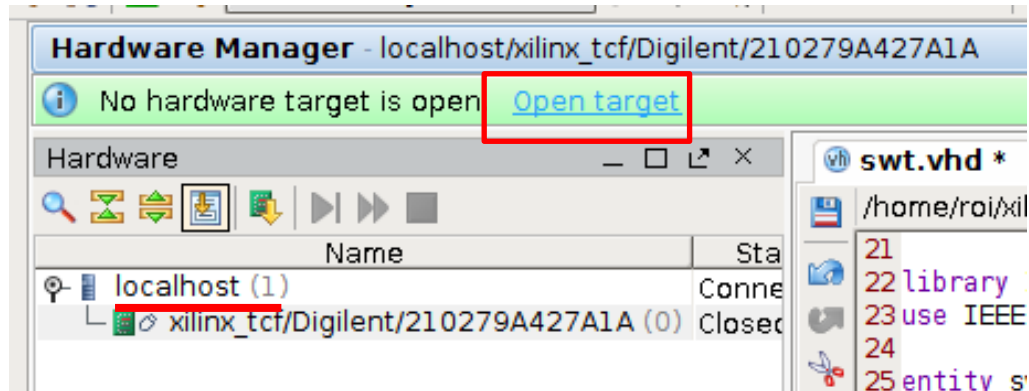


JB12 : PWR	JB06 : PWR
JB11 : GND	JB05 : GND
JB10 : W19	JB04 : W20
JB09 : W18	JB03 : V20
JB08 : Y19	JB02 : U20
JB07 : Y18	JB01 : T20

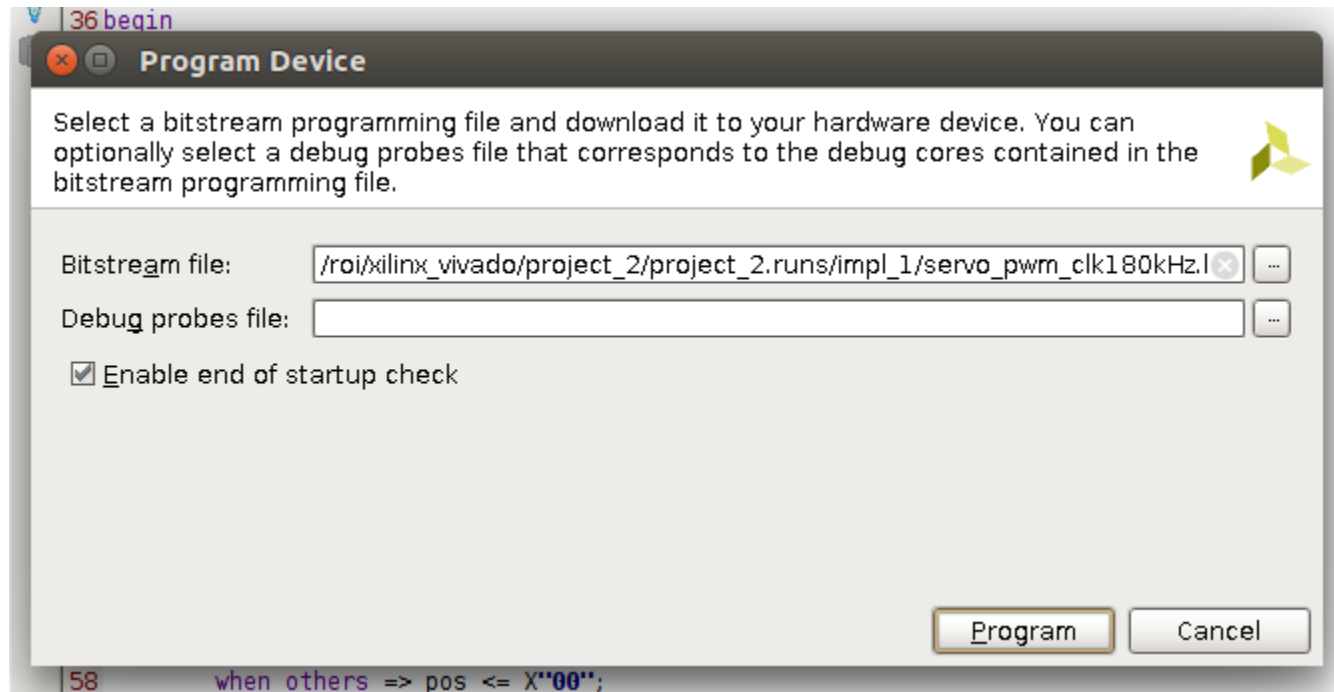
21. Flow Navigator>Project Manager>Program and Debug>Open Hardware Manager 클릭



22. ZYBO 의 전원을 켜고 Hardware Manager>Open target>Auto target 클릭하면 local host로 장치가 인식되는 것을 확인할 수 있음



23. Program device 클릭, Program Device 창이 뜨면 Program 클릭, 완료되면 ZYBO에 노란 led에 불이 들어옴



24. 스위치를 올리거나 내리면 모터가 동작하는 것을 확인할 수 있다.

## reference

1. <https://en.wikipedia.org/wiki/Servomotor>
2. [www.thomas.co.kr/tec/download.php?fn=1.서보의%20종류.pdf@A1161682707](http://www.thomas.co.kr/tec/download.php?fn=1.서보의%20종류.pdf@A1161682707)
3. <http://embedded-lab.com/blog/lab-21-servo-motor-control/>
4. [www.ti.com/lit/an/spraa88a/spraa88a.pdf](http://www.ti.com/lit/an/spraa88a/spraa88a.pdf)
5. [http://www.seekic.com/circuit diagram/Basic Circuit/DC motor servo circuit composed of %CE%BCA741.html](http://www.seekic.com/circuit_diagram/Basic_Circuit/DC_motor_servo_circuit_composed_of_%CE%BCA741.html)
6. <https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL>
7. <https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual>