***Phil Nevins******ECE 544 – Embedded Systems Programming with FPGAs******March 1, 2025***[*https://github.com/Dawgburt/ECE544-Project2*](https://github.com/Dawgburt/ECE544-Project2)

# PID Controller Design Report

## **1. Introduction**

This document outlines the design, implementation, and testing of a PID controller for luminosity regulation using the TSL2561 light sensor and PWM-driven LED on a Nexys A7 FPGA board. The PID controller dynamically adjusts LED brightness to maintain a user-defined light intensity (lux level). The system operates in real-time using FreeRTOS.

## **2. System Overview**

### **2.1 Components**

* **TSL2561 Light Sensor**: Measures ambient light intensity.
* **Nexys A7 FPGA Board**: Runs the PID controller and handles input/output.
* **PWM-controlled LED**: Adjusts brightness based on PID output.
* **FreeRTOS**: Provides task scheduling and real-time processing.
* **Push Buttons & Switches**: Used for setpoint adjustment and PID tuning.
* **7-Segment Display**: Displays current lux and target setpoint.

## **3. User Controls & Configuration**

### **3.1 Switches**

| **Switches** | **Function** |
| --- | --- |
| **[7:6]** | Select PID parameter to modify (Kp, Ki, Kd) |
| **[5:4]** | Set increment size (1, 5, 10) |
| **[3]** | Toggle setpoint adjustment mode |
| **[2]** | Enable/Disable Derivative control (D) |
| **[1]** | Enable/Disable Integral control (I) |
| **[0]** | Enable/Disable Proportional control (P) |

### **3.2 Buttons**

| **Button** | **Function** |
| --- | --- |
| **BtnU** | Increase selected parameter |
| **BtnD** | Decrease selected parameter |
| **BtnC (Center Button)** | Resets PID values and target\_lux to default settings |

## **4. Software Implementation**

### **4.1 Tasking Model**

| **Task** | **Function** |
| --- | --- |
| **PID Task** | Computes PWM based on PID algorithm |
| **Sensor Task** | Reads lux value from TSL2561 |
| **Display Task** | Updates 7-segment display with setpoint and current lux |
| **Input Task** | Reads switches/buttons and updates parameters |

### **4.2 PID Algorithm**

The PID controller operates using the following equation: Output=(Kp×Error)+(Ki×Integral)+(Kd×Derivative)Output = (Kp \times Error) + (Ki \times Integral) + (Kd \times Derivative)

Where:

* **Error = Setpoint - Current Lux**
* **Integral Term** prevents steady-state error but requires windup prevention.
* **Derivative Term** helps with stability and reducing overshoot.
* **Output is scaled to a PWM range of 0-255**.

### **4.3 Code Implementation**

* pid.c: Implements the PID algorithm.
* PID\_Controller.c: Handles system initialization and task scheduling.
* tsl2561.c: Driver for reading lux values from the sensor.
* nexys4io.c: Controls the FPGA I/O, including buttons, switches, and display.

## **5. Challenges & Solutions**

### **5.1 Display Scaling Issue**

* **Problem**: The displayed current\_lux value did not match expected values.
* **Solution**: We scaled the output of current\_lux by 2, which corrected the issue.
* **Assistance**: This was resolved with the help of ChatGPT debugging.

### **5.2 Nexys4IO Initialization Failure**

* **Problem**: Initialization of Nexys4IO caused heap and stack allocation issues, leading to system crashes.
* **Solution**: Increased heap size in BSP settings, which resolved the problem.

### **5.3 PID Tuning**

* **Problem**: Initial PID values led to poor tracking and oscillations.
* **Solution**: Through iterative testing and discussion with ChatGPT, we determined optimal values for **Kp, Ki, and Kd**.

## **6. Results & Performance**

### **6.1 System Behavior**

* The PID controller successfully adjusts the LED brightness to maintain the target lux level.
* The system responds dynamically to changes in ambient light and user input.
* **Setpoint Tracking**: The LED brightness adapts in real-time to keep current\_lux close to setpoint.

### **6.2 Performance Metrics**

| **Metric** | **Value** |
| --- | --- |
| **Max Response Time** | ~100ms |
| **Steady-State Error** | < 5% |
| **Max Lux Readable** | ~999 |
| **PWM Range** | 0-255 |

## **7. Conclusion**

This project successfully implemented a real-time PID controller for LED brightness regulation. The system utilizes FreeRTOS for task scheduling and achieves closed-loop control using the TSL2561 sensor. The design process involved resolving initialization failures, debugging display scaling, and refining PID tuning. The final implementation meets design expectations and provides a reliable control system for dynamic luminosity adjustment.

## **8.** **Future Work**

* Implement an **auto-tuning PID algorithm** for improved performance.
* Expand the system to support **multiple sensors** for more precise light control.
* Enhance the **graphical display interface** for better real-time monitoring.
* Optimize **power consumption** by dynamically adjusting PWM frequency based on need.

## **9. References**

**TSL2561 Light Sensor Datasheet**<https://cdn-shop.adafruit.com/datasheets/TSL2561.pdf>

**Xilinx FreeRTOS Documentation**

<https://www.freertos.org/RTOS-Xilinx.html>

**Nexys A7 FPGA Reference Manual (Digilent)**

<https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual>

**PID Controller Explanation (Control System Theory)**

<https://www.controleng.com/articles/pid-theory-explained/>

**ChatGPT Assistance for Debugging and PID Tuning**

<https://openai.com/chatgpt>

**Embedded Systems Heap & Stack Size Configuration in Xilinx SDK**

<https://support.xilinx.com/s/article/52848>

**Understanding Integral Windup in PID Controllers**

<https://en.wikipedia.org/wiki/Integral_windup>

**Using FreeRTOS for Real-Time Embedded Applications**

<https://freertos.org/Documentation/RTOS_book.html>