Polling vs Interrupts

CM0506 – Small Embedded Systems

Dr Alun Moon

Department of Computer and Information Science

Lecture 5

Polling

Periodically checking devices

```
while(1) {
  wait(200);
  if (buttonPressedAndReleased(&pin[JLEFT])) {
    flashing[LED1] = !flashing[LED1];
  }
  if (buttonPressedAndReleased(&pin[JDOWN])) {
    flashing[LED2] = !flashing[LED2];
  }
}
```

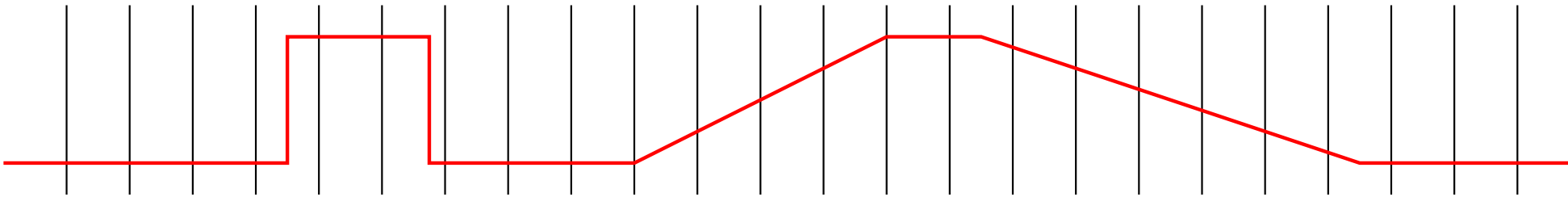
- Check for condition of device
- if data is available
- read the device or do something
- repeat

Polling

```
bool buttonPressedAndReleased(gpioPin_t *pin) {
  bool result = false;
  if (gpioPinVal(pin) == 0) {
          while (gpioPinVal(pin) == 0) {
                  /* skip */
          }
          result = true;
  }
  return result;
}
```

- Software has to keep checking status of device

Polling



Level sensitive Test value of input at sample points

- what is 1 or 0

Edge sensitive Test for changes in level

Rising Edge level goes from low (0) to high (1)

Falling Edge level goes from high (1) to low (0)

What happens if the signal changes between sample points?

Polling

- Also called Synchronous I/O
- For many applications: most polled samples will mean "nothing happening here"
- Considerable time may be spent executing code for long intervals when no input is available to work on
- This may be OK if there is nothing else to do . . . but . . .

Polling more than one input

- What if there are many inputs which have to be sampled?

Polling more than one input

- What if there are many inputs which have to be sampled?
- The application will have to scan each input and react if the input is active (interesting)

Polling more than one input

- What if there are many inputs which have to be sampled?
- The application will have to scan each input and react if the input is active (interesting)
- Manageable under some circumstances

Polling more than one input

- What if there are many inputs which have to be sampled?
- The application will have to scan each input and react if the input is active (interesting)
- Manageable under some circumstances
- It may become difficult if the number of inputs is large and other tasks require processing.

Polling more than one input

- What if there are many inputs which have to be sampled?
- The application will have to scan each input and react if the input is active (interesting)
- Manageable under some circumstances
- It may become difficult if the number of inputs is large and other tasks require processing.
- System may be slow to react to inputs

Polling more than one input

- What if there are many inputs which have to be sampled?
- The application will have to scan each input and react if the input is active (interesting)
- Manageable under some circumstances
- It may become difficult if the number of inputs is large and other tasks require processing.
- System may be slow to react to inputs
  - you have to wait until the next sample for that input

Interrupts

- A peripheral device may be connected to the CPU by an Interrupt request line
  - Identified by a number
  - May by *on chip* devices, timers, communications
  - May be *external* IO ports, dedicated interrupt pins
- The Interrupt vector table has a corresponding list of addresses for interrupt handler routines
  - an array of addresses

Actions on an interrupt

1. CPU finishes current instruction (machine level/assembler)
2. saves a snapshot of its state
3. looks up the address of the interrupt handler in the vector table
4. branches to this address
5. on exiting the interrupt handler, restores the saved state

Interrupts

- Asynchronous deviation of execution path
- can occur between *any two* machine instructions
- beware side effects of interrupt handler.

Interrupt Latency

- The Delay between the event occuring and the execution of the corresponding handler
- Many factors can influence:
  - Completion of current instruction
  - Fetch of interrupt vector
  - Completion of any higher priority interrupts

Interrupts and Complexity

- Interrupt based solutions can be very difficult to debug
- Hard to demonstrate correctness
- Interrupts can interfere in unexpected ways – especially if shared resources are involved

Part I

Code structure and hints

Interrupt handler

- No parameters passed or returned
- need to use globals to pass information between interrupts and other parts of code
- Usually some hardware related *boiler-plate* code to manage flags.

Split hardware interrupt related and application related

```
void (*timer0_handler)(uint32_t match);
void TIMER0_IRQHandler(void)
{
        timer0_handler(LPC_TIM0->IR);
        LPC_TIM0->IR |= (1UL << 0);
}
```

```
void flash(void)
{
        toggle_led(left_blue);
}
int main()
{
        initialise_timer0(ms,1000,flash)
}
```

Advantages

- Separated interrupt specific code from application code
- Easy and hard bits to write and test
- Keeps application logic together in same file.
- Scope rules help avoid unwanted interactions

scoping rules

```
static int flashing;
void flash(void)
{
        if( flashing ) toggle_led(left_blue);
}
int main()
{
        if( button_pressed() )
                flashing = 1;
        else
                flashing = 0;
}
```