arm Education

Embedded Systems Essentials with Arm: Getting Started

Module 4

KV1 (4): Interrupts

There are two ways to detect whether a switch is pressed or not: polling and interrupts.

Polling is the use of software to check the status of the switch regularly. However, polling is slow, inefficient and poorly scalable because it uses up CPU resources to check on a switch that may or may not have been pressed.

A better option is an interrupt, which uses special hardware in the MCU to detect a switch being pressed, and runs a piece of code in response. Interrupts are fast, efficient and scalable because they are only triggered when the switch is pressed.

The code forced into execution by an interrupt is called an interrupt service routine, or ISR.

The interrupt service routine directly executes addressed code.

When an interrupt is triggered, it stops or interrupts nearly everything, depending on its priority and configuration, in order to execute its own code.

After stopping the main code, the processor executes some hardwired routines, such as saving registers. The processor then executes the ISR, including a return-from-interrupt instruction at the end, enabling the regular program execution to continues afterwards.

Interrupts are a form of hardware-triggered asynchronous routine. In other words, an external device or hardware triggers a software routine that can happen asynchronously, meaning anywhere in the program. Once triggered, the code can be executed.

The use of asynchronous routines is fundamental for microcontroller programming. Efficient event-based processing, rather than polling, allows the controller to operate in a passive mode for as long as possible. The controller can respond to events quickly, regardless of program state, complexity, and location.

Based on this principle, hardware interrupts can emulate multithreading without the need for an operating system or task scheduler. The basic operation runs in a main task. Whenever something important happens, this task can be interrupted for a prioritized one.

Let’s consider an example.

Using a switch, we want to change the color of an RGB LED when it is pressed. To do this, we apply an external switch to a GPIO pin on the controller. In order to prevent the GPIO pin from being ‘open’, we will use a pull up resistor, as shown in the circuit diagram.

In this example, the RGB LED is connected to Vcc with the same resistor values. This is because we are assuming that the resistor values are enough to allow the voltage for each LED to be in an acceptable range.

P1, P2, and P3 are connected to the controller. In this example, the controller selects the color of the RGB LED by ‘current sinking’ the appropriate LED, meaning that the current flows from Vcc through the selected LED to a ground connection.

The program to operate this switch can be divided into three segments: the main code, the ISR and a global variable.

The main code does the initialization, and lights the RGB LED according to the global variable value.

When the switch is pressed, the ISR is triggered and increments the global variable value. This causes the main code to change the color of the RGB LED.