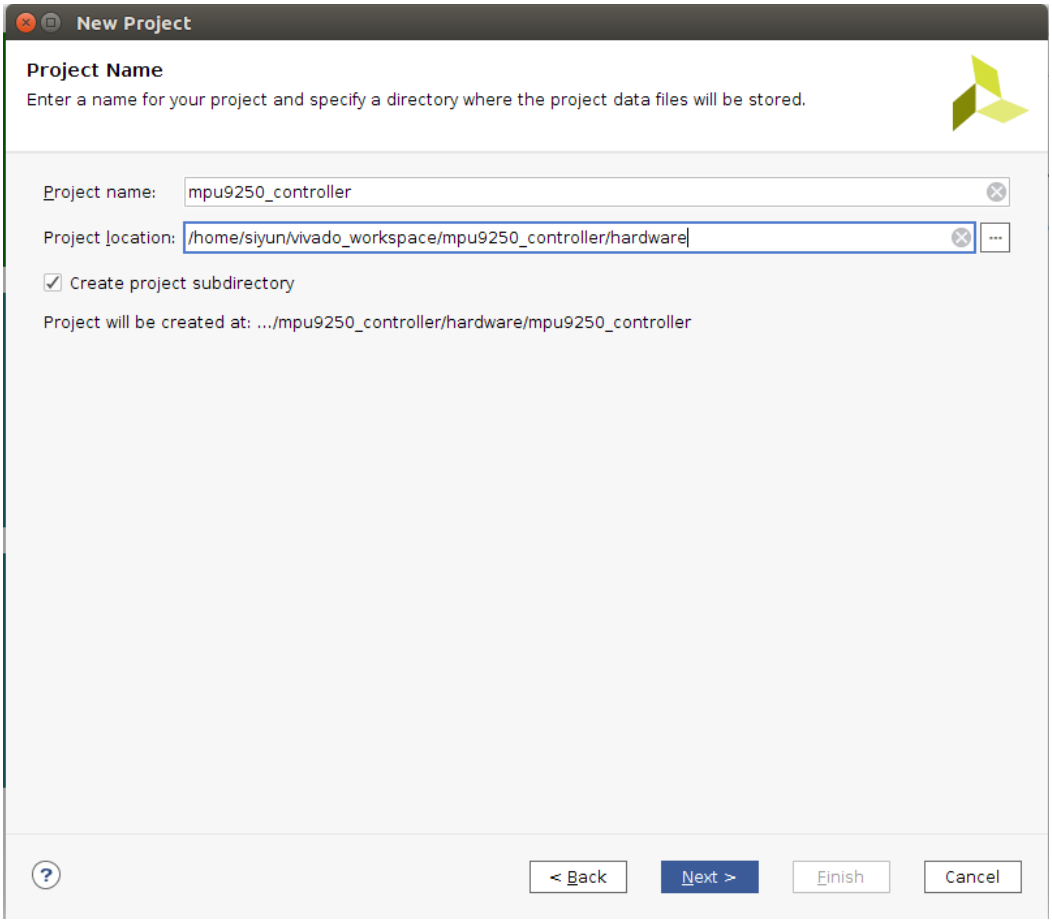
Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

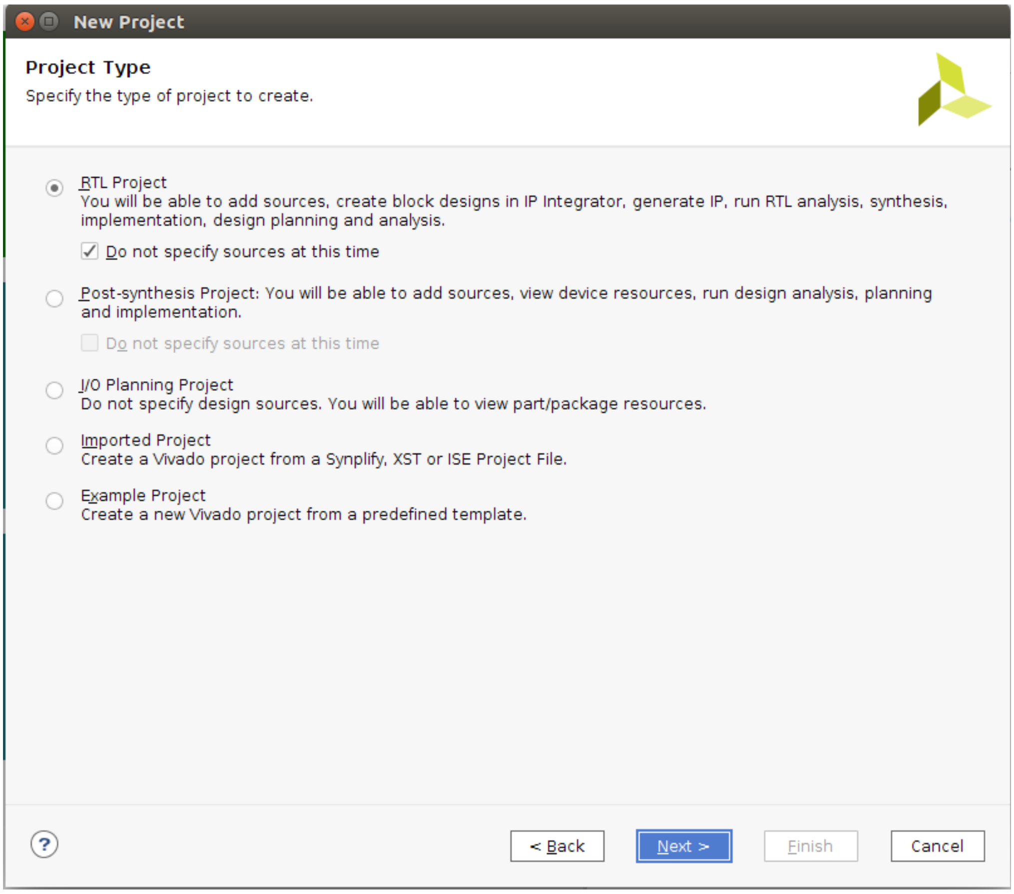
Use I2C device driver
MPU9250 Controller

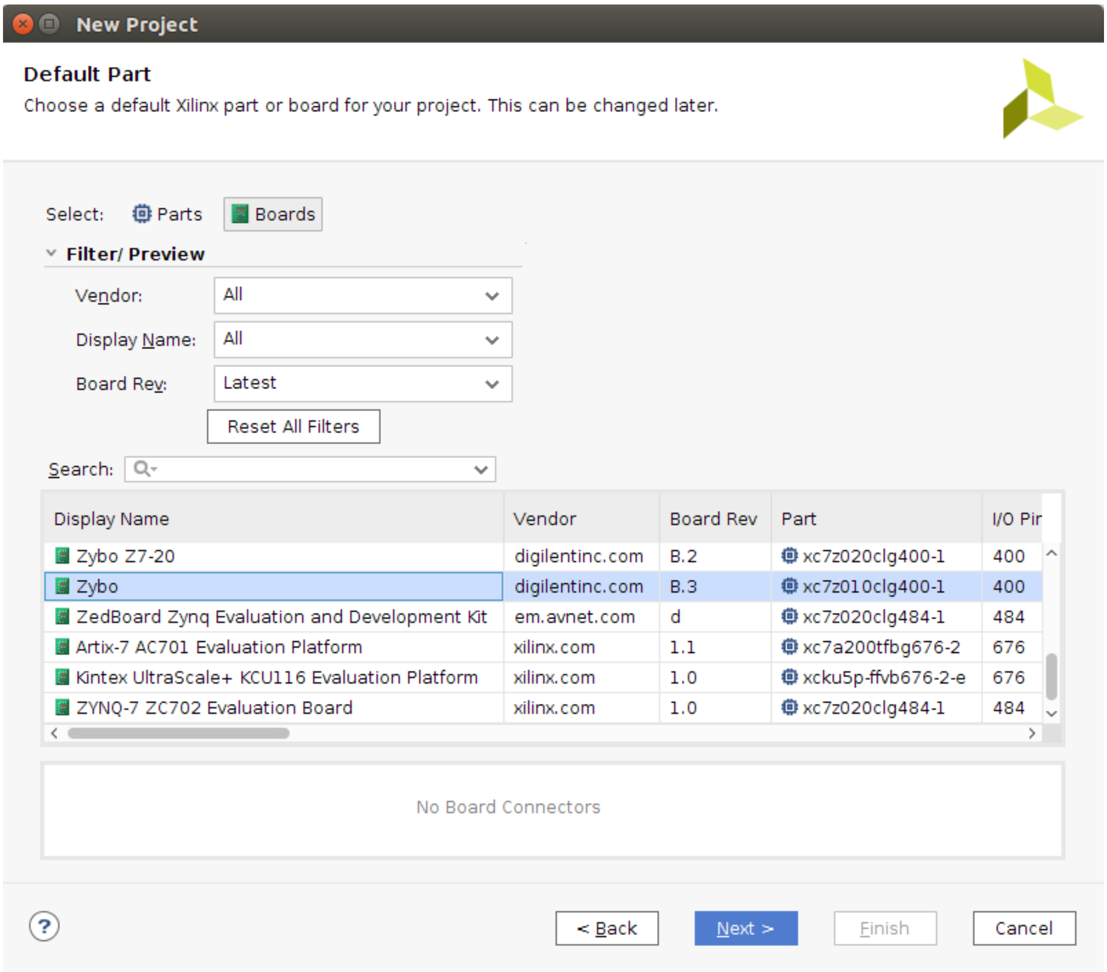
강사: 이상훈

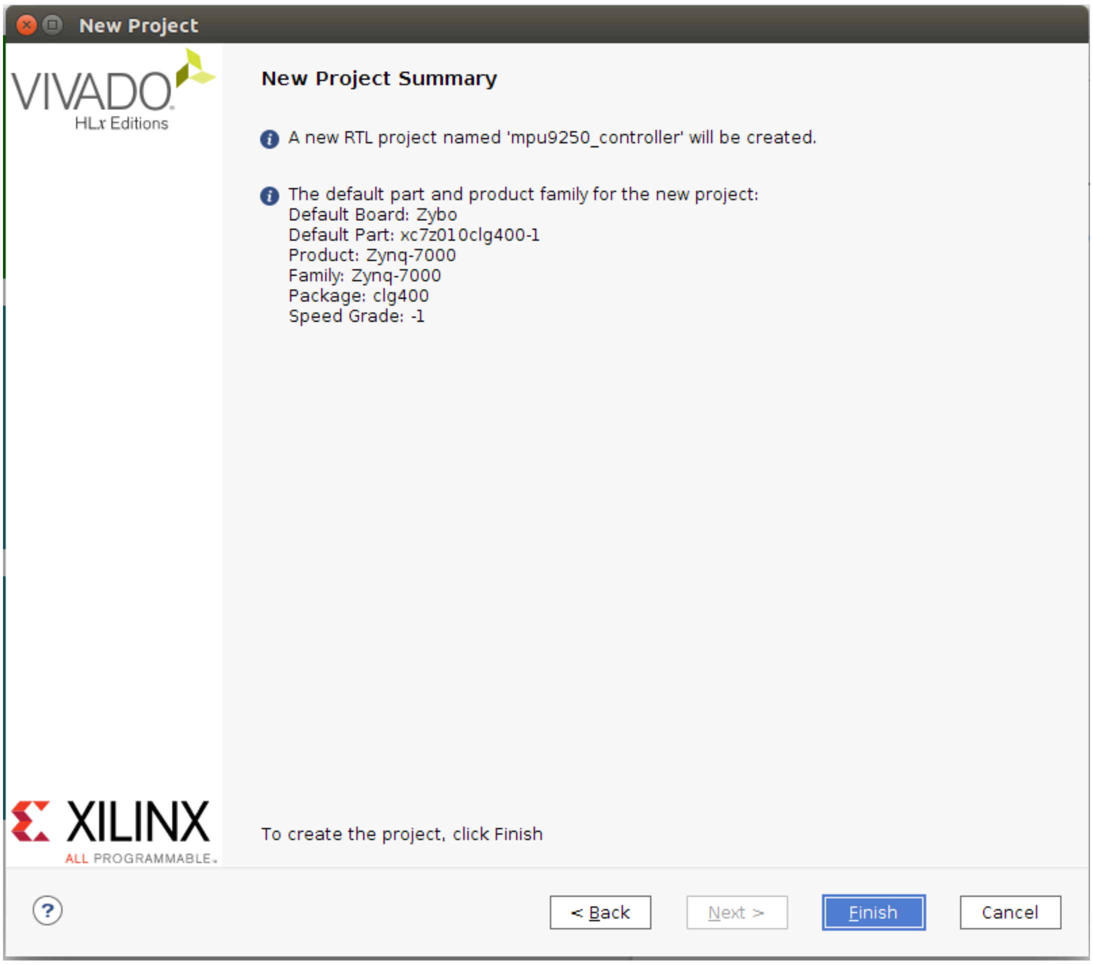
학생: 김시

1. Vivado 를 킨 후 프로젝트를 생성한다.



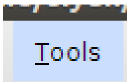


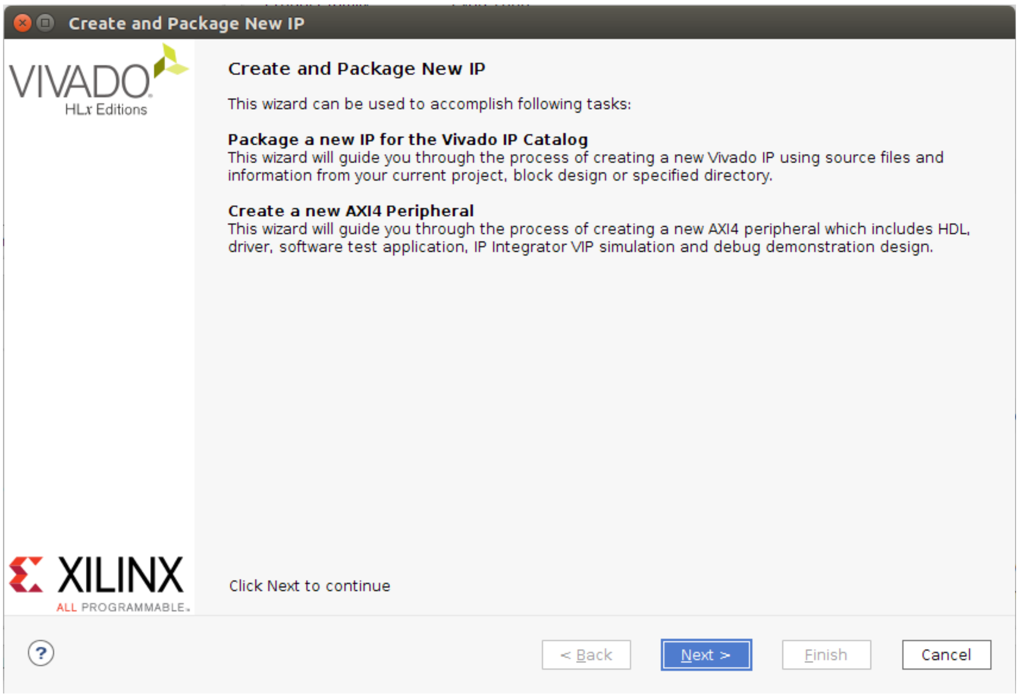

New Project
Default Part
Choose a default Xilinx part or board for your project. This can be changed later.
Select: Parts Boards
Filter/ Preview
Vendor: All
Display Name: All
Board Rev: Latest
Reset All Filters
Search:
Display Name | Vendor | Board Rev | Part | I/O Pir
Zybo Z7-20 | digilentinc.com | B.2 | xc7z020clg400-1 | 400
Zybo | digilentinc.com | B.3 | xc7z010clg400-1 | 400
ZedBoard Zynq Evaluation and Development Kit | em.avnet.com | d | xc7z020clg484-1 | 484
Artix-7 AC701 Evaluation Platform | xilinx.com | 1.1 | xc7a200tfbg676-2 | 676
Kintex UltraScale+ KCU116 Evaluation Platform | xilinx.com | 1.0 | xcku5p-ffvb676-2-e | 676
ZYNQ-7 ZC702 Evaluation Board | xilinx.com | 1.0 | xc7z020clg484-1 | 484
No Board Connectors
< Back
Next >
Finish
Cancel

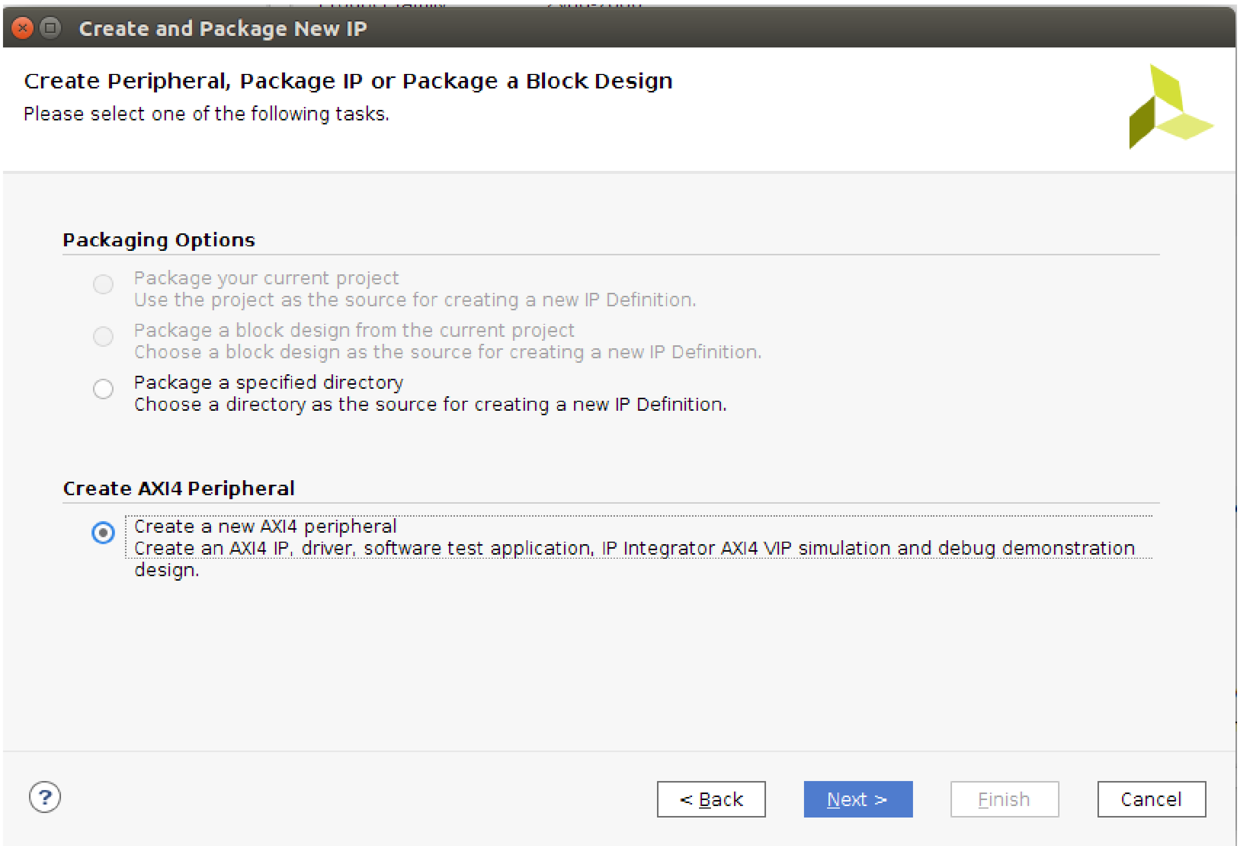

New Project
VIVADO
HLx Editions
New Project Summary
A new RTL project named 'mpu9250_controller' will be created.
The default part and product family for the new project:
Default Board: Zybo
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1
XILINX
ALL PROGRAMMABLE.
To create the project, click Finish
< Back
Next >
Finish
Cancel

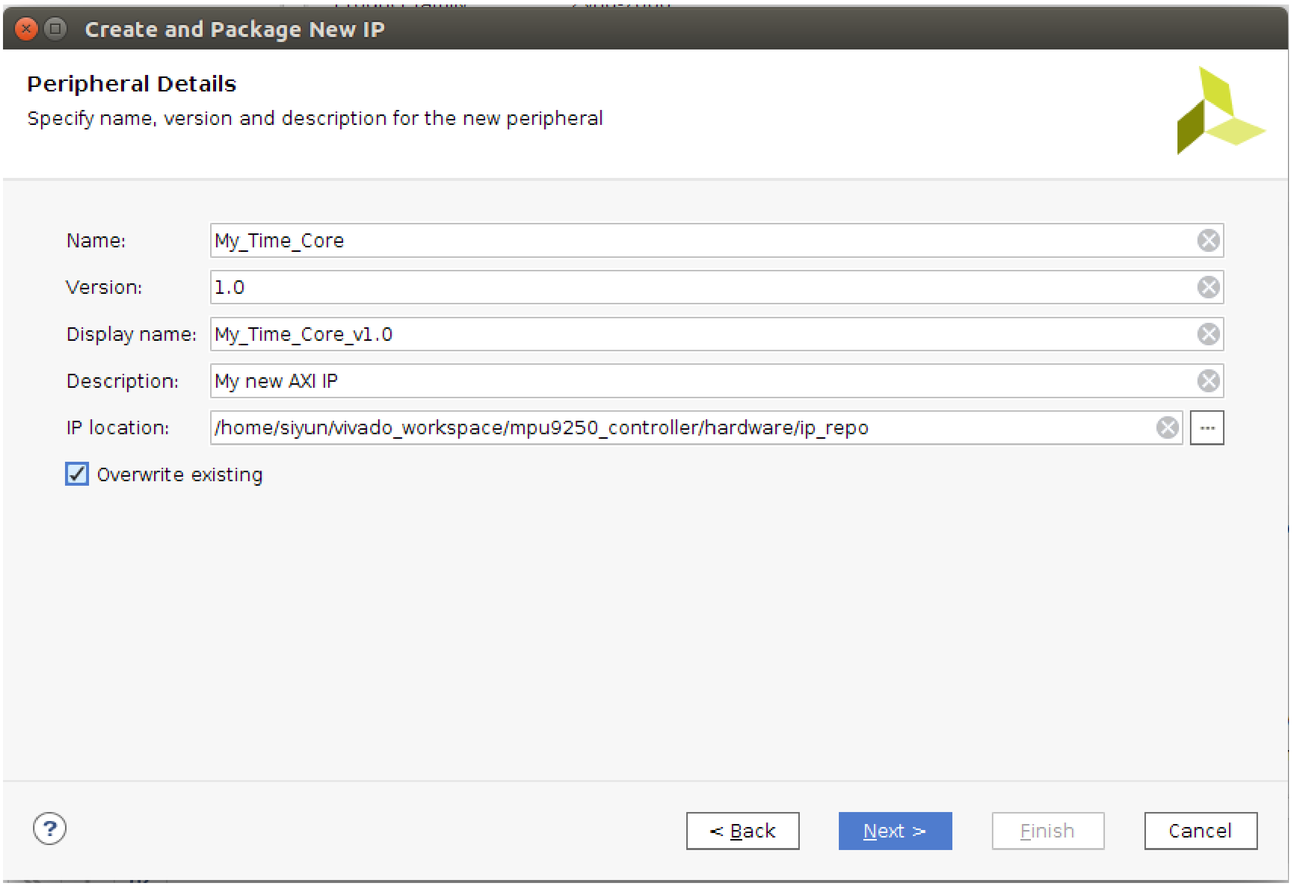
MPU9250 에서 시간을 얻어 계산하는 필터가 있는데, 리눅스 프로그래밍으로 시간을 얻으면 약간의 오차가 있기 때문에 정확한 시간을 얻기 위함 Custom Timer IP 를 만들 것이다.

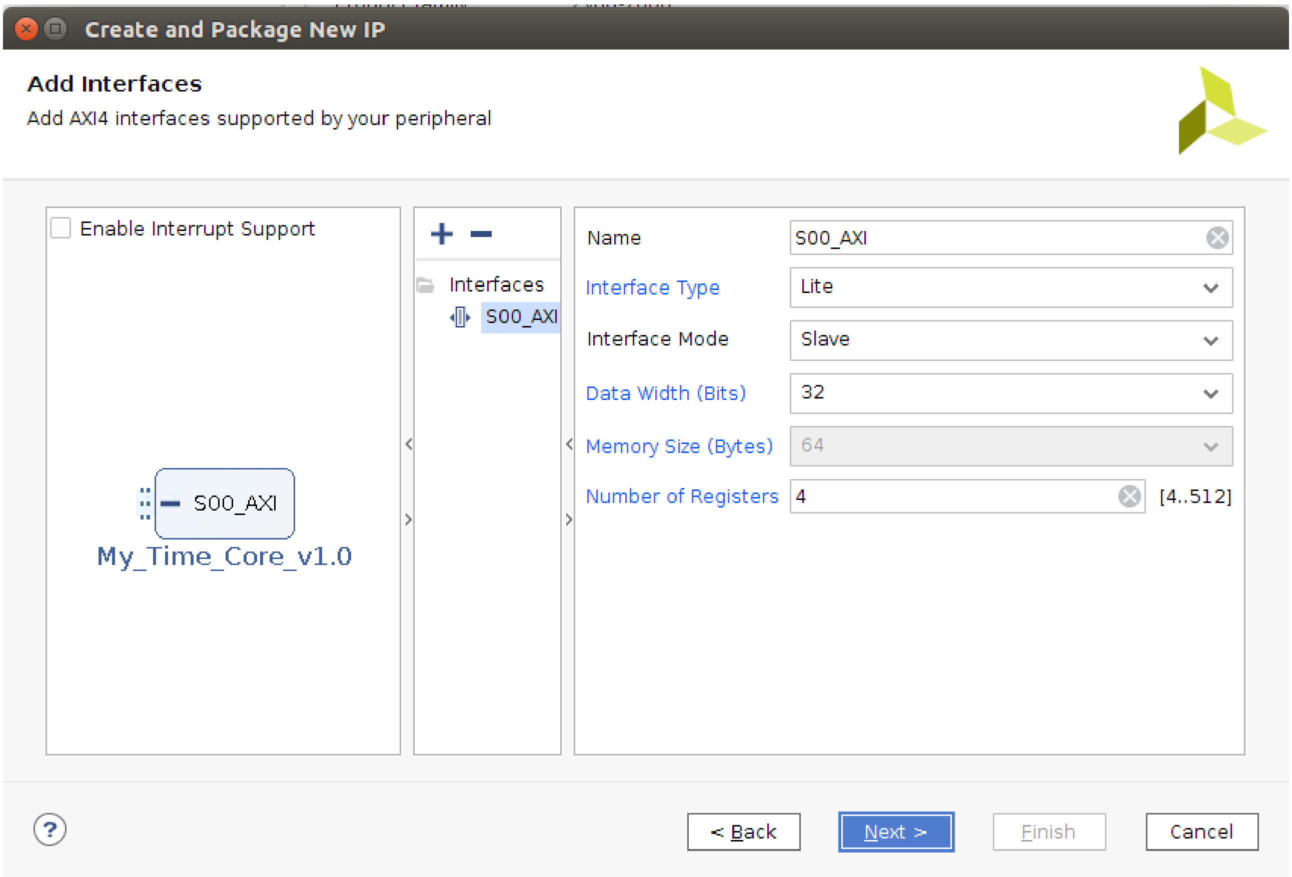
Vivado 상단에 Tools → Create and Package New IP
아래 사진과 같이 설정 후 Next 를 눌러주면 된다.

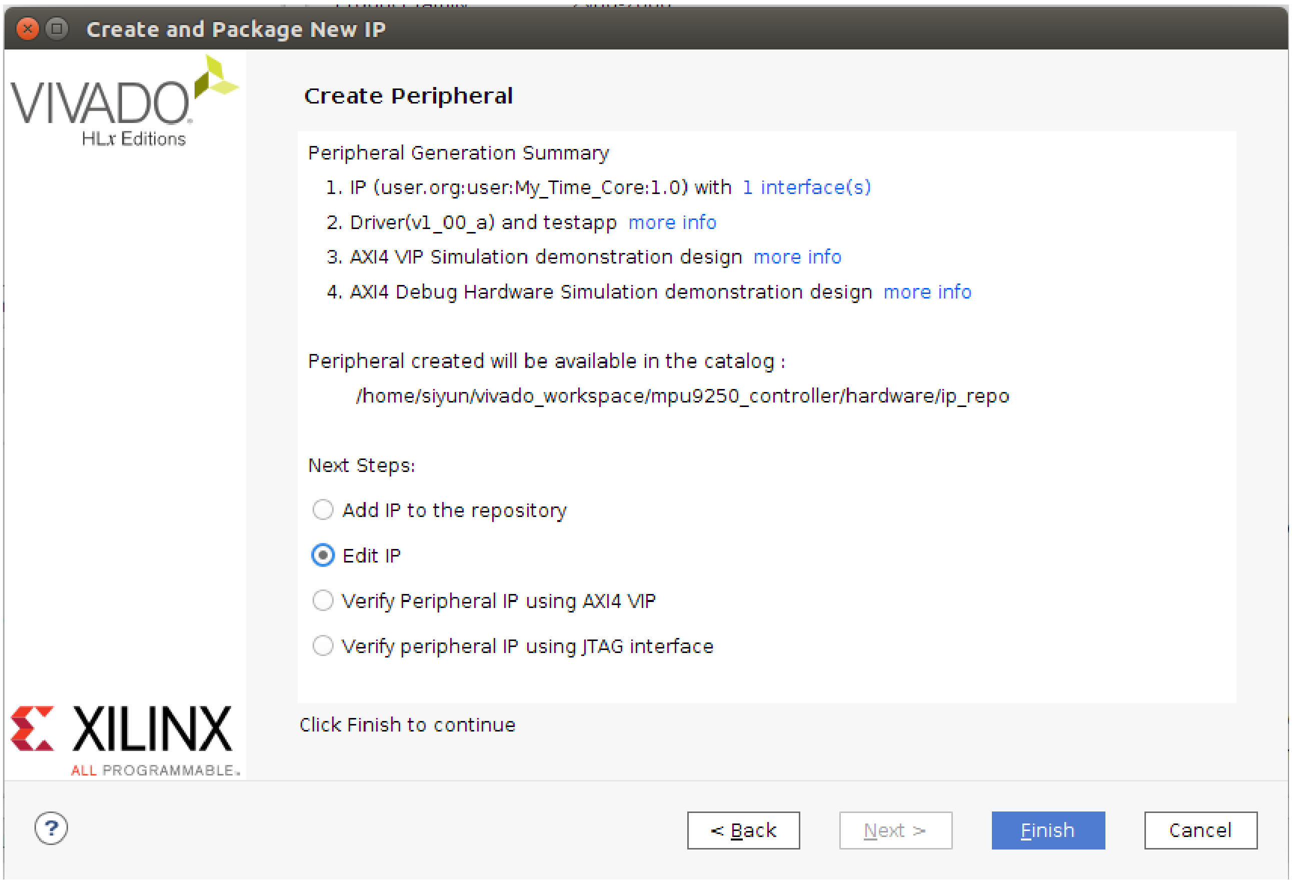






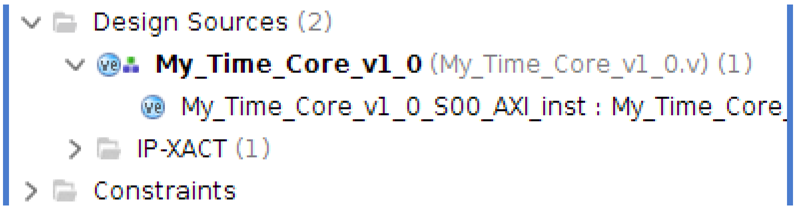

Create and Package New IP
Peripheral Details
Specify name, version and description for the new peripheral
Name: My_Time_Core
Version: 1.0
Display name: My_Time_Core_v1.0
Description: My new AXI IP
IP location: /home/siyun/vivado_workspace/mpu9250_controller/hardware/ip_repo
Overwrite existing
< Back
Next >
Finish
Cancel


Create and Package New IP
Add Interfaces
Add AXI4 interfaces supported by your peripheral
Enable Interrupt Support
S00_AXI
My_Time_Core_v1.0
Interfaces
S00_AXI
Name S00_AXI
Interface Type Lite
Interface Mode Slave
Data Width (Bits) 32
Memory Size (Bytes) 64
Number of Registers 4 [4..512]
< Back
Next >
Finish
Cancel



위 사진을 따라 한 후 Finish 를 누르게 디면 아래와같이 source 에 2 개의 베릴로그 파일이 보이게된다.

My_Time_Coore_v1_0 은 탑모듈, My_Time_Core_v1_0_s00_AXI_inst 는 하위모듈이다.
우리의 실질적인 로직은 하위모듈에 들어가고 타이머는 아웃포트가 없기 때문에 하위모듈만 수정하도록 한다.



하위모듈 베릴로그 코드.

```
`timescale 1 ns / 1 ps

	module My_Time_Core_v1_0_S00_AXI #
	(
		// Users to add parameters here

		// User parameters ends
		// Do not modify the parameters beyond this line

		// Width of S_AXI data bus
		parameter integer C_S_AXI_DATA_WIDTH	= 32,
		// Width of S_AXI address bus
		parameter integer C_S_AXI_ADDR_WIDTH	= 4
```

```
)
(
        // Users to add ports here
        // Time Base COUNTER //
        output reg [31:0] TBC,

        // User ports ends
        // Do not modify the ports beyond this line

        // Global Clock Signal
        input wire  S_AXI_ACLK,
        // Global Reset Signal. This Signal is Active LOW
        input wire  S_AXI_ARESETN,
        // Write address (issued by master, acceped by Slave)
        input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR,
        // Write channel Protection type. This signal indicates the
        // privilege and security level of the transaction, and whether
        // the transaction is a data access or an instruction access.
        input wire [2 : 0] S_AXI_AWPROT,
        // Write address valid. This signal indicates that the master signaling
        // valid write address and control information.
        input wire  S_AXI_AWVALID,
        // Write address ready. This signal indicates that the slave is ready
        // to accept an address and associated control signals.
        output wire  S_AXI_AWREADY,
        // Write data (issued by master, acceped by Slave)
        input wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_WDATA,
        // Write strobes. This signal indicates which byte lanes hold
        // valid data. There is one write strobe bit for each eight
        // bits of the write data bus.
        input wire [(C_S_AXI_DATA_WIDTH/8)-1 : 0] S_AXI_WSTRB,
        // Write valid. This signal indicates that valid write
        // data and strobes are available.
        input wire  S_AXI_WVALID,
        // Write ready. This signal indicates that the slave
        // can accept the write data.
        output wire  S_AXI_WREADY,
        // Write response. This signal indicates the status
        // of the write transaction.
        output wire [1 : 0] S_AXI_BRESP,
        // Write response valid. This signal indicates that the channel
        // is signaling a valid write response.
        output wire  S_AXI_BVALID,
        // Response ready. This signal indicates that the master
        // can accept a write response.
        input wire  S_AXI_BREADY,
        // Read address (issued by master, acceped by Slave)
        input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_ARADDR,
        // Protection type. This signal indicates the privilege
        // and security level of the transaction, and whether the
        // transaction is a data access or an instruction access.
        input wire [2 : 0] S_AXI_ARPROT,
        // Read address valid. This signal indicates that the channel
        // is signaling valid read address and control information.
        input wire  S_AXI_ARVALID,
        // Read address ready. This signal indicates that the slave is
        // ready to accept an address and associated control signals.
        output wire  S_AXI_ARREADY,
        // Read data (issued by slave)
```

```
		output wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_RDATA,
		// Read response. This signal indicates the status of the
		// read transfer.
		output wire [1 : 0] S_AXI_RRESP,
		// Read valid. This signal indicates that the channel is
		// signaling the required read data.
		output wire  S_AXI_RVALID,
		// Read ready. This signal indicates that the master can
		// accept the read data and response information.
		input wire  S_AXI_RREADY
	);


	// AXI4LITE signals
	reg [C_S_AXI_ADDR_WIDTH-1 : 0] 	axi_awaddr;
	reg  	axi_awready;
	reg  	axi_wready;
	reg [1 : 0] 	axi_bresp;
	reg  	axi_bvalid;
	reg [C_S_AXI_ADDR_WIDTH-1 : 0] 	axi_araddr;
	reg  	axi_arready;
	reg [C_S_AXI_DATA_WIDTH-1 : 0] 	axi_rdata;
	reg [1 : 0] 	axi_rresp;
	reg  	axi_rvalid;

	// Example-specific design signals
	// local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
	// ADDR_LSB is used for addressing 32/64 bit registers/memories
	// ADDR_LSB = 2 for 32 bits (n downto 2)
	// ADDR_LSB = 3 for 64 bits (n downto 3)
	localparam integer ADDR_LSB = (C_S_AXI_DATA_WIDTH/32) + 1;
	localparam integer OPT_MEM_ADDR_BITS = 1;
	//----------------------------------------------
	//-- Signals for user logic register space example
	//------------------------------------------------
	//-- Number of Slave Registers 4
	reg [C_S_AXI_DATA_WIDTH-1:0]	slv_reg0;
	reg [C_S_AXI_DATA_WIDTH-1:0]	slv_reg1;
	reg [C_S_AXI_DATA_WIDTH-1:0]	slv_reg2;
	reg [C_S_AXI_DATA_WIDTH-1:0]	slv_reg3;
	wire	 slv_reg_rden;
	wire	 slv_reg_wren;
	reg [C_S_AXI_DATA_WIDTH-1:0]	 reg_data_out;
	integer	 byte_index;
	reg	 aw_en;

	// I/O Connections assignments

	assign S_AXI_AWREADY	= axi_awready;
	assign S_AXI_WREADY	= axi_wready;
	assign S_AXI_BRESP	= axi_bresp;
	assign S_AXI_BVALID	= axi_bvalid;
	assign S_AXI_ARREADY	= axi_arready;
	assign S_AXI_RDATA	= axi_rdata;
	assign S_AXI_RRESP	= axi_rresp;
	assign S_AXI_RVALID	= axi_rvalid;
	// Implement axi_awready generation
	// axi_awready is asserted for one S_AXI_ACLK clock cycle when both
	// S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
```

```
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_awready <= 1'b0;
      aw_en <= 1'b1;
    end
  else
    begin
      if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
        begin
          // slave is ready to accept write address when
          // there is a valid write address and write data
          // on the write address and data bus. This design
          // expects no outstanding transactions.
          axi_awready <= 1'b1;
          aw_en <= 1'b0;
        end
        else if (S_AXI_BREADY && axi_bvalid)
            begin
              aw_en <= 1'b1;
              axi_awready <= 1'b0;
            end
      else
        begin
          axi_awready <= 1'b0;
        end
    end
end

// Implement axi_awaddr latching
// This process is used to latch the address when both
// S_AXI_AWVALID and S_AXI_WVALID are valid.

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_awaddr <= 0;
    end
  else
    begin
      if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
        begin
          // Write Address latching
          axi_awaddr <= S_AXI_AWADDR;
        end
    end
end

// Implement axi_wready generation
// axi_wready is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
```

```
	    if ( S_AXI_ARESETN == 1'b0 )
	      begin
	        axi_wready <= 1'b0;
	      end
	    else
	      begin
	        if (~axi_wready && S_AXI_WVALID && S_AXI_AWVALID && aw_en )
	          begin
	            // slave is ready to accept write data when
	            // there is a valid write address and write data
	            // on the write address and data bus. This design
	            // expects no outstanding transactions.
	            axi_wready <= 1'b1;
	          end
	        else
	          begin
	            axi_wready <= 1'b0;
	          end
	      end
	  end

	// Implement memory mapped register select and write logic generation
	// The write data is accepted and written to memory mapped registers when
	// axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write strobes are
used to
	// select byte enables of slave registers while writing.
	// These registers are cleared when reset (active low) is applied.
	// Slave register write enable is asserted when valid address and data are available
	// and the slave is ready to accept the write address and write data.
	assign slv_reg_wren = axi_wready && S_AXI_WVALID && axi_awready && S_AXI_AWVALID;

	always @( posedge S_AXI_ACLK )
	begin
	  if ( S_AXI_ARESETN == 1'b0 )
	    begin
	      slv_reg0 <= 0;
	      slv_reg1 <= 0;
	      slv_reg2 <= 0;
	      slv_reg3 <= 0;
	    end
	  else begin
	    if (slv_reg_wren)
	      begin
	        case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
	          2'h0:
	            for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index =
byte_index+1 )
	              if ( S_AXI_WSTRB[byte_index] == 1 ) begin
	                // Respective byte enables are asserted as per write strobes
	                // Slave register 0
	                slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
	              end
	          2'h1:
	            for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index =
byte_index+1 )
	              if ( S_AXI_WSTRB[byte_index] == 1 ) begin
	                // Respective byte enables are asserted as per write strobes
	                // Slave register 1
	                slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
```

```
                end
              2'h2:
                for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index =
byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                    // Respective byte enables are asserted as per write strobes
                    // Slave register 2
                    slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                  end
              2'h3:
                for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index =
byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                    // Respective byte enables are asserted as per write strobes
                    // Slave register 3
                    slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                  end
              default : begin
                          slv_reg0 <= slv_reg0;
                          slv_reg1 <= slv_reg1;
                          slv_reg2 <= slv_reg2;
                          slv_reg3 <= slv_reg3;
                        end
            endcase
          end
      end
    end

    // Implement write response logic generation
    // The write response and response valid signals are asserted by the slave
    // when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
    // This marks the acceptance of address and indicates the status of
    // write transaction.

    always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_bvalid  <= 0;
          axi_bresp   <= 2'b0;
        end
      else
        begin
          if (axi_awready && S_AXI_AWVALID && ~axi_bvalid && axi_wready && S_AXI_WVALID)
            begin
              // indicates a valid write response is available
              axi_bvalid <= 1'b1;
              axi_bresp  <= 2'b0; // 'OKAY' response
            end                   // work error responses in future
          else
            begin
              if (S_AXI_BREADY && axi_bvalid)
                //check if bready is asserted while bvalid is high)
                //(there is a possibility that bready is always asserted high)
                begin
                  axi_bvalid <= 1'b0;
                end
            end
        end
```

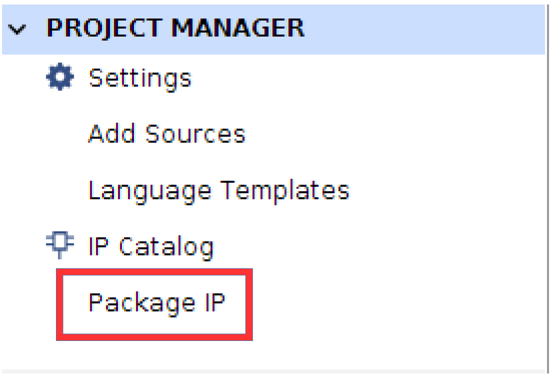
```
end

// Implement axi_arready generation
// axi_arready is asserted for one S_AXI_ACLK clock cycle when
// S_AXI_ARVALID is asserted. axi_awready is
// de-asserted when reset (active low) is asserted.
// The read address is also latched when S_AXI_ARVALID is
// asserted. axi_araddr is reset to zero on reset assertion.

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_arready <= 1'b0;
      axi_araddr  <= 32'b0;
    end
  else
    begin
      if (~axi_arready && S_AXI_ARVALID)
        begin
          // indicates that the slave has acceped the valid read address
          axi_arready <= 1'b1;
          // Read address latching
          axi_araddr  <= S_AXI_ARADDR;
        end
      else
        begin
          axi_arready <= 1'b0;
        end
    end
end

// Implement axi_arvalid generation
// axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_ARVALID and axi_arready are asserted. The slave registers
// data are available on the axi_rdata bus at this instance. The
// assertion of axi_rvalid marks the validity of read data on the
// bus and axi_rresp indicates the status of read transaction.axi_rvalid
// is deasserted on reset (active low). axi_rresp and axi_rdata are
// cleared to zero on reset (active low).
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_rvalid <= 0;
      axi_rresp  <= 0;
    end
  else
    begin
      if (axi_arready && S_AXI_ARVALID && ~axi_rvalid)
        begin
          // Valid read data is available at the read data bus
          axi_rvalid <= 1'b1;
          axi_rresp  <= 2'b0; // 'OKAY' response
        end
      else if (axi_rvalid && S_AXI_RREADY)
        begin
          // Read data is accepted by the master
          axi_rvalid <= 1'b0;
```

```
            end
          end
        end

        // Implement memory mapped register select and read logic generation
        // Slave register read enable is asserted when valid address is available
        // and the slave is ready to accept the read address.
        assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
        always @(*)
        begin
              // Address decoding for reading registers
              case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
                2'h0   : reg_data_out <= TBC;
                2'h1   : reg_data_out <= slv_reg1;
                2'h2   : reg_data_out <= slv_reg2;
                2'h3   : reg_data_out <= slv_reg3;
                default : reg_data_out <= 0;
              endcase
        end

        // Output register or memory read data
        always @( posedge S_AXI_ACLK )
        begin
          if ( S_AXI_ARESETN == 1'b0 )
            begin
              axi_rdata  <= 0;
            end
          else
            begin
              // When there is a valid read address (S_AXI_ARVALID) with
              // acceptance of read address by the slave (axi_arready),
              // output the read dada
              if (slv_reg_rden)
                begin
                  axi_rdata <= reg_data_out;     // register read data
                end
            end
        end

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

    TBC <= TBC + 32'd1;

    if(TBC == 32'd4294967295) begin
    TBC <=0;
    end

end

        // User logic ends

        endmodule
```

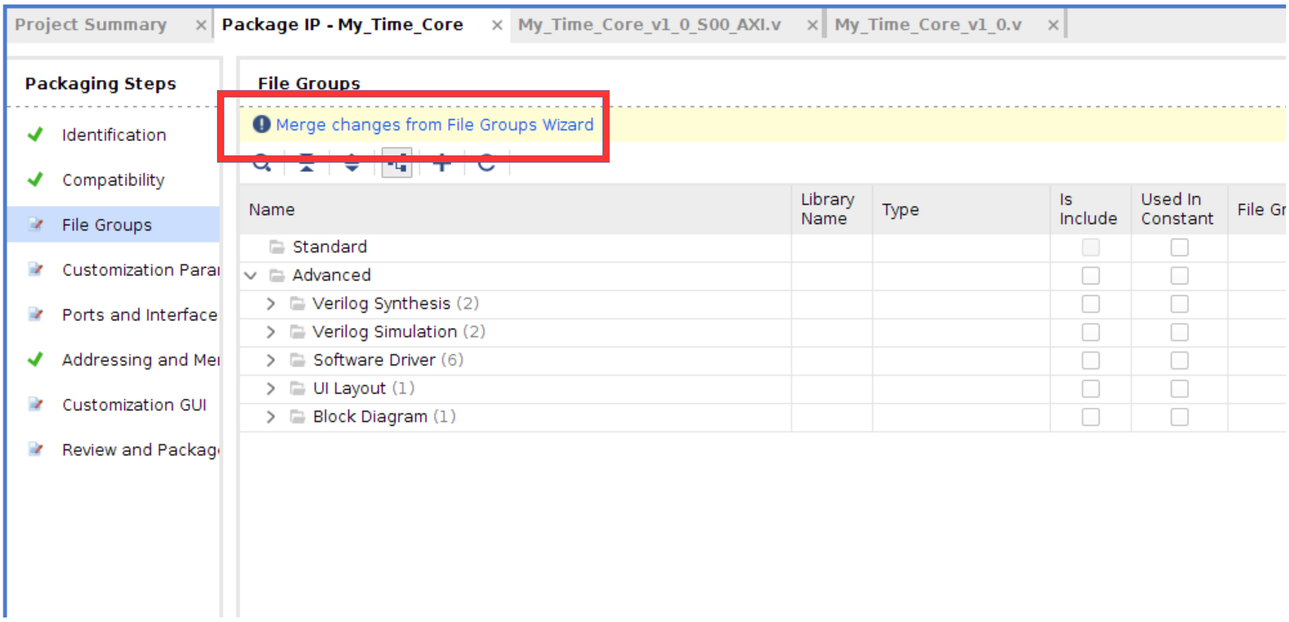
빨간색 글씨가 수정한 부분이므로 따라해주도록한다.

정확히 타이머 회로라기보다는 카운터 회로로 클럭이 들어올 때마다 1 씩 증가한다.

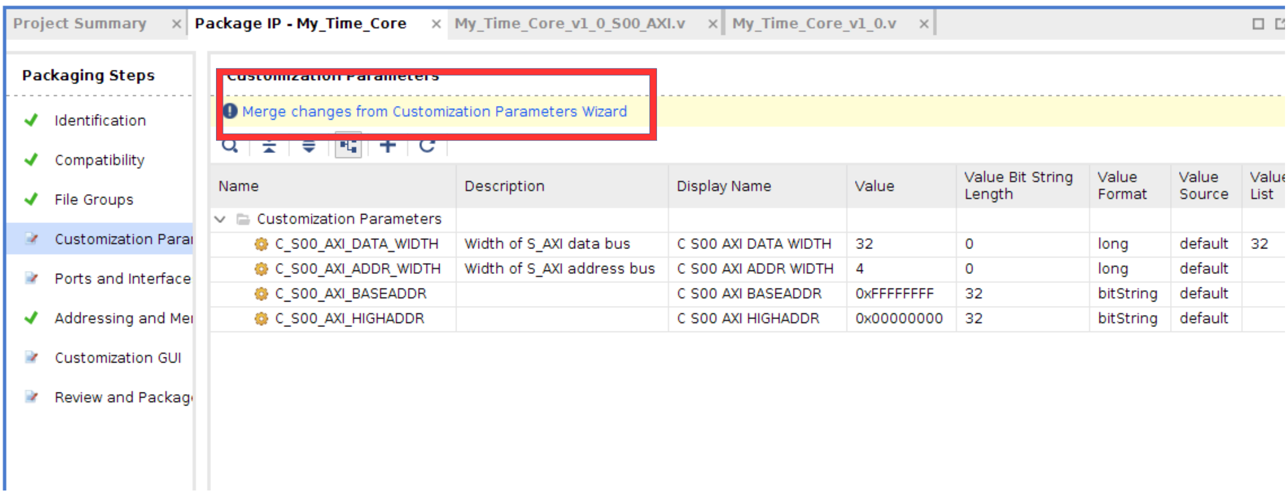
카운터 값을 얻어 시간을 계산하는 소스코드를 구현하기위해 카운터 값을 얻을수 있게 reg_data_out 에 카운터 값을 넣어준다.



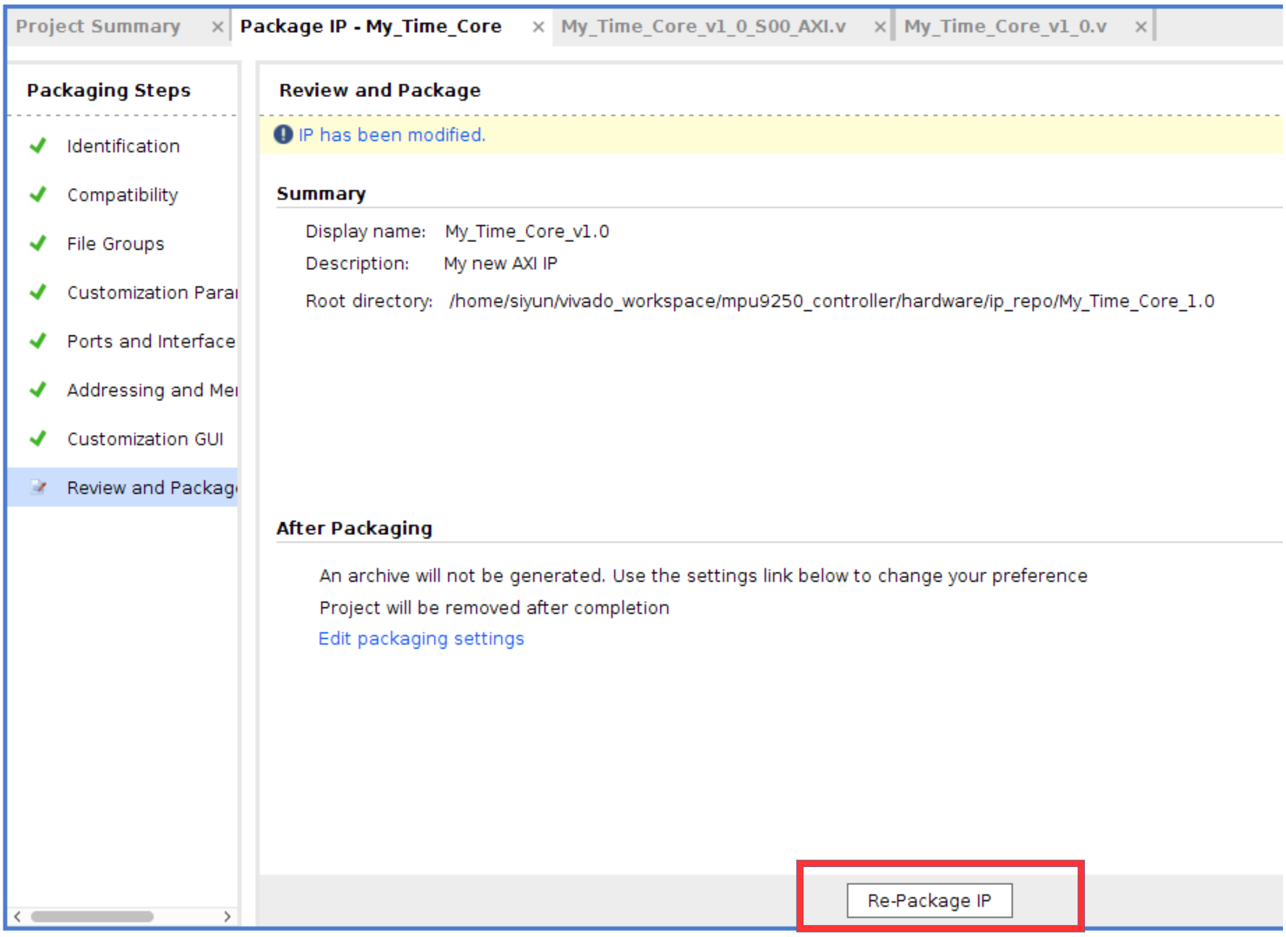
코드 수정을 완료하였으면 Project Manager 에서 Package IP 를 클릭한다.



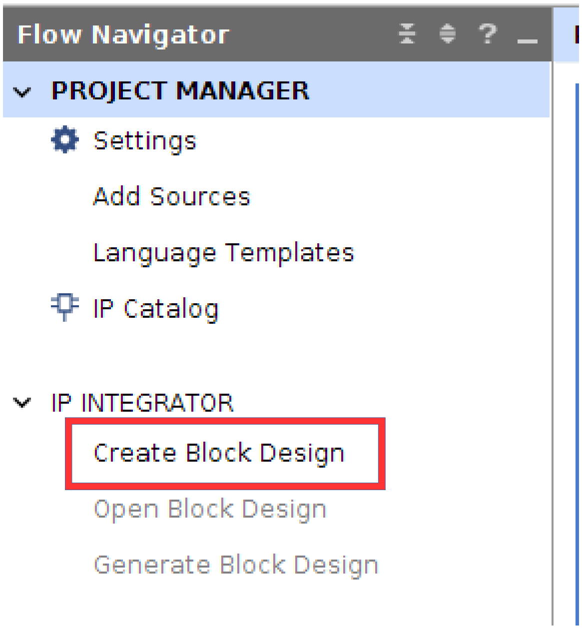
그 후 File Groups , Customization 등 체크표시가 없는 것들을 클릭하여 Merge Change 를 눌러준다

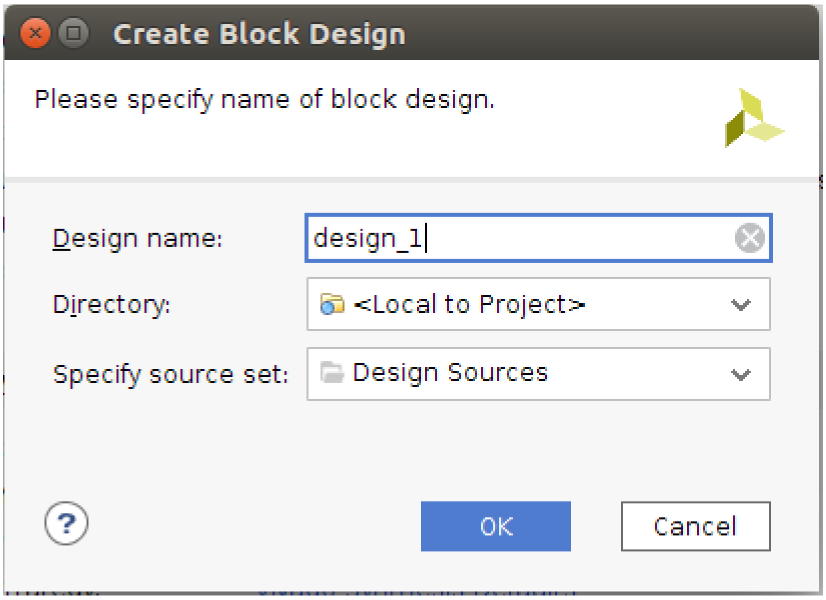


그 후 모든것이 초록색 체크표시가 뜨면 맨 밑에ㅔ Review and Package 를 클릭하고
Re-Package IP 를 클릭한다.

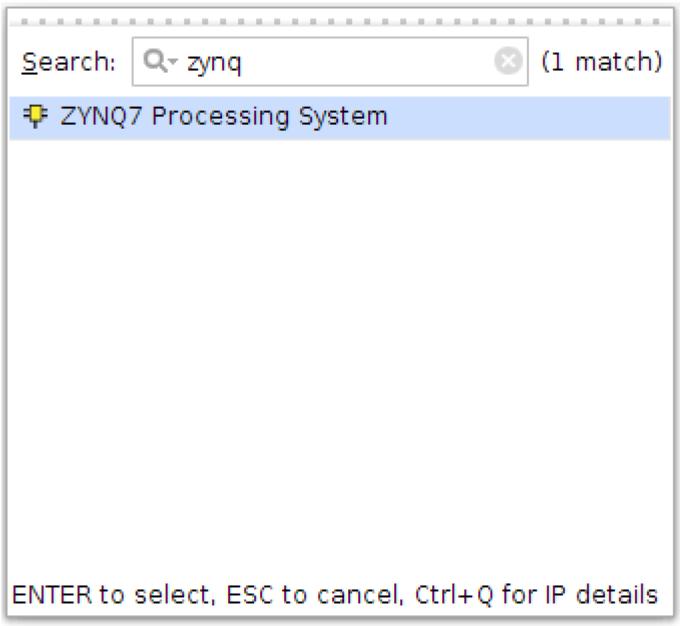


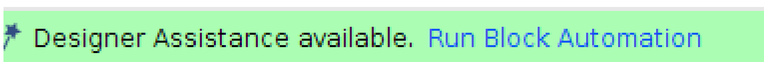
그럼 Create IP 가 종료되고 원래 프로젝트로 다시 넘어오게 된다.
이제 Block Design(H/W)을 설계한다. Create Block Design 을 클릭한다

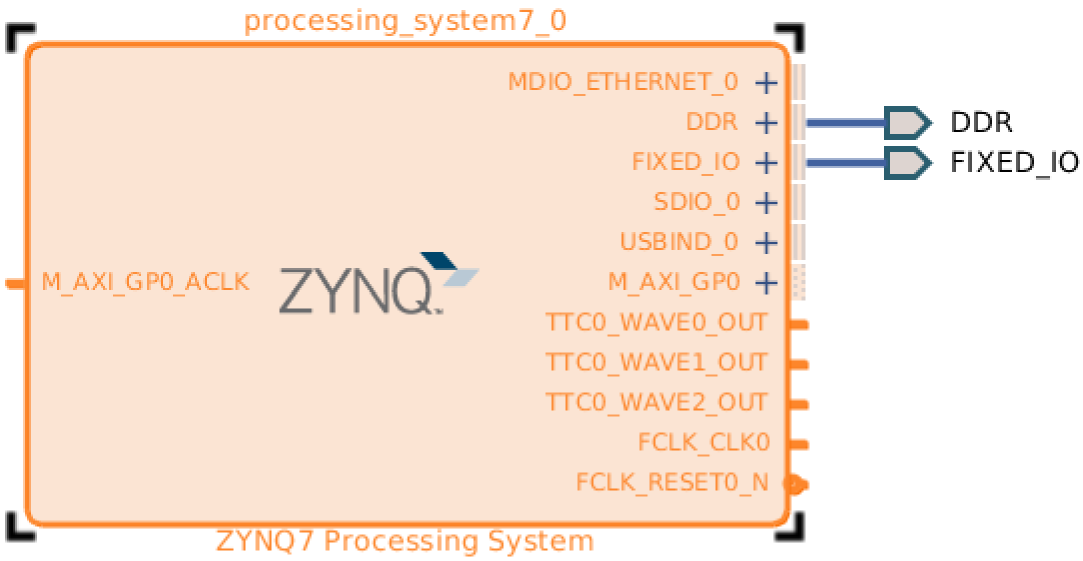




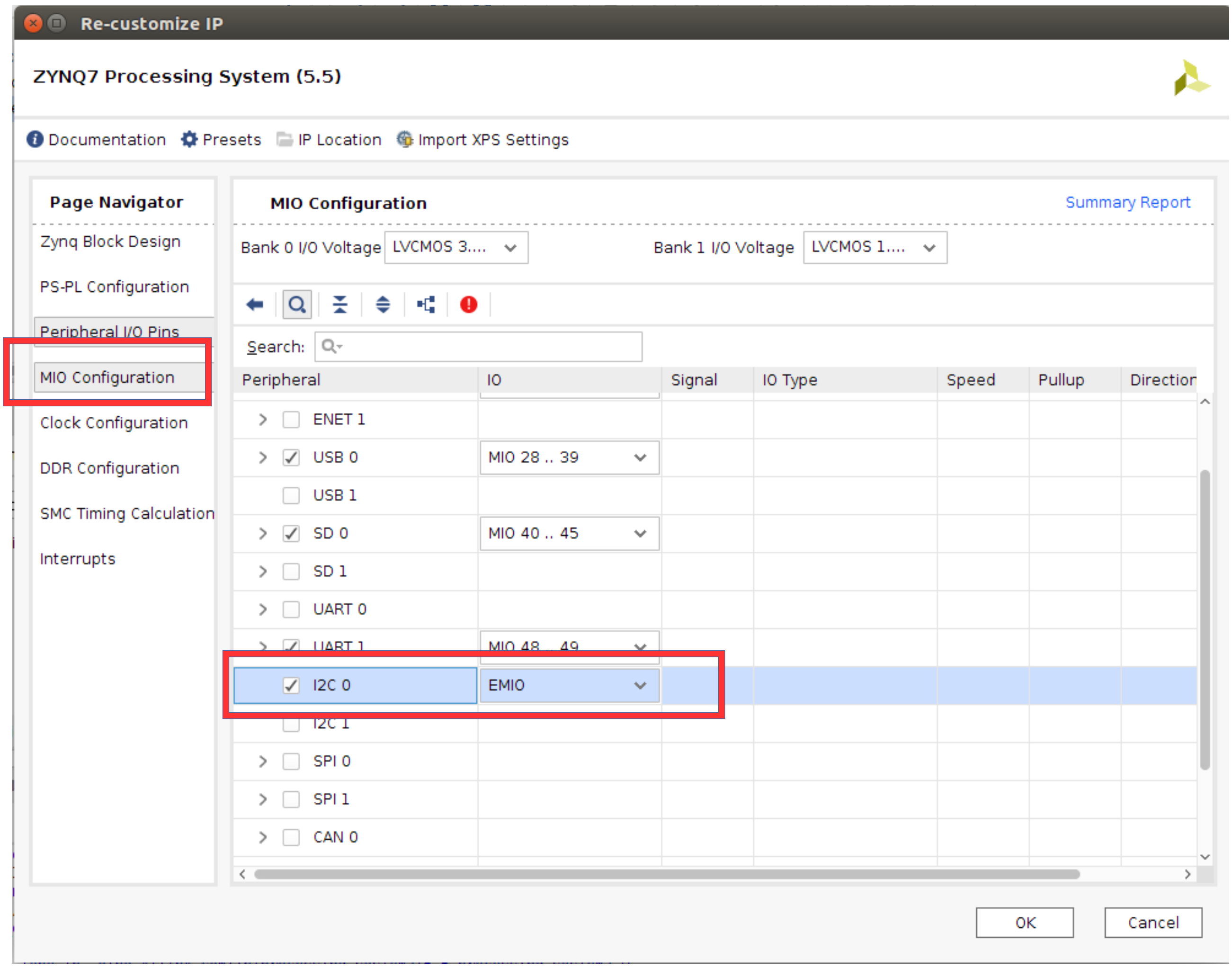
블록디자인을 만들었으면 IP 를 추가하여 아래의 과정을 따라한다.

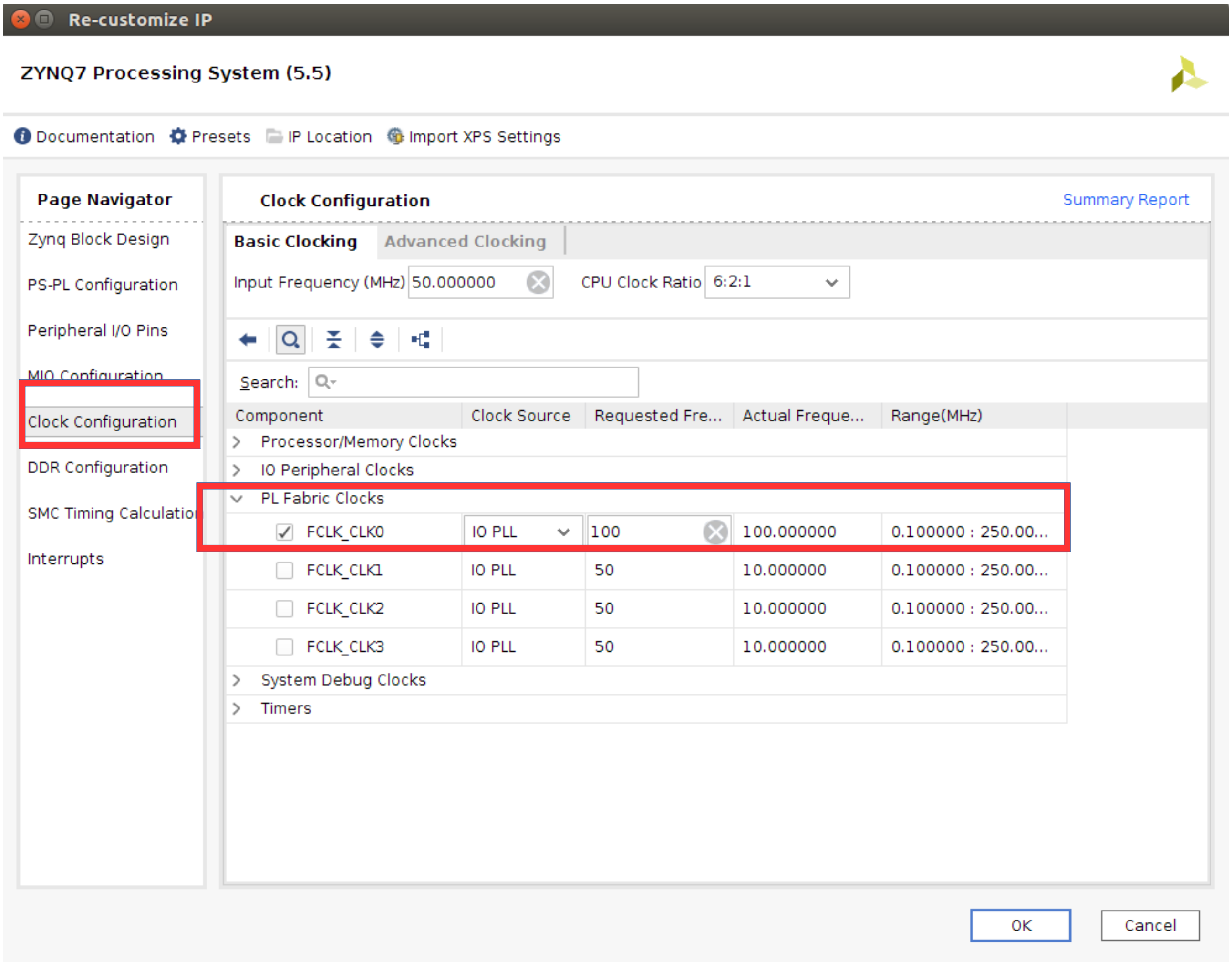






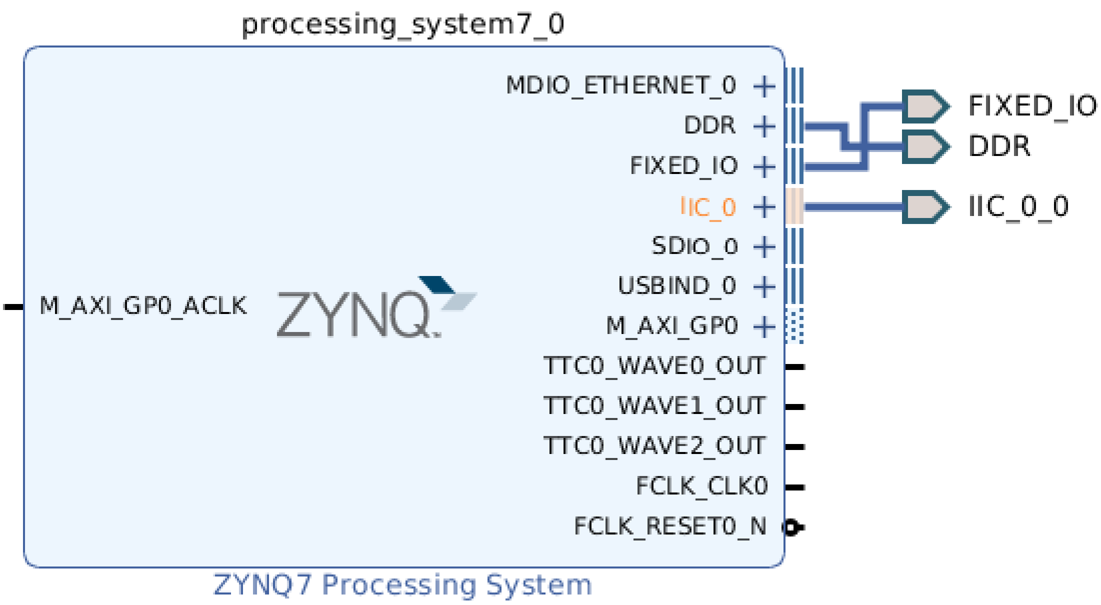
블록을 더블클릭하요 zynq 의 설정을 아래와 같이 바꾼다.

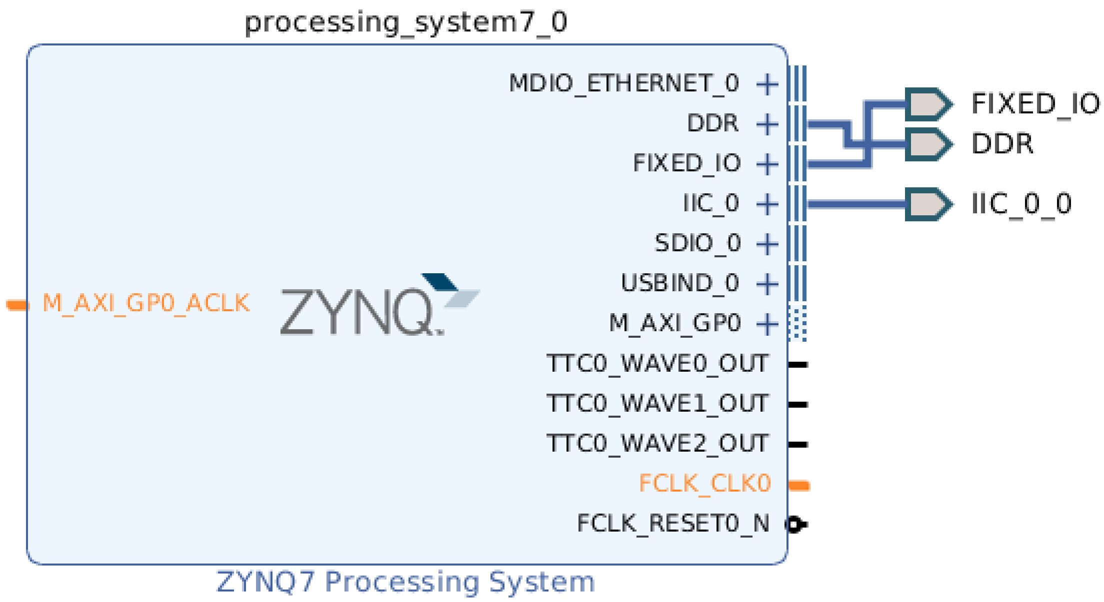


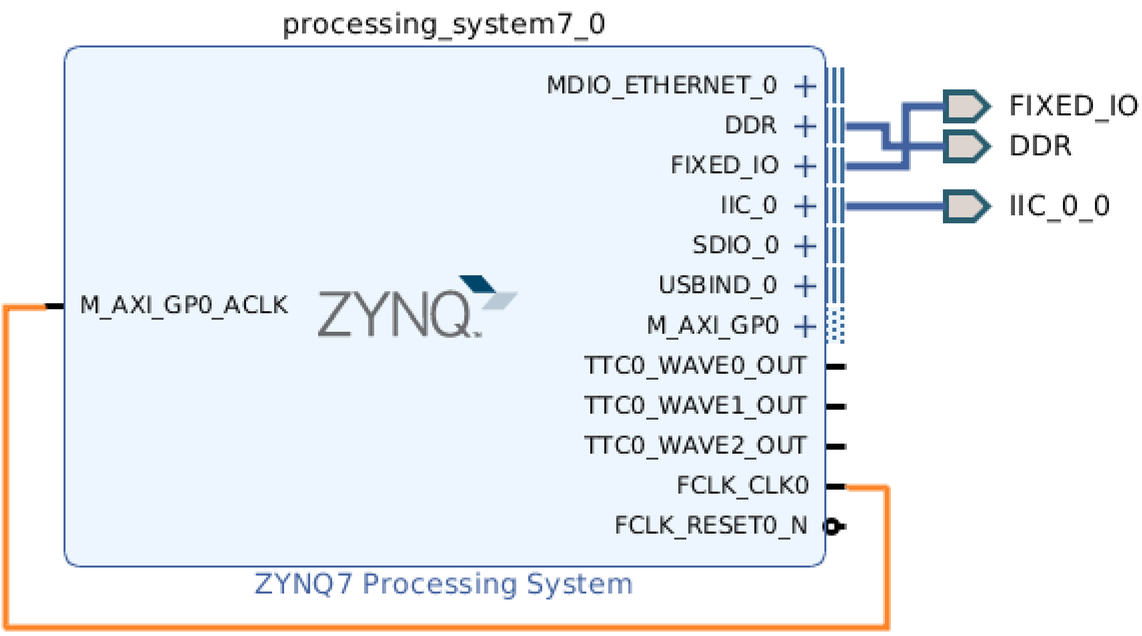


OK 를 눌러주어 설정을 적용한 후 아래를 따라한다.

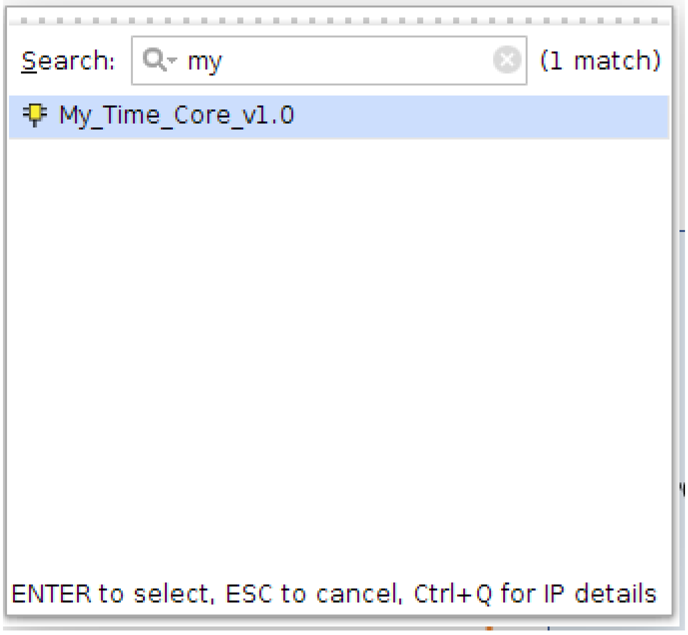
i2c 를 활성화 시켰기 때문에 포트가 생긴다. 포트를 외부로 빼준다. IIC_0 을 클릭 후 우클릭 하여 Make External 을 클릭하여 외부 버스선 포트를 만든다.



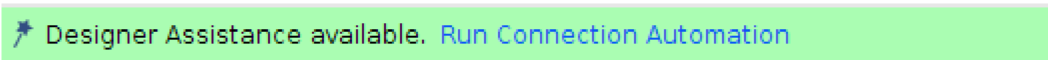




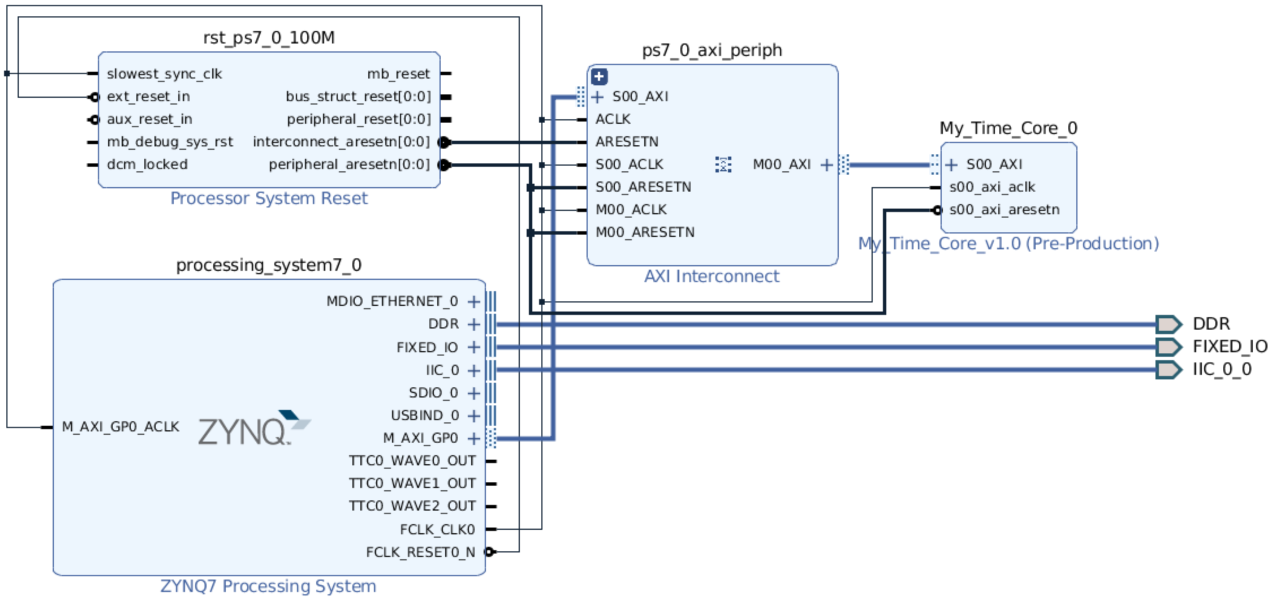
FCLK_CLK0 을 M_AXI_GP0_ACLK 와 연결해준다.

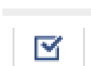


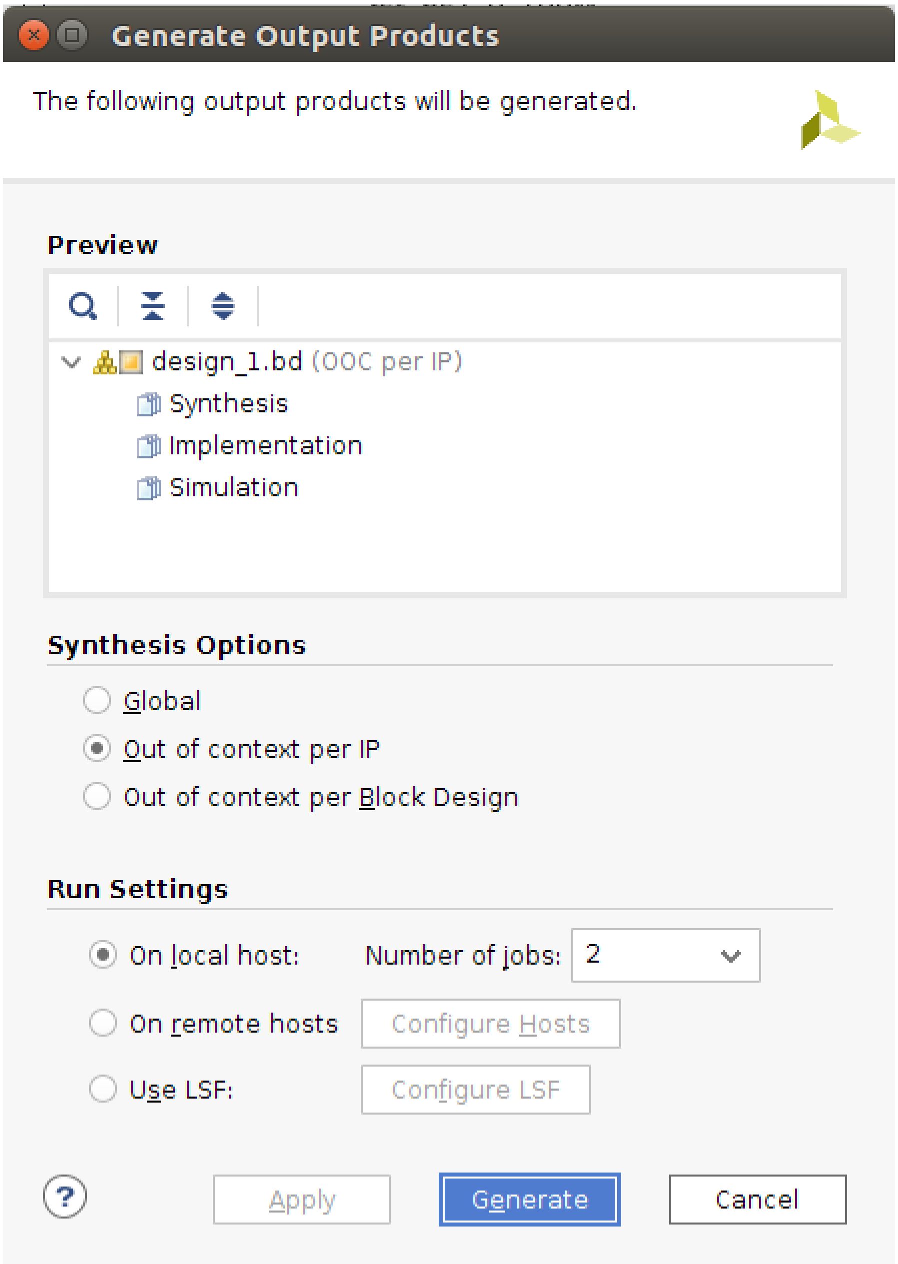
이제 우리가 만든 Custom IP 인 My_Time_Core 를 추가한다.

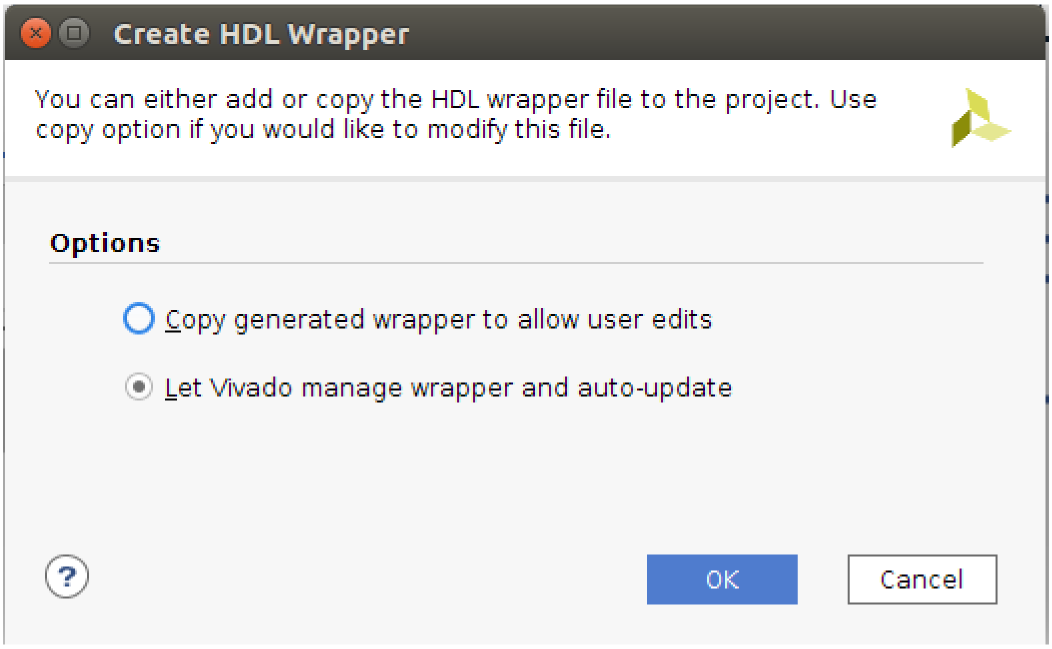


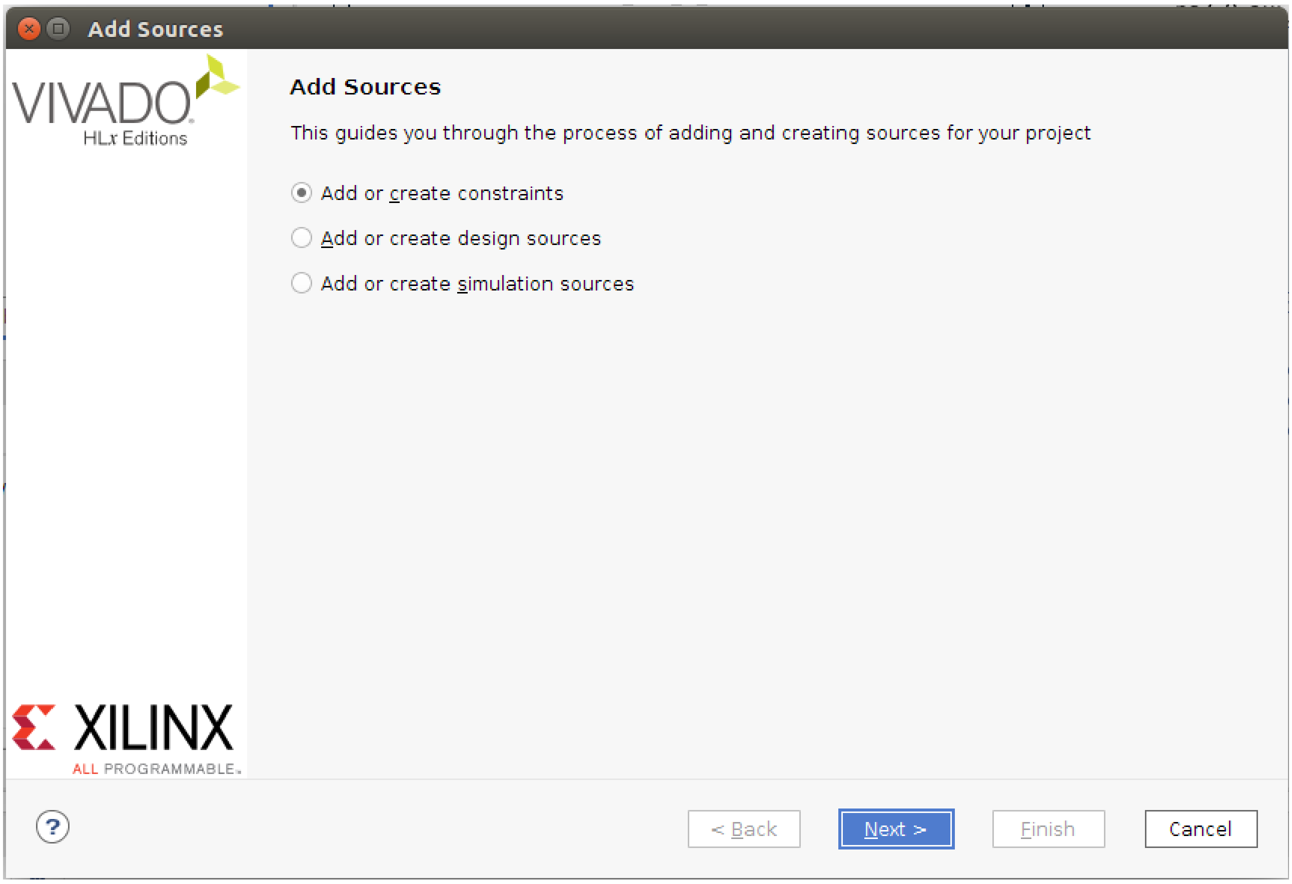

런 커넥션 후 새로고침 버튼을 눌러주면 아래와 같은 하드웨어가 완성된다.

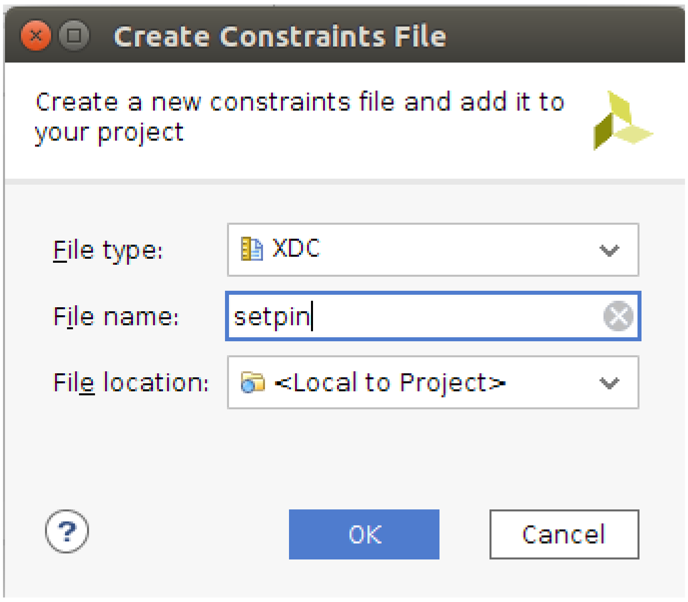



← 이 버튼을 클릭하여 회로의 오류가 있는지 없는지 체크한다. 오류가 난다면 한번더 눌러주면 없어질 것이다. 에러가 난다면 잘못된 회로이므로 위 과정을 다시 확인한다.

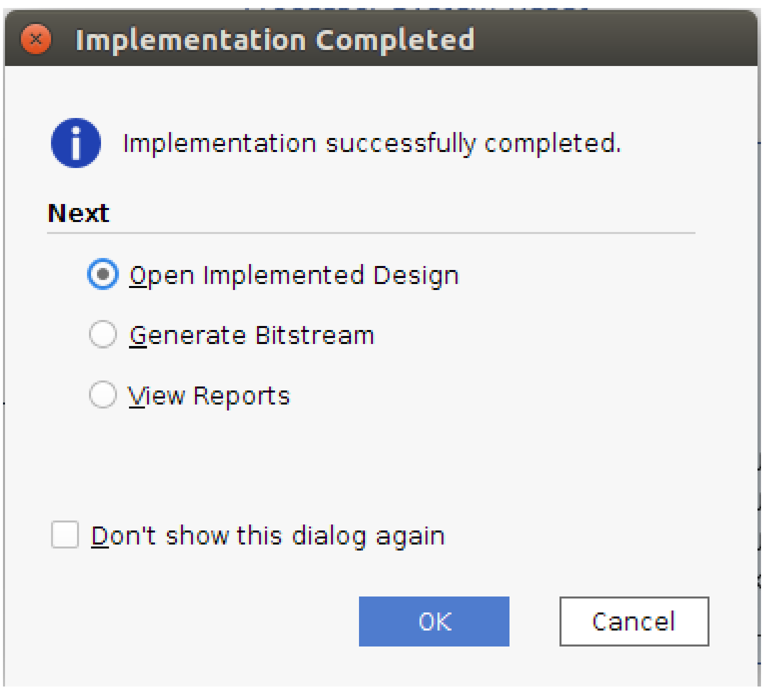

Generate Output Products
The following output products will be generated.
Preview
design_1.bd (OOC per IP)
Synthesis
Implementation
Simulation
Synthesis Options
Global
Out of context per IP
Out of context per Block Design
Run Settings
On local host:
Number of jobs: 2
On remote hosts
Configure Hosts
Use LSF:
Configure LSF
?
Apply
Generate
Cancel

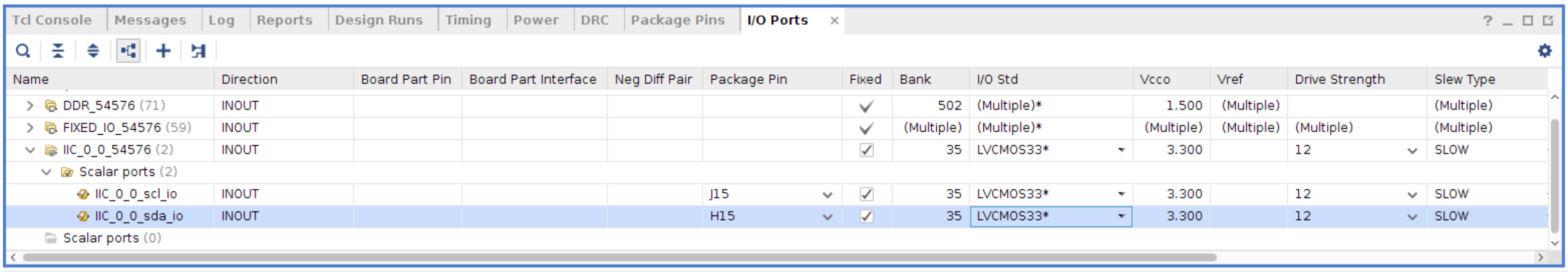

Create HDL Wrapper
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.
Options
Copy generated wrapper to allow user edits
Let Vivado manage wrapper and auto-update
OK
Cancel


Add Sources
VIVADO
HLx Editions
Add Sources
This guides you through the process of adding and creating sources for your project
Add or create constraints
Add or create design sources
Add or create simulation sources
XILINX
ALL PROGRAMMABLE.
< Back
Next >
Finish
Cancel

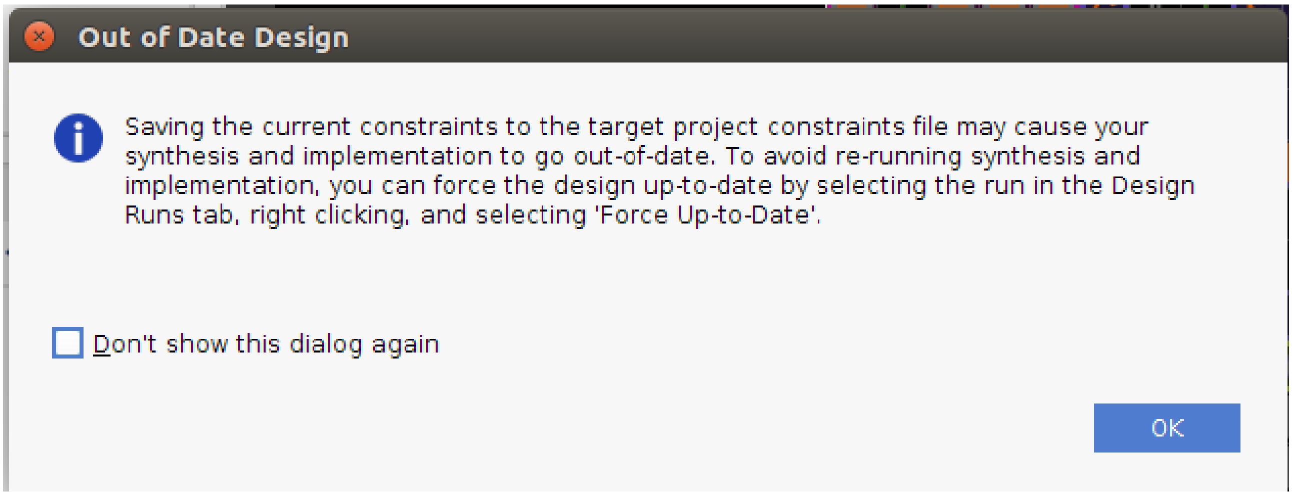

Create Constraints File
Create a new constraints file and add it to your project
File type:
XDC
File name:
setpin
File location:
<Local to Project>
OK
Cancel

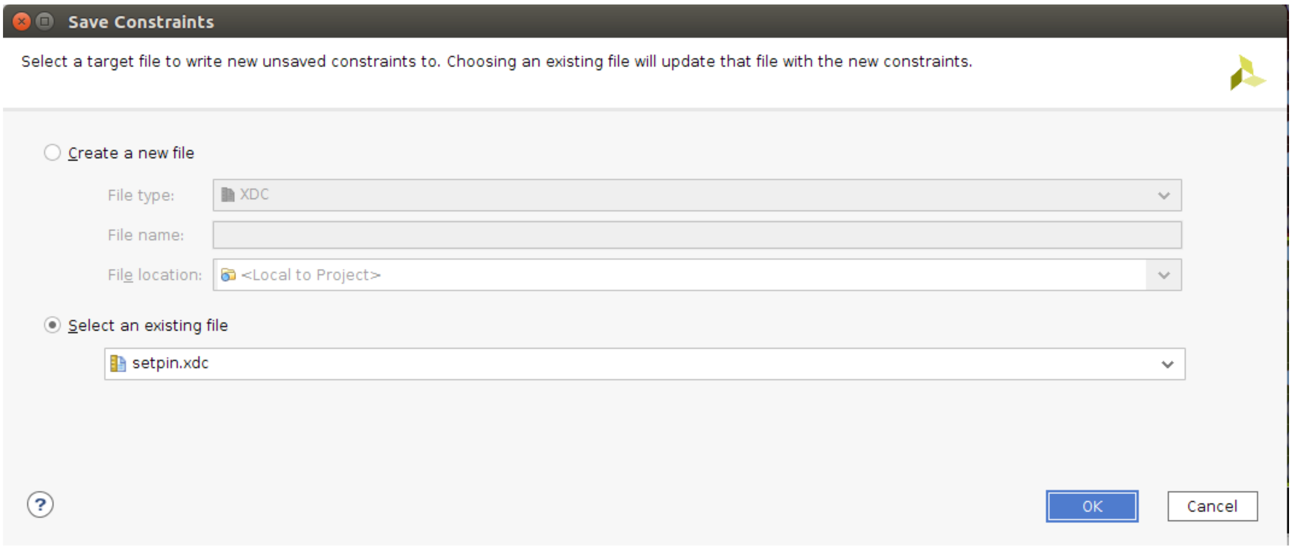

IMPLEMENTATION
Run Implementation
Open Implemented Design

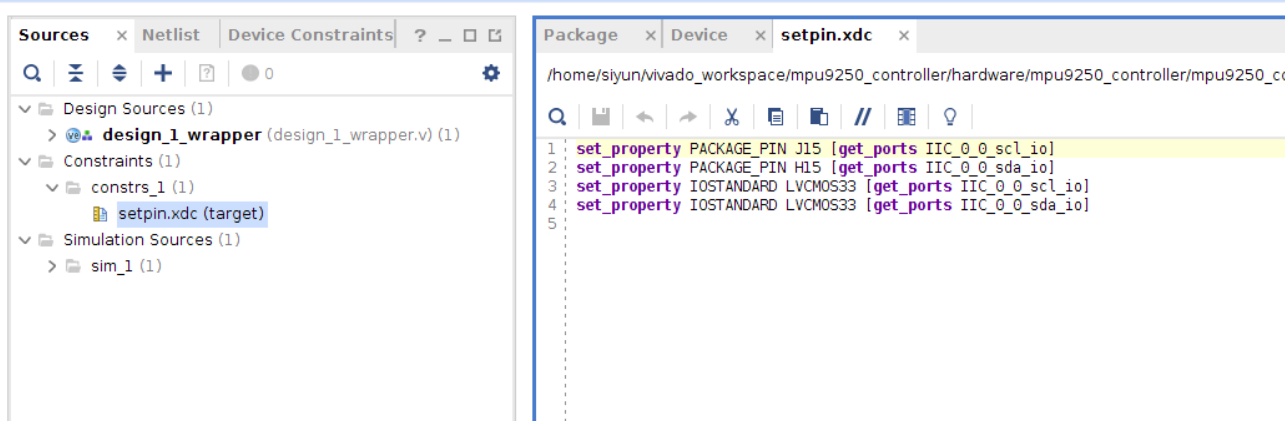

Implementation Completed
Implementation successfully completed.
Next
Open Implemented Design
Generate Bitstream
View Reports
Don't show this dialog again
OK
Cancel



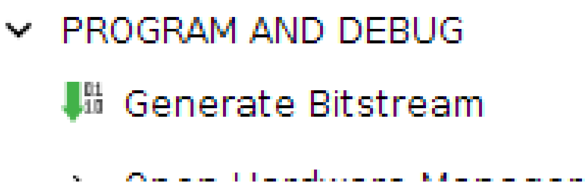
Save 하고 ok


Out of Date Design
Saving the current constraints to the target project constraints file may cause your synthesis and implementation to go out-of-date. To avoid re-running synthesis and implementation, you can force the design up-to-date by selecting the run in the Design Runs tab, right clicking, and selecting 'Force Up-to-Date'.
Don't show this dialog again
OK


Save Constraints
Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.
Create a new file
File type:
XDC
File name:
File location:
<Local to Project>
Select an existing file
setpin.xdc
OK
Cancel

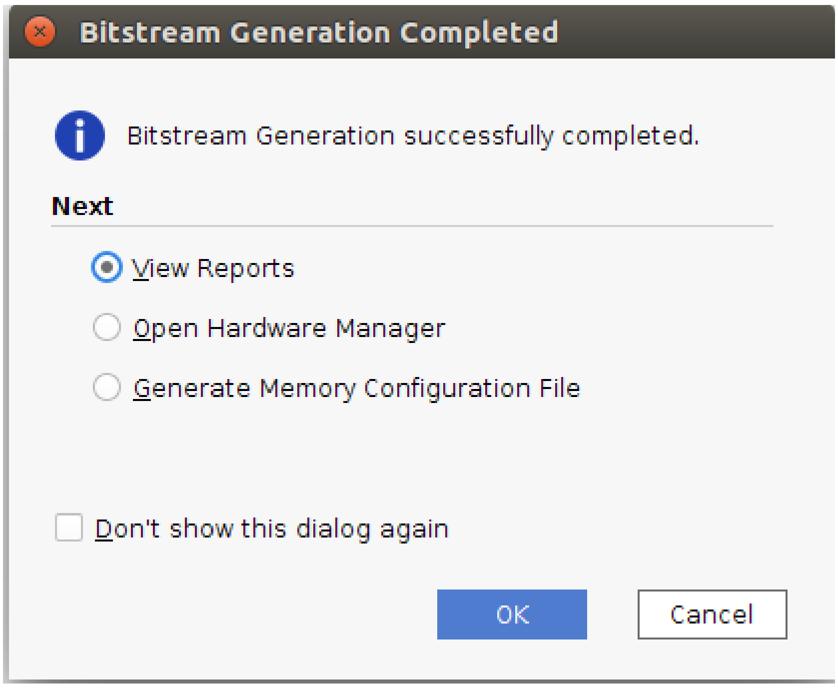


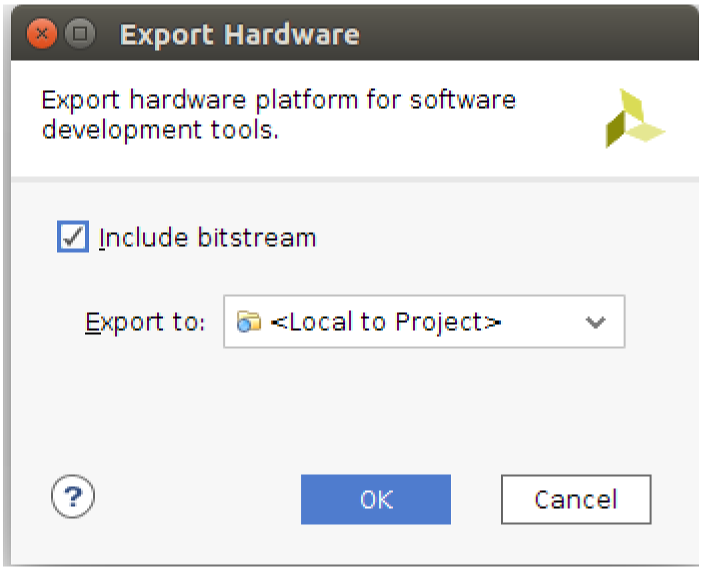
XDC 파일에 위와같이 생겼는지 확인한다.



비트스트림을 생성한다.

완료되면 File → export → export hardware 를 클릭한다.





이제 하드웨어 설계와 하드웨어 정보를 밖으로 빼내는 작업을 완료하였다.

페타리눅스 와 소프트웨어 설계로 넘어가도록 한다.

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller$ petalinux-create -t p
roject -n software --template zynq
INFO: Create project: software
INFO: New project successfully created in /home/siyun/vivado_workspace/mpu9250_c
ontroller/software
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller$
```

Petalinux-config –get-hw-description ../hardware/mpu9250_controller/mpu9250_controller.sdk

```
controller/software$ petalinux-config --get-hw-description ../hardware/mpu9250_controller/mpu9250_controller.sdk
```

```
< Exit >
```

```
[INFO ] oldconfig linux/u-boot
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config -c u-boot
```

```
    Architecture select (ARM architecture)  --->
    ARM architecture  --->
    General setup  --->
    Boot images  --->
    Command line interface  --->
    Device Tree Control  --->
-*- Networking support  --->
    Device Drivers  --->
[ ] Custom physical to bus address mapping
    File systems  ----
    Library routines  --->
[ ] Unit tests  ----
```

```
[*] Enable Driver Model for I2C drivers
[*] Enable I2C compatibility layer
```

```
< Save >        < Exit >
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config
```

```
[INFO ] oldconfig linux/u-boot
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config -c kernel
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config -c rootfs
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-build
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-create -t apps -n devic
e_driver --enable
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ cd component
s/apps/
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/components/ap
ps$ ls
device_driver
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/components/ap
ps$ cd device_driver/
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/components/ap
ps/device_driver$ ls
device_driver.c  Kconfig  Makefile  README
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/components/ap
ps/device_driver$ vi device_driver.c
```

device_driver.c

```
#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>
#include <sys/mman.h>
#include <signal.h>

void my_sig(int signo)
{
        printf("signal CTRL + C\n");
        exit(1);

}
```

```
// Using NOKIA 5110 monochrome 84 x 48 pixel display
// pin 9 - Serial clock out (SCLK)
// pin 8 - Serial data out (DIN)
// pin 7 - Data/Command select (D/C)
// pin 5 - LCD chip select (CS)
// pin 6 - LCD reset (RST)

// See also MPU-9250 Register Map and Descriptions, Revision 4.0, RM-MPU-9250A-00, Rev. 1.4, 9/9/2013 for registers not listed in
// above document; the MPU9250 and MPU9150 are virtually identical but the latter has a different register map
//
//Magnetometer Registers
#define AK8963_ADDRESS   0x0C
#define AK8963_WHO_AM_I  0x00 // should return 0x48
#define AK8963_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      0x66
#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 TIME_MAP_SIZE   0x1000
#define TIME_DATA_OFFSET 0x0
#define TIME_DATA2_OFFSET 0x4
#define TIME_TRI_OFFSET 0x4



// Using the MSENSR-9250 breakout board, ADO is set to 0
// Seven-bit device address is 110100 for ADO = 0 and 110101 for ADO = 1
#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

// Set initial input parameters
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
};

// Specify sensor full scale
uint8_t Gscale = GFS_250DPS;
uint8_t Ascale = AFS_2G;
uint8_t Mscale = MFS_16BITS; // Choose either 14-bit or 16-bit magnetometer resolution
uint8_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

// Pin definitions
int intPin = 12;  // These can be changed, 2 and 3 are the Arduinos ext int pins
int myLed = 13; // Set up pin 13 led for toggling

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
int16_t tempCount;      // temperature raw count output
float   temperature;    // Stores the real internal chip temperature in degrees Celsius
float   SelfTest[6];    // holds results of gyro and accelerometer self test

// global constants for 9 DoF fusion and AHRS (Attitude and Heading Reference System)
float GyroMeasError = M_PI * (40.0f / 180.0f);   // gyroscope measurement error in rads/s (start at 40 deg/s)
float GyroMeasDrift = M_PI * (0.0f  / 180.0f);   // gyroscope measurement drift in rad/s/s (start at 0.0 deg/s/s)
// There is a tradeoff in the beta parameter between accuracy and response speed.
// In the original Madgwick study, beta of 0.041 (corresponding to GyroMeasError of 2.7 degrees/s) was found to
give optimal accuracy.
// However, with this value, the LSM9SD0 response time is about 10 seconds to a stable initial quaternion.
// Subsequent changes also require a longish lag time to a stable output, not fast enough for a quadcopter or robot
car!
// By increasing beta (GyroMeasError) by about a factor of fifteen, the response time constant is reduced to ~2
sec
// I haven't noticed any reduction in solution accuracy. This is essentially the I coefficient in a PID control sense;
// the bigger the feedback coefficient, the faster the solution converges, usually at the expense of accuracy.
```

```
// In any case, this is the free parameter in the Madgwick filtering and fusion scheme.
float beta; // = sqrt(3.0 / 4.0) * GyroMeasError;   // compute beta
//float zeta = sqrt(3.0 / 4.0) * GyroMeasDrift;   // compute zeta, the other free parameter in the Madgwick
scheme usually set to a small or zero value
#define Kp 2.0f * 5.0f // these are the free parameters in the Mahony filter and fusion scheme, Kp for
proportional feedback, Ki for integral
#define Ki 0.0f

uint32_t delt_t = 0; // used to control display output rate
uint32_t count = 0, sumCount = 0; // used to control display output rate
float pitch, yaw, roll;
float deltat = 0.0f, sum = 0.0f;        // integration interval for both filter schemes
uint32_t lastUpdate = 0, firstUpdate = 0; // used to calculate integration interval
uint32_t Now = 0;        // used to calculate integration interval

float ax, ay, az, gx, gy, gz, mx, my, mz; // variables to hold latest sensor data values
float q[4] = {1.0f, 0.0f, 0.0f, 0.0f};    // vector to hold quaternion
float eInt[3] = {0.0f, 0.0f, 0.0f};       // vector to hold integral error for Mahony method


int fd =0;
int fd2 = 0;
void *ptr;

void MadgwickQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my, float
mz);

void MahonyQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my, float
mz);
void getMres();
void getAres();
void getGres();
void readAccelData(int fd,int16_t * destination);
void readGyroData(int fd,int16_t * destination);
void readMagData(int fd,int16_t * destination);
int16_t readTempData(int fd);
void initAK8963(int fd,float * destination);
void initMPU9250(int fd);
void calibrateMPU9250(int fd,float * dest1, float * dest2);
void MPU9250SelfTest(int fd,float * destination);
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);
int ioctl_mpu9250(int fd);
int ioctl_ak8963(int fd);
uint32_t millis(void *ptr,uint32_t start);
uint32_t micros(void *ptr,uint32_t start);

int main(void)
{
        uint32_t start = 0;

                if((fd2 = open("/dev/uio0",O_RDWR)) <0)
    {
        perror("Open Time Device Error!! \n");
        return -1;
    }

    ptr = mmap(NULL,TIME_MAP_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,fd2,0);
```

```
	printf("Time device malloc Success!!\n");

	start = *((unsigned *)(ptr + TIME_DATA_OFFSET));
	printf("Time start !\n");

			if((fd = open("/dev/i2c-0", O_RDWR)) <0)
	{
		perror("Open Device Error! \n");
		return -1;
	}

			ioctl_mpu9250(fd);
			uint8_t c = readByte(fd, WHO_AM_I_MPU9250);  // Read WHO_AM_I register for MPU-
9250
			printf("MPU9250 I AM %x\n",c);
			usleep(1000);

			if (c == 0x71) // WHO_AM_I should always be 0x68
			{
					printf("MPU9250 is online...\n");

					MPU9250SelfTest(fd,SelfTest); // Start by performing self test and reporting
values
					printf("x-axis self test: acceleration trim within : %f %% of factory
value\n",SelfTest[0]);
					printf("y-axis self test: acceleration trim within : %f %% of factory
value\n",SelfTest[1]);
					printf("z-axis self test: acceleration trim within : %f %% of factory
value\n",SelfTest[2]);
					printf("x-axis self test: gyration trim within :%f %% of factory value\n
",SelfTest[3]);
					printf("y-axis self test: gyration trim within :%f %% of factory value\n
",SelfTest[4]);
					printf("z-axis self test: gyration trim within :%f %% of factory value\n
",SelfTest[5]);

					calibrateMPU9250(fd,gyroBias, accelBias); // Calibrate gyro and
accelerometers, load biases in bias registers

					usleep(1000);

					initMPU9250(fd);
					printf("MPU9250 initialized for active data mode....\n"); // Initialize device for
active mode read of acclerometer, gyroscope, and temperature

					// Read the WHO_AM_I register of the magnetometer, this is a good test of
communication
					ioctl_ak8963(fd);
					uint8_t d = readByte(fd, AK8963_WHO_AM_I);  // Read WHO_AM_I
register for AK8963
					printf("AK8963 I AM %x\n",d);
					usleep(1000);

					// Get magnetometer calibration from AK8963 ROM
					initAK8963(fd,magCalibration); printf("AK8963 initialized for active data
mode....\n"); // Initialize device for active mode read of magnetometer

					if(SerialDebug) {
```

```
                                        //  Serial.println("Calibration values: ");
                                        printf("X-Axis sensitivity adjustment value 
%f\n",magCalibration[0]);
                                        printf("Y-Axis sensitivity adjustment value 
%f\n",magCalibration[1]);
                                        printf("Z-Axis sensitivity adjustment value 
%f\n",magCalibration[2]);
                              }
                              usleep(1000);
               }
               else
               {
                              printf("Could not connect to MPU9250: 0x%x\n",c);

                              while(1) ; // Loop forever if communication doesn't happen
               }


               while(1)
               {

                              signal(SIGINT,my_sig);

                              ioctl_mpu9250(fd);
                              // If intPin goes high, all data registers have new data
                              if (readByte(fd, INT_STATUS) & 0x01) {  // On interrupt, check if data ready 
interrupt
                                        readAccelData(fd,accelCount);  // Read the x/y/z adc values
                                        getAres();

                                        // Now we'll calculate the accleration value into actual g's
                                        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();

                                        // Calculate the gyro value into actual degrees per second
                                        gx = (float)gyroCount[0]*gRes;  // get actual gyro value, this 
depends on scale being set
                                        gy = (float)gyroCount[1]*gRes;
                                        gz = (float)gyroCount[2]*gRes;

                                        ioctl_ak8963(fd);
                                        readMagData(fd,magCount);  // Read the x/y/z adc values
                                        getMres();
                                        magbias[0] = +470.0;  // User environmental x-axis correction 
in milliGauss, should be automatically calculated
                                        magbias[1] = +120.0;  // User environmental x-axis correction 
in milliGauss
                                        magbias[2] = +125.0;  // User environmental x-axis correction 
in milliGauss

                                        // Calculate the magnetometer values in milliGauss
                                        // Include factory calibration per data sheet and user 
environmental corrections
                                        mx = (float)magCount[0]*mRes*magCalibration[0] -
```

```
magbias[0];  // get actual magnetometer value, this depends on scale being set
                        my = (float)magCount[1]*mRes*magCalibration[1] -
magbias[1];
                        mz = (float)magCount[2]*mRes*magCalibration[2] -
magbias[2];
                }

                Now = micros(ptr,start);
                deltat = ((Now - lastUpdate)/1000000.0f); // set integration time by time
elapsed since last filter update
                lastUpdate = Now;

                sum += deltat; // sum for averaging filter update rate
                sumCount++;

                // Sensors x (y)-axis of the accelerometer is aligned with the y (x)-axis of the
magnetometer;
                // the magnetometer z-axis (+ down) is opposite to z-axis (+ up) of
accelerometer and gyro!
                // We have to make some allowance for this orientationmismatch in feeding the
output to the quaternion filter.
                // For the MPU-9250, we have chosen a magnetic rotation that keeps the sensor
forward along the x-axis just like
                // in the LSM9DS0 sensor. This rotation can be modified to allow any
convenient orientation convention.
                // This is ok by aircraft orientation standards!
                // Pass gyro rate as rad/s
                //  MadgwickQuaternionUpdate(ax, ay, az, gx*PI/180.0f, gy*PI/180.0f,
gz*PI/180.0f,  my,  mx, mz);
                MahonyQuaternionUpdate(ax, ay, az, gx*M_PI/180.0f, gy*M_PI/180.0f,
gz*M_PI/180.0f, my, mx, mz);

                        ioctl_mpu9250(fd);
                if (!AHRS) {
                        delt_t = millis(ptr,start) - count;
                        if(delt_t > 500) {

                                if(SerialDebug) {
                                        // Print acceleration values
in milligs!
                                        printf("X-acceleration: %f
mg\n",1000*ax);
                                        printf("Y-acceleration: %f
mg\n",1000*ay);
                                        printf("Z-acceleration: %f
mg\n",1000*az);

                                        // Print gyro values in
degree/sec
                                        printf("X-gyro rate:%f
degrees/sec\n",gx);
                                        printf("Y-gyro rate:%f
degrees/sec\n",gy);
                                        printf("Z-gyro rate:%f
degrees/sec\n",gz);

                                        // Print mag values in
degree/sec
                                        printf("X-mag field:%f
```

```
mG\n",mx);
                                                                        printf("Y-mag field:%f
mG\n",my);
                                                                        printf("Z-mag field:%f
mG\n",mz);

                                                                        tempCount =
readTempData(fd);  // Read the adc values
                                                                        temperature = ((float)
tempCount) / 333.87 + 21.0; // Temperature in degrees Centigrade
                                                                        // Print temperature in
degrees Centigrade
                                                                        printf("Temperature is %f
degrees C \n",temperature); // Print T values to tenths of s degree C
                                                         }


                                                         count = millis(ptr,start);
                                           }
                             }
                             else {

                                           // Serial print and/or display at 0.5 s rate independent of data
rates
                                           delt_t = millis(ptr,start) - count;
                                           if (delt_t > 500) { // update LCD once per half-second
independent of read rate

                                                         if(SerialDebug) {
                                                                        printf("ax = %f \t",
(int)1000*ax);
                                                                        printf("ay = %f \t",
(int)1000*ay);
                                                                        printf("az = %f mg\n",
(int)1000*az);
                                                                        printf("gx =%f\t ",gx);
                                                                        printf("gy =%f\t ",gy);
                                                                        printf("gz =%f mg \n ",gz);
                                                                        printf("mx =%d \t",(int)mx);
                                                                        printf("my =%d \t",(int)my);
                                                                        printf("mz =%d mG\n",
(int)mz);

                                                                        printf("q0 =%f\t",q[0]);
                                                                        printf("qx =%f\t",q[1]);
                                                                        printf("qy =%f\t",q[2]);
                                                                        printf("qz =%f\t",q[3]);
                                                         }

                                                         // Define output variables from updated
quaternion---these are Tait-Bryan angles, commonly used in aircraft orientation.
                                                         // In this coordinate system, the positive z-
axis is down toward Earth.
                                                         // Yaw is the angle between Sensor x-axis and
Earth magnetic North (or true North if corrected for local declination, looking down on the sensor positive yaw
is counterclockwise.
                                                         // Pitch is angle between sensor x-axis and
Earth ground plane, toward the Earth is positive, up toward the sky is negative.
                                                         // Roll is angle between sensor y-axis and
```

```
Earth ground plane, y-axis up is positive roll.
                                        // These arise from the definition of the
homogeneous rotation matrix constructed from quaternions.
                                        // Tait-Bryan angles as well as Euler angles
are non-commutative; that is, the get the correct orientation the rotations must be
                                        // applied in the correct order which for this
configuration is yaw, pitch, and then roll.
                                        // For more see
http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles which has additional links.
                                        yaw   = atan2(2.0f * (q[1] * q[2] + q[0] *
q[3]), q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3]);
                                        pitch = -asin(2.0f * (q[1] * q[3] - q[0] *
q[2]));
                                        roll  = atan2(2.0f * (q[0] * q[1] + q[2] *
q[3]), q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3]);
                                        pitch *= 180.0f / M_PI;
                                        yaw   *= 180.0f / M_PI;
                                        yaw   +=8.23;//in Seoul//-= 13.8; //
Declination at Danville, California is 13 degrees 48 minutes and 47 seconds on 2014-04-04
                                        roll  *= 180.0f / M_PI;

                                        if(SerialDebug) {
                                                printf("Yaw, Pitch, Roll:%f\t
%f\t%f\n",yaw,pitch,roll);//pitch -5 = offset

                                                printf("rate = %f Hz\n",
(float)sumCount/sum);
                                        }

                                        // With these settings the filter is updating at a
~145 Hz rate using the Madgwick scheme and
                                        // >200 Hz using the Mahony scheme even
though the display refreshes at only 2 Hz.
                                        // The filter update rate is determined mostly
by the mathematical steps in the respective algorithms,
                                        // the processor speed (8 MHz for the 3.3V
Pro Mini), and the magnetometer ODR:
                                        // an ODR of 10 Hz for the magnetometer
produce the above rates, maximum magnetometer ODR of 100 Hz produces
                                        // filter update rates of 36 - 145 and ~38 Hz
for the Madgwick and Mahony schemes, respectively.
                                        // This is presumably because the
magnetometer read takes longer than the gyro or accelerometer reads.
                                        // This filter update rate should be fast
enough to maintain accurate platform orientation for
                                        // stabilization control of a fast-moving robot
or quadcopter. Compare to the update rate of 200 Hz
                                        // produced by the on-board Digital Motion
Processor of Invensense's MPU6050 6 DoF and MPU9150 9DoF sensors.
                                        // The 3.3 V 8 MHz Pro Mini is doing pretty
well!

                                        count = millis(ptr,start);
                                        sumCount = 0;
                                        sum = 0;
                                }
                        }

                }
```

```
}



/*********************************************************************************/
/* when you use kalmanfilter use this function
*
*
*/
/********************************************************************************/

// Implementation of Sebastian Madgwick's "...efficient orientation filter for... inertial/magnetic sensor arrays"
// (see http://www.x-io.co.uk/category/open-source/ for examples and more details)
// which fuses acceleration, rotation rate, and magnetic moments to produce a quaternion-based estimate of
absolute
// device orientation -- which can be converted to yaw, pitch, and roll. Useful for stabilizing quadcopters, etc.
// The performance of the orientation filter is at least as good as conventional Kalman-based filtering algorithms
// but is much less computationally intensive---it can be performed on a 3.3 V Pro Mini operating at 8 MHz!
    void MadgwickQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my,
float mz)
    {
        float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for readability
        float norm;
        float hx, hy, _2bx, _2bz;
        float s1, s2, s3, s4;
        float qDot1, qDot2, qDot3, qDot4;

        // Auxiliary variables to avoid repeated arithmetic
        float _2q1mx;
        float _2q1my;
        float _2q1mz;
        float _2q2mx;
        float _4bx;
        float _4bz;
        float _2q1 = 2.0f * q1;
        float _2q2 = 2.0f * q2;
        float _2q3 = 2.0f * q3;
        float _2q4 = 2.0f * q4;
        float _2q1q3 = 2.0f * q1 * q3;
        float _2q3q4 = 2.0f * q3 * q4;
        float q1q1 = q1 * q1;
        float q1q2 = q1 * q2;
        float q1q3 = q1 * q3;
        float q1q4 = q1 * q4;
        float q2q2 = q2 * q2;
        float q2q3 = q2 * q3;
        float q2q4 = q2 * q4;
        float q3q3 = q3 * q3;
        float q3q4 = q3 * q4;
        float q4q4 = q4 * q4;

                        beta = sqrt(3.0f / 4.0f) * GyroMeasError;   // compute beta
        // Normalise accelerometer measurement
        norm = sqrtf(ax * ax + ay * ay + az * az);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        ax *= norm;
        ay *= norm;
        az *= norm;
```

```
// Normalise magnetometer measurement
norm = sqrtf(mx * mx + my * my + mz * mz);
if (norm == 0.0f) return; // handle NaN
norm = 1.0f/norm;
mx *= norm;
my *= norm;
mz *= norm;

// Reference direction of Earth's magnetic field
_2q1mx = 2.0f * q1 * mx;
_2q1my = 2.0f * q1 * my;
_2q1mz = 2.0f * q1 * mz;
_2q2mx = 2.0f * q2 * mx;
hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 + _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;
hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 * mz * q4 - my * q4q4;
_2bx = sqrtf(hx * hx + hy * hy);
_2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 - mz * q3q3 + mz * q4q4;
_4bx = 2.0f * _2bx;
_4bz = 2.0f * _2bz;

// Gradient decent algorithm corrective step
s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4 + _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz * q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
norm = sqrtf(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    // normalise step magnitude
norm = 1.0f/norm;
s1 *= norm;
s2 *= norm;
s3 *= norm;
s4 *= norm;

// Compute rate of change of quaternion
qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

// Integrate to yield quaternion
q1 += qDot1 * deltat;
q2 += qDot2 * deltat;
q3 += qDot3 * deltat;
q4 += qDot4 * deltat;
norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise quaternion
norm = 1.0f/norm;
```

```
        q[0] = q1 * norm;
        q[1] = q2 * norm;
        q[2] = q3 * norm;
        q[3] = q4 * norm;

    }



 // Similar to Madgwick scheme but uses proportional and integral filtering on the error between estimated
reference vectors and
 // measured ones.
        void MahonyQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my,
float mz){
        float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for readability
        float norm;
        float hx, hy, bx, bz;
        float vx, vy, vz, wx, wy, wz;
        float ex, ey, ez;
        float pa, pb, pc;

        // Auxiliary variables to avoid repeated arithmetic
        float q1q1 = q1 * q1;
        float q1q2 = q1 * q2;
        float q1q3 = q1 * q3;
        float q1q4 = q1 * q4;
        float q2q2 = q2 * q2;
        float q2q3 = q2 * q3;
        float q2q4 = q2 * q4;
        float q3q3 = q3 * q3;
        float q3q4 = q3 * q4;
        float q4q4 = q4 * q4;

        // Normalise accelerometer measurement
        norm = sqrtf(ax * ax + ay * ay + az * az);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f / norm;        // use reciprocal for division
        ax *= norm;
        ay *= norm;
        az *= norm;

        // Normalise magnetometer measurement
        norm = sqrtf(mx * mx + my * my + mz * mz);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f / norm;        // use reciprocal for division
        mx *= norm;
        my *= norm;
        mz *= norm;

        // Reference direction of Earth's magnetic field
        hx = 2.0f * mx * (0.5f - q3q3 - q4q4) + 2.0f * my * (q2q3 - q1q4) + 2.0f * mz * (q2q4 + q1q3);
        hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f * my * (0.5f - q2q2 - q4q4) + 2.0f * mz * (q3q4 - q1q2);
        bx = sqrtf((hx * hx) + (hy * hy));
        bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f * my * (q3q4 + q1q2) + 2.0f * mz * (0.5f - q2q2 - q3q3);

        // Estimated direction of gravity and magnetic field
        vx = 2.0f * (q2q4 - q1q3);
        vy = 2.0f * (q1q2 + q3q4);
        vz = q1q1 - q2q2 - q3q3 + q4q4;
```

```
        wx = 2.0f * bx * (0.5f - q3q3 - q4q4) + 2.0f * bz * (q2q4 - q1q3);
        wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f * bz * (q1q2 + q3q4);
        wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f * bz * (0.5f - q2q2 - q3q3);

        // Error is cross product between estimated direction and measured direction of gravity
        ex = (ay * vz - az * vy) + (my * wz - mz * wy);
        ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
        ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
        if (Ki > 0.0f)
        {
            eInt[0] += ex;      // accumulate integral error
            eInt[1] += ey;
            eInt[2] += ez;
        }
        else
        {
            eInt[0] = 0.0f;     // prevent integral wind up
            eInt[1] = 0.0f;
            eInt[2] = 0.0f;
        }

        // Apply feedback terms
        gx = gx + Kp * ex + Ki * eInt[0];
        gy = gy + Kp * ey + Ki * eInt[1];
        gz = gz + Kp * ez + Ki * eInt[2];

        // Integrate rate of change of quaternion
        pa = q2;
        pb = q3;
        pc = q4;
        q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz) * (0.5 * deltat);
        q2 = pa + (q1 * gx + pb * gz - pc * gy) * (0.5 * deltat);
        q3 = pb + (q1 * gy - pa * gz + pc * gx) * (0.5 * deltat);
        q4 = pc + (q1 * gz + pa * gy - pb * gx) * (0.5 * deltat);

        // Normalise quaternion
        norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
        norm = 1.0f / norm;
        q[0] = q1 * norm;
        q[1] = q2 * norm;
        q[2] = q3 * norm;
        q[3] = q4 * norm;

}

//===============================================================================================================================
//====== Set of useful function to access acceleration. gyroscope, magnetometer, and temperature data
//===============================================================================================================================

void getMres() {
                switch (Mscale)
                {
                                // Possible magnetometer scales (and their register bit settings) are:
                                // 14 bit resolution (0) and 16 bit resolution (1)
                                case MFS_14BITS:
                                                mRes = 10.*4912./8190.; // Proper scale to return milliGauss
                                                break;
```

```
			case MFS_16BITS:
				mRes = 10.*4912./32760.0; // Proper scale to return 
milliGauss
				break;
	}
}

void getGres() {
	switch (Gscale)
	{
			// Possible gyro scales (and their register bit settings) are:
			// 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS  (11).
			// Here's a bit of an algorith to calculate DPS/(ADC tick) based on that 2-bit 
value:
			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 getAres() {
	switch (Ascale)
	{
			// Possible accelerometer scales (and their register bit settings) are:
			// 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
			// Here's a bit of an algorith to calculate DPS/(ADC tick) based on that 2-bit 
value:
			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 readAccelData(int fd,int16_t * destination)
{
	uint8_t rawData[6];  // x/y/z accel register data stored here
	readBytes(fd, ACCEL_XOUT_H, 6, &rawData[0]);  // Read the six raw data registers into data 
array
	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 readGyroData(int fd,int16_t * destination)
{
            uint8_t rawData[6];  // x/y/z gyro register data stored here
            readBytes(fd, GYRO_XOUT_H, 6, &rawData[0]);  // Read the six raw data registers
sequentially into data array
            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];  // x/y/z gyro register data, ST2 register stored here, must read ST2 at end
of data acquisition
            if(readByte(fd, AK8963_ST1) & 0x01) { // wait for magnetometer data ready bit to be set
                        readBytes(fd, AK8963_XOUT_L, 7, &rawData[0]);  // Read the six raw data
and ST2 registers sequentially into data array
                        uint8_t c = rawData[6]; // End data read by reading ST2 register
                        if(!(c & 0x08)) { // Check if magnetic sensor overflow set, if not then report
data
                                    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] ;
                        }
            }
}

int16_t readTempData(int fd)
{
            /* fd = ioctl mpu9250*/
            uint8_t rawData[2];  // x/y/z gyro register data stored here
            readBytes(fd, TEMP_OUT_H, 2, &rawData[0]);  // Read the two raw data registers sequentially
into data array
            return ((int16_t)rawData[0] << 8) | rawData[1] ;  // Turn the MSB and LSB into a 16-bit value
}

void initAK8963(int fd,float * destination)
{
            // First extract the factory calibration for each magnetometer axis
            uint8_t rawData[3];  // x/y/z gyro calibration data stored here
            writeByte(fd, AK8963_CNTL, 0x00); // Power down magnetometer
            usleep(10000);
            writeByte(fd, AK8963_CNTL, 0x0F); // Enter Fuse ROM access mode
            usleep(10000);
            readBytes(fd, AK8963_ASAX, 3, &rawData[0]);  // Read the x-, y-, and z-axis calibration
values
            destination[0] =  (float)(rawData[0] - 128)/256. + 1.;   // Return x-axis sensitivity adjustment
values, etc.
            destination[1] =  (float)(rawData[1] - 128)/256. + 1.;
            destination[2] =  (float)(rawData[2] - 128)/256. + 1.;
            writeByte(fd, AK8963_CNTL, 0x00); // Power down magnetometer
```

```
		usleep(10000);
		// Configure the magnetometer for continuous read and highest resolution
		// set Mscale bit 4 to 1 (0) to enable 16 (14) bit resolution in CNTL register,
		// and enable continuous mode data acquisition Mmode (bits [3:0]), 0010 for 8 Hz and 0110 for 
100 Hz sample rates
		writeByte(fd, AK8963_CNTL, Mscale << 4 | Mmode); // Set magnetometer data resolution and 
sample ODR
		usleep(10000);
}


void initMPU9250(int fd)
{
		// wake up device
		writeByte(fd, PWR_MGMT_1, 0x00); // Clear sleep mode bit (6), enable all sensors
		usleep(100000); // Wait for all registers to reset

		// get stable time source
		writeByte(fd, PWR_MGMT_1, 0x01);  // Auto select clock source to be PLL gyroscope 
reference if ready else
		usleep(200000);

		// Configure Gyro and Thermometer
		// Disable FSYNC and set thermometer and gyro bandwidth to 41 and 42 Hz, respectively;
		// minimum delay time for this setting is 5.9 ms, which means sensor fusion update rates cannot
		// be higher than 1 / 0.0059 = 170 Hz
		// DLPF_CFG = bits 2:0 = 011; this limits the sample rate to 1000 Hz for both
		// With the MPU9250, it is possible to get gyro sample rates of 32 kHz (!), 8 kHz, or 1 kHz
		writeByte(fd, CONFIG, 0x03);

		// Set sample rate = gyroscope output rate/(1 + SMPLRT_DIV)
		writeByte(fd, SMPLRT_DIV, 0x04);  // Use a 200 Hz rate; a rate consistent with the filter 
update rate
		// determined inset in CONFIG above

		// Set gyroscope full scale range
		// Range selects FS_SEL and GFS_SEL are 0 - 3, so 2-bit values are left-shifted into positions 
4:3
		uint8_t c = readByte(fd, GYRO_CONFIG); // get current GYRO_CONFIG register value
		// c = c & ~0xE0; // Clear self-test bits [7:5]
		c = c & ~0x03; // Clear Fchoice bits [1:0]
		c = c & ~0x18; // Clear GFS bits [4:3]
		c = c | Gscale << 3; // Set full scale range for the gyro
		// c =| 0x00; // Set Fchoice for the gyro to 11 by writing its inverse to bits 1:0 of 
GYRO_CONFIG
		writeByte(fd, GYRO_CONFIG, c ); // Write new GYRO_CONFIG value to register

		// Set accelerometer full-scale range configuration
		c = readByte(fd, ACCEL_CONFIG); // get current ACCEL_CONFIG register value
		// c = c & ~0xE0; // Clear self-test bits [7:5]
		c = c & ~0x18;  // Clear AFS bits [4:3]
		c = c | Ascale << 3; // Set full scale range for the accelerometer
		writeByte(fd, ACCEL_CONFIG, c); // Write new ACCEL_CONFIG register value

		// Set accelerometer sample rate configuration
		// It is possible to get a 4 kHz sample rate from the accelerometer by choosing 1 for
		// accel_fchoice_b bit [3]; in this case the bandwidth is 1.13 kHz
		c = readByte(fd, ACCEL_CONFIG2); // get current ACCEL_CONFIG2 register value
		c = c & ~0x0F; // Clear accel_fchoice_b (bit 3) and A_DLPFG (bits [2:0])
```

```
        c = c | 0x03;  // Set accelerometer rate to 1 kHz and bandwidth to 41 Hz
        writeByte(fd, ACCEL_CONFIG2, c); // Write new ACCEL_CONFIG2 register value
        // The accelerometer, gyro, and thermometer are set to 1 kHz sample rates,
        // but all these rates are further reduced by a factor of 5 to 200 Hz because of the SMPLRT_DIV
setting

        // Configure Interrupts and Bypass Enable
        // Set interrupt pin active high, push-pull, hold interrupt pin level HIGH until interrupt cleared,
        // clear on read of INT_STATUS, and enable I2C_BYPASS_EN so additional chips
        // can join the I2C bus and all can be controlled by the Arduino as master
        writeByte(fd, INT_PIN_CFG, 0x22);
        writeByte(fd, INT_ENABLE, 0x01);  // Enable data ready (bit 0) interrupt
        usleep(100000);
}


// Function which accumulates gyro and accelerometer data after device initialization. It calculates the average
// of the at-rest readings and then loads the resulting offsets into accelerometer and gyro bias registers.
void calibrateMPU9250(int fd,float * dest1, float * dest2)
{
        uint8_t data[12]; // data array to hold accelerometer and gyro x, y, z, data
        uint16_t ii, packet_count, fifo_count;
        int32_t gyro_bias[3]  = {0, 0, 0}, accel_bias[3] = {0, 0, 0};

        // reset device
        writeByte(fd, PWR_MGMT_1, 0x80); // Write a one to bit 7 reset bit; toggle reset device
        usleep(100000);

        // get stable time source; Auto select clock source to be PLL gyroscope reference if ready
        // else use the internal oscillator, bits 2:0 = 001
        writeByte(fd, PWR_MGMT_1, 0x01);
        writeByte(fd, PWR_MGMT_2, 0x00);
        usleep(200000);

        // 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(15000);

        // 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

		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]);

		// Output scaled gyro biases for display in the main program
```

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

// Construct the accelerometer biases for push to the hardware accelerometer bias registers. These registers contain
// factory trim values which must be added to the calculated accelerometer biases; on boot up these registers will hold
// non-zero values. In addition, bit 0 of the lower byte must be preserved since it is used for temperature
// compensation calculations. Accelerometer bias registers expect bias input as 2048 LSB per g, so that
// the accelerometer biases calculated above must be divided by 8.

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

// Apparently this is not working for the acceleration biases in the MPU-9250
// Are we handling the temperature correction bit properly?
// Push accelerometer biases to hardware 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;
}


// Accelerometer and gyroscope self test; check calibration wrt factory settings
void MPU9250SelfTest(int fd,float * destination) // Should return percent deviation from factory trim values, +/-
14 or less deviation is a pass
{
		uint8_t rawData[6] = {0, 0, 0, 0, 0, 0};
		uint8_t selfTest[6];
		int32_t gAvg[3] = {0}, aAvg[3] = {0}, aSTAvg[3] = {0}, gSTAvg[3] = {0};
		float factoryTrim[6];
		uint8_t FS = 0;
		int ii=0,i=0;

		writeByte(fd, SMPLRT_DIV, 0x00);    // Set gyro sample rate to 1 kHz
		writeByte(fd, CONFIG, 0x02);        // Set gyro sample rate to 1 kHz and DLPF to 92 Hz
		writeByte(fd, GYRO_CONFIG, FS<<3);  // Set full scale range for the gyro to 250 dps
		writeByte(fd, ACCEL_CONFIG2, 0x02); // Set accelerometer rate to 1 kHz and bandwidth to
92 Hz
		writeByte(fd, ACCEL_CONFIG, FS<<3); // Set full scale range for the accelerometer to 2 g

		for(ii = 0; ii < 200; ii++) {  // get average current values of gyro and acclerometer

				readBytes(fd, ACCEL_XOUT_H, 6, &rawData[0]);        // Read the six raw
data registers into data array
				aAvg[0] += (int16_t)(((int16_t)rawData[0] << 8) | rawData[1]) ;  // Turn the
MSB and LSB into a signed 16-bit value
				aAvg[1] += (int16_t)(((int16_t)rawData[2] << 8) | rawData[3]) ;
				aAvg[2] += (int16_t)(((int16_t)rawData[4] << 8) | rawData[5]) ;

				readBytes(fd, GYRO_XOUT_H, 6, &rawData[0]);       // Read the six raw data
registers sequentially into data array
				gAvg[0] += (int16_t)(((int16_t)rawData[0] << 8) | rawData[1]) ;  // Turn the
MSB and LSB into a signed 16-bit value
				gAvg[1] += (int16_t)(((int16_t)rawData[2] << 8) | rawData[3]) ;
				gAvg[2] += (int16_t)(((int16_t)rawData[4] << 8) | rawData[5]) ;
		}

		for ( ii =0; ii < 3; ii++) {  // Get average of 200 values and store as average current readings
				aAvg[ii] /= 200;
				gAvg[ii] /= 200;
		}

		// Configure the accelerometer for self-test
		writeByte(fd, ACCEL_CONFIG, 0xE0); // Enable self test on all three axes and set
accelerometer range to +/- 2 g
		writeByte(fd, GYRO_CONFIG,  0xE0); // Enable self test on all three axes and set gyro range
to +/- 250 degrees/s
		usleep(25000);  // Delay a while to let the device stabilize

		for(ii = 0; ii < 200; ii++) {  // get average self-test values of gyro and acclerometer

				readBytes(fd, ACCEL_XOUT_H, 6, &rawData[0]);  // Read the six raw data
```

```
registers into data array
                        aSTAvg[0] += (int16_t)(((int16_t)rawData[0] << 8) | rawData[1]) ;  // Turn the
MSB and LSB into a signed 16-bit value
                        aSTAvg[1] += (int16_t)(((int16_t)rawData[2] << 8) | rawData[3]) ;
                        aSTAvg[2] += (int16_t)(((int16_t)rawData[4] << 8) | rawData[5]) ;

                        readBytes(fd, GYRO_XOUT_H, 6, &rawData[0]);  // Read the six raw data
registers sequentially into data array
                        gSTAvg[0] += (int16_t)(((int16_t)rawData[0] << 8) | rawData[1]) ;  // Turn the
MSB and LSB into a signed 16-bit value
                        gSTAvg[1] += (int16_t)(((int16_t)rawData[2] << 8) | rawData[3]) ;
                        gSTAvg[2] += (int16_t)(((int16_t)rawData[4] << 8) | rawData[5]) ;
            }

            for (ii =0; ii < 3; ii++) {  // Get average of 200 values and store as average self-test readings
                        aSTAvg[ii] /= 200;
                        gSTAvg[ii] /= 200;
            }

            // Configure the gyro and accelerometer for normal operation
            writeByte(fd, ACCEL_CONFIG, 0x00);
            writeByte(fd, GYRO_CONFIG,  0x00);
            usleep(25000);  // Delay a while to let the device stabilize

            // Retrieve accelerometer and gyro factory Self-Test Code from USR_Reg
            selfTest[0] = readByte(fd, SELF_TEST_X_ACCEL); // X-axis accel self-test results
            selfTest[1] = readByte(fd, SELF_TEST_Y_ACCEL); // Y-axis accel self-test results
            selfTest[2] = readByte(fd, SELF_TEST_Z_ACCEL); // Z-axis accel self-test results
            selfTest[3] = readByte(fd, SELF_TEST_X_GYRO);  // X-axis gyro self-test results
            selfTest[4] = readByte(fd, SELF_TEST_Y_GYRO);  // Y-axis gyro self-test results
            selfTest[5] = readByte(fd, SELF_TEST_Z_GYRO);  // Z-axis gyro self-test results

            // Retrieve factory self-test value from self-test code reads
            factoryTrim[0] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[0] - 1.0) )); // FT[Xa]
factory trim calculation
            factoryTrim[1] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[1] - 1.0) )); // FT[Ya]
factory trim calculation
            factoryTrim[2] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[2] - 1.0) )); // FT[Za]
factory trim calculation
            factoryTrim[3] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[3] - 1.0) )); // FT[Xg]
factory trim calculation
            factoryTrim[4] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[4] - 1.0) )); // FT[Yg]
factory trim calculation
            factoryTrim[5] = (float)(2620/1<<FS)*(powf( 1.01 , ((float)selfTest[5] - 1.0) )); // FT[Zg]
factory trim calculation

            // Report results as a ratio of (STR - FT)/FT; the change from Factory Trim of the Self-Test
Response
            // To get percent, must multiply by 100
            for (i = 0; i < 3; i++) {
                        destination[i]   = 100.0*((float)(aSTAvg[i] - aAvg[i]))/factoryTrim[i] - 100.;   //
Report percent differences
                        destination[i+3] = 100.0*((float)(gSTAvg[i] - gAvg[i]))/factoryTrim[i+3] -
100.; // Report percent differences
            }

}
```

```
// Wire.h read and write protocols
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");
	}
```

```
}

uint32_t millis(void *ptr,uint32_t start)
{
   uint32_t end =0,result =0;

   end = *((unsigned *)(ptr + TIME_DATA_OFFSET));

   result = end - start;

   if(result < 0)
   {
      end = end + 4294967295;
      result = end - start;
   }

   return result/100000;

}

uint32_t micros(void *ptr,uint32_t start)
{
   uint32_t end = 0,result =0;

   end = *((unsigned *)(ptr + TIME_DATA_OFFSET));

   result = end - start;

   if(result < 0)
   {
      end = end + 4294967295;
      result = end - start;
   }

   return result/100;
}
```

알아보기 힘드니 파일을 같이 업로드 해놓았습니다.

Math.h 를 사용하였기 때문에 Makefile 에서 -lm 옵션을 추가해주어야한다.
또한 자동부트 (자동실행) 을 위해 Makefile 에 2 줄을 추가하여야한다.

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/components/apps/device_driver$ vi Makefile
```

Makefile

```
ifndef PETALINUX
$(error "Error: PETALINUX environment variable not set.  Change to the root of your PetaLinux install, and source the settings.sh file")
endif

include apps.common.mk

APP = device_driver
LDLIBS += -lm
```

```
# Add any other object files to this list below
APP_OBJS = device_driver.o

all: build install

build: $(APP)

$(APP): $(APP_OBJS)
	$(CC) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)

clean:
	-rm -f $(APP) *.elf *.gdb *.o

.PHONY: install image

install: $(APP)
	$(TARGETINST) -d $(APP) /bin/$(APP)
	$(TARGETINST) -d -p 0755 device_driver /etc/init.d/device_driver
	$(TARGETINST) -s /etc/init.d/device_driver /etc/rc5.d/S99device_driver

%.o: %.c
	$(CC) -c $(CFLAGS) -o $@ $<

help:
	@echo ""
	@echo "Quick reference for various supported build targets for $(INSTANCE)."
	@echo "----------------------------------------------------"
	@echo "  clean              clean out build objects"
	@echo "  all                build $(INSTANCE) and install to rootfs host copy"
	@echo "  build              build subsystem"
	@echo "  install            install built objects to rootfs host copy"
```

LDLIBS += -lm = -lm 옵션을 추가.

```
$(TARGETINST) -d -p 0755 device_driver /etc/init.d/device_driver
$(TARGETINST) -s /etc/init.d/device_driver /etc/rc5.d/S99device_driver
```

자동 로그인과 자동실행 명령어

원래의 Makefile

siyun@siyun-CR62-6M: ~/vivado_workspace/mpu9250_controller/software/components/apps/device_driver

```makefile
ifndef PETALINUX
$(error "Error: PETALINUX environment variable not set.  Change to the root of your PetaLinux install, and source the settings.sh file")
endif

include apps.common.mk

APP = device_driver

# Add any other object files to this list below
APP_OBJS = device_driver.o

all: build install

build: $(APP)

$(APP): $(APP_OBJS)
	$(CC) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)

clean:
	-rm -f $(APP) *.elf *.gdb *.o

.PHONY: install image

install: $(APP)
	$(TARGETINST) -d $(APP) /bin/$(APP)

%.o: %.c
	$(CC) -c $(CFLAGS) -o $@ $<

help:
	@echo ""
	@echo "Quick reference for various supported build targets for $(INSTANCE)."
	@echo "----------------------------------------------------"
	@echo "  clean                  clean out build objects"
	@echo "  all                    build $(INSTANCE) and install to rootfs host copy"
	@echo "  build                  build subsystem"
	@echo "  install                install built objects to rootfs host copy"
```

```makefile
ifndef PETALINUX
$(error "Error: PETALINUX environment variable not set.  Change to the root of your PetaLinux install, and source the settings.sh file")
endif

include apps.common.mk

APP = device_driver
LDLIBS += -lm


# Add any other object files to this list below
APP_OBJS = device_driver.o

all: build install

build: $(APP)

$(APP): $(APP_OBJS)
	$(CC) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)

clean:
	-rm -f $(APP) *.elf *.gdb *.o

.PHONY: install image

install: $(APP)
	$(TARGETINST) -d $(APP) /bin/$(APP)
	$(TARGETINST) -d -p 0755 device_driver /etc/init.d/device_driver
	$(TARGETINST) -s /etc/init.d/device_driver /etc/rc5.d/S99device_driver

%.o: %.c
	$(CC) -c $(CFLAGS) -o $@ $<

help:
	@echo ""
	@echo "Quick reference for various supported build targets for $(INSTANCE)."
	@echo "----------------------------------------------------"
	@echo "  clean                  clean out build objects"
	@echo "  all                    build $(INSTANCE) and install to rootfs host copy"
	@echo "  build                  build subsystem"
	@echo "  install                install built objects to rootfs host copy"
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ cd subsystems/linux/configs/devic
e-tree/
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/subsystems/linux/configs/device-tr
ee$ ls
pcw.dtsi  pl.dtsi  skeleton.dtsi  system-conf.dtsi  system-top.dts  zynq-7000.dtsi
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/subsystems/linux/configs/device-tr
ee$ vi system-top.dts
```

커스텀아이피의 내용을 디바이스트리에 추가해야한다. (장치파일)

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&clkc {
ps-clk-frequency = <50000000>;
};
&flash0{
compatible = "s25fl128s1";
};
&usb0{
dr_mode = "otg";
};
&gem0{
phy-handle = <&phy0>;
mdio{
#address-cells = <1>;#size-cells = <0>;
phy0: phy@1{
compatible = "realtek,RTL8211E";
device_type = "ethernet-phy";
reg = <1>;
};
};
};
&My_Time_Core_0 {
compatible = "generic-uio";
};
```

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&clkc {
        ps-clk-frequency = <50000000>;
};
&flash0{
        compatible = "s25fl128s1";
};
&usb0{
        dr_mode = "otg";
};
&gem0{
        phy-handle = <&phy0>;
        mdio{
#address-cells = <1>;#size-cells = <0>;
phy0: phy@1{
                compatible = "realtek,RTL8211E";
                device_type = "ethernet-phy";
                reg = <1>;
      };
        };
};
&My_Time_Core_0 {
        compatible = "generic-uio";
};
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config -c kernel
INFO: Checking component...
INFO: Config linux/kernel
[INFO ] config linux/kernel


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config -c rootfs
INFO: Checking component...
INFO: Config linux/rootfs
[INFO ] config linux/rootfs


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software$ petalinux-config
```

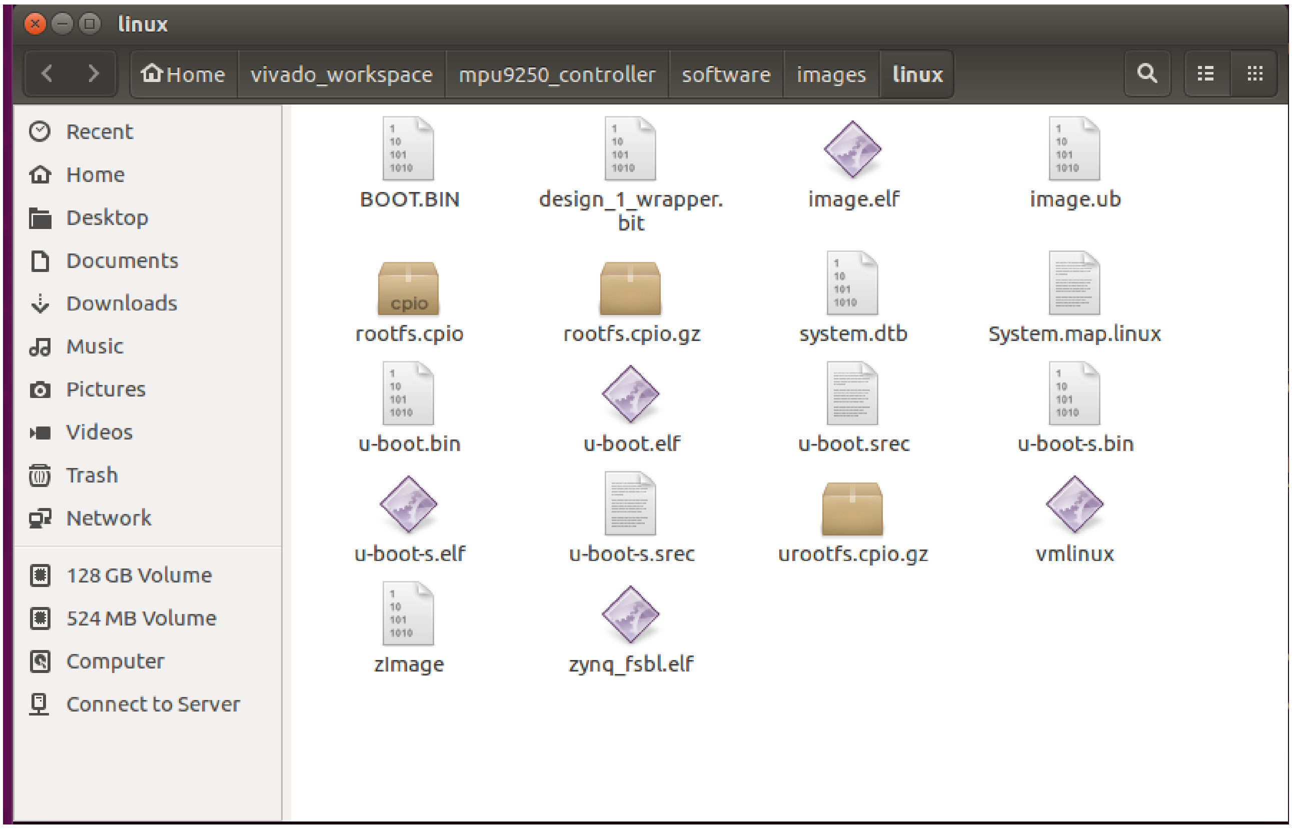
```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/images/linux$ petalinux-build
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/images/linux$ source ~/xilinx/Vivado/2017.4/settings64.sh
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/images/linux$ source ~/xilinx/SDK/2017.4/settings64.sh
```

Petalinux-package –boot –fsbl zynq_fsbl.elf –fpga ../../../hardware/mpu92500_controller/mpu9250_controller.runs/impl_l/design_1wrapper.bit –force

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/images/linux$ petalinux-package --boot --fsbl zynq_fsbl.elf --fpga ../../../hardware/mpu9250_controller/mpu9250_controller.runs/impl_1/design_1_wrapper.bit --force
```

```
siyun@siyun-CR62-6M:~/vivado_workspace/mpu9250_controller/software/images/linux$ nautilus ./
```



```
ax = -1.098633   ay = 1.525879    az = 908.081055 mg
gx =-0.030518     gy =0.068665     gz =-0.083923 mg
 mx =-475        my =56  mz =340 mG
q0 =0.740160     qx =0.000460     qy =0.000284     qz =0.672431    Yaw, Pitch, Roll:92.739967      -0.011341       0.060911
rate = 5428.313965 Hz
ax = -0.976562   ay = -0.427246   az = 911.499023 mg
gx =-0.045776     gy =0.083923     gz =0.015259 mg
 mx =-466        my =51  mz =352 mG
q0 =0.744702     qx =0.000300     qy =0.000203     qz =0.667397    Yaw, Pitch, Roll:91.962921      -0.005666       0.041160
rate = 5412.404297 Hz
ax = -0.915527   ay = 1.953125    az = 911.865234 mg
gx =0.076294      gy =0.068665     gz =-0.045776 mg
 mx =-484        my =55  mz =341 mG
q0 =0.744302     qx =-0.000018    qy =0.000357     qz =0.667843    Yaw, Pitch, Roll:92.031631      0.031842        0.025750
rate = 5429.653320 Hz
ax = -1.464844   ay = -0.610352   az = 907.470703 mg
gx =0.030518      gy =0.106812     gz =-0.038147 mg
 mx =-484        my =33  mz =341 mG
q0 =0.740857     qx =-0.001120    qy =0.001226     qz =0.671661    Yaw, Pitch, Roll:92.620972      0.190287        -0.000715
rate = 5369.720215 Hz
ax = -1.647949   ay = -0.122070   az = 910.461426 mg
gx =0.000000      gy =0.068665     gz =-0.053406 mg
 mx =-478        my =49  mz =333 mG
q0 =0.738965     qx =-0.001209    qy =0.001351     qz =0.673741    Yaw, Pitch, Roll:92.943115      0.207776        0.001956
rate = 5374.665527 Hz
ax = -2.563477   ay = 0.183105    az = 914.672852 mg
gx =-0.030518     gy =0.114441     gz =0.000000 mg
 mx =-462        my =51  mz =345 mG
q0 =0.739660     qx =-0.000665    qy =0.000681     qz =0.672980    Yaw, Pitch, Roll:92.824959      0.109056        -0.003837
rate = 5475.780273 Hz
ax = 0.732422    ay = -1.464844   az = 914.062500 mg
gx =-0.007629     gy =0.015259     gz =0.000000 mg
 mx =-493        my =53  mz =333 mG
q0 =0.745736     qx =0.000180     qy =0.000442     qz =0.666242    Yaw, Pitch, Roll:91.785324      0.024001        0.049095
rate = 5400.741699 Hz
ax = -1.098633   ay = 1.220703    az = 911.865234 mg
gx =0.000000      gy =0.091553     gz =-0.007629 mg
 mx =-484        my =40  mz =341 mG
q0 =0.743843     qx =0.000064     qy =0.000649     qz =0.668354    Yaw, Pitch, Roll:92.110390      0.050414        0.055098
rate = 5468.955078 Hz
ax = 1.037598    ay = 0.732422    az = 909.912109 mg
gx =0.122070      gy =0.137329     gz =-0.007629 mg
 mx =-475        my =53  mz =350 mG
q0 =0.744823     qx =0.000529     qy =-0.000118    qz =0.667262    Yaw, Pitch, Roll:91.942162      -0.050487       0.036129
rate = 5341.667480 Hz
ax = -1.159668   ay = -1.831055   az = 909.667969 mg
gx =0.038147      gy =0.045776     gz =-0.053406 mg
 mx =-471        my =53  mz =340 mG
q0 =0.742286     qx =0.001073     qy =-0.000497    qz =0.670083    Yaw, Pitch, Roll:92.376877      -0.124699       0.053131
rate = 5464.869141 Hz
ax = 0.305176    ay = 0.671387    az = 910.705566 mg
gx =0.053406      gy =0.045776     gz =-0.076294 mg
 mx =-473        my =47  mz =348 mG
q0 =0.742000     qx =0.000027     qy =0.000402     qz =0.670400    Yaw, Pitch, Roll:92.425858      0.032097        0.033220
rate = 5396.099121 Hz
ax = 0.061035    ay = -0.366211   az = 909.423828 mg
gx =0.015259      gy =0.068665     gz =-0.076294 mg
 mx =-477        my =55  mz =345 mG
q0 =0.743211     qx =0.000993     qy =-0.000518    qz =0.669056    Yaw, Pitch, Roll:92.218552      -0.120313       0.044866
rate = 5397.894531 Hz
ax = 1.342773    ay = -0.549316   az = 911.743164 mg
gx =-0.007629     gy =0.083923     gz =-0.061035 mg
 mx =-466        my =55  mz =348 mG
q0 =0.743932     qx =0.001441     qy =-0.001077    qz =0.668253    Yaw, Pitch, Roll:92.094894      -0.202196       0.040413
rate = 5421.584961 Hz
ax = -2.197266   ay = -0.427246   az = 913.085938 mg
gx =-0.007629     gy =0.144958     gz =-0.053406 mg
 mx =-487        my =40  mz =362 mG
q0 =0.740402     qx =-0.001690    qy =0.001792     qz =0.672159    Yaw, Pitch, Roll:92.698196      0.282236        -0.005315
rate = 5403.092773 Hz
ax = 1.342773    ay = -1.464844   az = 908.386230 mg
gx =-0.038147     gy =0.114441     gz =0.015259 mg
 mx =-493        my =49  mz =343 mG
q0 =0.742283     qx =-0.000404    qy =0.000741     qz =0.670086    Yaw, Pitch, Roll:92.377388      0.094056        0.022466
rate = 5391.085449 Hz
ax = 1.464844    ay = -0.183105   az = 910.766602 mg
gx =0.106812      gy =0.022888     gz =-0.053406 mg
 mx =-475        my =46  mz =343 mG
q0 =0.742810     qx =-0.001126    qy =0.001328     qz =0.669500    Yaw, Pitch, Roll:92.287109      0.199455        0.006043
rate = 5400.425293 Hz
```