Christian J. Busca

May 25, 2025

CS 350 Milestone Two – UART LED Control Lab Report

**Lab Questions: Technical Analysis**

**1. Why do both the SerialTest-Write.py and SerialLightControl-Client.py scripts use the encode() method of the string datatype when writing data to the serial port?**

The encode() method converts a human-readable string into a byte stream, which is the format required by the pyserial library to transmit data over a serial interface. Since UART communication operates at the byte level, encoding textual commands such as "ON" and "OFF" ensures proper data formatting for transmission and accurate interpretation by the receiving system.

**2. Why does the SerialTest-Read.py script use the decode() method of the string datatype when reading the data from the serial port?**

The decode() method is necessary to transform the incoming byte stream received from the serial interface into a readable string format. This allows the server application to process the received command and respond accordingly. Without decoding, the program would be unable to logically evaluate the input data for command execution.

**3. What is the purpose of the try/except block in both the SerialLightControl-Client.py and SerialLightControl-Server.py scripts?**

The try/except construct facilitates robust error handling within both scripts. It allows the applications to gracefully manage common exceptions such as keyboard interrupts (CTRL+C), device communication errors, and data decoding failures. Incorporating this structure helps maintain system stability and ensures that resources such as the GPIO and serial interfaces are properly released, even when the program is interrupted.

**4. Why is it necessary to ensure that the GPIO pins are returned to their original state at the end of the program run?**

Ensuring that the GPIO pins are reset upon program termination is critical for maintaining hardware integrity and predictable behavior. Failure to return the pins to their default states may leave connected components in an unintended or active condition, potentially causing damage or electrical conflicts. The GPIO.cleanup() function ensures a safe reset of all GPIO resources.

**Troubleshooting Summary: Diagnostic Process and Resolution**

During the implementation of this milestone, several challenges were encountered related to UART communication between the Raspberry Pi and the USB-to-TTL interface. Below is a refined overview of the successful strategies employed to resolve these issues.

UART Configuration and Validation

Utilized raspi-config to:

Disable the login shell on the serial interface.

Enable serial hardware support (UART).

Edited /boot/cmdline.txt to remove the serial console directive:

console=ttyS0,115200

This prevented system-level interference with UART communication.

Verified serial hardware activation post-reboot via:

dmesg | grep ttyS0

Corrected USB-to-TTL cable connections as follows:

TX (White) → GPIO15 (Raspberry Pi RX)

RX (Green) ← GPIO14 (Raspberry Pi TX)

GND (Black) → Raspberry Pi GND

The red VCC wire was intentionally left disconnected to prevent overvoltage damage.

Serial Communication Testing

Confirmed bi-directional UART communication by executing:

SerialTest-Write.py on /dev/ttyUSB0 (sending side).

SerialTest-Read.py on /dev/ttyS0 (receiving side).

Once basic transmission was verified, enhanced control scripts were implemented:

Client script was updated to present a numeric menu (1–4) for input clarity and command consistency.

Server script included read buffering, command parsing, and robust exception handling.

Functional Validation

Issued commands such as "ON", "OFF", and "EXIT" via the client script.

Observed the following behaviors confirming system functionality:

Server correctly recognized and printed each command.

The LED responded accurately to the corresponding GPIO signal.

The program exited cleanly upon receiving termination commands, and hardware resources were properly released.

This structured troubleshooting process ensured stable UART communication, accurate LED control, and a robust implementation suitable for embedded system applications.