Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

학생 : 김시윤

04 진행상황 및 문제점

김시윤 – FPGA Device Driver

Device Driver 란?

운영체제 내에서 응용프로그램 과 제어하고자 하는 장치를 연결해주는 프로그램.

리눅스 운영체제에서 디바이스(장치)도 파일로 인식한다.
즉 제어하고자 하는 장치 파일의 메모리주소로 접근하여 값을 읽고 쓰는 것이 디바이스 드라이버다.

Vivado에서 설계한 디바이스를 BitStream 하여 Export 하면 장치 정보가 담긴 bit 파일을 설정한 경로에 생성하게 된다. 이 생성된 장치를 페타리눅스 Config 시 읽어오고 Device-Tree 를 작성하여 파일을 생성하여 그 파일을 제어하면 장치가 제어된다.

Device Driver 란?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#define PWM_MAP_SIZE     0x10000
#define PWM_DATA_OFFSET     0x00
#define PWM_TRI_OFFSET        0x04

int num=0;

int main(int argc, char *argv[])
{
    int  fd,i;
    char *uiod;
    void *ptr;
    printf("PWM UIO Test₩n");

    fd = open("/dev/uio0", O_RDWR);

    ptr = mmap(NULL, PWM_MAP_SIZE, PROT_READ|
          PROT_WRITE,MAP_SHARED, fd, 0);
```

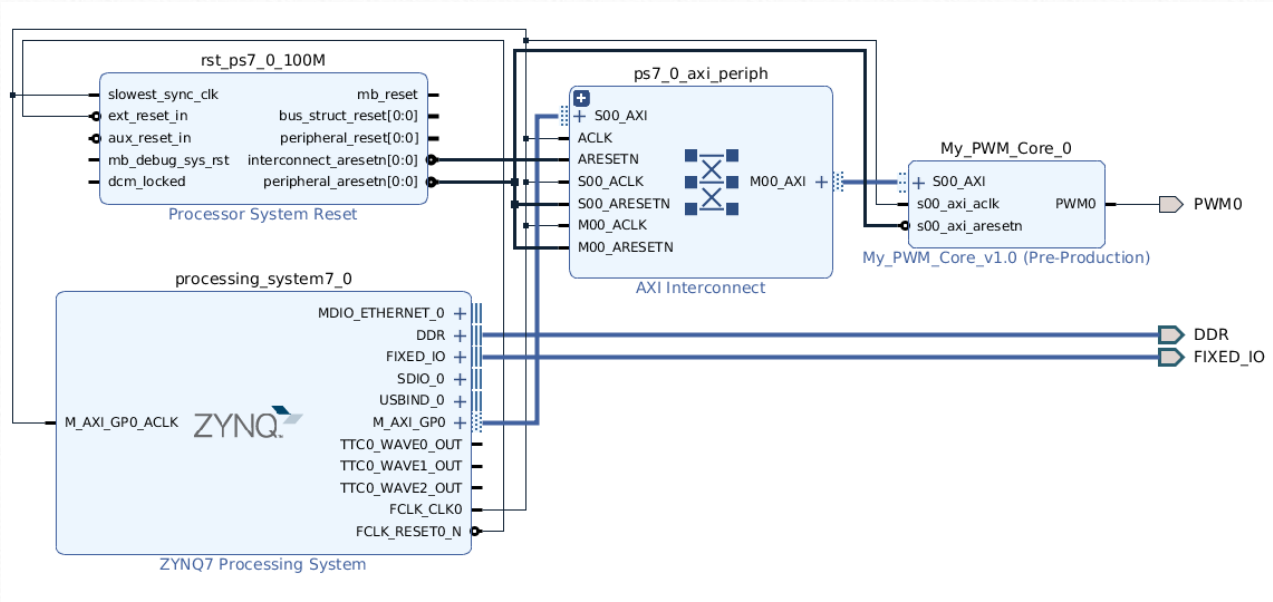
```
        while(1)
        {
            if(num == 2000000)
                num = 0;
            else
            {
                num = num + 1000;


            }
        *((unsigned *)(ptr + PWM_TRI_OFFSET))=0;
        *((unsigned *)(ptr + PWM_DATA_OFFSET))=num;
            for(i = 0; i<30000; i++);
        printf("device_driver TEST₩n₩r");
        }

    munmap(ptr, PWM_MAP_SIZE);
    return 0;
}
```

Write

FPGA PWM


rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
My_PWM_Core_0
S00_AXI
s00_axi_aclk
s00_axi_aresetn
PWM0
My_PWM_Core_v1.0 (Pre-Production)
PWM0
processing_system7_0
MDIO_ETHERNET_0
DDR
FIXED_IO
SDIO_0
USBIND_0
M_AXI_GP0
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
M_AXI_GP0_ACLK
ZYNQ
ZYNQ7 Processing System
DDR
FIXED_IO

FPGA PWM

```
// Add user logic here

reg [31:0] counter = 0;

always @(posedge S_AXI_ACLK) begin

if(counter < PWM_COUNTER_MAX -1)
        counter <= counter +1;
else
        counter <= 0;
end

assign PWM0 = slv_reg0 < counter ? 1'b0 : 1'b1;
// User logic ends
```

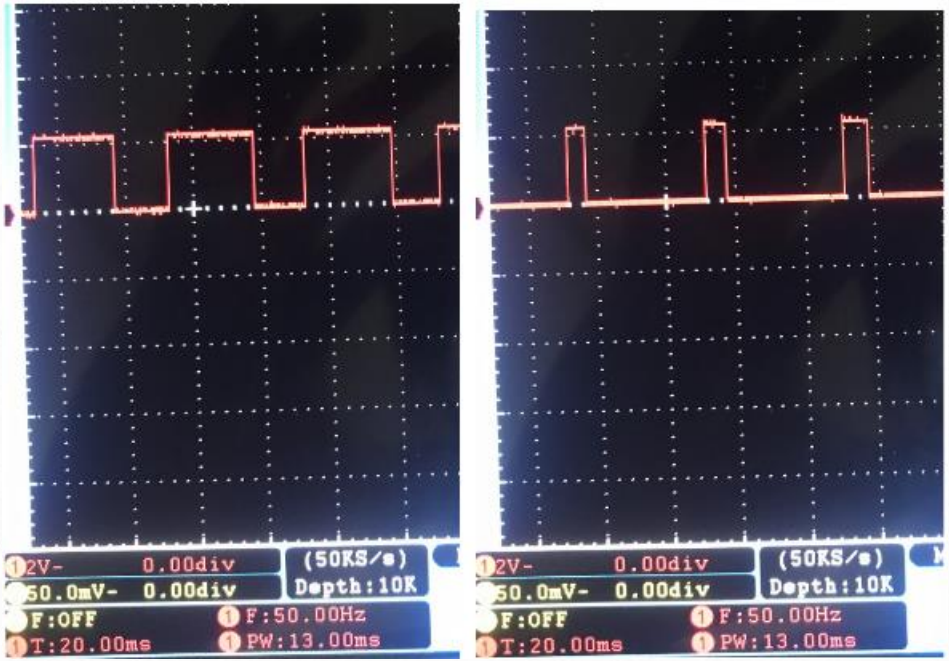
PWM_COUNTER_MAX = 2000000 이다.(주기설정)

S_AXI_ACLK(100MHz) 클럭이 들어올 때 마다 카운터 레지스터의 값을 1씩 증가시킨다.

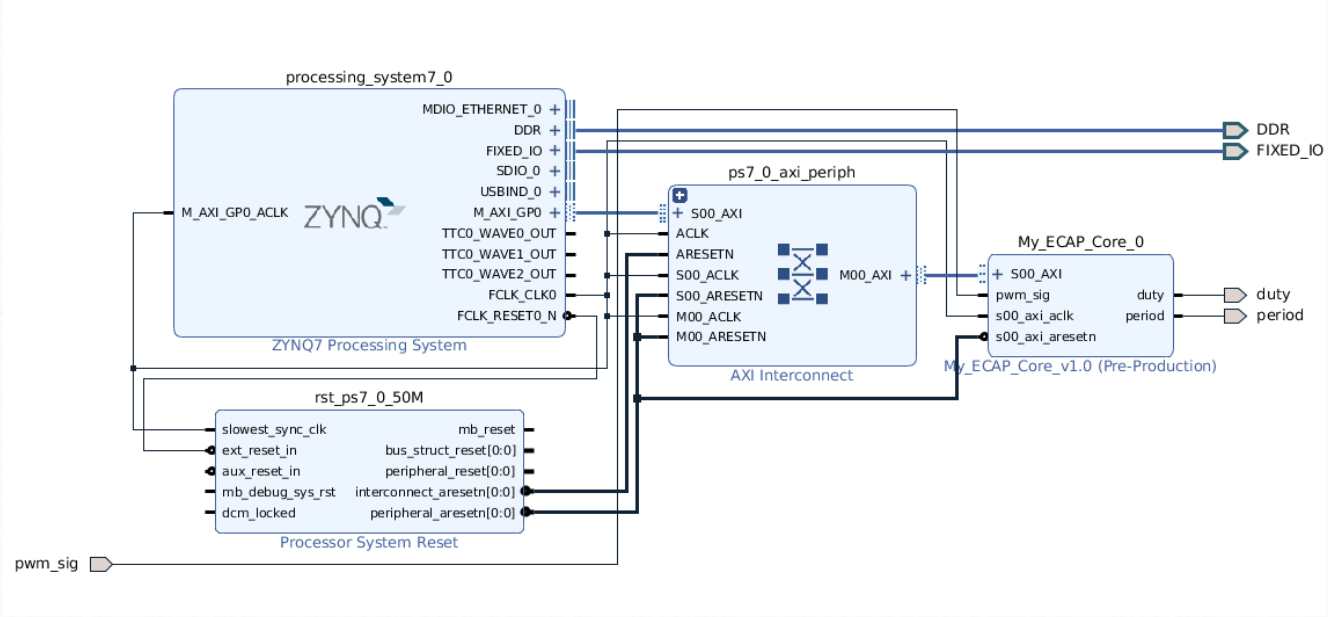
밑에 assign 문에서 slv_reg0 보다 카운터가 크면 0이 출력 카운터가 작으면 1이 출력되는 로직.

여기서 slv_reg0은 zybo 내부 레지스터로 우리는 이 레지스터 값을 읽고 쓴다.

PWM 모듈에서 이 레지스터는 듀티 가 된다.


2V- 0.00div
50.0mV- 0.00div
(50KS/s)
Depth:10K
F:OFF
T:20.00ms
F:50.00Hz
PW:13.00ms
2V- 0.00div
50.0mV- 0.00div
(50KS/s)
Depth:10K
F:OFF
T:20.00ms
F:50.00Hz
PW:13.00ms

FPGA eCAP


processing_system7_0
MDIO_ETHERNET_0
DDR
FIXED_IO
SDIO_0
USBIND_0
M_AXI_GP0_ACLK
M_AXI_GP0
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
My_ECAP_Core_0
S00_AXI
pwm_sig
s00_axi_aclk
s00_axi_aresetn
duty
period
My_ECAP_Core_v1.0 (Pre-Production)
DDR
FIXED_IO
duty
period
rst_ps7_0_50M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
pwm_sig

FPGA eCAP

```
// Add user logic here
always @(posedge S_AXI_ACLK) begin

if(pwm_sig == 1 && past_pwm_sig ==0) begin
                rising_edge_flag = rising_edge_flag + 4'd1;

                if(rising_edge_flag == 4'd1) begin

                                cap1<=counter;
                end
                else if(rising_edge_flag == 4'd2) begin

                                cap3<=counter;
                                duty_e <= cap2 - cap1;
                                duty <= duty_e;
                if(cap3 > cap1) begin

                                period_e <= cap3 - cap1;
                end
                else if(cap1 > cap3) begin

                                period_e <= cap1 - cap3;
                end
                                period <= period_e;
                                counter <= 32'd0;
                                rising_edge_flag <= 4'd0;
                end
end
                else if(pwm_sig == 0 && past_pwm_sig ==1) begin
                                cap2<=counter;
                end
                                past_pwm_sig <= pwm_sig;
                                counter <= counter + 32'd1;
end
// User logic ends
```

2'h0 : reg_data_out <= period;
2'h1 : reg_data_out <= duty;
레지스터에 값을 넣어준다. 이제 이 레지스터를 읽으면 캡쳐한 듀티값과 주기값을 알 수 있다.

FPGA eCAP

```
 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913

 ECAP_TEST
period = 1999825
duty = 999913
```

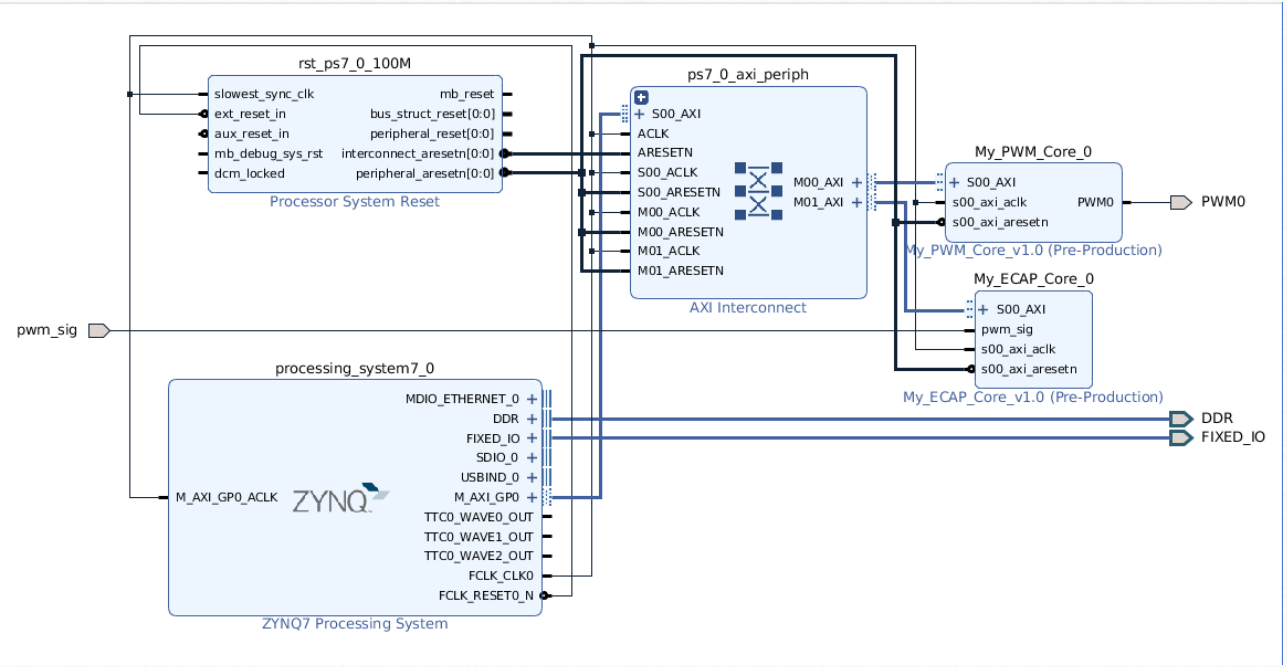
MCU eTPWM 을 활성화 시킨다.
Period 20ms
Duty 50%

FPGA V15 를 eCAP 입력핀으로 사용하였다.
MCU pwm <-> ZYBO V15
MCU GND <-> ZYBO GND
핀은 위와 같이 물려준다.

그리고 Putty 를 실행하면 다음과 같은 동작을 한다.

Zynq 프로세서를 100MHz 로 설정하였으므로 주기 20ms(50Hz)를 카운트 하면 2000000 이 나와야하고 duty 50% 를 카운트 하면 1000000 이 나와야 하지만 MCU에서 나오는 period 는 약간의 오차가 있다.

FPGA eCAP+PWM



FPGA eCAP+PWM

```
pwm : 188800, period = 2000000, duty = 186201
pwm : 188900, period = 2000000, duty = 186201
pwm : 189000, period = 2000000, duty = 186201
pwm : 189100, period = 2000000, duty = 186201
pwm : 189200, period = 2000000, duty = 186201
pwm : 189300, period = 2000000, duty = 186201
pwm : 189400, period = 2000000, duty = 186201
pwm : 189500, period = 2000000, duty = 186201
pwm : 189600, period = 2000000, duty = 186201
pwm : 189700, period = 2000000, duty = 186201
pwm : 189800, period = 2000000, duty = 186201
pwm : 189900, period = 2000000, duty = 186201
pwm : 190000, period = 2000000, duty = 186201
pwm : 190100, period = 2000000, duty = 186201
pwm : 190200, period = 2000000, duty = 186201
pwm : 190300, period = 2000000, duty = 186201
pwm : 190400, period = 2000000, duty = 186201
pwm : 190500, period = 2000000, duty = 186201
pwm : 190600, period = 2000000, duty = 186201
pwm : 190700, period = 2000000, duty = 186201
pwm : 190800, period = 2000000, duty = 186201
pwm : 190900, period = 2000000, duty = 186201
pwm : 191000, period = 2000000, duty = 186201
pwm : 191100, period = 2000000, duty = 186201
pwm : 191200, period = 2000000, duty = 186201
pwm : 191300, period = 2000000, duty = 186201
pwm : 191400, period = 2000000, duty = 186201
pwm : 191500, period = 2000000, duty = 186201
pwm : 191600, period = 2000000, duty = 186201
pwm : 191700, period = 2000000, duty = 186201
pwm : 191800, period = 2000000, duty = 186201
pwm : 191900, period = 2000000, duty = 186201
```

위의 두개를 합쳐 디바이스 드라이버를 생성하여 실행시켜주면 이와 같이 동작한.

소스코드에 딜레이가 있기때문에 캡쳐는 pwm 의 이전 듀티를 캡쳐한다.

문제점 및 개선사항.

PWM 디바이스 드라이버 작성 시 printf 가 없으면 동작하지 않는 현상.
-아직 정확한 이유는 파악 못함.

eCAP 디바이스 드라이버 제작시 레지스터 주소 접근에 대한 어려움.

- void* 타입일 때 는 +4, int * 타입일 때는 +1
- Int * 일때 +1 하면 4바이트가 증가하기 때문.

eCAP + PWM 디바이스 드라이버 제작시 uio0 과 uio1의 순서가 주소 높은게 0 주소 낮은게 1로 생기는 현상.

-디버깅을 통해 알애냄.

감사합니다