Interrupt-Driven vs Polling in Embedded Design

Embedded systems often need to respond to external events—such as user input, sensor readings, or communication data. The choice between **interrupt-driven** and **polling-based** mechanisms to handle these events is fundamental to system performance, power consumption, and responsiveness.

🔁 What is Polling?

Polling is a **synchronous** method where the processor continuously checks (or "polls") a device or register to see if an event has occurred.

📌 How It Works

- CPU repeatedly reads a status register in a loop.
- If the event hasn't occurred, the CPU keeps checking.
- When the event occurs (e.g., data ready), the CPU handles it.

✅ Pros

- Simple to implement.
- Predictable timing.
- No concurrency issues; all logic is in a single thread/loop.

❌ Cons

- Inefficient for low-frequency events (wastes CPU cycles).
- Higher power consumption.
- Reduces processor availability for other tasks.

Example:

```
while (1) {
    if (UART_DataReady()) {
        char c = UART_Read();
        process_char(c);
    }
}
```

🧠 Choosing Between the Two

Criteria	Polling	Interrupt
Event Frequency	High and consistent	Low or sporadic
Power Sensitivity	Not ideal	More efficient
Code Simplicity	Simple	Complex
CPU Usage	Constant	Only on event
Real-Time Responsiveness	Poor	Excellent (with priorities)

🧩 Hybrid Approach

Some systems use a **hybrid** model:

- Use interrupts to wake the CPU.
- Use polling in tight timing loops (e.g., reading multiple bytes from SPI quickly).

🧵 Summary

Aspect	**Polling**	**Interrupt**
CPU Efficiency	❌	✅
Simplicity	✅	❌
Real-Time Use	❌	✅
Power Saving	❌	✅

Choose **interrupts** when:

- Events are asynchronous.
- You want better power/performance efficiency.
- You have limited CPU resources.

Choose **polling** when:

- You need deterministic timing.
- The system has a simple loop and no multitasking.
- Peripheral access is fast and frequent.