# Foreground/Background Processing

**ENCE361 Embedded Systems 1**

Course Coordinator: Ciaran Moore (ciaran.moore@Canterbury.ac.nz)

Lecturer: Le Yang (le.yang@canterbury.ac.nz)

Department of Electrical and Computer Engineering

# Where we're going today

- **Another recap on interrupts**
- Foreground/background processing examples
- Still more on shared-data problem
- Homework

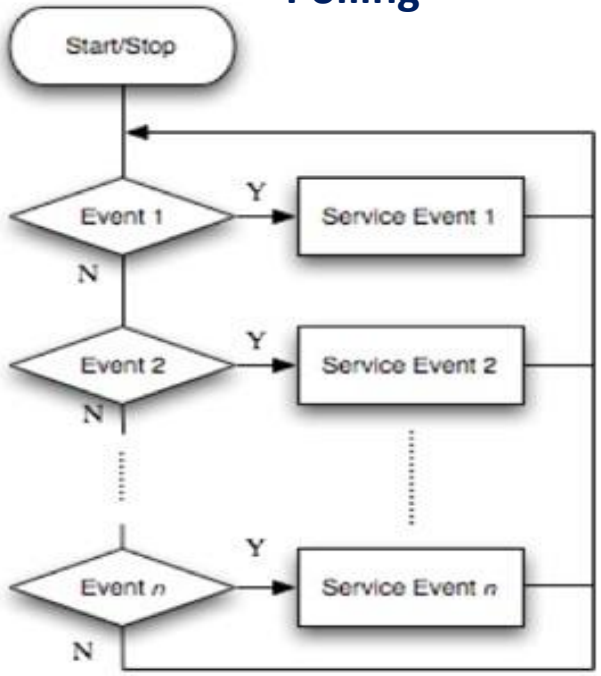
# Another Recap on Interrupts (1)

- For each interrupt requiring attention, an interrupt service routine (ISR)/handler is registered/associated
- Starting address of each ISR is stored in the interrupt vector table
- ARM Cortex-M4 microprocessor saves (and restores) automatically some critical registers (context) in servicing an interrupt
  - PUSH and POP
- ISR should execute quickly and only include the code essential to servicing the interrupt
  - Avoid delay loop and operations that halt/hang the system …

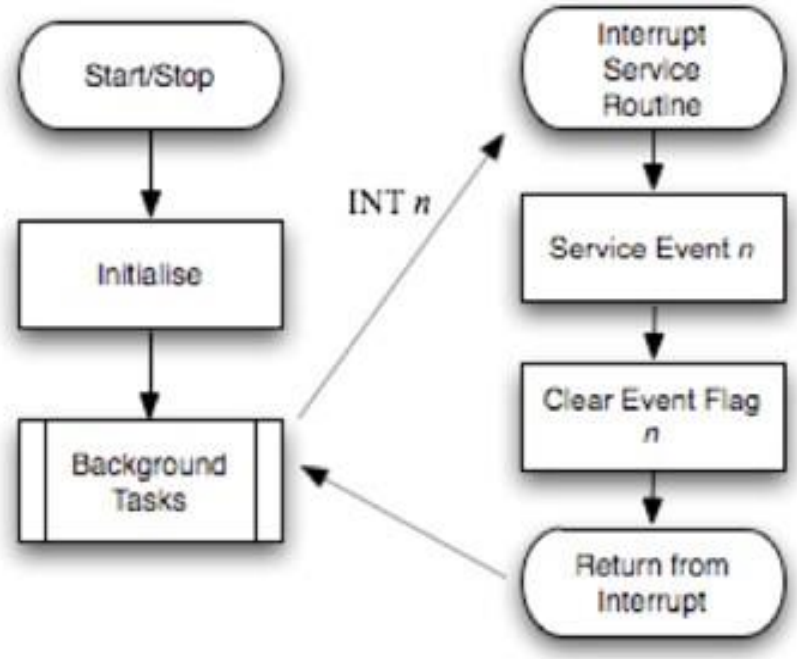
# Another Recap on Interrupts (2)

- Interrupts are more efficient than polling in handling asynchronous events

**Polling**




**Interrupts**



# Another Recap on Interrupts (3)

- Scenarios where using interrupts should be considered
  - Events are asynchronous and their service needs to be completed within a short time
  - Events occur at a high average repetition rate or at varying intervals
  - Microcontroller is relatively busy with the task it must perform
  - A foreground/background processing model can simplify the control code for a real-time system

# Where we're going today

- Recap on interrupts
- **Foreground/background processing examples**
- Still more on shared-data problem
- Homework

# Foreground/Background Model

- In a foreground/background model for embedded systems, all processing tasks are classified into two groups
- Foreground group
  - Interrupt-driven part of the processing, including ISRs
    - Tasks that need to be conducted frequently and/or with low latency with respect to external events (e.g., in real-time applications)
- Background group
  - All the other processing without strict latency requirement
    - Time-consuming tasks
    - Display tasks
    - Tasks coupled to human needs instead of machine needs …

# Example A: Anti-Lock Braking System (1)

- Anti-lock braking system (ABS)
  - Sudden braking can cause wheels to stop spinning and vehicle to skid
  - ABS detects rapid deceleration in any wheel and reduces the pressure on the brakes if needed
    - To prevent wheels from locking up
    - To help maintain control of vehicle
    - To avoid uncontrolled skidding of vehicle

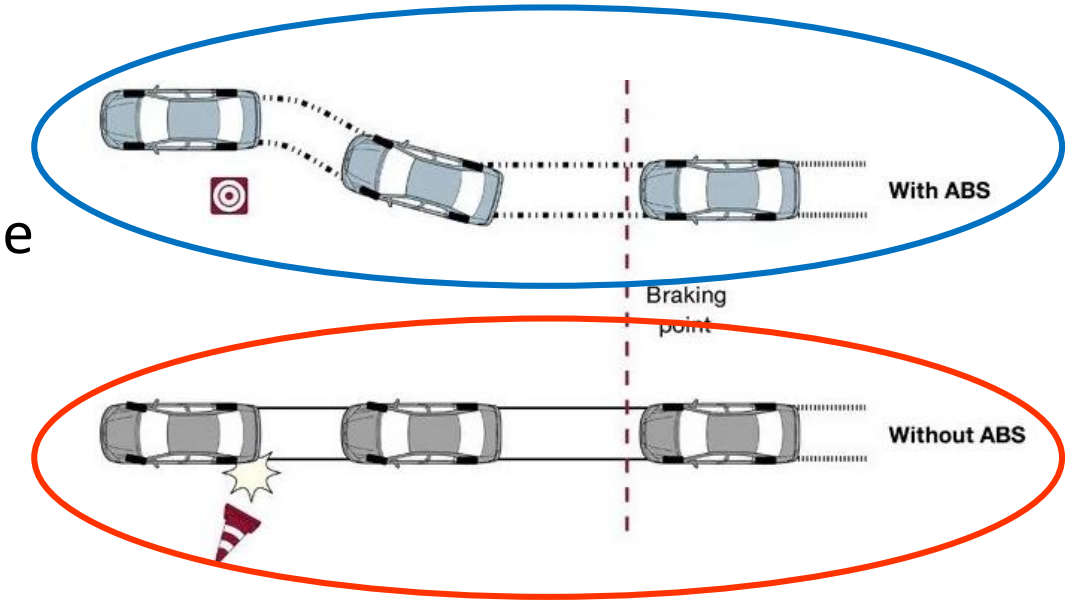


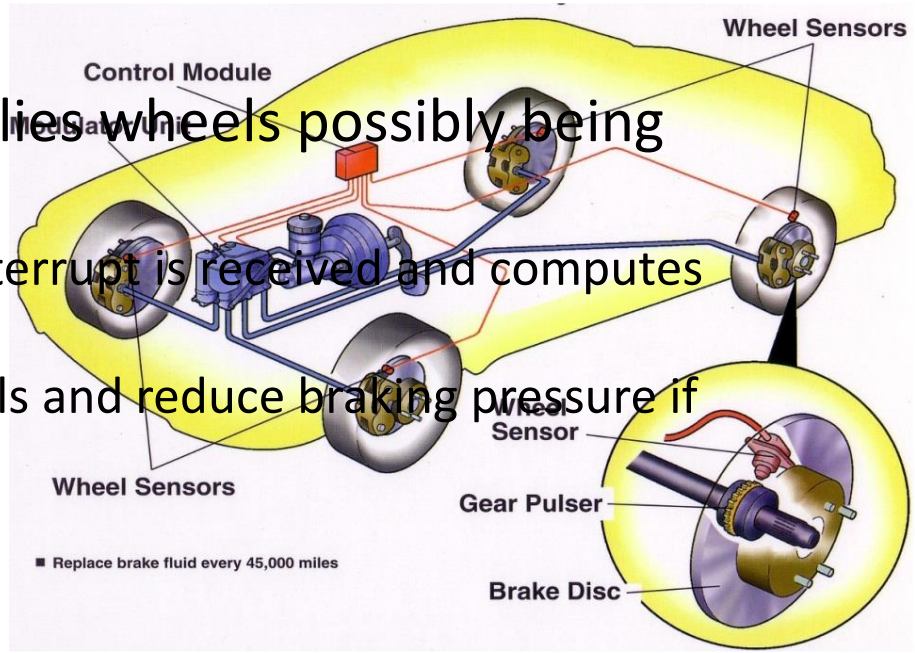



Figure from https://www.scienceabc.com/innovation/abs-sensors-anti-lock-breaking-system-technology-cars-work.html

# Example A: Anti-Lock Braking System (2)

- ABS uses wheel sensors to accurately monitor the speed of all wheels
  - Wheel sensor generates pulses as teeth of gear pulser pass in front of it
  - Every wheel revolution produces (say) 40 pulses
    - Pulse repetition rate up to 1,320/s at 200kph

- Abrupt increase in pulse repetition interval implies wheels possibly being locked up
  - **Foreground tasks**: ISR logs the time when a pulse interrupt is received and computes the interval since the previous pulse
  - **Background tasks**: monitor the last few such intervals and reduce braking pressure if necessary



# Example B: LAN Packet Routing Switch (1)

- Routing switch connects (say) 16 PCs together via packet switching to receive, process and forward data

- Decide which "client" can transmit
  - Transmission from every PC is asynchronous
  - Polling all PCs for transmission request (TR) waste processor cycles

- Monitor all packets from its "clients" and Internet
  - Packets arrives asynchronously
  - Check if any packets are destined for any of its "client"

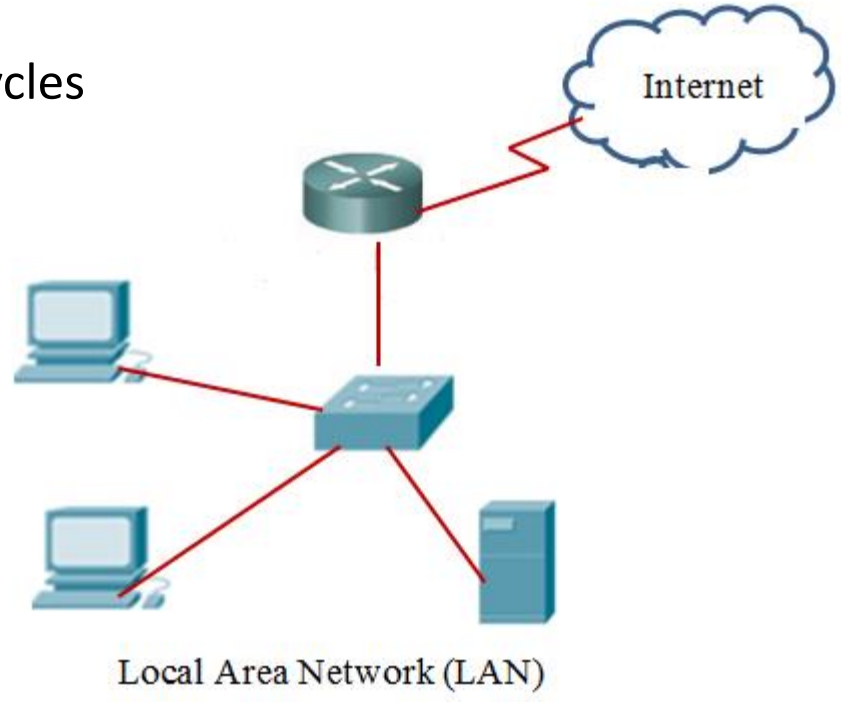





Figure from https://www.orbit-computer-solutions.com/local-area-network-lan/

# Example B: LAN Packet Routing Switch (2)

- Foreground tasks
  - ISR for handling transmission request (TR) interrupt
    - Sends an ACK to the initiating PC and adds TR to a FIFO of pending TRs
  - ISR for packet monitoring
    - Check destination address field and adds it to a FIFO of pending packets

- Background tasks
  - Process received packets
  - Deal with pending TR and packet at the top of respective FIFOs

# Where we're going today

- Recap on interrupts
- Foreground/background processing examples
- **Still more on shared-data problem**
- Homework

# Characteristics of Shared-Data Bug

- Shared-data problem can lead to unexpected program behavior
- It may have a low but definitely non-zero probability of occurring

- Bugs like this are famous for happening at times such as:
  - 5 o'clock in the afternoon, usually on Friday
  - Any time you are not actually watching
  - Whenever no debugging equipment is attached to the system
  - After your product has landed on Mars
  - When you are conducting a demonstration to Ciaran and myself …

# Example A (1)

```
static int iTemperatures[2];                    // Nuclear reactor temperature readings from 2 sensors

void interrupt vReadTemperatures (void)         // Foreground processing
{
   iTemperatures[0] = !! read in value from hardware
   iTemperatures[1] = !! read in value from hardware
}
```

```
void main (void)                                // Background processing
{
   int iTemp0, iTemp1;

   while (TRUE)
   {
      iTemp0 = iTemperatures[0];
      iTemp1 = iTemperatures[1];
      if (iTemp0 != iTemp1)
         !! Set off howling alarm;              // Alarm if two readings are not identical
   }
}
```

Critical Section



From Page 93 in D. E. Simon's book "An Embedded Software Primer"

# Example Code: tick64Test.c (1)

```
//*****************************************************************
// tick64Test.c - Test for real-time 64-bit clock


// Global variables – 64-bit counter
volatile static uint32_t gui32ClockHigh = 0;    // high word } 64-bit
volatile static uint32_t gui32ClockLow = 0;     // low word  } tick count


// The interrupt handler for the for SysTick interrupt (Foreground Program)
void SysTickIntHandler (void)
{
    // Update the 64-bit tick count
    gui32ClockLow++;                            // Increase tick count.

    if (gui32ClockLow == 0)
      gui32ClockHigh++;
}
```

# Example Code: tick64Test.c (2)

```
int main (void)                                  // Background program
{
  uint32_t ui32HighTmp, ui32LowTmp;     // For loading high and low words of tick count
  uint32_t ui32Num, ui32Denom;              // For calculating time in minutes and seconds
  .......
  while (1)
  {
    // Critical section: load current 64-bit count
    IntMasterDisable ();
    ui32HighTmp = gui32ClockHigh;
    ui32LowTmp  = gui32ClockLow;
    IntMasterEnable ();
    //
    // Time conversion
    ui32Num = (ui32HighTmp << (32 - BITS_TO_TRIM)) | (ui32LowTmp >> BITS_TO_TRIM);
    ui32Denom = SYSTICK_RATE_HZ >> BITS_TO_TRIM;

    displayTime ((ui32Num / ui32Denom) / 60,  (ui32Num / ui32Denom) % 60);
  }
}
```

**Atomic critical section**

**Read and understand**

# Improve tick64Test.c → tick64Test1.c

```
// Global 64-bit tick counter
volatile static uint64_t gui64Clock = 0;

void SysTickIntHandler (void)      // Foreground program
{
      gui64Clock++;                // Update the 64-bit tick count
}

int main (void)                    // Foreground program
{
    uint64_t ui64Tmp;              // For loading 64-bit tick count
    while (1)
    {
      IntMasterDisable ();
      ui64Tmp = gui64Clock;
      IntMasterEnable ();
      //
      displayTime ((ui64Tmp / SYSTICK_RATE_HZ) / 60, (ui64Tmp / SYSTICK_RATE_HZ) % 60);
    }
}
```

Are these operations necessary?

No Lecture on next Monday

# Homework

1. Consider a system which inspects parcels moving along a conveyor belt through 3 stations.

At the 1st station, the 5 visible faces of the parcel are inspected. If no label is found, the parcel is flipped by a robot arm.

At the 2nd station, an optical character recognition system is used to find the destination address.

At the 3rd station, the parcel is sorted into one of *N* destination bins, including one which is "address unknown".

Which parts of the system could be considered *foreground* and which parts *background*?

2. Could the maintenance of the 64-bit counter demonstrated in **tick64Test.c** be performed without interrupts? If so, what would be the consequences?

3. If **tick64Test.c** was written without the pair of disable/enable interrupt function calls, describe the most serious error that could occur.