Interrupt Service Routines

CM0506 Small Embedded Systems

Dr Alun Moon

11th July 2016

Interrupt Service Routines (ISR)

Interrupt Service Routines (ISR) are unlike normal functions. Normal functions have an entry-point that is called by other functions (with parameters), and an exit-point that returns control (and a value) to the calling routine. The point in a program where a function is called is known and predicable. An ISR by its nature departs from these patterns.

An ISR:

- Takes no parameters passed into it.
- Returns no values
- Can occur at any (random) time – is *asynchronous*

The prototype, as seen in a header file for an ISR is:-

```
void someISR(void);
```

Code size

An ISR can occur at any time, and most likely happens between critical operations[1], there is a need to keep the time the ISR takes to execute to a minimum.

[1] For example between the *increment* and *test* of a `for` loop

In event driven systems, most of the behaviour can occur in interrupts. While the execution time may not be as so constrained, care must be taken that the execution-time of the ISR is known in relation to the time available for it to execute.

If the ISR takes too long to execute, the behaviour of the system can be compromised. *Priority-inversion* can occur where a low priority ISR can block a higher-priority interrupt from occuring. Nested interrupts and interrupt masks can help aleviate symptoms here. If the ISR takes too long to handle high-frequency interrupts, then successive interrupts may not be processed.

It follows that slow operations, such as drawing to the display, should be avoided in ISRs.

Communications

The ISR takes no parameters and returns no values. This makes communicating with the rest of the program difficult. The only way to pass information from the IST to other parts of the program is to use *global variables*. This introduces many potential problems so extreme care is needed. Good use of `static` and scoping rules restricts access to only those parts that need it. Ideally use variables that can be updated in a single (assembly) operation.

There may be other cases, such as an ISR handling communications via serial, ethernet, or CAN. In these cases a shared buffer can be used, and care taken to avoid the problems raised by simultaneous access of producer and consumer parts of the program.

Exercise 1: Git download of initial code

```
$ git
```

Question 1: enum and switch

Solution