Summary Description: Reflecting On Embedded Systems Concepts With Arduino And FreeRTOS

Tags: electronics; software; FreeRTOS; Arduino; nano; stepper motor; UART

Why I did this:

In one of my university’s courses, we learned about real-time embedded systems. Real-time operating systems are systems that strive to complete tasks within hard time limits; these can be both scheduled and interrupt triggered tasks. They are important systems because not only do they express high performance with their ability to respond, but they are suitable for mission or safety critical systems (like vehicle accident black boxes, aircraft systems, medical devices).

FreeRTOS is an example of a real-time operating system (it’s in the name). I learned that you could import the FreeRTOS library on the Arduino IDE, so I wanted to reflect on the concepts in learned in my course and how I could apply them to an Arduino project.

The main concepts our course focused on were: multi-tasking/concurrency and priorities; queue containers; UART and SPI communications; device control (motors with opto-isolator, displays, buttons, leds); hardware interrupts and polling;

(summary pic of system)

Design Walkthrough:

Parts: 2x Arduino microcontroller (nano); FreeRTOS library; 2x OLED displays; 5x LEDs; stepper motor (and ULN2003 driver)

I wanted to showcase as much of the concepts I listed above, so I made a scene where one Arduino nano is holding a hardware button and 2 prioritized (but concurrently acting) tasks that log and trigger a UART communication (to a second Arduino nano) when the button is pressed. The first nano, when hitting the button, will increment the display with ‘number of button presses’, and trigger a led array to light sequentially along the array (1 to 5, after a second each); when the final led is lit, a second display number will increment the ‘UART counts’ and the UART communication signal will send to the other nano.

The second Arduino will receive the UART communication to display on the OLED an increment of the ‘UART counts' log and trigger a stepper motor’s rotation. After the stepper is done a revolution, the display will increment the ‘number of stepper revolutions’.

The first nano’s display will also show an incrementing fast counter to act as a ‘heartbeat counter’.

This dual nano scene performs:

- multi-tasking/concurrency with the two tasks on the first Arduino that:

Task 1: increment the heartbeat counter and handle displaying all OLED information;

Task 2: handle the LED array animation and send UART communication to the other nano;

- button triggering interrupts that increment a counter and trigger the led array.

- UART communication (with the aforementioned second task).

- stepper control from the second nano.

The scene is missing concrete displays of:

- task prioritizations (the ‘led array’ task is higher priority that the ‘OLED’ task, but they don’t really show their realized use in this example scene).

- queue containers (since I was passing single and int data between tasks and UART lines, it isn’t the best example for me to use queues).

- SPI communications (I was using I2C for displays and UART for Arduino comms but didn’t find an appropriate example to use SPI).

- polling (I didn’t have a good example of hardware attention checking).

I can probably make another scene on a different article to showcase these missing features, but for now I will use what I included.

(pics of system)

- Button is pressed, triggering an interrupt that increment the first counter (Display 1, left number).

- Increment causes another task (LED array) to animate; after completing, causes an increment of another counter (Display 1, middle number) and trigger UART communication to other Arduino.

- Nano receives UART and increments another counter (Display 2, left number).

- Increment causes stepper motor to revolve; after completing, causes an increment of another counter (Display 2, middle number).

- First display has counter (Display 1, right number) that acts as a heartbeat, but also displays concurrency is its increment isn't interrupted when the led array is activated.

Lessons Learned and Future Changes:

Power of concurrency. Arduino alone is great for hardware/peripheral control/communication, and interrupts, but with adding FreeRTOS, you can make multiple tasks seem to work at the same time; this helps improve the value of a cheap single board microcontroller.

Opto-mise it. I had at first attempted to use an opt-isolator with the stepper motor, but it wasn’t receiving the input signals, so I hard wired the inputs between the nano and stepper. This is bad practise as motors can induce back voltage through the signal lines that will fry the board, but I was ok with risking it for a cheap nano for the sake of this demo.

References:

FreeRTOS website: <https://www.freertos.org/index.html>

FreeRTOS on Arduino: <https://www.arduino.cc/reference/en/libraries/freertos/>

Arduino UART tutorial: <https://docs.arduino.cc/learn/built-in-libraries/software-serial>

Stepper tutorial: <https://www.arduino.cc/reference/en/libraries/stepper/>

Robotics Back-End’s Arduino button interrupt tutorial: <https://roboticsbackend.com/arduino-interrupts/>