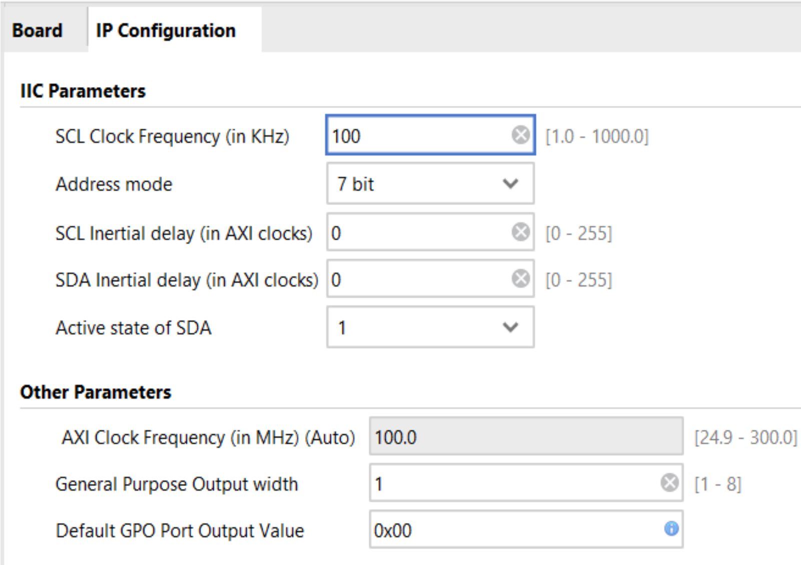
Project 2 Outline

Creating the Embedded System

You can create the embedded system the same way you did so for Project #1 by creating the block diagram in the IP Integrator and modifying the target FPGA platform top level and constraints files to accommodate the addition of the Nexys4IO inputs and outputs. Specify the FPGA part number for your target FPGA platform instead of using the board files. Add the bidirec module and nexysa7 module into the sources list. A sample constraints file is also provided.

The embedded system should have this minimum configuration:

- Microblaze, mdm, etc. Configure the Microblaze for 128 kB of local (BRAM) memory. Configure the mdm (debug port) to include a UART. Enable the AXI bus and interrupts.
- Nexys4IO: Add a Nexys4IO IP block to your embedded system. This IP block and driver interfaces to the green LED's, switches, pushbuttons, RGB LEDs, and 7-segment display on the target FPGA platform.
- AXI Timer/counter: Add a 32-bit AXI timer.
- AXI Interrupt Controller & `xl_concat` block with number of interrupt sources set to 1 and connected to the FIT timer interrupt output.
- AXI UART Lite - We will be using an external serial terminal to grab data from the application and save it. Double click on the module to make the following changes
  - Board->Board Interface->usb uart
  - IP Configuration -> AXI CLK Freq = 100.0, Baud Rate = 115200, Data Bits = 8, Select No Parity
- AXI IIC - Used to interface the MPU 6050 Sensor. Configuration:
  - Board->Board Interface -> Custom
  - IP Configuration->



IIC Parameters

SCL Clock Frequency (in KHz): 100 [1.0 - 1000.0]

Address mode: 7 bit

SCL Inertial delay (in AXI clocks): 0 [0 - 255]

SDA Inertial delay (in AXI clocks): 0 [0 - 255]

Active state of SDA: 1

Other Parameters

AXI Clock Frequency (in MHz) (Auto): 100.0 [24.9 - 300.0]

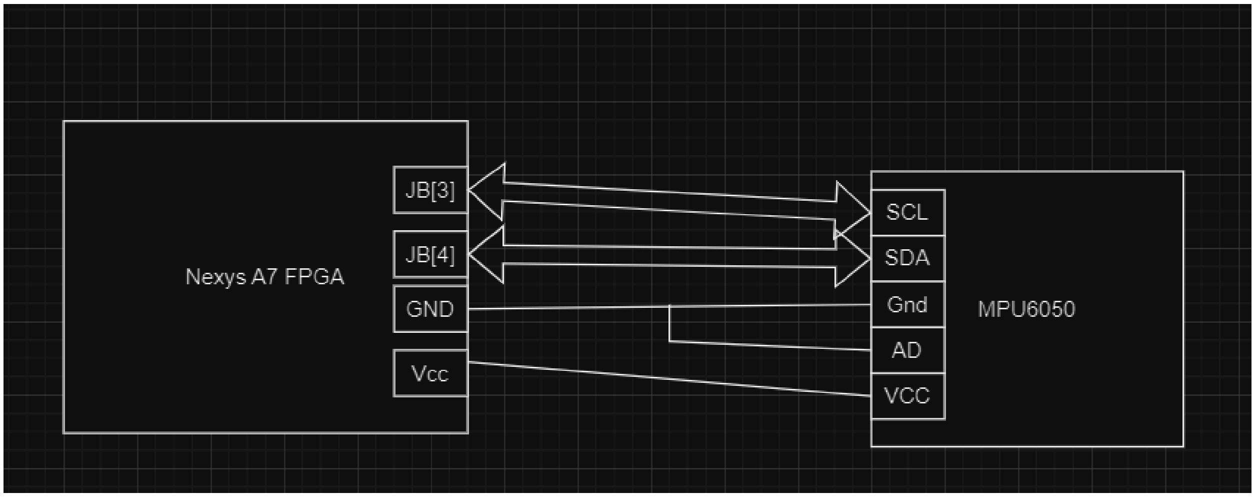
General Purpose Output width: 1 [1 - 8]

Default GPO Port Output Value: 0x00

Refer to the schematic to verify the design.

Hardware Setup (MPU-6050)

There are two I2C addresses that the MPU 6050 can be used with. You will be using I2C Address as 0x68. Connect the AD Pin to ground



Creating the MPU-6050 driver

Open the MPU-6050 Register Map to understand what registers need to be referenced. We recommend that you create two files mpu6050.c and mpu6050.h.

`mpu6050.c`:
Should contain the code for the functions described in the table below:
a. `void mpu6050_init(XIic* i2c);`
b. `void mpu6050_getGyroData(XIic* i2c, u8 *angle_actual, int axis);`
c. `void mpu6050_gyroCfg(XIic* i2c);`
d. `void mpu6050_setSleepMode(XIic* i2c);`
e. `void mpu6050_clearSleepMode(XIic* i2c);`

The functionality is described in the below table:

Function Name	Function Description
mpu6050_init	Identify the address of the MPU-6050 Sensor. Reset the MPU-6050 Sensor through PWR_MGMNT_1 Register
mpu6050_getGyroData	Get 2 bytes of raw Gyro data corresponding to a particular axis
mpu6050_gyroCfg	Configure the Gyro Sensitivity to enable Full scale Range

mpu6050.h:
Should contain any structure declarations, list of all the functions used in `mpu6050.c` and any macros used.

Introduction to MPU6050

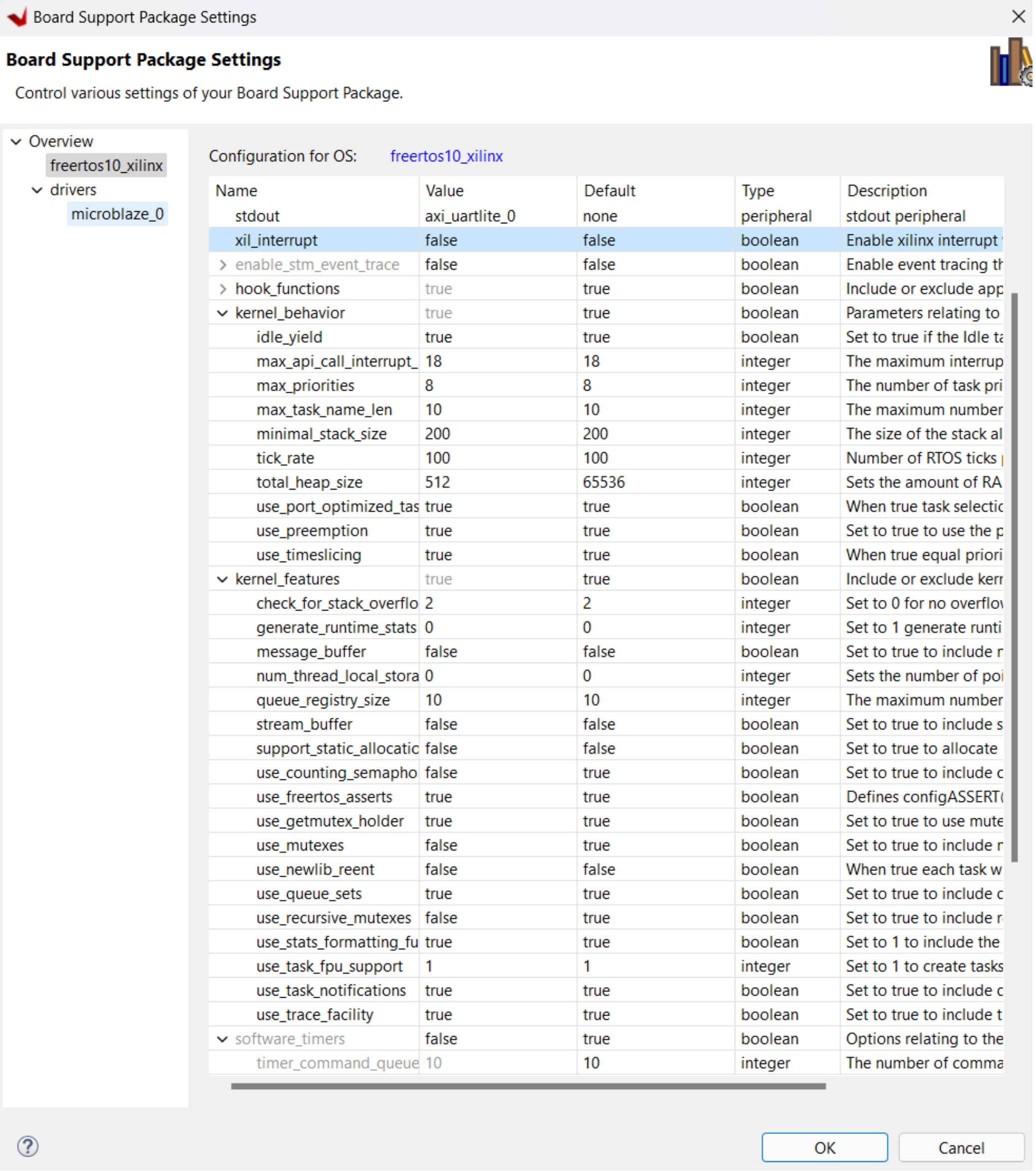
https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

Calibration and angle measurement:
https://www.youtube.com/watch?v=7VW_XVbtu9k
https://www.youtube.com/watch?v=Yh6mYF3VdFQ

Firmware setup in Vitis

Prepare the initial setup as done in Project #1. (Including build of both Platform Project and Application Project with a main). While creating the Platform Project in Vitis, please select the OS as FreeRTOS instead of the standalone OS.

After creating the Application Project in Vitis, go to the <application_name>.spr tab -> Board Support Package -> Modify BSP Settings -> overview -> freertos10_xilinx ->kernel features -> Make the following changes:



Board Support Package Settings

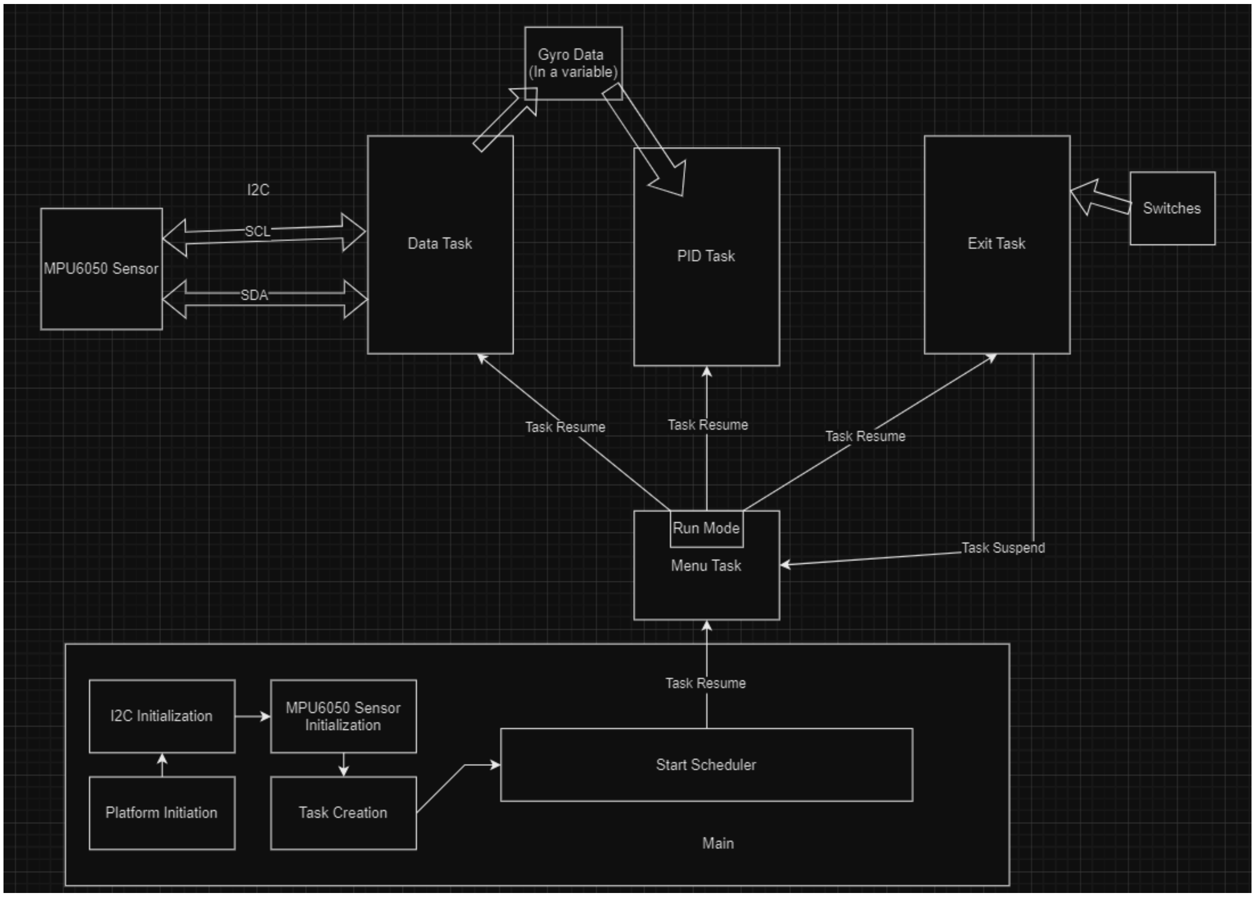
Control various settings of your Board Support Package.

Configuration for OS: freertos10_xilinx

Name	Value	Default	Type	Description
stdout	axi_uartlite_0	none	peripheral	stdout peripheral
xil_interrupt	false	false	boolean	Enable xilinx interrupt
enable_stm_event_trace	false	false	boolean	Enable event tracing th
hook_functions	true	true	boolean	Include or exclude app
kernel_behavior	true	true	boolean	Parameters relating to
idle_yield	true	true	boolean	Set to true if the Idle ta
max_api_call_interrupt_	18	18	integer	The maximum interrup
max_priorities	8	8	integer	The number of task pri
max_task_name_len	10	10	integer	The maximum number
minimal_stack_size	200	200	integer	The size of the stack al
tick_rate	100	100	integer	Number of RTOS ticks
total_heap_size	512	65536	integer	Sets the amount of RA
use_port_optimized_tas	true	true	boolean	When true task selectic
use_preemption	true	true	boolean	Set to true to use the p
use_timeslicing	true	true	boolean	When true equal priori
kernel_features	true	true	boolean	Include or exclude kerr
check_for_stack_overflo	2	2	integer	Set to 0 for no overflov
generate_runtime_stats	0	0	integer	Set to 1 generate runti
message_buffer	false	false	boolean	Set to true to include r
num_thread_local_stora	0	0	integer	Sets the number of poi
queue_registry_size	10	10	integer	The maximum number
stream_buffer	false	false	boolean	Set to true to include s
support_static_allocatic	false	false	boolean	Set to true to allocate
use_counting_semapho	false	true	boolean	Set to true to include c
use_freertos_asserts	true	true	boolean	Defines configASSERT(
use_getmutex_holder	true	true	boolean	Set to true to use mute
use_mutexes	false	true	boolean	Set to true to include r
use_newlib_reent	false	false	boolean	When true each task w
use_queue_sets	true	true	boolean	Set to true to include c
use_recursive_mutexes	false	true	boolean	Set to true to include r
use_stats_formatting_fu	true	true	boolean	Set to 1 to include the
use_task_fpu_support	1	1	integer	Set to 1 to create tasks
use_task_notifications	true	true	boolean	Set to true to include c
use_trace_facility	true	true	boolean	Set to true to include t
software_timers	false	true	boolean	Options relating to the
timer_command_queue	10	10	integer	The number of comma

In the `src` folder of your application project, create a new folder mpu6050 and add your driver files.

Application Architecture



FPGA target board inputs:
Slide Switches: The 16-slide switches to exit the run mode.
Pushbuttons: `BTNU` (`BTN0` on Boolean board) and `BTND` (`BTN3` on Boolean board) buttons are used to cycle to increment/decrement target angle while the PID is running.
`LEDs`, `RGB LEDs`, and `7-segment display` are not used in the application
`BtnCpuReset` (`BTN0` & `BTN1`) should restart the application

Menu Task:
Print the user name and email address (similar to Project #1) and ask the user for Run mode (`r` or `R`). If the character is identified, query the user for the target angle. Then, resume the `Data` task, `PID` task and `Exit` task

Data Task:
Get the X-axis raw data from MPU-6050 and compute the angle

PID Task:
Get the computed angle and implement PID Control. Check if BTNU and BTND are pressed. If yes, Increment/decrement the target_angle by 1 in degrees.

Exit Task:
Check if the switches give a non-zero value and suspend `Data` , `PID` and `Exit` tasks when true. Resume `Menu` task after suspension.

Application Approach

Please ensure that each step works correctly and only then move on to the next step.
Also, Please try not to copy firmware elements from Project #1. They are not going to help you move forward with the project.

Keep the FreeRTOS Developer docs (https://www.freertos.org/a00106.html) open.

a. Platform initialization - platform init
b. I2C Initialization – Initialize the AXI I2C controller Note: it may not be necessary to much initialization if you use the low level drivers, but check the Product Guide and API documentation.
c. mpu-6050 init function - Read the MPU-6050 ID
d. MPU-6050 Configuration - Set to Gyro to Full Scale Range (250deg/s)
e. Create the `Data`, `PID`, `Exit` and `Run` tasks using the `xTaskCreate()` API
f. Implement the task functions for the Run, `Data`, `PID`, and `Exit` tasks
g. Suspend Data, PID and Exit tasks using the FreeRTOS Suspend API
h. Start the scheduler using the FreeRTOS StartScheduler API

Important Points to note:

a. The major challenge to this project is time. Use your time wisely. Decide early on about the responsibilities of each team member. Allow plenty of time for system integration (bringing the HW and Firmware together); that is where most projects fail to meet their functionality and delivery schedule.
b. Use `xil_printf()` to send data over UART. To do this you will need to map `stdin` and `stdout` to the UART when you configure the BSP.
c. I2C Interface test with MPU-6050 Sensor is successful only when you can read the ID from the WHO_AM_I register. Don't perform the initialization in the `Data` task.
d. You can use `XIic_Send()` and `XIic_Recv()` functions for data transfer. These are the low level API functions.
e. Use only `Create`, `Resume`, `Suspend`, `StartScheduler`, `Delete` APIs for the basic implementation.
f. Feel free to use mutex or message queues etc. later after you are done with the basic deliverables - **extra credit**!!!

Time Management

Task Timeline	Student - 1 Tasks	Student - 2 Tasks
1-2 days	Generate the embedded System and be ready with sensor purchase done	Generate the embedded System and be ready with sensor purchase done
1-2 days	Create MPU-6050 driver referring the datasheet and the links given	Use the `.xsa` and develop the firmware application for initially printing on the PUTTY Terminal after platform initialization; at a minimum your app should be able to read the buttons and switches
1-2 days	Develop/implement the PID Controller	Create the Application architecture. Your application should be capable of setting up the `Menu` task. Check that you can run 3 tasks and then switch back to `Menu` task.
1-2 days	Prepare the `Data` task (i.e. getting the data from the sensors and performing computation to get Roll	Debug the previous task
1-2 days	Complete firmware integration and tests	Complete firmware integration and tests
1-2 days	PID Tuning and graphs	PID Tuning and graphs