PID-Controlled Angle Sensing System on the RealDigital BooleanBoard

By: Ibrahim Binmahfood and Robert Wilcox
Date: 03/03/2024

Abstract

Implemented a PID-controlled system on the RealDigital BooleanBoard, utilizing the MPU6050 sensor for precise angle stabilization. This project demonstrated the effectiveness of PID control for real-time adjustments, making it suitable for real world applications requiring precise gyroscopic positioning information, such as motor position sensors.

Introduction

This project aimed to address the challenge of achieving precise angle stabilization in embedded systems. By developing a FreeRTOS-based application on the BooleanBoard and integrating the MPU6050 sensor, we sought to enable real-time angle measurement and stabilization. The system's design and implementation focused on leveraging PID control algorithms to meet the demands of applications in need of precise orientational control.

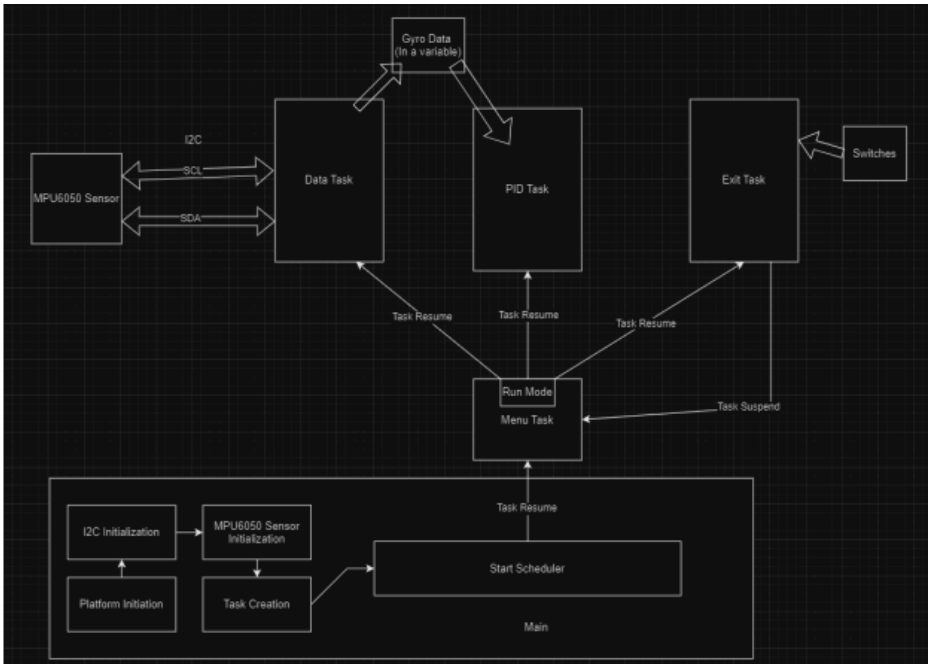
System Overview

The BooleanBoard, equipped with a MicroBlaze processor and FreeRTOS, served as the platform. The MPU6050 sensor provided gyroscope and accelerometer data for calculating angles. The system architecture integrated sensor data acquisition, PID control logic, and a user interface for real-time control.

Application Code Description

- Task Management:

  The tasks Menu, Data, PID, and Exit follow a state machine traversing through its states (tasks) based upon the inputs.



The required resources in using FreeRTOS's API is found in its documentation available in the references section of this paper. To set the application up, the tasks must have separate handles instantiated by the ***TaskHandle_t*** type and initialized to ***NULL***. This is required so that when suspending/resuming specific tasks at certain points of the application. The ***main()*** function is where the system initialization, microblaze platform initialization and creation of tasks is done. The ***xTaskCreate()*** function creates the tasks with specified aliases to the tasks, stack sizes, arguments provided to the tasks, tasks priority level, and tasks handles. The Data task required more stack space inorder to process the Fusion libraries math operations. So, its stack size is 3x the minimum. Inorder to begin the application at the Menu task, all other tasks were suspended. The Menu task was resumed and the Task scheduler was called.

For the Menu task, it provided a user interface through stdout of the UART. Querying the user to activate different modes of operation and set the target angle for the system. This task then resumes other tasks based on user input. To activate the run mode, the user must type 'r' or 'R' into the UART. Then to activate read mode or data gather mode, the user must type 'r' or 'R' and or 'd' or 'D' respectively. The read mode prints information about the data given. Whereas the data gather mode prints the raw data to the UART, allowing the user to put the data into a spreadsheet.

For the Data task, it primarily is for collecting and processing the data from the MPU6050 sensor. This task periodically reads data from the gyroscope and accelerometer, which processes the data to calculate the current angle. Then updates the system state with this new angle.

For the PID task, it primarily is for the control loop. This task periodically calculates the required correction to minimize the error between the target angle and current angle. The task also handles user input to adjust the target angle via BTN0 for increase and BTN3 for decrease.

For the Exit task, it primarily handles system exit conditions. This task will monitor for a switch asserted to suspend data collection and PID control tasks. The result of this will lead to stopping the system's active operations and returning control to the user in the Menu task.

- MPU6050 Driver:

  Developed a driver for I2C communication, enabling sensor initialization and data reading. Integrated this driver within the FreeRTOS framework for continuous data acquisition. The first step in developing this driver was to read the MPU6050 documentation. The MPU6050 can be controlled by writing to registers over the I2C bus; a list of registers relevant to this project was made and defined in the header file for the driver. Next, functions to read and write to a generic register over the I2C bus were created. At first, XIix_MasterSend and receive commands were tried, but when these caused some issues XIic_Send and XIixRecv were used instead.

  The read and write register functions could then be used to create functions to initialize the gyro, configure it, put it to sleep, wake it up, and read the gyroscope data. Because the project only needed 1 gyroscope axis, this function was set up to return data from 1 axis. Later on in the project’s development, it was discovered that the MPU6050 suffers from fairly severe drift issues. This was causing errors in the angle measurement, and steps were taken to correct this. A package on GitHub called Fusion AHRS by X-IO technologies was implemented in order to stabilize the drift. The package has a free license, and functions as an attitude and heading reference system. [See Appendix B for citation].

After implementing the package, the gyroscope angle readings were *very* precise and stable, albeit the system now has a little extra latency when stabilizing to the current angle. Because this application is non-specific, we opted for accuracy over latency in this case. In order to add the Fusion package, a function to get the MPU6050 accelerometer data was needed. Fusion also supports magnetometer data, but the MPU6050 lacks this sensor. Because it was known all 3 axes of accelerometer data were going to be needed for fusion, the getAccelData function returns all 3 axes data. The getGyroData function is simply called 3 times in the data task to get all 3 axes. An improvement to this driver would be to modify getGyroData to return all 3 axes, similar to the acceleration function.

- PID Controller:

  Implemented a PID controller to calculate control efforts based on sensor data, adjusting system behavior to stabilize the angle at a user-defined target. After reading Tim Wescott's paper, *PID without a PhD*, the PID application was fairly straightforward to develop. Wescott provides an example function of UpdataPID, which was used. The header file for the PID controller defines an SPid struct, which will contain the PID data.

- User Interface:

  The input mechanisms such as buttons and switches are processed through the Nexys4IO peripheral within the embedded system. However, for user input which activates certain modes and sets the target angle is done via UART. The UART Lite peripheral is configured to have stdout and stdin mapped to it. So when the user types in a character, the program uses UART Lite's lower level function ***XUartLite_RecvByte()*** function. It will return a byte character from stdin of the UART. To read the target angle, it is done in a similar fashion using the same function but through a wrapper function ***readAngle()***. The ***readAngle()*** function captures the numeric input from the UART console until a ***'\r'*** or ***'\n'*** is detected. Then it stores this input in a buffer as a string which can be later converted to a number value.

Problems Encountered and Solutions

- I2C Communication:

  We faced many challenges when it came to I2C communication. An AXI_IIC IP block was added to our embedded system design, which was to be used to communicate with the MPU6050 sensor. However, upon hooking up the sensor we received no communications over the bus. At first, we looked at whether it was an issue in the MPU6050 driver code. This is when the change away from the master send and receive calls was made, but it did not solve the issue. A logic analyzer was used to determine the I2C lines were experiencing some noise issues, causing the bus to unintentionally go into a busy state and get stuck there. A post in the Xilinx forums indicated that using the inertial delay feature of the AXI_IIC block could help with this problem. The inertial delay essentially provides some signal filtering. After implementing an inertial delay of 5 clocks for both the SCL and SDA lines, the bus freed up and we were able to communicate with the sensor. It is notable that teams using the Nexys A7 board did not need inertial delay, so it seems like an issue specific to the BooleanBoard. After this fix was made there were no more issues with I2C communication.

- PID Tuning:

  Following Professor Kravit's PID tuning instructions carefully, one pass of the PID variable was made and no more. The system seems fairly responsive, and because we are not actually feeding the output of the PID calculations into an exact mechanical system, it is certainly good enough for this case. If this program were to be used as control for a mechanical system, further tuning for that system would be needed.

- Task Synchronization:

  When in any of the tasks, the task you wish to go to next would always hang. This occurred in the Exit task mostly. The cause of this issue was the ordering of suspending/resuming functions. To prevent this issue, you must suspend the task you are in last to be called and the resumption or suspension of other tasks before it. Another problem was the data task kept experiencing stack overflows. So, the stack size created for the data task is 3x the minimum stack size to prevent this.

- System Integration:

The integration of both our parts didn't lead to any major problems. Since we had communicated our plans together earlier on. Also updating each other as we went along with our tasks to complete.

Performance Evaluation

- The PID performance was graphed, and the system appears to be fairly responsive. Again, it is my hand moving towards the target, and I have *a lot* of latency and inaccuracy. To really test this system, we would need to implement the control on some mechanical system which can make adjustments at computer-level speeds.

Suggestions for Improvement

- Hardware Upgrades: Using a more advanced sensor which includes a magnetometer, or using a magnetometer in conjunction with the MPU6050, would allow even more accurate angle measurements. Actually implementing a mechanical system for the PID to control would be the obvious next hardware step for this platform. Adding a motor or some sort of balancing system would be an interesting application.
- Software Optimization: Some of the function calls in the drivers written could be cleaned up for neater code, but as it stands that would not really improve the design much. An obvious improvement would be to add mutexes or semaphores, or some other form of data protection. As it stands, multiple tasks do not interfere with each other because they are running at different times, but if we wanted a more realistic real-time system then some form of data protection would need to be added.

- Feature Expansion: We would have liked to add the feature to display the target angle on one set of the SSEG LEDs, and the current angle on the other. This would have provided even more user feedback without the use of the terminal. This would essentially be a step in allowing the system to operate independent from a computer, which it needs right now both for the menu input and to display the target and current angles.

Conclusion

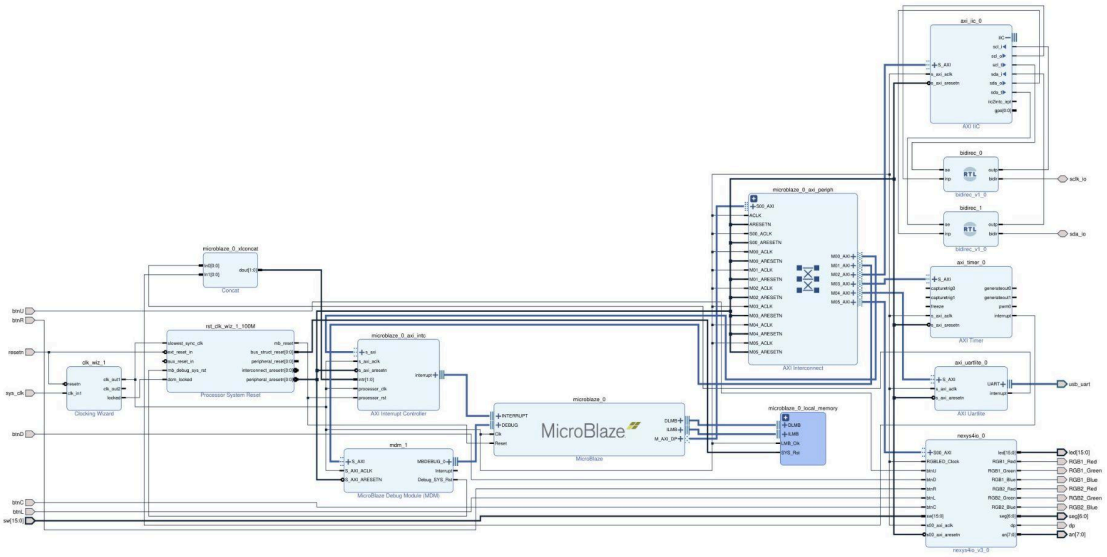
The objectives and achievements of this project are met. The firmware application uses Free RTOS on a Microblaze-based system to measure the roll of an MPU-6050 Accelerometer/Gyroscope module. The sensor reading of the angle closely matches the target angle when placed at that angle. Using the closed loop PID controller, guided us in how to reach the target angle and match it closely as possible. The log of the angles were put into a spreadsheet which then a graph was generated. The graph is shown in the ***PID_Data/*** directory. The project timeline and tasks were done using GitHub's Projects. The source code can also be found on GitHub. The demo of the project can be seen here.

References

- MPU-6000 Datasheet
    - Invensense, TDK. (2015). MPU-6000 and MPU-6050 Product Specification. Revision 3.4. Retrieved from https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
- RealDigital BooleanBoard Documentation:
    - RealDigital. (2024). BooleanBoard Documentation. Retrieved from https://www.realdigital.org/doc/02013cd17602c8af749f00561f88ae21
- Xilinx IIC API Documentation
    - Xilinx. (2024). IIC API Documentation. Retrieved from https://xilinx.github.io/embeddedsw.github.io/iic/doc/html/api/index.html
- FreeRTOS Documentation
    - FreeRTOS. (2024). Real Time Kernels. Retrieved from https://www.freertos.org/a00106.html
- Xio Technologies (2024). Fusion: An open-source sensor fusion library. [Online]. Available: https://github.com/xioTechnologies/Fusion

Appendices

Appendix A: Embedded System



Appendix B: Outside code sources

B.1 Fusion Library

The source code for the Fusion library, used for sensor data fusion in this project, can be found on GitHub: https://github.com/xioTechnologies/Fusion.

This library is written in C and provides an algorithm for combining gyroscope, accelerometer, and magnetometer data to determine a sensor's orientation. It is also available as a Python package, making it suitable for various development environments. The Fusion library is licensed under the GNU General Public License v3.0 and is a valuable resource for embedded systems projects requiring sensor fusion.