I2C Controlled PWM Driver

Liukee Liu, **liukee@pdx.edu**
Mohammad Alshaiji, **alshaiji@pdx.edu**
Riley Cameron, **rileycam@pdx.edu**
Sal Esmaeil, **esmaeil@pdx.edu**

ECE 571
Introduction to SystemVerilog for design and Verification
Venkatesh Patil & Brian Cruikshank

Electrical and Computer Engineering Department
Maseeh College of Engineering and Computer Science

Portland State UNIVERSITY

Project Final Presentation
I2C Controlled PWM Driver:

Introduction

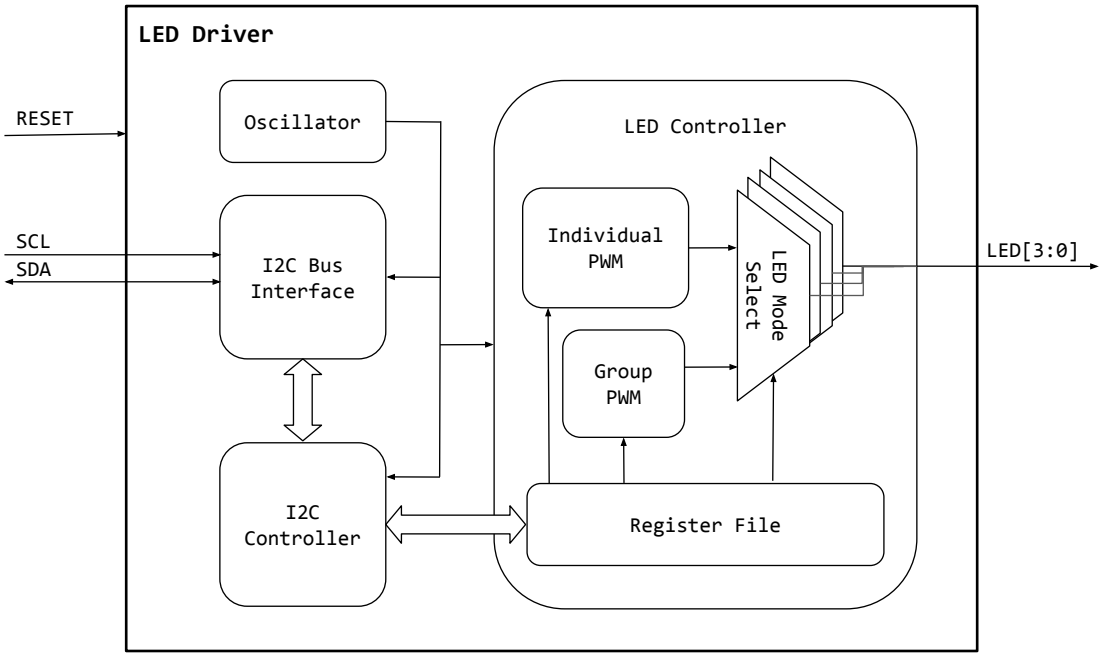
- What the project is about?
  - Designing a simplified I²C-controlled LED driver
  - Based on the PCA9632 device architecture
  - Converts register values into LED brightness and operating modes
  - Uses SystemVerilog modules for timing, PWM, and output control
- Applications of the project/design
  - Used in embedded systems and IoT devices for status and alert LEDs
  - Common in consumer electronics such as keyboards, routers, and displays
  - Helpful in automotive dashboards and industrial control panels
  - Useful for learning and prototyping digital hardware designs

Process

Step-by-step workflow followed for designing, simulating, and integrating the I²C-controlled PWM LED driver

1. Requirements & Research
   a. Reviewed PCA9632 datasheet
   b. Selected implementable digital features
2. System Partitioning
   a. Divided design into I²C interface, I²C controller, LED/PWM logic, and oscillator
3. Module-Level Design
   a. Implemented bit-level I²C logic and START/STOP detection
   b. Designed controller and register decoding
   c. Built PWM engines, group logic, and output mux
4. Verification & Debugging
   a. Individual module testbenches
   b. Tested I²C transfers and register operations
   c. Checked register writes, reads, and LED output modes
5. System Integration
   a. Connected all modules in top-level driver
   b. Ran full waveform simulations
   c. Verified PWM, group modes, and sleep behavior
6. Final Validation
   a. Checked correct register behavior
   b. Cleaned code and captured waveforms
   c. Completed report and presentation

Process



I²C Bus Interface receives SCL/SDA, detects START/STOP, shifts data in/out

I²C Controller decodes bytes, determines register address/data

Register File stores MODE, PWM, and LEDOUT settings

Oscillator provides timing for all PWM engines

LED Controller generates Individual and Group PWM signals

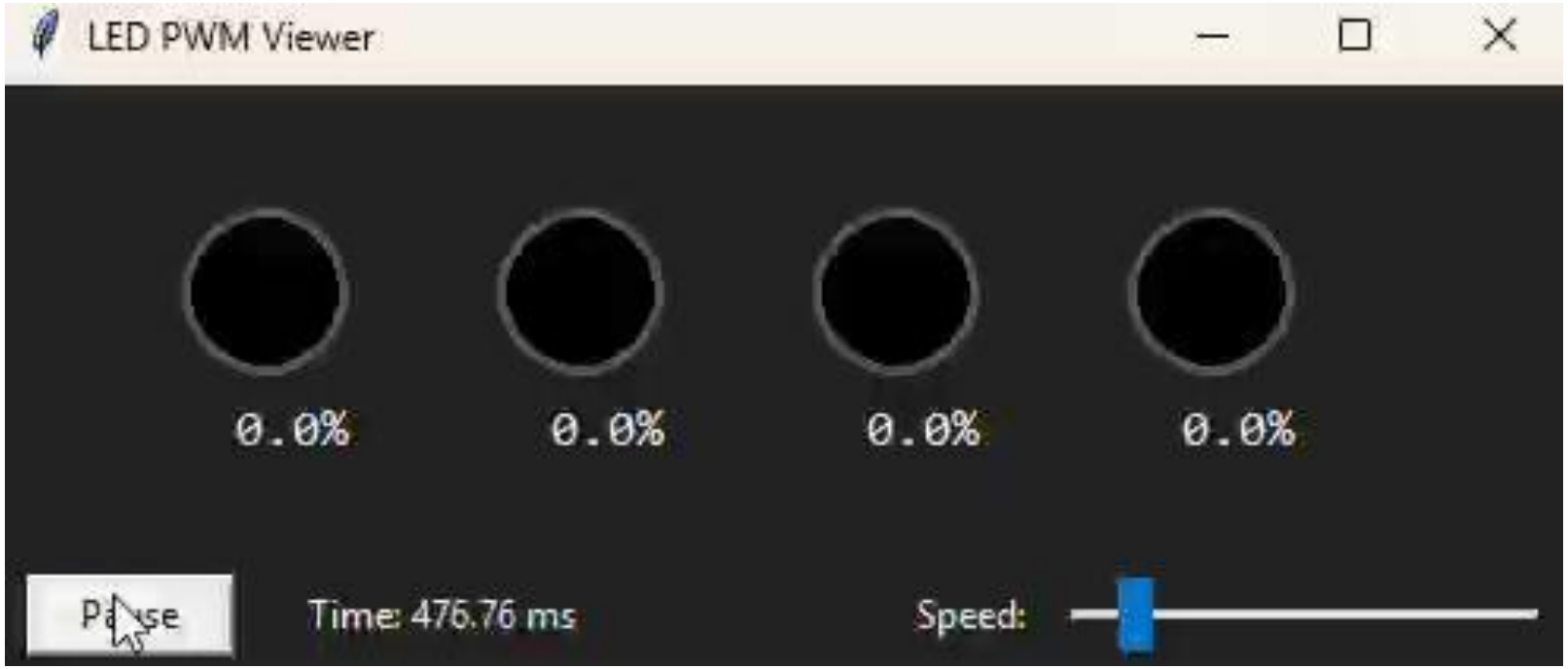
LED Mode Select picks the correct signal for each LED output

LED[3:0] updates based on configured mode and duty cycle

Demonstration

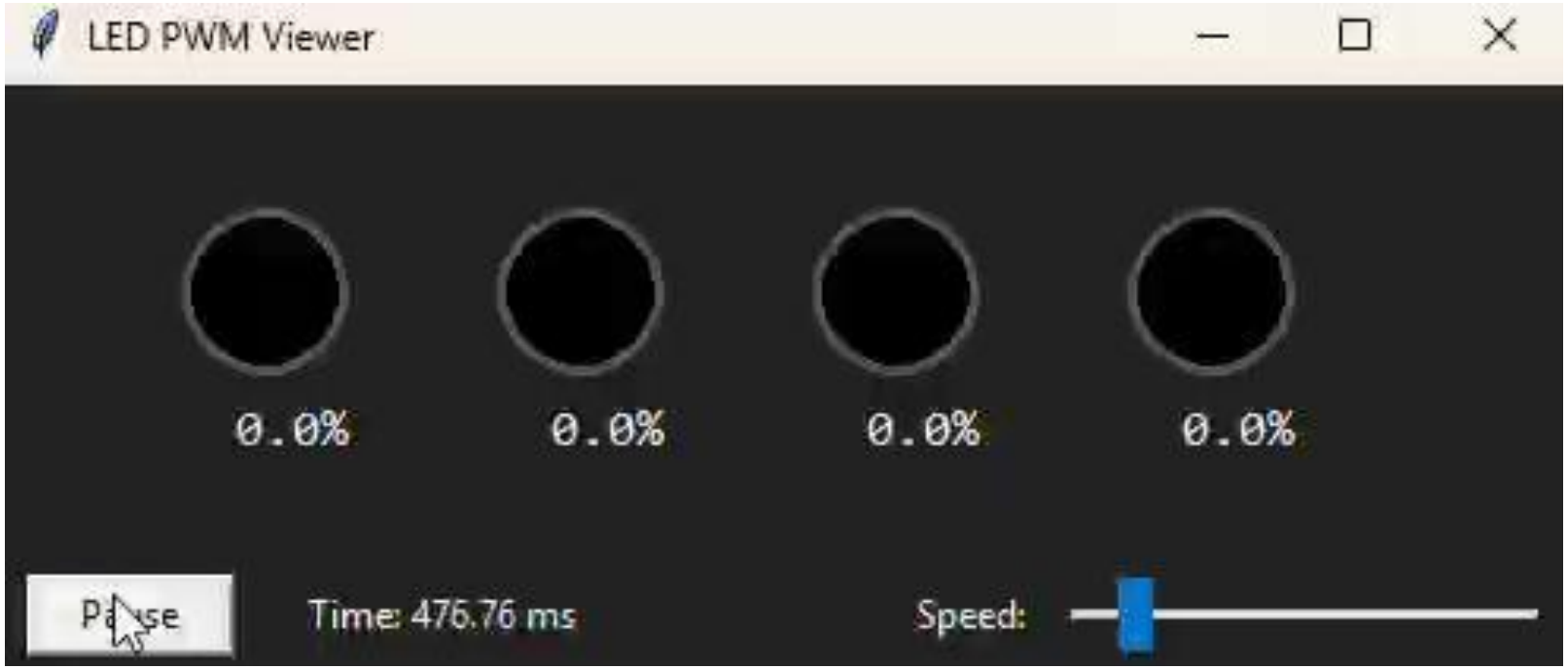
**Demo videos were created by exporting the output signals to a wave.vcd file and parsing it with a python script. Full Video: https://drive.google.com/file/d/1cur0XTjT6hOeXpAsOVk7cibnPJeYMFsJ/view?usp=sharing*

- Off -> On



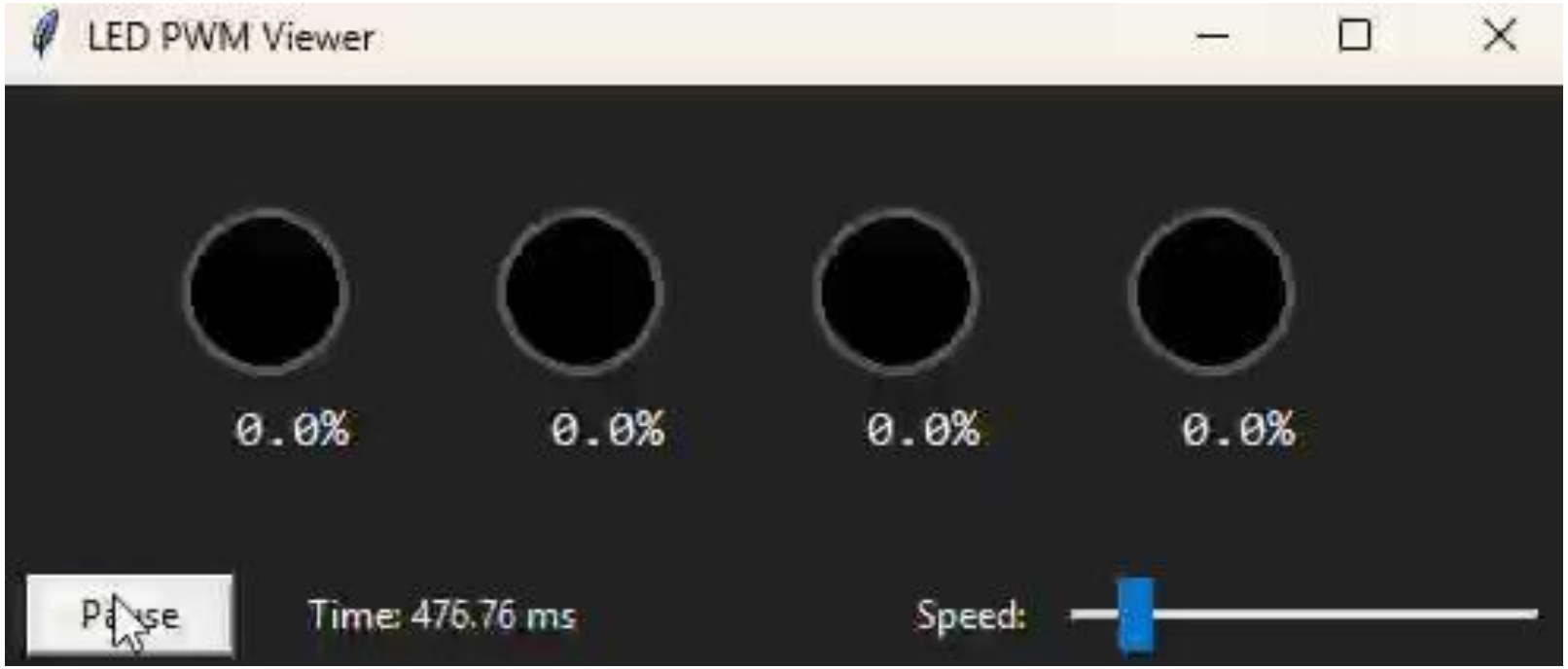
Demonstration

- Individual Mode



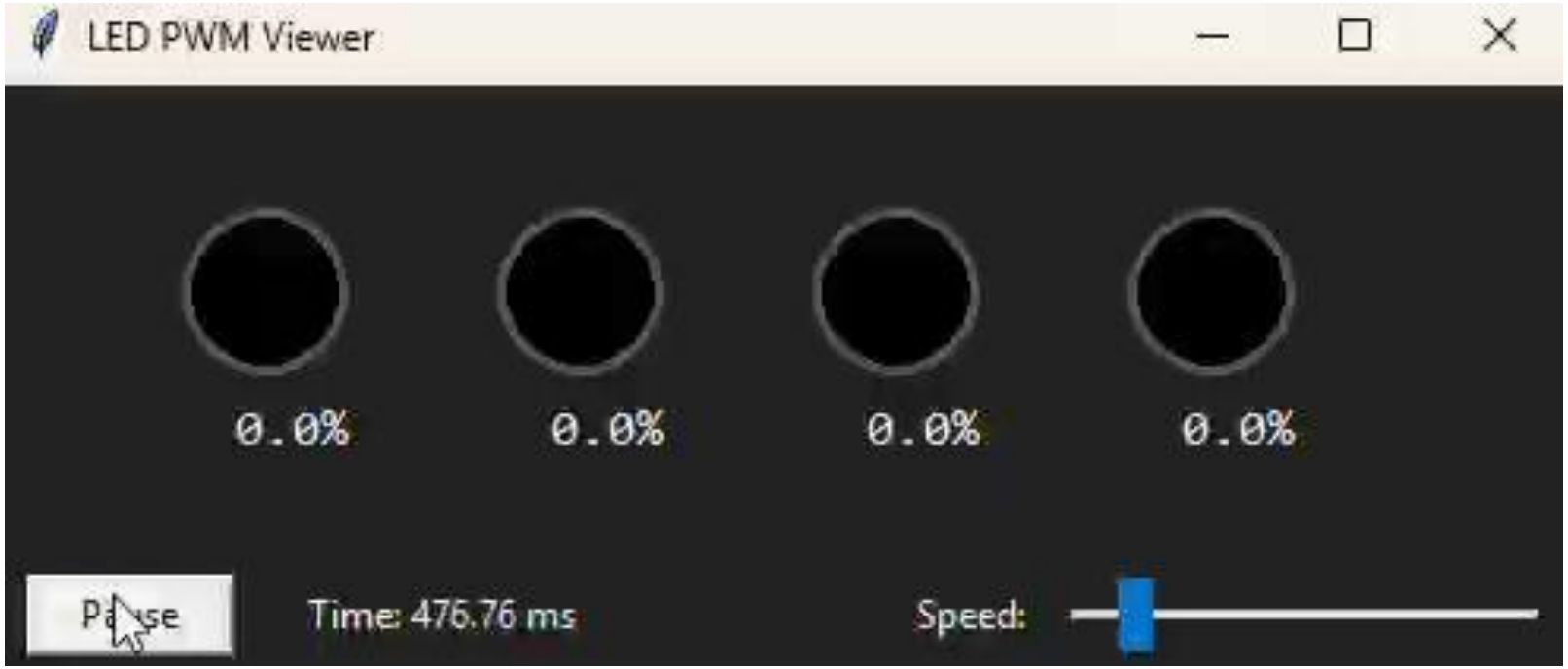
Demonstration

- Group Dimming Mode



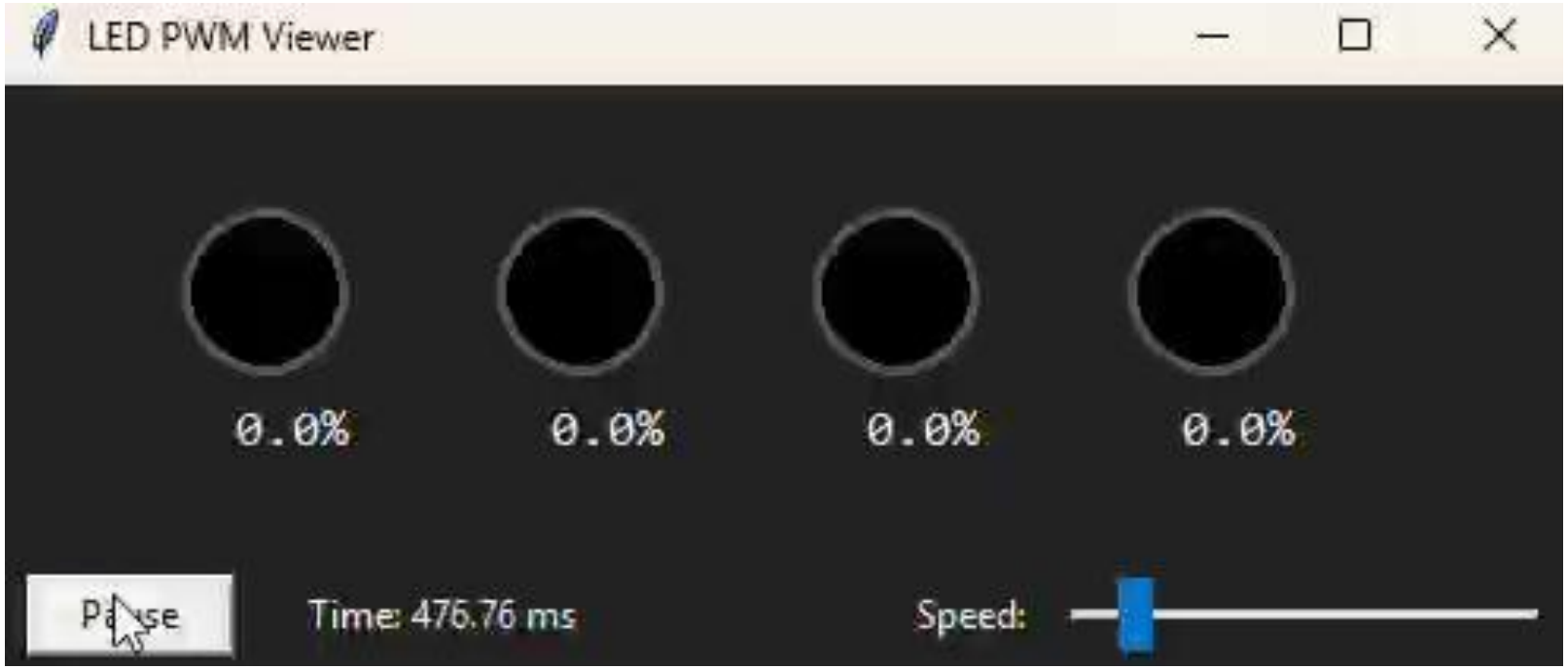
Demonstration

- Group Blinking Mode



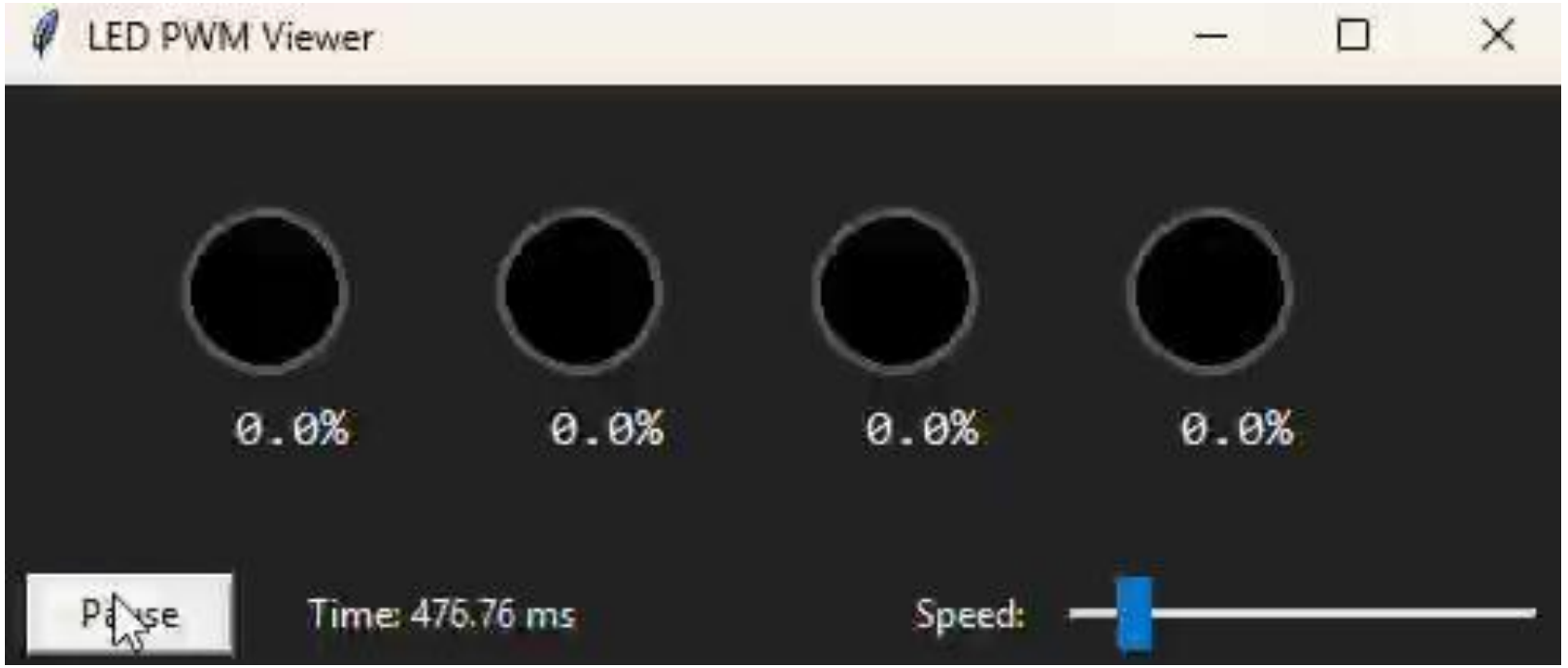
Demonstration

- Sleep Mode



Demonstration

- Inverted Output



Demonstration

- Results from end of transcript: **0 errors**

```
# -- Invert Output --
# [Write-Check PASS]    REG_LEDOUT  addr=7  data=10101010
# [Write-Check PASS]    REG_MODE    addr=0  data=00000100
# [Write-Check PASS]    REG_MODE    addr=0  data=00000000
#
# -- Register Read --
# [Read-Check PASS] REG_PWM0    data=11111111
# [Read-Check PASS] REG_PWM1    data=11000000
# [Read-Check PASS] REG_PWM2    data=10000000
# [Read-Check PASS] REG_PWM3    data=01000000
# [Read-Check PASS] REG_GRPFREQ data=11111111
# [Read-Check PASS] REG_GRPPWM  data=10000000
# [Reset]
# [Read-Check PASS] REG_PWM0    data=00000000
# [Read-Check PASS] REG_PWM1    data=00000000
# [Read-Check PASS] REG_PWM2    data=00000000
# [Read-Check PASS] REG_PWM3    data=00000000
# [Read-Check PASS] REG_GRPFREQ data=00000000
# [Read-Check PASS] REG_GRPPWM  data=00000000
#
# === ALL TESTS PASSED ===
```

Challenges

- Challenges Faced & Resolutions
  - False STOP Condition: Readback failed on even addresses due to an SCL/SDA race condition.
    - Resolution: Implemented "bus parking" in the testbench to force SCL low before data transitions, preventing invalid stop detection.
  - Time Precision Mismatch: Simulator rounded fractional delays to zero, causing simultaneous signal transitions.
    - Resolution: Enforced a strict timescale directive for accurate microsecond timing.
  - FSM Rigidity vs. Standard I2C: The simplified 4-state FSM prevented standard "Current Address Reads."
    - Resolution: Enforced a "Write-Pointer-then-Read" protocol. This design choice ensures deterministic register access and minimizes silicon footprint, intentionally deviating from standard I2C for efficiency.

- Future Work & Open Items
  - Standard Compliance: Expand FSM to support Repeated Start and ACK/NACK handshaking to enable compatibility with broader off-the-shelf masters.
  - Hardware Realization: Perform Synthesis and Static Timing Analysis (STA) to validate asynchronous signal synchronization on physical hardware.

- What Worked, What Did Not, Learnings
  - What Worked: The streamlined architecture (omitting complex ACK/NACK logic) proved highly effective for this write-intensive application, reducing gate count. The modular design also successfully isolated physical layer issues.
  - What Did Not: Reliance on default simulation time units caused misleading debugging data; combining high-speed I2C logic with long-duration blink tests created unmanageable waveform files (>6GB).
  - Learnings: Testbench logic must model physical bus physics (hold times), not just logical transitions, to prevent race conditions in mixed-speed simulations.

Distribution

Liukee – I²C Bus Interface (Bit-level)
Mohammad – I²C Controller (Byte-level)

Riley – LED Controller & Group Logic
Sal – PWM Engines & Mode Muxing

Reference

[1] NXP Semiconductors, "PCA9632 4-bit Fm+ I²C-bus low power LED driver - Product data sheet, Rev. 6, 21 September 2021." https://www.nxp.com/docs/en/data-sheet/PCA9632.pdf

[2] Wikipedia Contributors. "I2C." Wikipedia, Wikimedia Foundation, https://en.wikipedia.org/wiki/I2C

Question?

Thank you!