TI DSP,MCU 및 Xilinux Zynq

FPGA 프로그래밍 전문가 과정

Zynq Zybo - Vivado + PetaLinux

2018/08/22

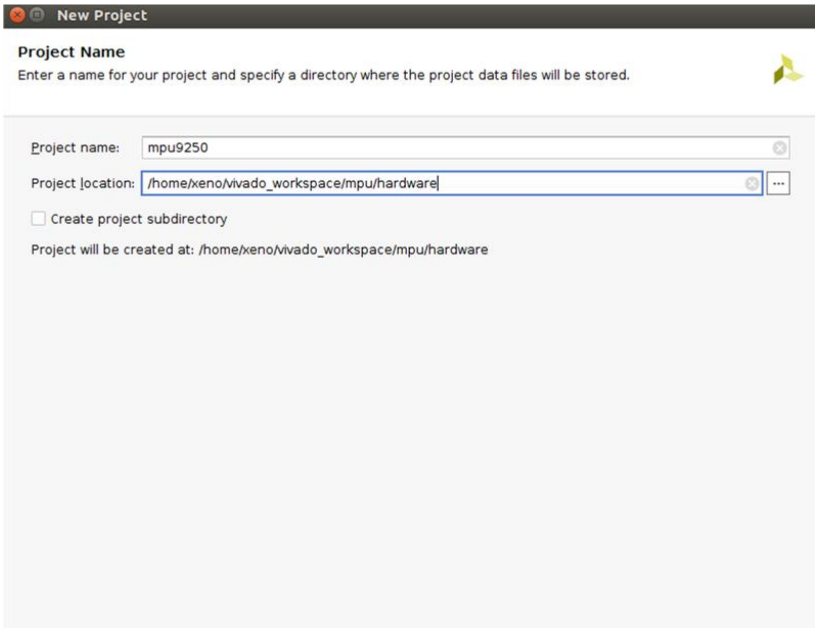
강사 : Innova Lee(이상훈)
강사 메일 : gcccompil3r@gmail.com
학생 : 문지희
학생 메일 : mjh8127@naver.com

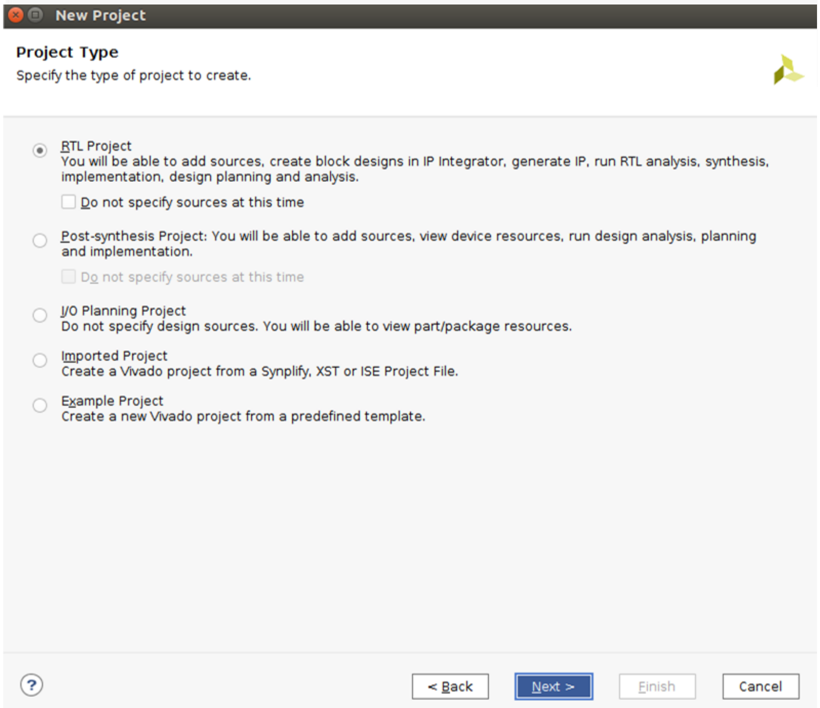
[Vivado]

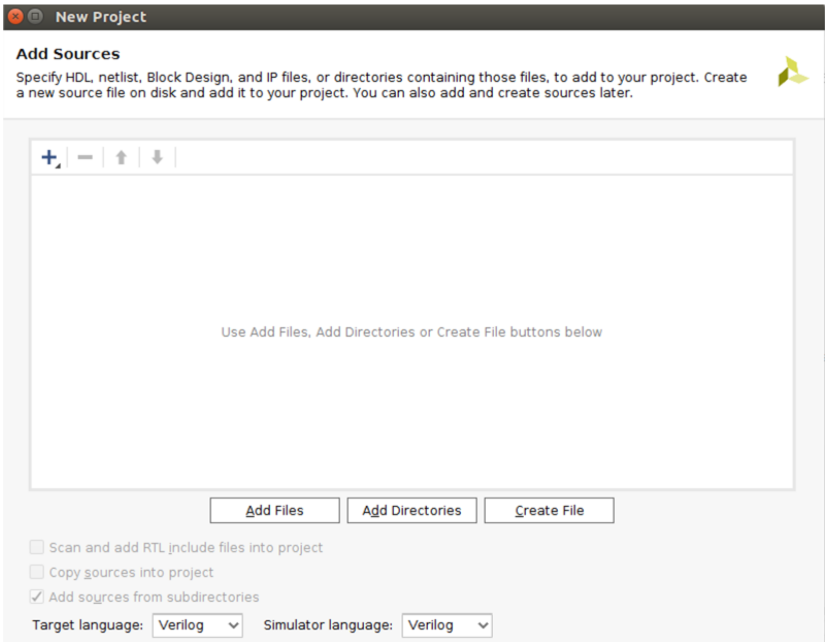
1. 프로젝트 생성

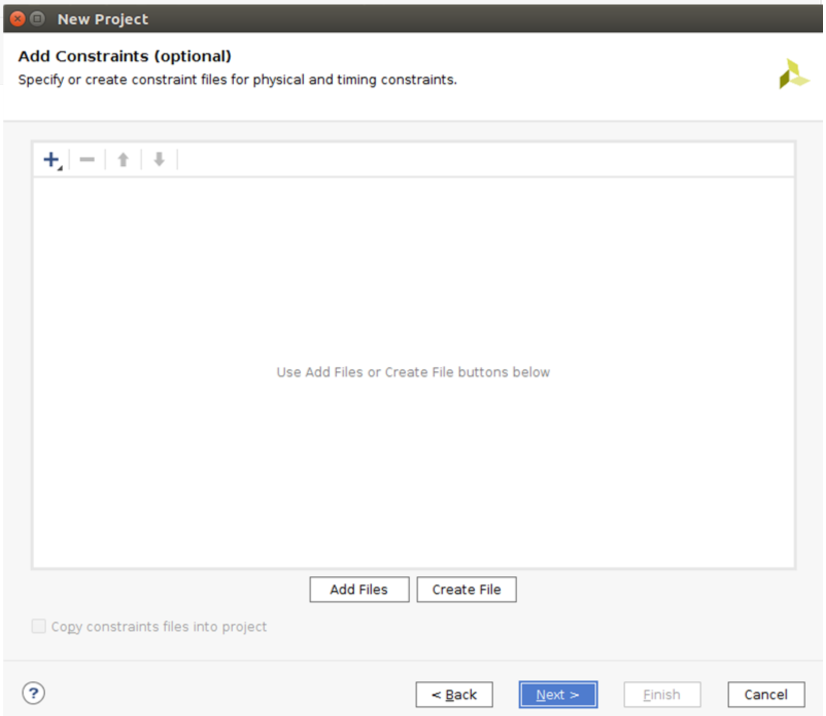
[File - New Project]

mpu9250 전용 폴더를 만든 뒤 hardware 정보를 저장할 폴더를 만든 뒤, 아래 사진처럼 프로젝트 이름과 경로를 설정한다.

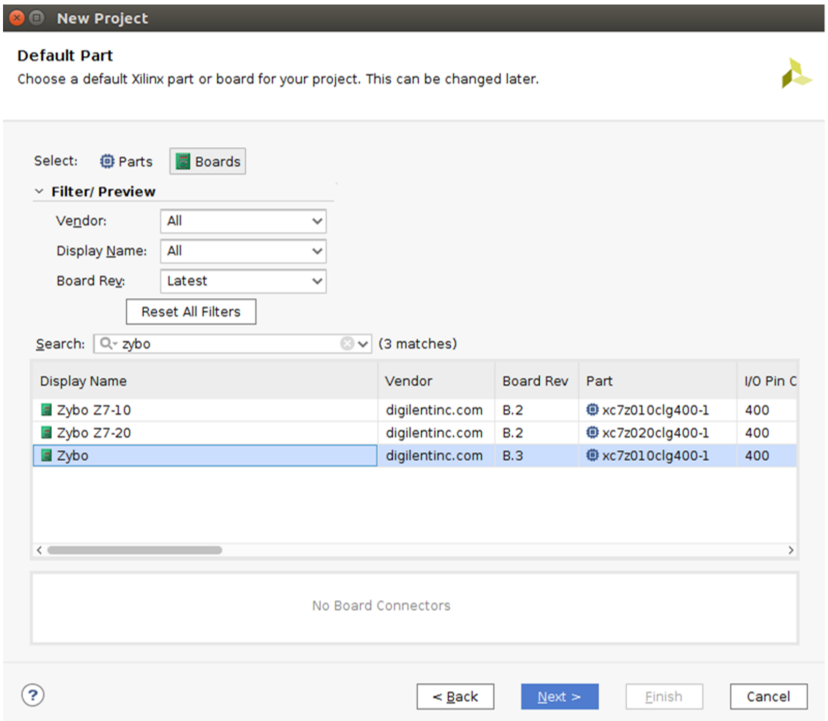




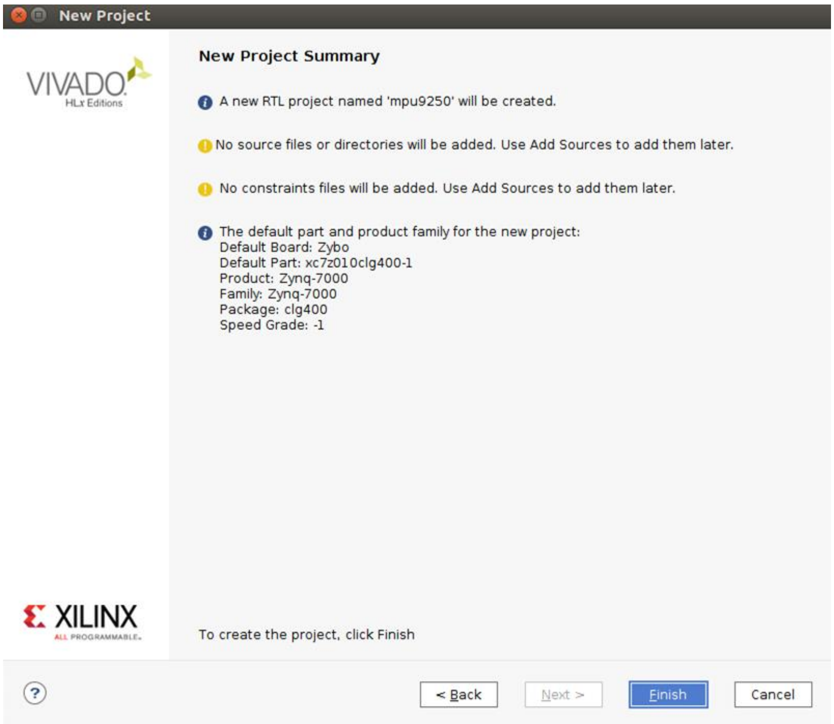




constraints 는 추후 추가함

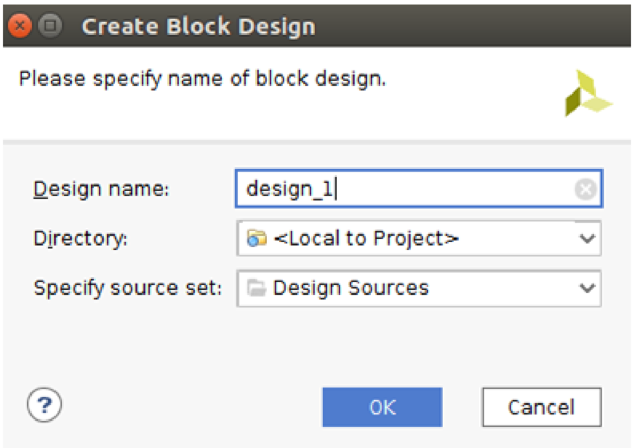


Default Part 에서
Zybo 보드 선택



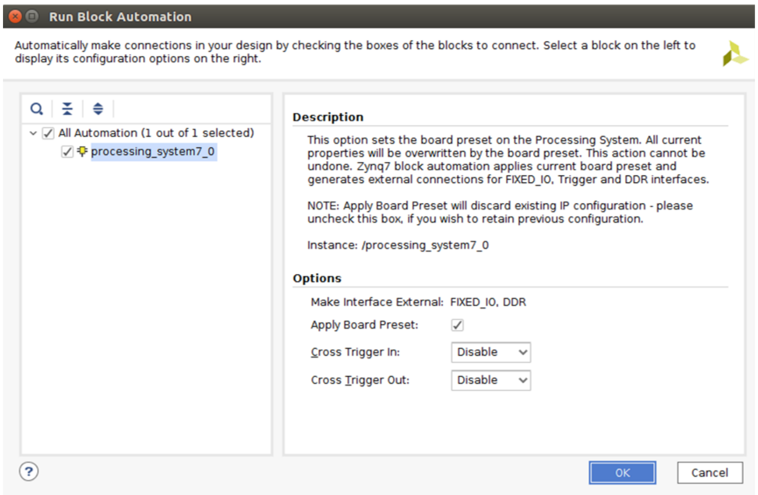
위와 같은 과정을 따라하면
이와 같이 설정이 완료되고
Finish 를 클릭한다.

2. Block Design 설정

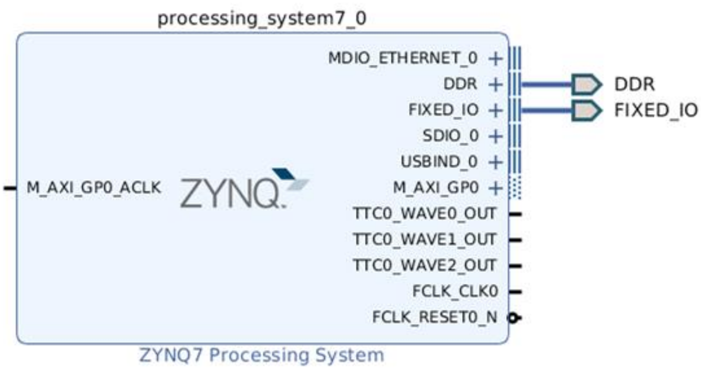


[IP INTEGRATOR - Create Block Design]

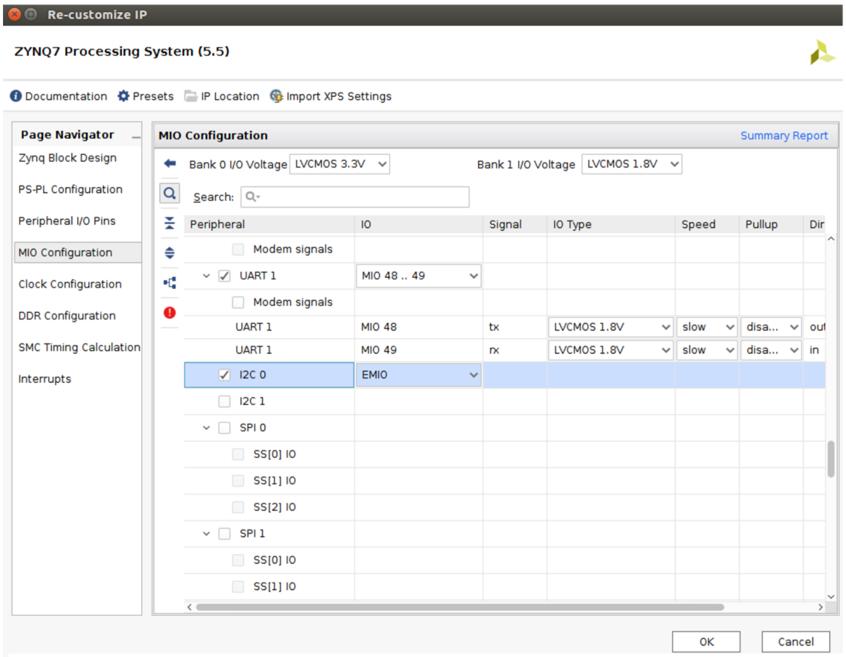
Diagram 에서 ‘ +‘ 를 눌러
‘ZYNQ7 Processing System ‘ 추가



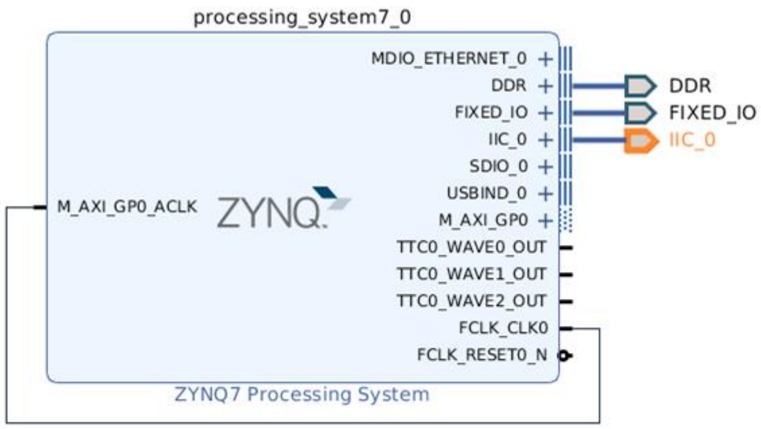
Run Block Automation 을 클릭해
설정한 뒤 OK 를 클릭



Run Block Automation 을
완료하면 왼쪽과 같이 변화한다

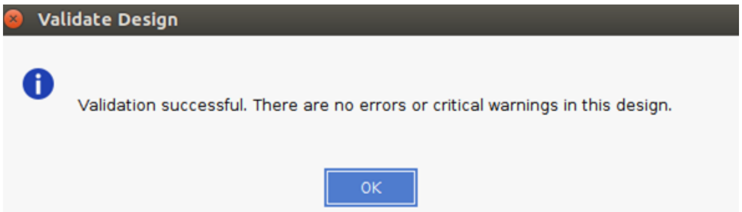


I2C 0 를 선택

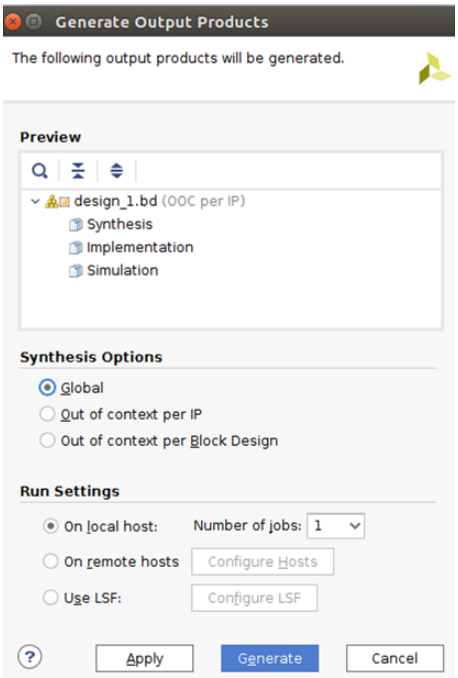


IIC_0 를 우클릭하여
‘ Create Interface Port’ 를
눌러 포트를 생성하고,
M_AXI_GP0_ACLK 와
FCLK_CLK0와 연결한다.

3. Validate Design

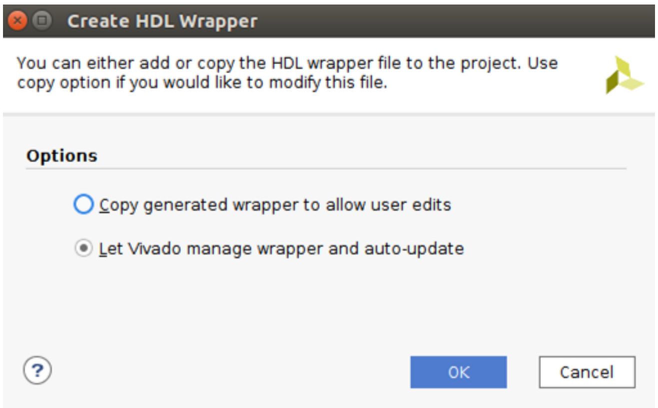


4. Generate Output Products

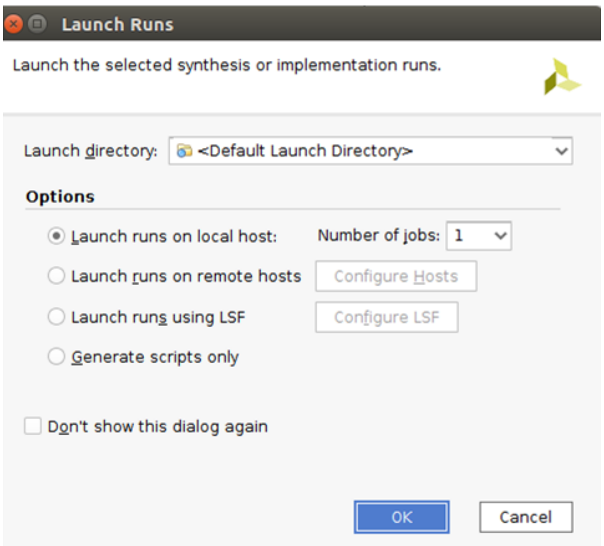


[Sources - Design Sources - Generate Output Products]
Global 을 선택 후 Generate 함

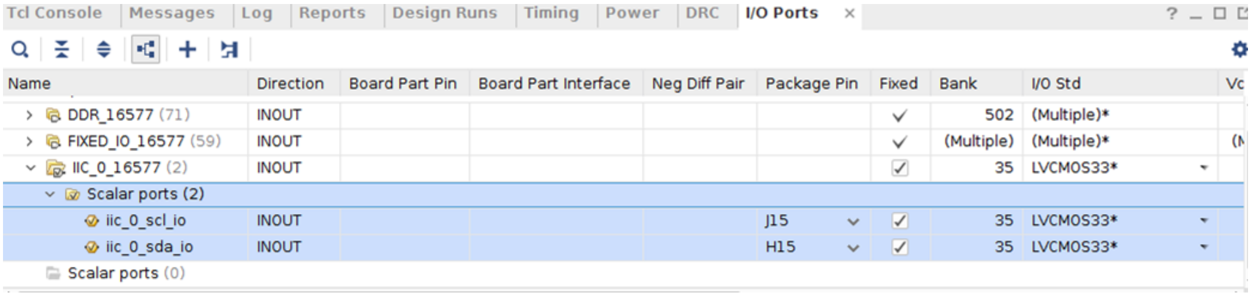
5. Create HDL Wrapper
[Sources - Design Sources - Create HDL Wrapper]



6. Run Implementation



7. I/O Ports 설정

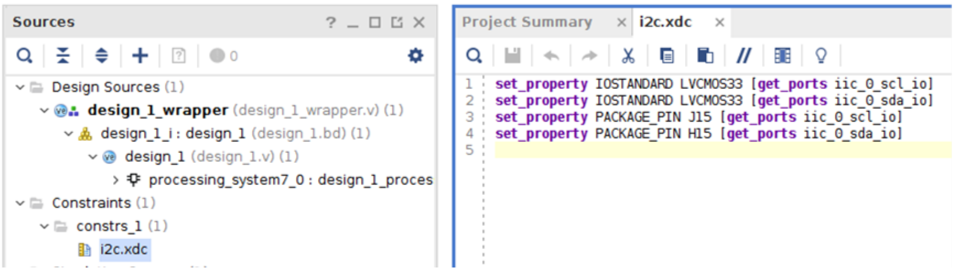


scl - J15

sda - H15

8. Create Constraints

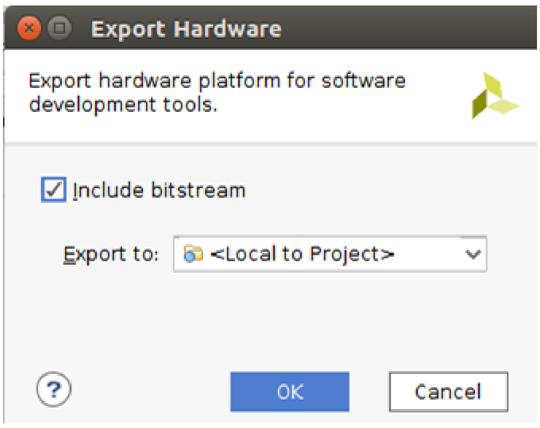
I/O Ports 설정을 하고 Ctrl+S 를 누르면 자동으로 생성된다.



9. Generate Bitstream

[PROGRAM AND DEBUG - Generate Bitstream]

10. Export Hardware



[File - Export - Export Hardware]

‘include bitstream’옵션을 클릭하고 OK

[PetaLinux]

1. **petalinux-create -t project -n software --template zynq**
→ petalinux 프로젝트 생성
mpu 9250 프로젝트 폴더 내에서 실행한다. ls 를 누르면 hardware 폴더와 software 폴더가 존재해야한다.
파란색 글씨는 본인의 임의대로 설정해도 된다.

2. **cd hardware/mpu9250.sdk**
→ 아까 만들었던 하드웨어 정보의 sdk 폴더에 위치

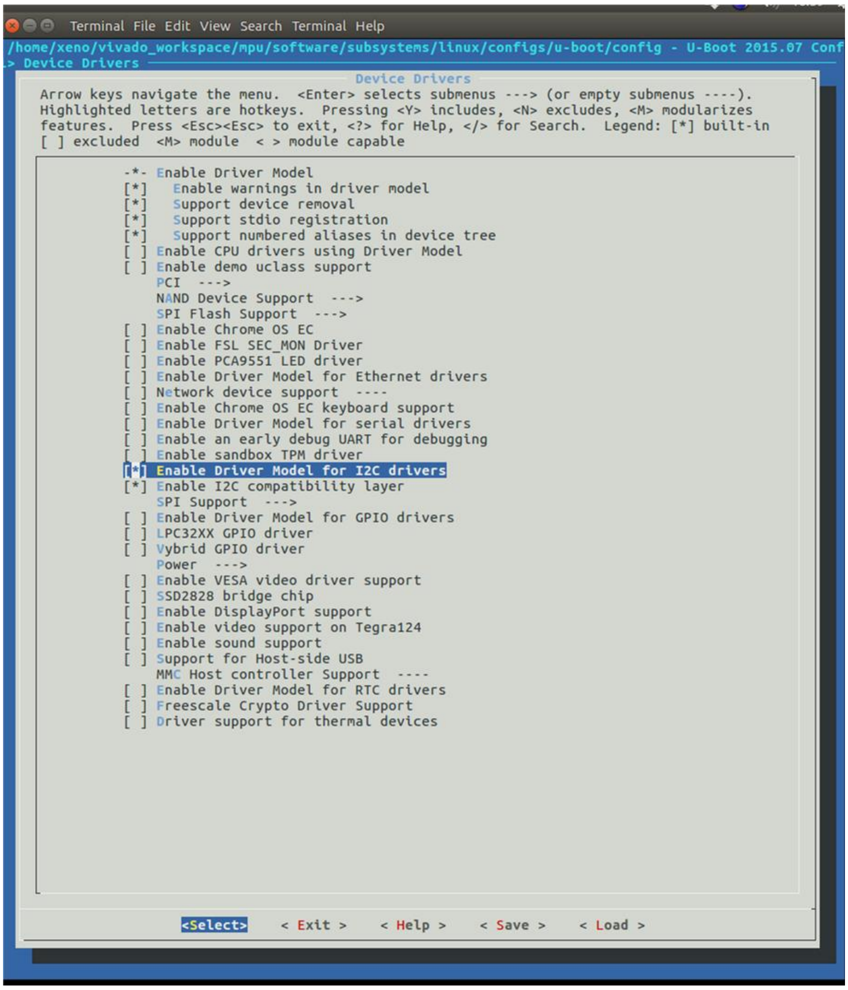
3. **petalinux-config --get-hw-description -p ../../software**
→ 하드웨어 정보를 아까 만들었던 petalinux 프로젝트 폴더에 가져온다.

4. 파란색 배경 창이 뜨면 save 하고 exit 를 눌러 나가준다.

5. **cd ../../software**
→ petalinux 프로젝트 폴더로 이동

6. **petalinux-build**

7. **petalinux-config -c u-boot**



'Device Drivers' 내로 들어오면
왼쪽과 같은 사진이 뜨게 되는데
'Enable Driver Model for
 I2C drivers' 와
'Enable I2C compatiblity layer'
 2개를 선택

petalinux-config -c kernel
petalinux-config -c rootfs
petalinux-config

→ 페타리눅스 설정. petalinux-config -c u-boot 을 제외한 나머지는 save 를 하고 exit 를 누른다.

8. **petalinux-create -t apps -n device_driver --enable**
→ 디바이스 드라이버 소스코드를 작성할 폴더 생성

9. **cd compatible/apps/device_driver**
→ 방금 생성한 디바이스 드라이버 소스코드가 위치할 폴더로 이동

10. 소스코드 작성
- device_driver.c

```
#include "mpu9250.h"

#define    MPU9250_ADDRESS          0x68
#define    MAG_ADDRESS              0x0C

#define    GYRO_FULL_SCALE_250_DPS    0x00
#define    GYRO_FULL_SCALE_500_DPS    0x08
#define    GYRO_FULL_SCALE_1000_DPS   0x10
#define    GYRO_FULL_SCALE_2000_DPS   0x18

#define    ACC_FULL_SCALE_2_G       0x00
#define    ACC_FULL_SCALE_4_G       0x08
#define    ACC_FULL_SCALE_8_G       0x10
#define    ACC_FULL_SCALE_16_G       0x18

#define R2D  180 / M_PI
#define GYRO2DEGREE_PER_SEC  65.5

#define R2D  180 / M_PI
#define GYRO2DEGREE_PER_SEC  65.5


uint32_t rx_data = 0;
uint32_t tmp = 0;
uint32_t value = 0;


volatile char acc_data[6];
volatile char gyro_data[6];
volatile char mag_data[8];


#define IDX     2

int32_t duty_arr[IDX] = {1000, 2000};

//Declaring some global variables
float ax, ay, az, gx, gy, gz, mx, my, mz; // variables to hold latest sensor data values
```

```
double pitch;
double roll;

int16_t accelCount[3];  // Stores the 16-bit signed accelerometer sensor output
int16_t gyroCount[3];   // Stores the 16-bit signed gyro sensor output
int16_t magCount[3];    // Stores the 16-bit signed magnetometer sensor output

float magCalibration[3] = {0, 0, 0} , magbias[3] = {0, 0, 0};  // Factory mag calibration and mag bias
float gyroBias[3] = {0, 0, 0}, accelBias[3] = {0, 0, 0};// Bias corrections for gyro and accelerometer


int fd = 0;

int main(void)
{

	if( (fd = open(I2C_FILE_NAME, O_RDWR)) <0)
	{
		perror("Open Device Error! ₩n");
		return -1;
	}

	ioctl_mpu9250(fd);

	usleep(1000000);

	uint8_t c = readByte(fd, WHO_AM_I_MPU9250 );
	printf("I AM = %x ₩n₩r", c);

	usleep(1000000);
	if(c == 0x71){

		calibrateMPU9250(gyroBias, accelBias, fd);
		printf("MPU9250 calibration Success!!!!!!₩n₩r");

		initMPU9250(fd);
		printf("MPU9250 Init Success!!!!!!₩n₩r");

		ioctl_ak8963(fd);
		initAK8963(fd, magCalibration);
		printf("MPU9250 AK8963 Init Success!!!!!!₩n₩r");

		ioctl_mpu9250(fd);
		get_offset_value(fd);
		printf("gyro_offset_setting Success!!₩n");

		wait(1000000);
	}
	else
	{
		printf("MPU9250 doesn't work!₩n");
		while(1);
```

```
	}
	for(;;)
	{
		ioctl_mpu9250(fd);
		if(readByte(fd, INT_STATUS) & 0x01)
		{
			readAccelData(fd, accelCount);// Read the x/y/z adc values
			getAres();

			ax = (float) accelCount[0] * aRes; // - accelBias[0];
			// get actual g value, this depends on scale being set
			ay = (float) accelCount[1] * aRes; // - accelBias[1];
			az = (float) accelCount[2] * aRes; // - accelBias[2];

			readGyroData(fd, gyroCount);  // Read the x/y/z adc values
			getGres();

			gx = (float) gyroCount[0] * gRes;
			gy = (float) gyroCount[1] * gRes;
			gz = (float) gyroCount[2] * gRes;

			get_roll_pitch(fd);
			printf("roll = %d ₩t pitch = %d  ₩n",
			    (int)angle_roll_acc, (int)angle_pitch_acc);

			ioctl_ak8963(fd);
			readMagData(fd, magCount );
			getMres();
			magbias[0] = +470.0;
			magbias[1] = +120.0;
			magbias[2] = +125.0;

			mx = (float) magCount[0] * mRes * magCalibration[0] - magbias[0];
			my = (float) magCount[1] * mRes * magCalibration[1] - magbias[1];
			mz = (float) magCount[2] * mRes * magCalibration[2] - magbias[2];

			printf("acc_x = %f₩t acc_y = %f₩t acc_z = %f₩n",
			    1000*ax, 1000*ay, 1000*az);
			usleep(1000000);
			printf("GYRO_x = %d₩t₩t, GYRO_y = %d₩t₩t, GYRO_z = %d₩n",
			      ((int)gx), ((int)gy), ((int)gz));
			usleep(1000000);
			printf("MAG_x = %d₩t₩t, MAG_y = %d₩t₩t, MAG_z = %d₩n",
			      ((int)mx), ((int)my), ((int)mz));
			printf("₩n₩n");
			usleep(1000000);


		}
	}

	return 0;
}
```

- mpu9250.h

```
#ifndef INCLUDE_MPU9250_H
#define INCLUDE_MPU9250_H

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <stdbool.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>

#define I2C_FILE_NAME        "/dev/i2c-0"

//Magnetometer Registers
#define AK8963_ADDRESS   0x0C
#define WHO_AM_I_AK8963  0x00 // should return 0x48
#define INFO             0x01
#define AK8963_ST1       0x02  // data ready status bit 0
#define AK8963_XOUT_L    0x03  // data
#define AK8963_XOUT_H    0x04
#define AK8963_YOUT_L    0x05
#define AK8963_YOUT_H    0x06
#define AK8963_ZOUT_L    0x07
#define AK8963_ZOUT_H    0x08
#define AK8963_ST2       0x09  // Data overflow bit 3 and data read error status bit 2
#define AK8963_CNTL      0x0A  // Power down (0000), single-measurement (0001), self-test (1000) and Fuse
ROM (1111) modes on bits 3:0
#define AK8963_ASTC      0x0C  // Self test control
#define AK8963_I2CDIS    0x0F  // I2C disable
#define AK8963_ASAX      0x10  // Fuse ROM x-axis sensitivity adjustment value
#define AK8963_ASAY      0x11  // Fuse ROM y-axis sensitivity adjustment value
#define AK8963_ASAZ      0x12  // Fuse ROM z-axis sensitivity adjustment value

#define SELF_TEST_X_GYRO 0x00
#define SELF_TEST_Y_GYRO 0x01
#define SELF_TEST_Z_GYRO 0x02



/*
#define X_FINE_GAIN      0x03 // [7:0] fine gain
#define Y_FINE_GAIN      0x04
#define Z_FINE_GAIN      0x05
#define XA_OFFSET_H      0x06 // User-defined trim values for accelerometer
```

```
#define XA_OFFSET_L_TC   0x07
#define YA_OFFSET_H      0x08
#define YA_OFFSET_L_TC   0x09
#define ZA_OFFSET_H      0x0A
#define ZA_OFFSET_L_TC   0x0B
 */

#define SELF_TEST_X_ACCEL 0x0D
#define SELF_TEST_Y_ACCEL 0x0E
#define SELF_TEST_Z_ACCEL 0x0F

#define SELF_TEST_A      0x10

#define XG_OFFSET_H      0x13  // User-defined trim values for gyroscope
#define XG_OFFSET_L      0x14
#define YG_OFFSET_H      0x15
#define YG_OFFSET_L      0x16
#define ZG_OFFSET_H      0x17
#define ZG_OFFSET_L      0x18
#define SMPLRT_DIV       0x19
#define CONFIG           0x1A
#define GYRO_CONFIG      0x1B
#define ACCEL_CONFIG     0x1C
#define ACCEL_CONFIG2    0x1D
#define LP_ACCEL_ODR     0x1E
#define WOM_THR          0x1F

#define MOT_DUR          0x20  // Duration counter threshold for motion interrupt generation, 1 kHz rate, LSB = 1
ms
#define ZMOT_THR         0x21  // Zero-motion detection threshold bits [7:0]
#define ZRMOT_DUR        0x22  // Duration counter threshold for zero motion interrupt generation, 16 Hz rate,
LSB = 64 ms

#define FIFO_EN          0x23
#define I2C_MST_CTRL     0x24
#define I2C_SLV0_ADDR    0x25
#define I2C_SLV0_REG     0x26
#define I2C_SLV0_CTRL    0x27
#define I2C_SLV1_ADDR    0x28
#define I2C_SLV1_REG     0x29
#define I2C_SLV1_CTRL    0x2A
#define I2C_SLV2_ADDR    0x2B
#define I2C_SLV2_REG     0x2C
#define I2C_SLV2_CTRL    0x2D
#define I2C_SLV3_ADDR    0x2E
#define I2C_SLV3_REG     0x2F
#define I2C_SLV3_CTRL    0x30
#define I2C_SLV4_ADDR    0x31
#define I2C_SLV4_REG     0x32
#define I2C_SLV4_DO      0x33
#define I2C_SLV4_CTRL    0x34
#define I2C_SLV4_DI      0x35
#define I2C_MST_STATUS   0x36
```

```
#define INT_PIN_CFG      0x37
#define INT_ENABLE       0x38
#define DMP_INT_STATUS   0x39  // Check DMP interrupt
#define INT_STATUS       0x3A
#define ACCEL_XOUT_H     0x3B
#define ACCEL_XOUT_L     0x3C
#define ACCEL_YOUT_H     0x3D
#define ACCEL_YOUT_L     0x3E
#define ACCEL_ZOUT_H     0x3F
#define ACCEL_ZOUT_L     0x40
#define TEMP_OUT_H       0x41
#define TEMP_OUT_L       0x42
#define GYRO_XOUT_H      0x43
#define GYRO_XOUT_L      0x44
#define GYRO_YOUT_H      0x45
#define GYRO_YOUT_L      0x46
#define GYRO_ZOUT_H      0x47
#define GYRO_ZOUT_L      0x48
#define EXT_SENS_DATA_00 0x49
#define EXT_SENS_DATA_01 0x4A
#define EXT_SENS_DATA_02 0x4B
#define EXT_SENS_DATA_03 0x4C
#define EXT_SENS_DATA_04 0x4D
#define EXT_SENS_DATA_05 0x4E
#define EXT_SENS_DATA_06 0x4F
#define EXT_SENS_DATA_07 0x50
#define EXT_SENS_DATA_08 0x51
#define EXT_SENS_DATA_09 0x52
#define EXT_SENS_DATA_10 0x53
#define EXT_SENS_DATA_11 0x54
#define EXT_SENS_DATA_12 0x55
#define EXT_SENS_DATA_13 0x56
#define EXT_SENS_DATA_14 0x57
#define EXT_SENS_DATA_15 0x58
#define EXT_SENS_DATA_16 0x59
#define EXT_SENS_DATA_17 0x5A
#define EXT_SENS_DATA_18 0x5B
#define EXT_SENS_DATA_19 0x5C
#define EXT_SENS_DATA_20 0x5D
#define EXT_SENS_DATA_21 0x5E
#define EXT_SENS_DATA_22 0x5F
#define EXT_SENS_DATA_23 0x60
#define MOT_DETECT_STATUS 0x61
#define I2C_SLV0_DO      0x63
#define I2C_SLV1_DO      0x64
#define I2C_SLV2_DO      0x65
#define I2C_SLV3_DO      0x6g6
#define I2C_MST_DELAY_CTRL 0x67
#define SIGNAL_PATH_RESET  0x68
#define MOT_DETECT_CTRL  0x69
#define USER_CTRL        0x6A  // Bit 7 enable DMP, bit 3 reset DMP
#define PWR_MGMT_1       0x6B // Device defaults to the SLEEP mode
#define PWR_MGMT_2       0x6C
```

```
#define DMP_BANK         0x6D  // Activates a specific bank in the DMP
#define DMP_RW_PNT       0x6E  // Set read/write pointer to a specific start address in specified DMP bank
#define DMP_REG          0x6F  // Register in DMP from which to read or to which to write
#define DMP_REG_1        0x70
#define DMP_REG_2        0x71
#define FIFO_COUNTH      0x72
#define FIFO_COUNTL      0x73
#define FIFO_R_W         0x74
#define WHO_AM_I_MPU9250 0x75 // Should return 0x71
#define XA_OFFSET_H      0x77
#define XA_OFFSET_L      0x78
#define YA_OFFSET_H      0x7A
#define YA_OFFSET_L      0x7B
#define ZA_OFFSET_H      0x7D
#define ZA_OFFSET_L      0x7E

#define ADO 0
#if ADO
#define MPU9250_ADDRESS 0x69  // Device address when ADO = 1
#else
#define MPU9250_ADDRESS 0x68  // Device address when ADO = 0
#define AK8963_ADDRESS 0x0C   //  Address of magnetometer
#endif

#define AHRS true         // set to false for basic data read
#define SerialDebug true  // set to true to get Serial output for debugging

enum Ascale {
                AFS_2G = 0,
                AFS_4G,
                AFS_8G,
                AFS_16G
};

enum Gscale {
                GFS_250DPS = 0,
                GFS_500DPS,
                GFS_1000DPS,
                GFS_2000DPS
};

enum Mscale {
                MFS_14BITS = 0, // 0.6 mG per LSB
                MFS_16BITS      // 0.15 mG per LSB
};

int8_t Gscale = GFS_500DPS;
int8_t Ascale = AFS_2G;
int8_t Mscale = MFS_16BITS; // Choose either 14-bit or 16-bit magnetometer resolution
int8_t Mmode = 0x02;        // 2 for 8 Hz, 6 for 100 Hz continuous magnetometer data read

float aRes, gRes, mRes;      // scale resolutions per LSB for the sensors
```

```
//Declaring some global variables for roll, pitch
int16_t g_xyz[3];
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
bool set_gyro_angles;

int16_t a_xyz[3];
long acc_total_vector;
float angle_roll_acc, angle_pitch_acc;

float angle_pitch, angle_roll;
float angle_pitch_output, angle_roll_output;


//fucntion
void writeByte(int fd, uint8_t regAddr, uint8_t data);
void readBytes(int fd, uint8_t regAddr, int length, uint8_t *data);
uint8_t readByte(int fd, uint8_t regAddr);

void wait(int delay);

void initAK8963(int fd, float * destination);
void initMPU9250(int fd);
void calibrateMPU9250(float *dest1, float *dest2, int fd);

void getAres(void);
void getGres(void);
void getMres(void);

void readAccelData(int fd, int16_t *destination);
void readGyroData(int fd, int16_t * destination);
void readMagData(int fd, int16_t *destination);

void get_roll_pitch(int fd);
void get_offset_value(int fd);

int ioctl_mpu9250(int fd);
int ioctl_ak8963(int fd);



void calibrateMPU9250(float *dest1, float *dest2, int fd)
{
	int a;
	printf("start calibrate₩n");
	uint8_t data[12];
	int16_t ii, packet_count, fifo_count;
	int32_t gyro_bias[3] = {0,0,0};
	int32_t accel_bias[3] = {0,0,0};

	writeByte(fd, PWR_MGMT_1, 0x80);
	usleep(100000);
```

```
	writeByte(fd, PWR_MGMT_1, 0x01);
	writeByte(fd, PWR_MGMT_2, 0x00);
	usleep(100000);

	// Configure device for bias calculation
	writeByte(fd, INT_ENABLE, 0x00);   // Disable all interrupts
	writeByte(fd, FIFO_EN, 0x00);      // Disable FIFO
	writeByte(fd, PWR_MGMT_1, 0x00);   // Turn on internal clock source
	writeByte(fd, I2C_MST_CTRL, 0x00); // Disable I2C master
	writeByte(fd, USER_CTRL, 0x00);    // Disable FIFO and I2C master modes
	writeByte(fd, USER_CTRL, 0x0C);    // Reset FIFO and DMP
	usleep(100000);

	// Configure MPU6050 gyro and accelerometer for bias calculation
	writeByte(fd, CONFIG, 0x01);      // Set low-pass filter to 188 Hz
	writeByte(fd, SMPLRT_DIV, 0x00);  // Set sample rate to 1 kHz
	writeByte(fd, GYRO_CONFIG, 0x00);  // Set gyro full-scale to 250 degrees per second, maximum
sensitivity
	writeByte(fd, ACCEL_CONFIG, 0x00); // Set accelerometer full-scale to 2 g, maximum sensitivity

	uint16_t  gyrosensitivity  = 131;   // = 131 LSB/degrees/sec
	uint16_t  accelsensitivity = 16384;  // = 16384 LSB/g

	// Configure FIFO to capture accelerometer and gyro data for bias calculation
	writeByte(fd, USER_CTRL, 0x40);   // Enable FIFO
	writeByte(fd, FIFO_EN, 0x78);     // Enable gyro and accelerometer sensors for FIFO  (max size 512 bytes in
MPU-9150)
	usleep(40000); // accumulate 40 samples in 40 milliseconds = 480 bytes

	// At end of sample accumulation, turn off FIFO sensor read
	writeByte(fd, FIFO_EN, 0x00);        // Disable gyro and accelerometer sensors for FIFO


	readBytes(fd, FIFO_COUNTH, 2, &data[0]); // read FIFO sample count
	fifo_count = ((uint16_t)data[0] << 8) | data[1];
	packet_count = fifo_count/12;// How many sets of full gyro and accelerometer data for averaging
	printf("fifo count : 0x%x, packet_count : 0x%x₩n", fifo_count, packet_count);

	for(a=0; a<12; a++)
	{
		printf("%x",data[a]);
	}


	for(ii = 0; ii < packet_count; ii++)
	{

		int16_t accel_temp[3] = {0, 0, 0}, gyro_temp[3] = {0, 0, 0};
		readBytes(fd, FIFO_R_W, 12, &data[0]); // read data for averaging

		accel_temp[0] = (int16_t) (((int16_t)data[0] << 8)  | data[1] ) ;  // Form signed 16-bit integer for
each sample in FIFO
		accel_temp[1] = (int16_t) (((int16_t)data[2] << 8)  | data[3] ) ;
```

```
		accel_temp[2] = (int16_t) (((int16_t)data[4] << 8)  | data[5] ) ;
		gyro_temp[0]  = (int16_t) (((int16_t)data[6] << 8)  | data[7] ) ;
		gyro_temp[1]  = (int16_t) (((int16_t)data[8] << 8)  | data[9] ) ;
		gyro_temp[2]  = (int16_t) (((int16_t)data[10] << 8) | data[11]) ;
		accel_bias[0] += (int32_t) accel_temp[0]; // Sum individual signed 16-bit biases to get
accumulated signed 32-bit biases
		accel_bias[1] += (int32_t) accel_temp[1];
		accel_bias[2] += (int32_t) accel_temp[2];
		gyro_bias[0]  += (int32_t) gyro_temp[0];
		gyro_bias[1]  += (int32_t) gyro_temp[1];
		gyro_bias[2]  += (int32_t) gyro_temp[2];

	}


	accel_bias[0] /= (int32_t) packet_count; // Normalize sums to get average count biases
	accel_bias[1] /= (int32_t) packet_count;
	accel_bias[2] /= (int32_t) packet_count;
	gyro_bias[0]  /= (int32_t) packet_count;
	gyro_bias[1]  /= (int32_t) packet_count;
	gyro_bias[2]  /= (int32_t) packet_count;

	if(accel_bias[2] > 0L) {
		accel_bias[2] -= (int32_t) accelsensitivity;
	}  // Remove gravity from the z-axis accelerometer bias calculation
	else {
		accel_bias[2] += (int32_t) accelsensitivity;
	}

	// Construct the gyro biases for push to the hardware gyro bias registers, which are reset to zero upon
device startup
	data[0] = ((-gyro_bias[0]/4)  >> 8) & 0xFF;
	// Divide by 4 to get 32.9 LSB per deg/s to conform to expected bias input format
	data[1] = (-gyro_bias[0]/4)       & 0xFF;
	// Biases are additive, so change sign on calculated average gyro biases
	data[2] = ((-gyro_bias[1]/4)  >> 8) & 0xFF;
	data[3] = ((-gyro_bias[1]/4))       & 0xFF;
	data[4] = ((-gyro_bias[2]/4)  >> 8) & 0xFF;
	data[5] = (-gyro_bias[2]/4)       & 0xFF;

	// Push gyro biases to hardware registers
	writeByte(fd, XG_OFFSET_H, data[0]);
	writeByte(fd, XG_OFFSET_L, data[1]);
	writeByte(fd, YG_OFFSET_H, data[2]);
	writeByte(fd, YG_OFFSET_L, data[3]);
	writeByte(fd, ZG_OFFSET_H, data[4]);
	writeByte(fd, ZG_OFFSET_L, data[5]);

	dest1[0] = (float) gyro_bias[0]/(float) gyrosensitivity;
	dest1[1] = (float) gyro_bias[1]/(float) gyrosensitivity;
	dest1[2] = (float) gyro_bias[2]/(float) gyrosensitivity;

	int32_t accel_bias_reg[3] = {0, 0, 0}; // A place to hold the factory accelerometer trim biases
```

```
	readBytes(fd, XA_OFFSET_H, 2, &data[0]); // Read factory accelerometer trim values
	accel_bias_reg[0] = (int32_t) (((int16_t)data[0] << 8) | data[1]);
	readBytes(fd, YA_OFFSET_H, 2, &data[0]);
	accel_bias_reg[1] = (int32_t) (((int16_t)data[0] << 8) | data[1]);
	readBytes(fd, ZA_OFFSET_H, 2, &data[0]);
	accel_bias_reg[2] = (int32_t) (((int16_t)data[0] << 8) | data[1]);

	uint32_t mask = 1uL;
	// Define mask for temperature compensation bit 0 of lower byte of accelerometer bias registers
	uint8_t mask_bit[3] = {0, 0, 0}; // Define array to hold mask bit for each accelerometer bias axis

	for(ii = 0; ii < 3; ii++) {
		if((accel_bias_reg[ii] & mask)) mask_bit[ii] = 0x01;
		// If temperature compensation bit is set, record that fact in mask_bit
		}

	// Construct total accelerometer bias, including calculated average accelerometer bias from above
	accel_bias_reg[0] -= (accel_bias[0]/8);
	// Subtract calculated averaged accelerometer bias scaled to 2048 LSB/g (16 g full scale)
	accel_bias_reg[1] -= (accel_bias[1]/8);
	accel_bias_reg[2] -= (accel_bias[2]/8);

	data[0] = (accel_bias_reg[0] >> 8) & 0xFF;
	data[1] = (accel_bias_reg[0])      & 0xFF;
	data[1] = data[1] | mask_bit[0];
	// preserve temperature compensation bit when writing back to accelerometer bias registers
	data[2] = (accel_bias_reg[1] >> 8) & 0xFF;
	data[3] = (accel_bias_reg[1])      & 0xFF;
	data[3] = data[3] | mask_bit[1];
	// preserve temperature compensation bit when writing back to accelerometer bias registers
	data[4] = (accel_bias_reg[2] >> 8) & 0xFF;
	data[5] = (accel_bias_reg[2])      & 0xFF;
	data[5] = data[5] | mask_bit[2];
	// preserve temperature compensation bit when writing back to accelerometer bias registers

	writeByte(fd, XA_OFFSET_H, data[0]);
	writeByte(fd, XA_OFFSET_L, data[1]);
	writeByte(fd, YA_OFFSET_H, data[2]);
	writeByte(fd, YA_OFFSET_L, data[3]);
	writeByte(fd, ZA_OFFSET_H, data[4]);
	writeByte(fd, ZA_OFFSET_L, data[5]);

	// Output scaled accelerometer biases for display in the main program
	dest2[0] = (float)accel_bias[0]/(float)accelsensitivity;
	dest2[1] = (float)accel_bias[1]/(float)accelsensitivity;
	dest2[2] = (float)accel_bias[2]/(float)accelsensitivity;

}

void initMPU9250(int fd)
{

	writeByte(fd, PWR_MGMT_1, 0x00);
```

```
	usleep(1000);

	writeByte(fd, PWR_MGMT_1, 0x01);
	usleep(1000);

	writeByte(fd, CONFIG, 0x03);

	writeByte(fd, SMPLRT_DIV, 0x04);

	uint8_t c = readByte(fd, GYRO_CONFIG);

	writeByte(fd, GYRO_CONFIG, (c & ~(0x02 | 0x18)) | Gscale << 3);

	c = readByte(fd, ACCEL_CONFIG);

	writeByte(fd, ACCEL_CONFIG, (c & ~0x18) | Ascale << 3);

	c = readByte(fd, ACCEL_CONFIG2);

	writeByte(fd, ACCEL_CONFIG2, (c & ~0x0F));

	writeByte(fd, INT_PIN_CFG, 0x22);
	writeByte(fd, INT_ENABLE, 0x01);
	usleep(1000);
}

void initAK8963(int fd, float * destination)
{
	uint8_t rawData[3];

	writeByte(fd, AK8963_CNTL, 0x00);
	usleep(1000);
	writeByte(fd, AK8963_CNTL, 0x0F);
	usleep(1000);
	readBytes(fd, AK8963_ASAX, 3, &rawData[0]);
	destination[0] =  (((float)(rawData[0] - 128))/256.0) + 1.0;
	// Return x-axis sensitivity adjustment values, etc.
	destination[1] =  (((float)(rawData[1] - 128))/256.0) + 1.0;
	destination[2] =  (((float)(rawData[2] - 128))/256.0) + 1.0;
	writeByte(fd, AK8963_CNTL, 0x00); // Power down magnetometer
	usleep(1000);
	writeByte(fd, AK8963_CNTL, Mscale << 4 | Mmode);
	// Set magnetometer data resolution and sample ODR
	usleep(1000);

}

void get_offset_value(int fd)
{
	int cnt;
	for (cnt = 0; cnt < 1000; cnt++)
		{
		readGyroData(fd, g_xyz);
```

```
            gyro_x_cal += g_xyz[0];
            gyro_y_cal += g_xyz[1];
            gyro_z_cal += g_xyz[2];
            usleep(100);
        }

        gyro_x_cal /= 1000; //Divide the gyro_x_cal variable by 2000 to get the avarage offset
        gyro_y_cal /= 1000; //Divide the gyro_y_cal variable by 2000 to get the avarage offset
        gyro_z_cal /= 1000;


}

void get_roll_pitch(int fd)
{
        readAccelData(fd, a_xyz);
        readGyroData(fd, g_xyz);

        g_xyz[0] -= gyro_x_cal;
        g_xyz[1] -= gyro_y_cal;
        g_xyz[2] -= gyro_z_cal;

        angle_pitch += angle_roll * sin( ((float)g_xyz[2]) * 0.000001066);
        //If the IMU has yawed transfer the roll angle to the pitch angel
        angle_roll -= angle_pitch * sin( ((float)g_xyz[2]) * 0.000001066);
        //If the IMU has yawed transfer the pitch angle to the roll angel

        acc_total_vector = sqrt( pow(a_xyz[0],2)+pow(a_xyz[1],2)+pow(a_xyz[2],2) );
        //Calculate the total accelerometer vector
          //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
        angle_pitch_acc = asin( (float)a_xyz[1]/acc_total_vector) * 57.296;//Calculate the pitch angle
        angle_roll_acc = asin( (float)a_xyz[0]/acc_total_vector) * -57.296;//Calculate the roll angle

}

void readGyroData(int fd, int16_t * destination)
{
        uint8_t rawData[6];
        readBytes(fd, GYRO_XOUT_H, 6, &rawData[0]);
        destination[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;
        // Turn the MSB and LSB into a signed 16-bit value
        destination[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
        destination[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;
}

void readAccelData(int fd, int16_t *destination)
{
        uint8_t rawData[6];
        readBytes(fd, ACCEL_XOUT_H, 6, &rawData[0]);
        destination[0] = ((int16_t)rawData[0] << 8) | rawData[1];
        // Turn the MSB and LSB into a signed 16-bit value
        destination[1] = ((int16_t)rawData[2] << 8) | rawData[3];
        destination[2] = ((int16_t)rawData[4] << 8) | rawData[5];
```

```
}

void readMagData(int fd, int16_t *destination)
{
	uint8_t rawData[7];
	if(readByte(fd, AK8963_ST1) & 0x01)
	{
		readBytes(fd, AK8963_XOUT_L, 7, &rawData[0]);
		uint8_t c = rawData[6];
		if(!(c & 0x08))
		{
			destination[0] = ((int16_t)rawData[1] << 8) | rawData[0] ;
			// Turn the MSB and LSB into a signed 16-bit value
			destination[1] = ((int16_t)rawData[3] << 8) | rawData[2] ;
			// Data stored as little Endian
			destination[2] = ((int16_t)rawData[5] << 8) | rawData[4] ;
		}
	}
}

void getMres(void)
{
	switch (Mscale)
	{
		case MFS_14BITS:
			mRes = 10.0 * 4219.0/ 8190.0; // Proper scale to return milliGauss
							break;
		case MFS_16BITS:
			mRes = 10.0 * 4219.0/ 32760.0; // Proper scale to return milliGauss
			break;
	}
}

void getAres(void)
{
	switch(Ascale)
	{
		case AFS_2G:
			aRes = 2.0/32768.0;
			break;
		case AFS_4G:
			aRes = 4.0/32768.0;
			break;
		case AFS_8G:
			aRes = 8.0/32768.0;
			break;
		case AFS_16G:
			aRes = 16.0/32768.0;
			break;
	}
}

void getGres(void)
```

```
{
	switch (Gscale)
	{
		case GFS_250DPS:
			gRes = 250.0/32768.0;
			break;
		case GFS_500DPS:
			gRes = 500.0/32768.0;
			break;
		case GFS_1000DPS:
			gRes = 1000.0/32768.0;
			break;
		case GFS_2000DPS:
			gRes = 2000.0/32768.0;
			break;
	}
}

void wait(int delay)
{
	int i;
	for(i=0; i<delay; i++)
		;
}

void writeByte(int fd, uint8_t regAddr, uint8_t data)
{
	int8_t buf[2] = {regAddr,data};

	if(write(fd, buf ,sizeof(buf)) != sizeof(buf))
	{
		printf("write register error - writeByte₩n");
	}
}

void readBytes(int fd, uint8_t regAddr, int length, uint8_t *data)
{
	uint8_t buf[1] = {regAddr};
	if(write(fd, buf, 1) != 1)
	{
		perror("read register error - readBytes₩n");
	}
	if(read(fd, data, length) != length)
	{
		printf("recieve data error - readBytes₩n");
	}
}

uint8_t readByte(int fd, uint8_t regAddr)
{
	uint8_t buf[1] = {regAddr};
	uint8_t data[1] = {0};
	if(write(fd,buf,1) != 1)
```

```
	{
		perror("read register error -w ₩n");
		return -1;
	}
	if(read(fd, data, 1) != 1)
	{
		perror("read register error -r ₩n");
		return -1;
	}

	return data[0];

}

int ioctl_mpu9250(int fd)
{
	if(ioctl(fd,I2C_SLAVE_FORCE, MPU9250_ADDRESS) < 0)
	{
		perror("slave address connect error - ioctl_mpu9250₩n");
	}
}

int ioctl_ak8963(int fd)
{
	if(ioctl(fd, I2C_SLAVE_FORCE, AK8963_ADDRESS) < 0)
	{
		perror("slave address connect error - ioctl_ak8963₩n");
	}
}

#endif
```

11. Makefile 수정

```
Terminal File Edit View Search Terminal Help
 1 ifndef PETALINUX
 2 $(error "Error: PETALINUX environment variable not set.  Change to the root of your PetaLinux inst
   all, and source the settings.sh file")
 3 endif
 4
 5 include apps.common.mk
 6
 7 APP = device_driver
 8 LDLIBS += -lm
 9
10 # Add any other object files to this list below
11 APP_OBJS = device_driver.o
12
13 all: build install
14
15 build: $(APP)
16
17 $(APP): $(APP_OBJS)
18     $(CC) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)
19
20 clean:
21     -rm -f $(APP) *.elf *.gdb *.o
22
23 .PHONY: install image
24
25 install: $(APP)
26     $(TARGETINST) -d $(APP) /bin/$(APP)
27
28 %.o: %.c
29     $(CC) -c $(CFLAGS) -o $@ $<
30
31 help:
32     @echo ""
33     @echo "Quick reference for various supported build targets for $(INSTANCE)."
34     @echo "----------------------------------------------------"
35     @echo "  clean                  clean out build objects"
36     @echo "  all                    build $(INSTANCE) and install to rootfs host copy"
37     @echo "  build                  build subsystem"
38     @echo "  install                install built objects to rootfs host copy"
~
~
```

8번째 라인의 '**LDIBS += -lm**'을 추가하여 -lm 옵션을 사용할 수 있도록 한다.

gcc mpu9250.h
gcc device_driver -lm 을 하여 문제없이 디버깅이 되어야한다.

12. cd ../../../
→ 페타리눅스 폴더로 이동

13. **petalinux-build**

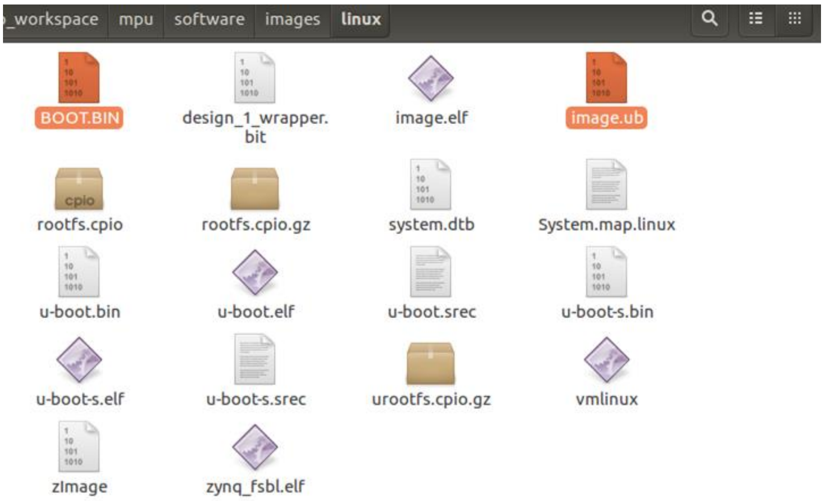
14. **cd image/linux**

15. **petalinux-build**
→ 빌드를 한번 더하 는 이유는 여러 번 빌드했을 때 수정한 디바이스 드라이버 코드가 제대로 적용되지 않을때가 있어 한번 더 해준다.

16. **source /opt/Xilinx/Vivado/2017.1/settings64.sh**

17. **petalinux-package --boot --fsbl zynq_fsbl.elf --fpga ../../../hardware/mpu9250.runs/impl_1/design_1_wrapper.bit --u-boot --force**
→ 팩킹하여 BOOT.BIN 파일 생성

18. **nautilus ./**
→ 해당 터미널의 폴더 열기

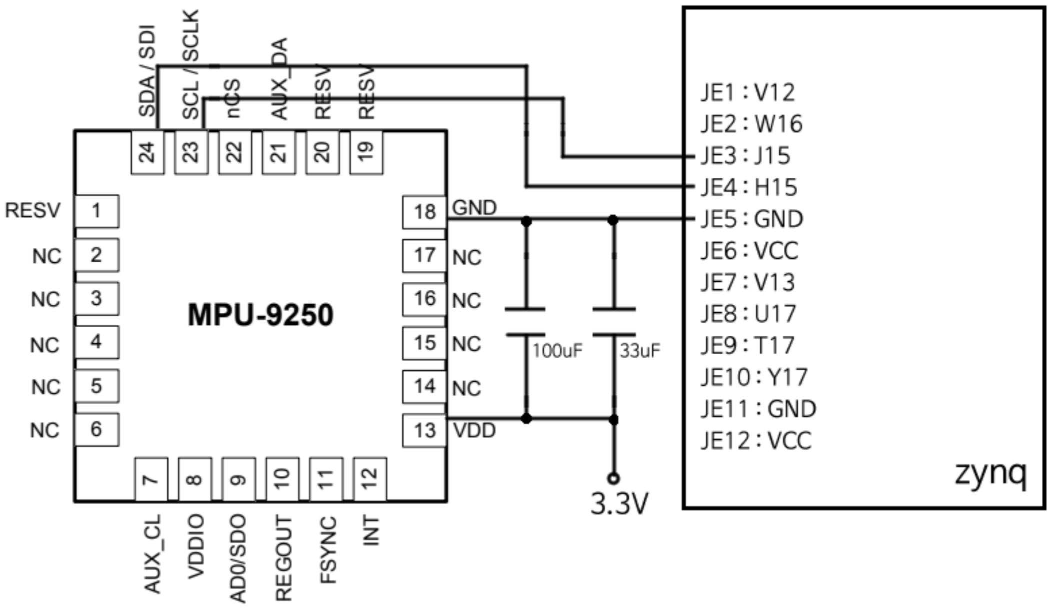


19. 미리 BOOT 와 rootfs 으로 파티션을 분할해 놓았던 sd 카드의 BOOT 파티션에 위의 두 파일(BOOT.BIN 과 image.ub)을 넣는다.

[실행시키기]

1. BOOT.BIN 과 image.ub 파일을 담은 sd 카드를 zybo 보드에 삽입한다.

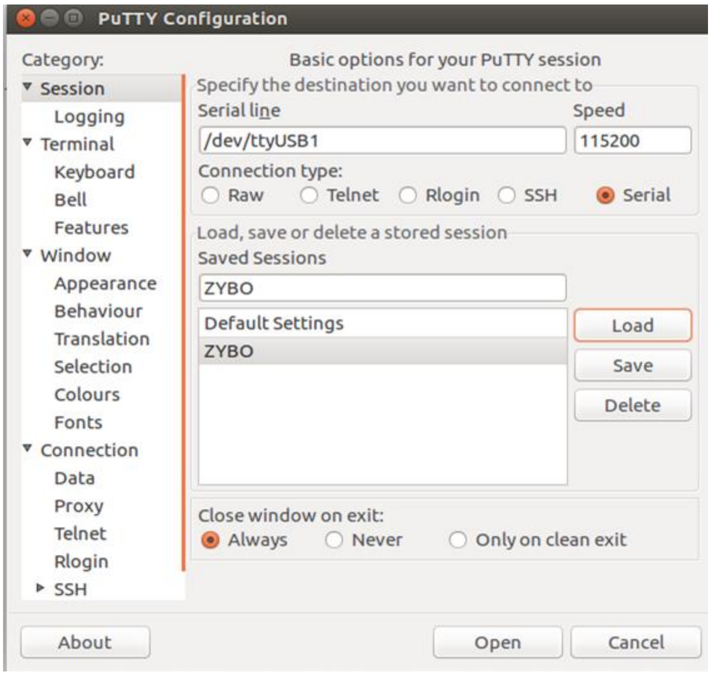
2. 회로를 아래와 같이 결선한다.



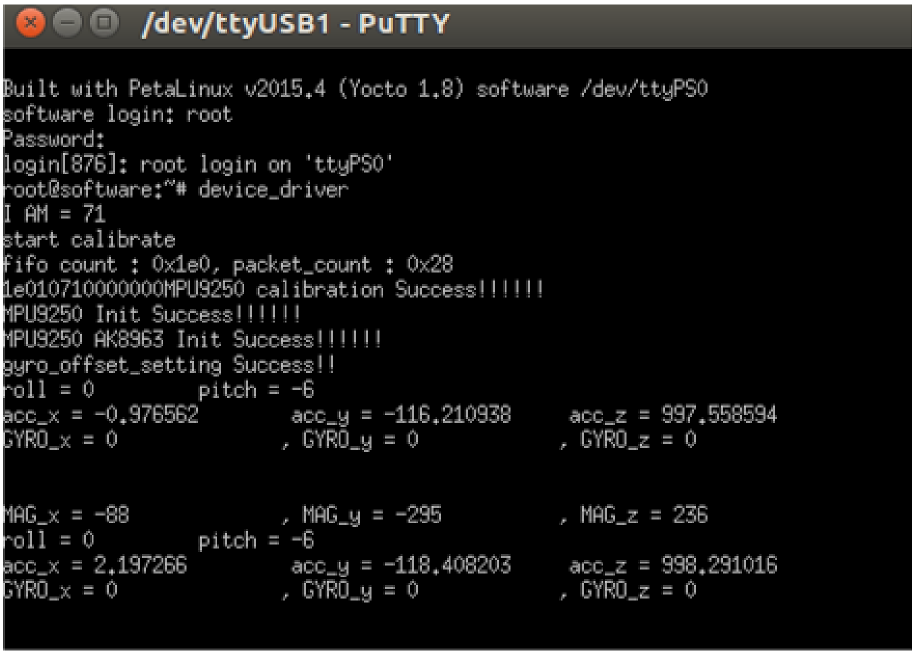
3. zybo 보드를 연결하여 전원을 킨 뒤 권한을 준다.

```
sudo chmod 666 /dev/ttyUSB1
```

4. putty 실행



아이디와 비밀번호를 **root** 로 입력하여 로그인 한 뒤 ‘device_driver’를 입력하여 실행시킨다.



mpu9250을 꽂은 빵판을 90도로 회전시켜서 **roll 이나 pitch 값이 90 또는 -90**이 나오는 것을 확인하면 된다. Acc 값들은 중력가속도 때문에 mpu9250이 움직이지 않아도 값을 가진다. MAG 값은 mpu9250에 x,y,z 축이 표시되어 있어 자기장의 방향에 따라 값이 달라지게 된다.