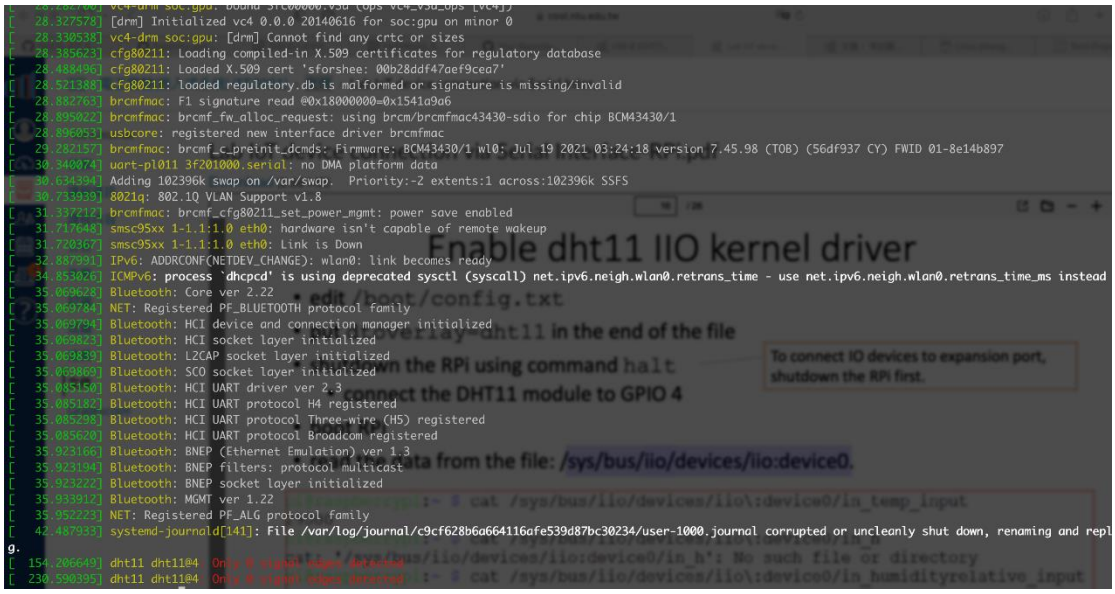
Lab8 report
B09901055 楊倢綺

Experiment result:



1. What is Linux IIO subsystem? Please explain IIO in terms of its purpose and user interface provision.

   The Linux Industrial I/O (IIO) subsystem is a kernel subsystem that provides support for devices that handle analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC), as well as other types of data acquisition devices. The IIO subsystem is designed to be a generic framework for supporting a wide range of sensors, converters, and similar devices used in embedded systems.

   The primary purpose of the IIO subsystem is to provide a unified interface for data acquisition devices. This includes sensors that measure physical quantities (such as temperature, pressure, or acceleration) and converters that transform analog signals to digital values or vice versa. Also, it abstracts the underlying hardware details, providing a consistent API for application developers regardless of the specific hardware being used. This abstraction allows developers to write applications that can work with various IIO-compatible devices without needing to know the intricacies of each device. Finally, IIO is integrated into the Linux kernel, ensures that the subsystem is well-supported and benefits from the Linux kernel's development and maintenance processes.

User Interface Provision:

a. sysfs Interface: Device-specific information, configuration options, and data can be accessed and manipulated by interacting with the sysfs entries associated with IIO devices.

b. Character Device Interface: IIO also supports a character device interface, allowing users to interact with devices through standard file operations (read, write, etc.). This interface is more flexible than sysfs and provides a way to stream data from sensors.

c. Industrial I/O Framework API: For application developers, the IIO subsystem provides a programming interface through the Industrial I/O (IIOD) framework. This framework includes a set of APIs that applications can use to communicate with IIO devices, retrieve sensor data, and configure device settings.

2. How is the efficiency difference when compared between interrupt-driven I/O and programming I/O (polling I/O)?

Interrupt-Driven I/O use CPU time more efficiently, since the CPU can perform other tasks while waiting for I/O operations to complete. Also, it can have lower latency, for the system can respond quickly to I/O events, as the CPU is notified as soon as data is ready. However, Interrupt-Driven I/O can suffer from more overhead while handling interrupts, especially when there are many interrupts.

On the other hand, Polling I/O is more predictable and is free from unexpected interruptions. Also, it is simpler to implement. But the CPU utilization can be less efficient, as the CPU may spend time repeatedly checking the status of the device even when there is no data to process. And because CPU might not be aware of the device immediately, there may be longer latency.