Final Project

Working with the digital accelerometer sensor module (ADXL345) with a Hybrid Hardware-Software System

Introduction

The goal of this project is to bring together the software and hardware elements you have been working so far in previous laboratories and combine them in a hardware/software co-designed implementation. In this project you well obtain measurements from the digital accelerometer sensor module (ADXL345), commonly known as G-sensor, which is connected to the HPS on the DE1SoC. A digital accelerometer is an integrated circuit used to measure the acceleration of an object to which it is attached. The G-sensor in the DE1SoC board is a small, thin, ultralow power assumption 3-axis accelerometer with high-resolution measurement. The measurements from the G-sensor will be processed in the FPGA and converted into the inclination angle of the board. Then the corresponding angle will be displayed in the 7-segment display and also returned to the HPS.

In other words, you will write a C++ program that obtains the measurements from the G-sensor given by X, Y and Z coordinates and passes the X coordinate to the FPGA. The FPGA will implement the necessary hardware circuit to convert the X coordinate obtained by G-sensor in an angle that represents the inclination of the board. When the board is in the regular horizontal position (on a table), the angle is 90 degrees. When the board is inclined in vertical position, the angle is 0 or 180 degrees, depending on the direction of the inclination. This angle will be displayed in the 7-segment display and also returned to the HPS to be displayed in the terminal, too.

Background material

Here are some steps that will guide you through the design of this project:

1. The G-Sensor module uses the I^2C communication protocol to communicate with the HPS System in the DE1-SoC board. Scan through Chapter 21 (I^2C Controller) from the *Cyclone V Hard Processor System Technical Reference Manual* included under "Labs Reference Documents" on Canvas, and linked in the *Appendix*.

Using the base address for devices mounted on the HPS below, and the reading assignment above, compute the offsets of the following I^2C Module 0 (I2C0) registers from the `HPS_BRIDGE_BASE` address.

```
/* Cyclone V HPS devices */
const unsigned int HPS_BRIDGE_BASE      = 0xFF700000;   // base
const unsigned int HPS_BRIDGE_SPAN      = 0x006FFFFF;   // span

/* I2C0 Peripheral Registers */
const unsigned int I2C0_BASE        = 0x00504000;      // I2C0 Base Address
const unsigned int I2C0_CON         = 0x00504000;      // Control Register
const unsigned int I2C0_TAR         = ? ;  // Target Address Register
const unsigned int I2C0_DATA_CMD    = ? ;  // Tx Rx Data and Command Register
const unsigned int I2C0_FS_SCL_HCNT = ? ;  // Fast Spd Clock SCL HCNT Register
const unsigned int I2C0_FS_SCL_LCNT = ? ;  // Fast Spd Clock SCL LCNT Register
```

```
const unsigned int I2C0_CLR_INTR    = ? ;  // Combined and Individual Interrupt
Register
const unsigned int I2C0_ENABLE      = ? ;  // Enable Register
const unsigned int I2C0_TXFLR       = ? ;  // Transmit FIFO Level Register
const unsigned int I2C0_RXFLR       = ? ;  // Receive FIFO Level Register
const unsigned int I2C0_ENABLE_STATUS     = ? ;   // Enable Status Register
```

2. Accelerometers (G-Sensor) are devices that measure acceleration in units of meters per second squared (m/s2) or in G-forces (g). A single G-force is equivalent to 9.8 m/s2. The ADXL345 G-Sensor (included in the DE1-SoC) provides 3-axis acceleration measurements, and we will use it in this project as a tilt sensor. Think of how your phone detects if it's in portrait or landscape orientation.

   Read the ADXL345 Datasheet (Digital Accelerometer) to understand how the device works. Pay attention to register memory mapping and register settings for correct setup and operation. You can skip the section on SPI communication since we are using 12C instead. You can also skip the graphs on performance measurements.
3. Read the DE1-SoC User Manual on the sections talking about the G-Sensor for more insight on how the device is configured on the DE1-SoC board.
4. Read the document: Using the Accelerometer on DE SoC Boards for an example of how to implement the G-Sensor device with a C/C++ code. The document takes the information from the above 3 referenced documents and builds up the pseudo algorithm for the G-Sensor. Use the steps outlined in this document to write your own **C++ object oriented program** to configure and read from the G-Sensor.

   Using the document referenced in this part, compute the offsets of the Pin Multiplexer registers below from the `HPS_BRIDGE_BASE` address.

```
// The Pin Multiplexer selection
const unsigned int PIN_MUX_GEN_IO7      = ? ; // GENERALIO7 reg offset
const unsigned int PIN_MUX_GEN_IO8      = ? ; // GENERALIO8 reg offset
const unsigned int PIN_MUX_GPLMUX55     = ? ; // GPLMUX55 reg offset
const unsigned int PIN_MUX_GPLMUX56     = ? ; // GPLMUX56 reg offset
const unsigned int PIN_MUX_I2C0USEFPGA  = ? ; // I2C0USEFPGA  reg offset
```

5. Complete the tasks outlined in the following sections

Part 1 The G-Sensor (ADXL345) Output Data

We will pass raw accelerometer values for the X-axis only from the HPS to the FPGA for processing and display. A typical usable range of values read from the G-Sensor will be [-1000 to 1000] mg. You may get smaller or larger values than these if you tilt your board too much. Come up with an algorithm to map this range to an angle between 0 and 180 degrees, i.e. if the X-axis value is around -1000mg, we will want to display and angle of 0 degrees on the HEX displays, a value of 0mg maps to 90 degrees, and 1000 mg maps to 180 degrees. The range should translate linearly to the new range [0^o to 180^o] degrees.

Assignment 1

Write down your algorithm. You will implement this algorithm with logic blocks in Quartus Schematic in Part 5.

Part 2 Software Design: Creating the Base Class

Create a folder called *lab_project* and inside, create another folder called *SW_project*. Inside *SW_project*, copy your `DE1SoChps` class implemented in a header file (`DE1SoChps.h` file) and source file (`DE1SoChps.cpp` file). The class should have the following functions as implemented in the previous labs :

- Constructor initializing the memory-mapped I/O.
- Destructor finalizing the memory-mapped I/O.
- Function RegisterWrite(offset, value), writing a value into a register given its offset.
- Function RegisterRead(offset), returning the value read from a register given its offset.

The class `DE1SoChps` will be the base class initializing memory and controlling register access. All the writing and reading of registers should be done with the two functions provided by this class.

Replace the register offsets from the header file with the I2CO register offsets and Pin Multiplexer register offsets obtained in *part 1* and *part 4* listed above. Verify that the `HPS_BRIDGE_BASE` address and `HPS_BRIDGE_SPAN` remain the same.

Assignment 2

Create a parent folder named *lab_project*. Inside create another folder *SW_project* to contain all your C++ programs. At this point you should have your `DE1SoChps.h` and `DE1SoChps.cpp` files.

Part 3 Software Design: The G-sensor module

Create a new class `GSensor` in a header file `GSensor.h` as shown below, and source file `GSensor.cpp` which will be used to setup the G-Sensor module for reading data. Implement the class given below in the corresponding `GSensor.cpp` as described in the *Background* section, and the referenced document. Note that the class also includes a header file `ADXL345.h`. This file is provided for you on Canvas and it includes all the G-Sensor configurations and register addresses as described in ADXL345 Datasheet.

The class `GSensor` inherits the class `DE1SoChps` and uses the functions `RegisterWrite(…)` and `RegisterRead(…)` to write or read I2C0 or ADXL345 registers as needed.

```
/**
 * @file   GSensor.h
 * @brief  Process accelerometer input from from the DE1-SoC
 * Reads the XYZ accelerometer data from ADXL345 via the I2C0 line
 */

#ifndef GSENSOR_H
#define GSENSOR_H

 // project includes
#include <stdint.h>
#include "ADXL345.h"
#include "DE1SoChps.h"

class GSensor : public DE1SoChps
{
```

```
private:
   /* Prototypes for functions used to configure Pin Mux and I2C0 */
   /**
    * Configure I2C Pin Multiplexer to use I2C for ADXL345.
    * - Writes the settings to the corresponding pin_mux registers
    */
   void PinmuxConfig();

   /**
    * Configure I2C0 for communication with ADXL345
    *- Writes the settings to the corresponding I2C0 registers
    */
   void I2C0_Init();

public:
   /**
    * Constructor Initializes Pin Multiplexer I2C0
    * by calling the two functions
    */
   GSensor();         // Constructor
   /**
    * Destructor to finalize G-Sensor module
    * Does nothing other than print a finalizing message
    */
   ~GSensor();        // Destructor

   /* Prototypes for functions used to configure ADXL345 */
   /**
    * Initialize the ADXL345 chip
    *- Writes the settings to the corresponding I2C0 registers
    */
   void ADXL345_Init();

   /**
    * Read value from internal register at address
    *
    * @param address  Address of register to read.
    * @param value    Address of value to read
    */
   void ADXL345_RegRead(uint8_t address, uint8_t *value);

   /**
    * Write value to internal register at address
    * @param address  Address of register to write to.
    * @param value    Value to write to register
    */
   void ADXL345_RegWrite(uint8_t address, uint8_t value);

   /**
    * Reads values from multiple internal registers
    *
    * @param address  Address of the first register to read.
    * @param values   Buffer to read data into
    * @param len      Number of registers to read from
    */
   void ADXL345_RegMultiRead(uint8_t address, uint8_t values[], uint8_t len);

   /**
    * Read acceleration data of all three axes
```

```
     *
     * @param szData16 Buffer to read data into
     * @param mg_per_lsb     Resolution multiplier factor
     */
    void ADXL345_XYZ_Read(int16_t szData16[3], int16_t mg_per_lsb);

    /**
     * Return true if there is new data
     *
     * @param None
     * @return None
     */
    bool ADXL345_IsDataReady();
};

#endif
```

Write a `GSensorMain.cpp` file that can be used to test the functionality of the G-Sensor module and files.

The main function should perform the following tasks (refer to the reference document in *Background section part 4* for an example) :

- Instantiate an object of the `GSensor` class using dynamic memory allocation. This object automatically opens and maps the device memory from the parent `DE1SoChps` class, and configures the Pin Multiplexer for the I2CO, then initializes the I2C0 itself from the `GSensor` constructor.
- Reads the ADXL345 Device ID (Reg 0x00) and verifies the correct ID. If the ID is not correct, then print an error message and exit. Otherwise, Initialize the ADXL345 module by calling the appropriate function.
- Then, in an infinite while loop, read and print the X, Y, Z data when new data is ready. The digitalized output is formatted as 16-bit in two's complement and needs to be multiplied by 4 representing the resolution for the LSB (this is an approximate multiplier, actual value is 3.9 mg)

Write a `Makefile` to compile all your files.

Assignment 3

1. In your folder, you should have the program implemented in the following files: `DE1SoChps.h`, `DE1SoChps.cpp`, `GSensor.h`, `GSensor.cpp`, & `GSensorMain.cpp`. Copy and include the `ADXL345.h` file in this folder too.
2. Create a `Makefile` to compile your programs, including a clean rule accessible through command `make clean` in order to get rid of all generated binary files.
3. Compile and run your program and verify that you are getting valid data (in mg) within the range of about -1000 mg to 1000 mg on each of the X, Y, and Z axis after moving your board around (tilting to different axis).
4. Screenshot of terminal output and include all your C++ files as part of your project report submission.

Part 4 Hardware design: HPS-FPGA bridge

This section describes how to develop a Quartus project to design the needed hardware in the FPGA, so

the HPS and FPGA can communicate with each other. The Quartus project will be created based on the "HW_project" Quartus project included in the laboratory materials you will find in Canvas. This project includes basic implementation, which already includes the HPS component. The HPS component has been well-configured according to the hardware design of the DE1SoC HPS. This project includes also the required pins declared for the HPS. Note that the pins declared for HPS only needs to specify pin direction and IO standard. Pin location is not required for those HPS pins.

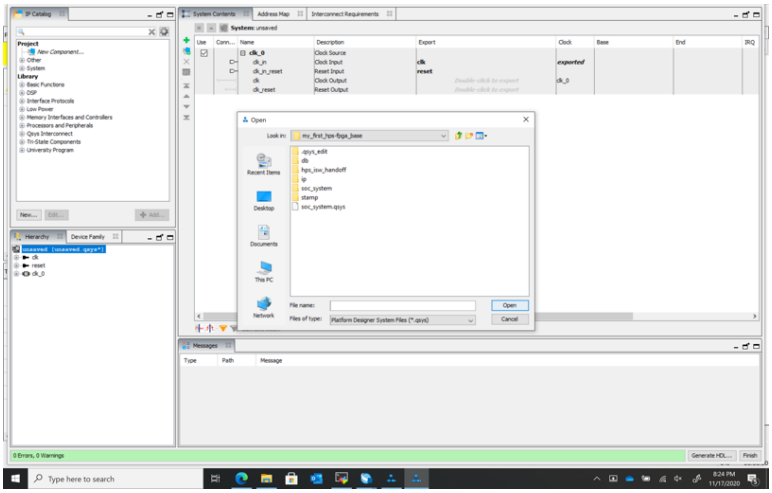
Follow the next steps to get ready:

1. Download the Quartus project "HW_project" to use it as a starting point. Download the compress file and decompress it.
2. Include this folder inside your *lab_project* folder.
3. Launch Quartus Prime and open the project "HW_project". For that, in Quartus -> Open project and double click in "soc_system.qpf" file.
4. Now you are ready to start working with this project. Do not change anything unless requested in this guided document.

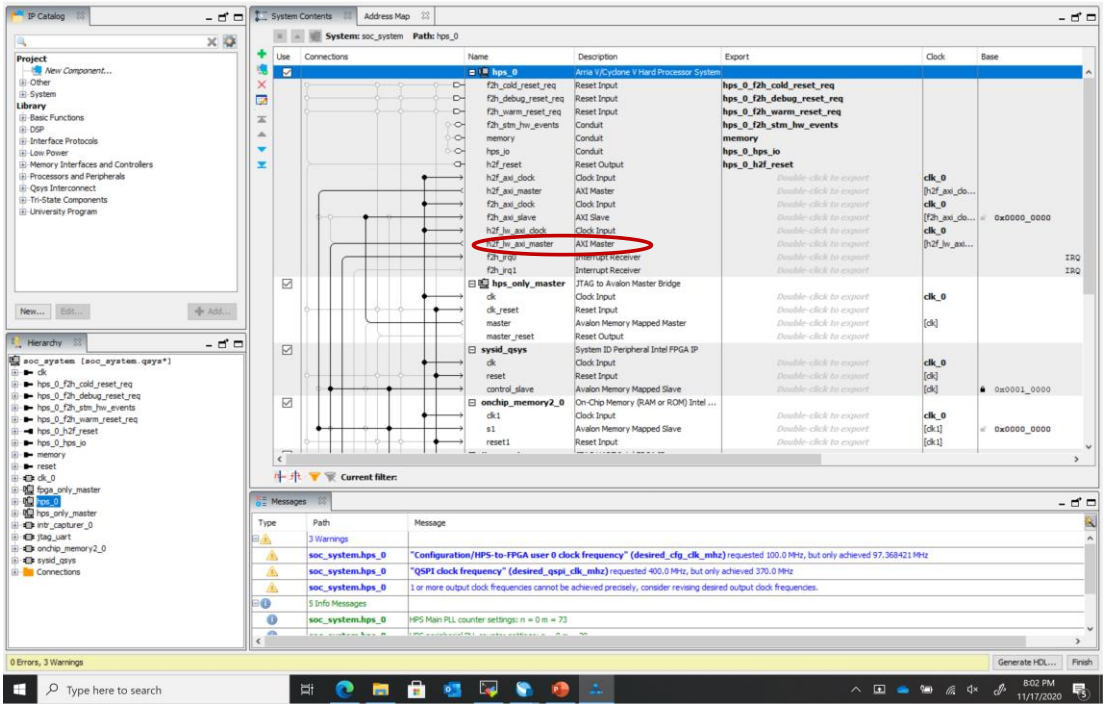
In order to generate the necessary connection between the HPS and the FPGA, we will use the "Platform Designer" tool provided in Quartus (previously known as the Qsys tool). The Platform Designer is a system integration tool that makes it easy to add the desired components and configure how they connect together. The project "soc_system" already includes a system created with the Platform Designer, with the necessary components and connections to create the connection between the HPS and the FPGA.

To open the system in the Platform Designer, do the following:

1. To open the Platform Designer, choose Tools -> Platform Designer. A new window will be launched.



2. When the Platform Designer is launched, it will ask you to select a target system file. In this case, select the file "soc_system.qsys". The figure below shows the content of the soc_system.qsys system. As you can see, among others, it contains the hps_0 HPS component. The port "h2f_lw_axi_master" is the lightweight HPS-to-FPGA AXI Master port of the HPS component. This port can be connected to any memory-mapped slave port of hardware components we want to access from the HPS.



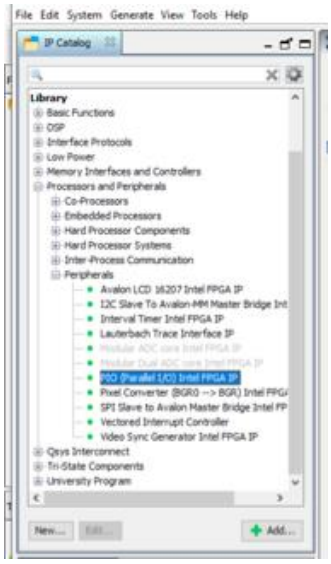
Now, you need to add the necessary components, so the HPS can drive output signals to the FPGA and receive input signals from the FPGA. The component that let you achieve this goal is the PIO (Parallel Input/Output) component. We will add two PIO components.

a) Output PIO component: the HPS drives the X coordinate obtained from the G-sensor and stored in the corresponding register in the HPS. This output signal will be passed to the FPGA hardware to calculate the corresponding angle.

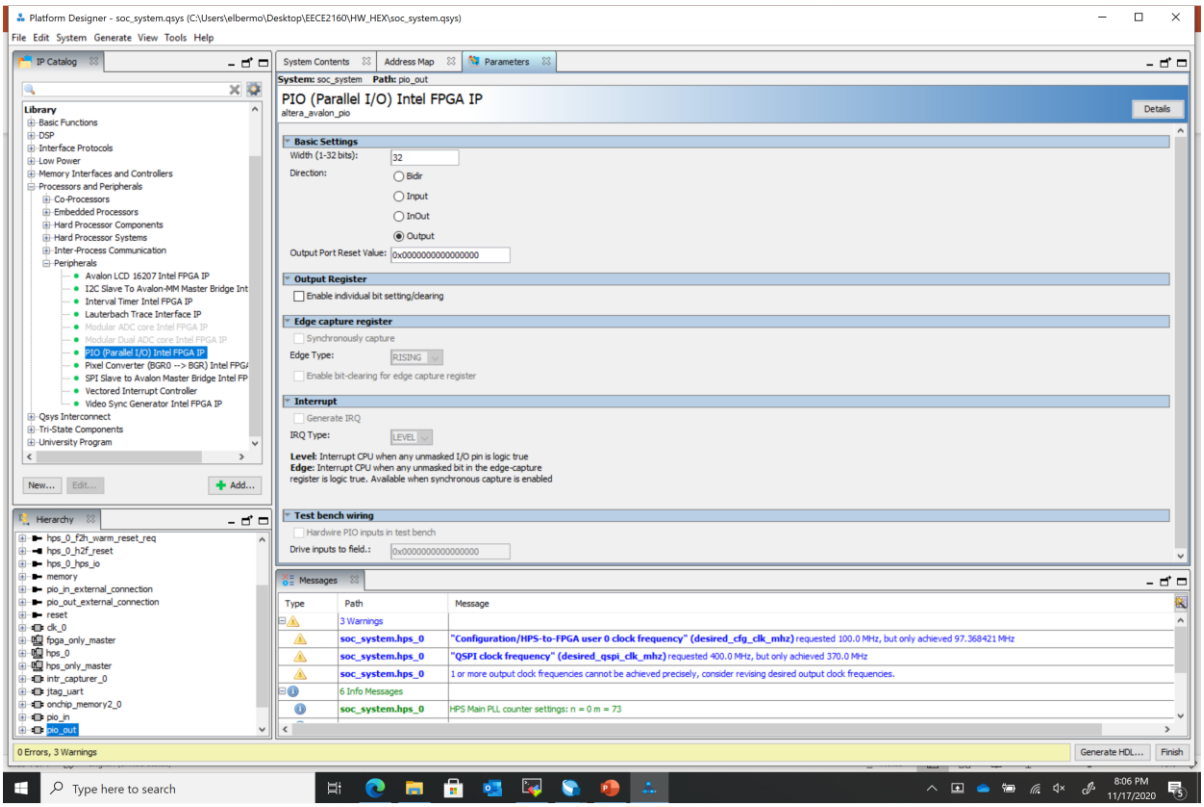
b) Input PIO component: the HPS receives the angle calculated by the FPGA hardware. The HPS can obtain the angle reading in the corresponding register.

In order to add a the output PIO Component in the Platform Designer and connect the PIO component to the HPS follow the next steps:

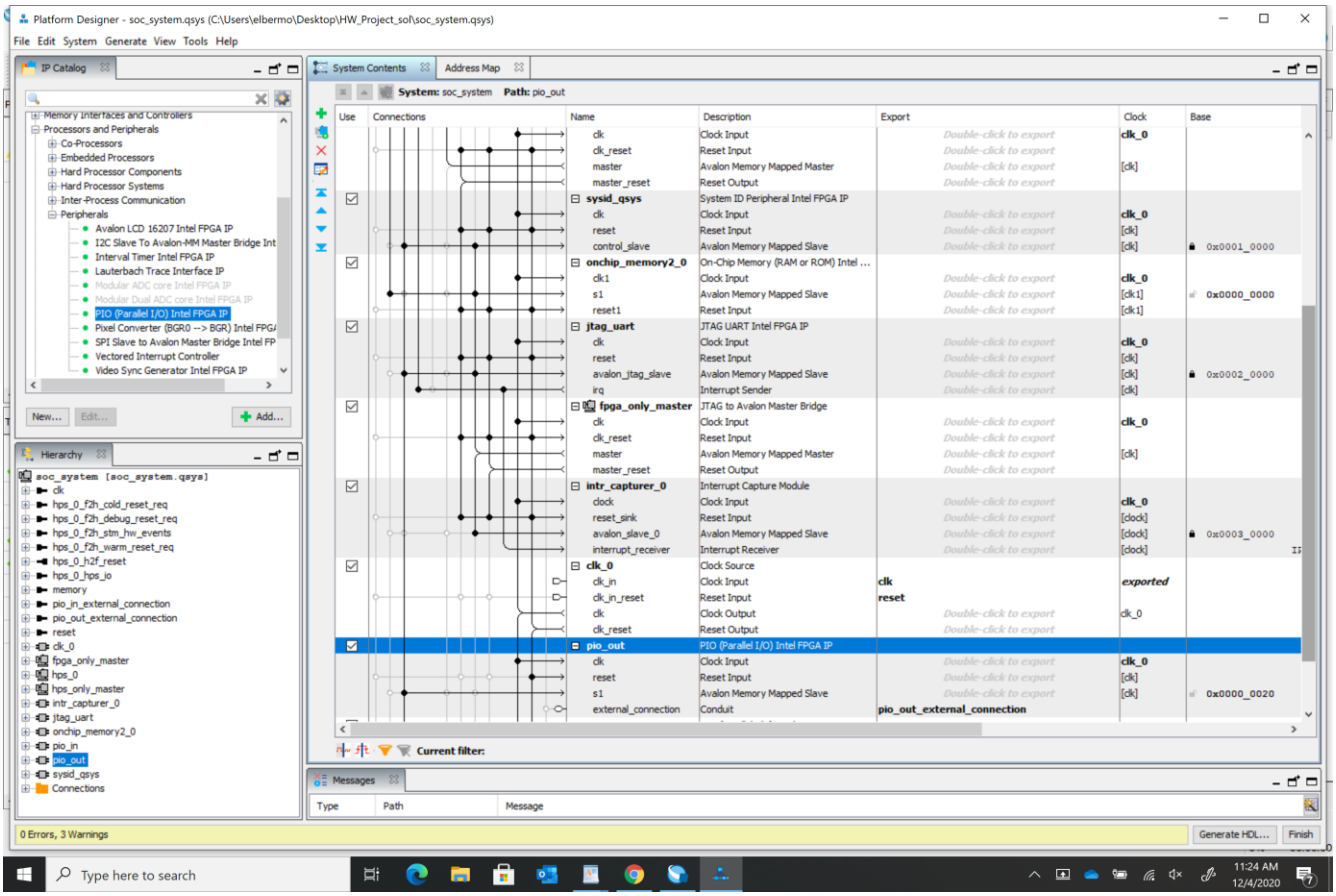
1. In Library, Processors and Peripherals -> Peripherals -> PIO (Parallel I/O) Intel FPGA IP.



2. In Basic Settings, assign 32 bits and output direction to the PIO. Click finish.



3. Rename pio_0 to pio_out. For that, right click in pio_0 -> renane
4. When the PIO component is added into the system, connect the “h2f_lw_axi_master” AXI port of the hps_0 component to the “s1” slave port of the PIO component. To do that click in the dot that connects both ports.
5. Connect the “clk” port of the PIO component to the “clk” port of the clk_0 component.
6. Connect the “reset” port of the PIO component to the “clk_reset” port of the clk_0 component.
7. Export the “external connection” as “pio_out_external_connection”. For that in the column “Export” double click in the external connection position. Make sure the name exported is “pio_out_external_connection”.
8. The Base address of the pio_out is very important. The HPS program will access this PIO component according to this base address. Change the base address and assign value 0x0000_0020. For that, double click in the column “Base” and change it accordingly.
9. For you reference, check that your system looks like the following figure:



Now, you need to add another input PIO component. Follow the same steps and use the following values:

1. 32 bits, input.
2. Rename the component to pio_in
3. Same port connections as before
4. Export with name “pio_in_external_connection”
5. Base address: 0x0000_0040

Your Qsys design is ready. Save the file and in the top menu, click Generate -> Generate HDL… to generate the source code for the system that will be used in the top-level file of your project. Do not change any parameter and click “Generate”. It may take a couple of minutes.

After the generation is complete (you will have some warnings, that is fine), click “Close”. Close the Platform Designer tool window and go back to the Quartus project. A message will pop up with some information. The needed .qip and .sip files are already integrated in the project, so you do not have to do any action. Ignore this message.

Assignment 4

Take a screenshot of the Qsys system with both PIO components and include it in your report. Include several screenshots with different parts if needed.

Part 5 Hardware design: Hardware design to obtain inclination angle of the board

This section describes how to develop the hardware design in your Quartus project in order to convert an X coordinate from the HPS into an angle that shows the inclination of the board. This angle will be displayed in the 7-segment display and will be sent to the HPS, so a program running in the HPS can obtain the angle from the X coordinate of the G-sensor.

In order to achieve our goal, we will add the necessary hardware components in the already created "soc_system" project. In the previous section you develop the necessary logic to connect the HPS and the FPGA with the Platform Designer, now you will create the new hardware component by using schematics.

First of all, take some time to understand the files that we have already in the project. In Quartus, in the project navigator, choose Files. As you can see, there are many different files. The most important files are the following:

a) "soc_system/synthesis/soc_system.qip": this package contains all the files generated by the Platform Designer system.

b) "soc_top.v": this is the top-level file that defines the input/output pins connected to the FPGA pins. This file is a Verilog file, where the hardware is described in Verilog HDL. This is an advanced topic that we do not cover in this course, so this file has been already prepared to make the necessary connections among the different components needed and also includes all the input/output pins needed for the pin planner. In this file, the Platform Designer system is instantiated and connected to other components in the project. Also, the Verilog file includes the connections needed to integrate the hardware component block, "angle_calculator", you will create in this section. Just for your reference, this block corresponds with the following lines in the Verilog file:

```
angle_calculator angle_inst (
      .coorX(out),
      .angle(in),
      .display0(HEX0),
      .display1(HEX1),
      .display2(HEX2)
);
```

Our next step is to create a new component by using Quartus schematic, where we will add the necessary logic to convert an X coordinate into an angle that shows the inclination of the board. The angle must be displayed in the 7-segment displays HEX0, HEX1 and HEX2. In order to do that, you will need the files you used in Lab7. In order to implement this new component, follow these steps:

1. Copy the **display7.bdf**, **display7.bsf**, **display7_en.bdf**, **display7_en.bsf**, **display_8bits.bdf**, **display_8bits.bsf**, all the **div8u** files, **comp0** files, and **const10** files inside the folder "HW_project".
2. In the Quartus project, create a new .bdf schematic file and name it *angle_calculator*.
3. Your schematic **must have** the following input and output pins and names
    a. Input: 32-bit pin called "coorX". Coordinate X passed from the HPS.
    b. Output: 32-bit pin called "angle". Angle calculated that will be passed to the HPS.
    c. Output: 7-bit pin called "display0". These bits will be assigned to the HEX0 pins.
    d. Output: 7-bit pin called "display1". These bits will be assigned to the HEX1 pins.
    e. Output: 7-bit pin called "display2". These bits will be assigned to the HEX2 pins.

4. Add the necessary Arithmetic LPM IP blocks (Adder, Multiplier, …) or other necessary IP blocks to calculate the angle given by the X coordinate. Follow the equation you obtained in Part 1. Connect the necessary input/output pins accordingly. Here you have some tips:
    a. The X coordinate and the angle are represented with 32 bits integers, choose the IP blocks input parameters accordingly (32 bits).
    b. Note that the inputs/outputs of the arithmetic block must be signed. Choose the IP blocks parameters accordingly.
    c. For adders and multipliers, you can set as the second operand a fixed value. Choose the IP blocks parameters accordingly.
    d. We are working with signed integers. When you design your circuit, make sure that any of the partial results is rounded to 0.
5. We want to display the angle value in the 7-segment display. For that, you will use the "display_8bits" block you created in previous labs. Note that the angle will always have a positive value (0-180) that can be represented with only the lower-order 8 bits of the angle represented by a 32-bit integer. Connect the corresponding output pins accordingly.
6. Save the file and make sure that it is in the list of files in the project navigator (it must have the name *angle_calculator.bdf*)

Now your project is ready to be compiled. Follow the next final steps:

1. Compile your project. It may take several minutes.
2. Add the pin location and parameters for the pins of HEX0, HEX1, HEX2 and CLOCK50 (CLOCK50=PIN_AF14). Do not remove any pin and do not change the parameters or location of other pins. If you open the pin planner, you will see that there are many HPS pins that are not connected. This is correct, the HPS pins do not need the pin location. Do not change any parameter of the HPS pins.
3. Compile your project again. It may take several minutes. Once it finishes, you have your hardware design ready to program the board. But first, you will need to add some functions to your software project in Part 6.

Assignment 5

Take screenshots of the component designed in Quartus schematic and include them in the project report.

Part 6 Software Design: Prepare software for hardware integration

In this section, you will develop C++ functions to control the "pio_out" and the "pio_in" PIO components from the HPS. For a program to control the PIO components, the base address of the PIO components is required. The "/dev/mem" memory space and mmap system call will be used in the same way you used it in previous labs, so the physical addresses of the PIO components is mapped to a virtual address that can be directly accessed by the software program. The difference between previous labs and this project is that now, you have implemented your own hardware system in the FPGA instead of using the default hardware used to program the FPGA when the board is turned on.

In order to prepare your software program for hardware integration, we will reuse the files you develop

in previous labs.

Inside the folder *SW_project*, copy your `DE1SoCfpga` class implemented in a header file (`DE1SoCfpga.h` file) and source file (`DE1SoCfpga.cpp` file).

The class should have the following functions as implemented in the previous labs:

- Constructor initializing the memory-mapped I/O.
- Destructor finalizing the memory-mapped I/O.
- Function RegisterWrite(offset, value), writing a value into a register given its offset.
- Function RegisterRead(offset), returning the value read from a register given its offset.

In `DE1SoCfpga.h`, remove all the offsets for the LEDR, SW and HEX displays. Add the register offsets for the PIO components:

```
const unsigned int OUT_BASE         = 0x00000020;   // PIO output
const unsigned int IN_BASE          = 0x00000040;   // PIO input
```

Verify that the `LW_BRIDGE_BASE` address and the `LW_BRIDGE_SPAN` remain, `0xFF200000` and `0x00005000`, respectively.

The class `DE1SoCfpga` will be the base class initializing memory and controlling register access to the PIO components in the FPGA. All the writing and reading of registers should be done with the two functions provided by this class.

Inside the folder *SW_project*, copy your `LEDControl` class implemented in a header file (`LEDControl.h` file) and source file (`LEDControl.cpp` file). Rename the class to `PIOControl`. Rename all the files and class/members accordingly.

We will need the following functions to be in class `PIOControl` as implemented in `LEDControl`:

- Two private data member: `unsigned int out_regValue` and `unsigned int in_regValue` representing the state of the output/input PIO registers. The `out_regValue` variable should be updated every time a new value is written to the corresponding register.
- Constructor initializing the private data members.
- Destructor printing the message: *"Closing PIOs..."*.
- Function `WritePIOout(int value)`, which writes a value in the output PIO register. This function is the same as `WriteAllLeds(int value)`, you only need to update the name of the function, the offset variable and the private data member name.
- Function `ReadPIOin()`, which returns the value stored in the input PIO register. This function is the same as `ReadAllSwitches()`, you only need to update the name of the function, the offset variable and the private data member name.
- Remove other function you had in `LEDControl`.

Assignment 6

Add the files `PIOControl.h`, `PIOControl.cpp`, `DE1SoCfpga.h`, and `DE1SoCfpga.cpp` to the folder *SW_project.* You do not need to include anything in the report.

Part 7 Software-Hardware Integration

You have developed your hardware design and the software programs. Now it is time to integrate them.

First, modify the main function you developed in Part 3 with the following additions:

1. Add the new header/s needed.
2. Instantiate an object of the `PIOControl` class using dynamic memory allocation
3. The coordinate X obtained from the G-sensor is stored in the PIO output register.
4. The angle value that represents the inclination of the board, which is stored in the PIO input register, is read and displayed in the terminal.

Now, everything is ready. Follow these final steps:

1. Connect your computer to the FPGA with the white cable and program the FPGA with the "soc_system.sof" file that you will find in the "HW_project".
2. You can disconnect the white cable and connect your computer to the HPS with the black cable. Connect to the HPS, compile and run your software program.

Assignment 7

a) In your *SW_project* folder, you should have the program implemented in the following files: `ADXL345.h`, `DE1SoChps.h`, `DE1SoChps.cpp`, `GSensor.h`, `GSensor.cpp`, `DE1SoCfpga.h`, `DE1SoCfpga.cpp`, `PIOControl.h`, `PIOControl.cpp` & `GSensorMain.cpp`.

b) Modify the `Makefile` to compile your programs, including a clean rule accessible through command `make clean` in order to get rid of all generated binary files.

c) Compile and run your programs and verify the correct behavior. Record a video with a maximum duration of 20 seconds where you demonstrate the behavior of the program. Upload your video as part of your project submission in a file named Assign7.mov.

d) In the project report, include also the output terminal showing the compilation and execution of the program.

e) Download all the .h and .cpp files, and Makefile into your local machine, and create a ZIP file named Project.zip. Include this file with your report submission.

Project Report

You should follow the lab report outline provided on Canvas. Your report should be developed on a wordprocessor (e.g., OpenOffice or LaTeX), and should include graphics when trying to present a large amount of data. Upload the report on canvas. Include screenshots of your design in your report body or the appendix of your report as requested in the assignments. **Submit also your C++ files and videos in a zipped folder as requested in the assignments.**

Appendix:

1. DE1-SoC User Manual
2. Chapter 21 (I^2C Controller) from the *Cyclone V Hard Processor System Technical Reference Manual*
3. Using the Accelerometer on DE SoC Boards
4. ADXL345 Datasheet (Digital Accelerometer)