Interrupt Processing II

ENCE361 Embedded Systems 1

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

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

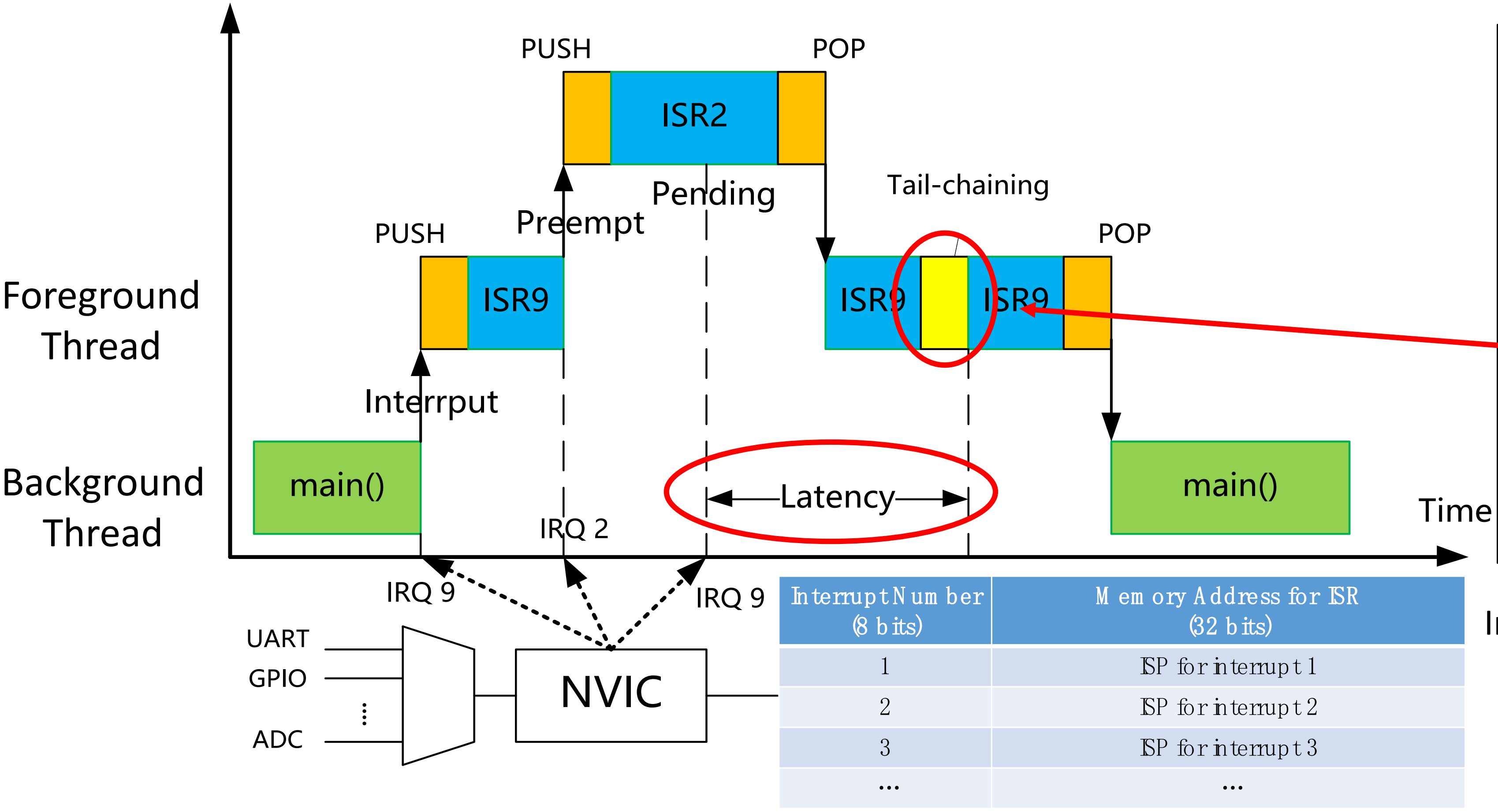
Department of Electrical and Computer Engineering

What we have learned

- Interrupt
  - (Mostly peripheral) events requiring attention → interrupt requests
  - Microcontroller stops to run interrupt service routine (ISR)
    - ISR can be executed between **any** two instructions
  - Microcontroller returns to code prior to interrupt
  - More efficient than polling for handling **asynchronous** events

What we have learned

- **Interrupt vector table**
  - Interrupt vector: starting memory address of an ISR
  - Table lookup via interrupt number (8 bits, signed)
- **Nested vectored interrupt controller (NVIC)**
  - Prioritizes and handles all interrupts
  - Preempt priority number for preemption
  - Sub-priority number for ordering interrupts with same preempt priority



Interrupt Number (8 bits)	Memory Address for ISR (32 bits)
1	ISP for interrupt 1
2	ISP for interrupt 2
3	ISP for interrupt 3
…	…

Interrupt Vector Table

- **PUSH, POP**
  - 12 clock cycles each
  - PSR, PC, LR, R12, R0, R1, R2, R3
- **Tail-chaining**
  - 6 clock cycles
  - ISR runs immediately after another

- **Real-time systems require bounded worst case interrupt latency**
  - Max time of being disabled and handling higher-priority interrupts
  - PUSH, POP, tail-chaining
  - ISR “work” time

Where we’re going today

- **ISR coding basics**
- Inter-thread communication
- Shared data problem
- Homework

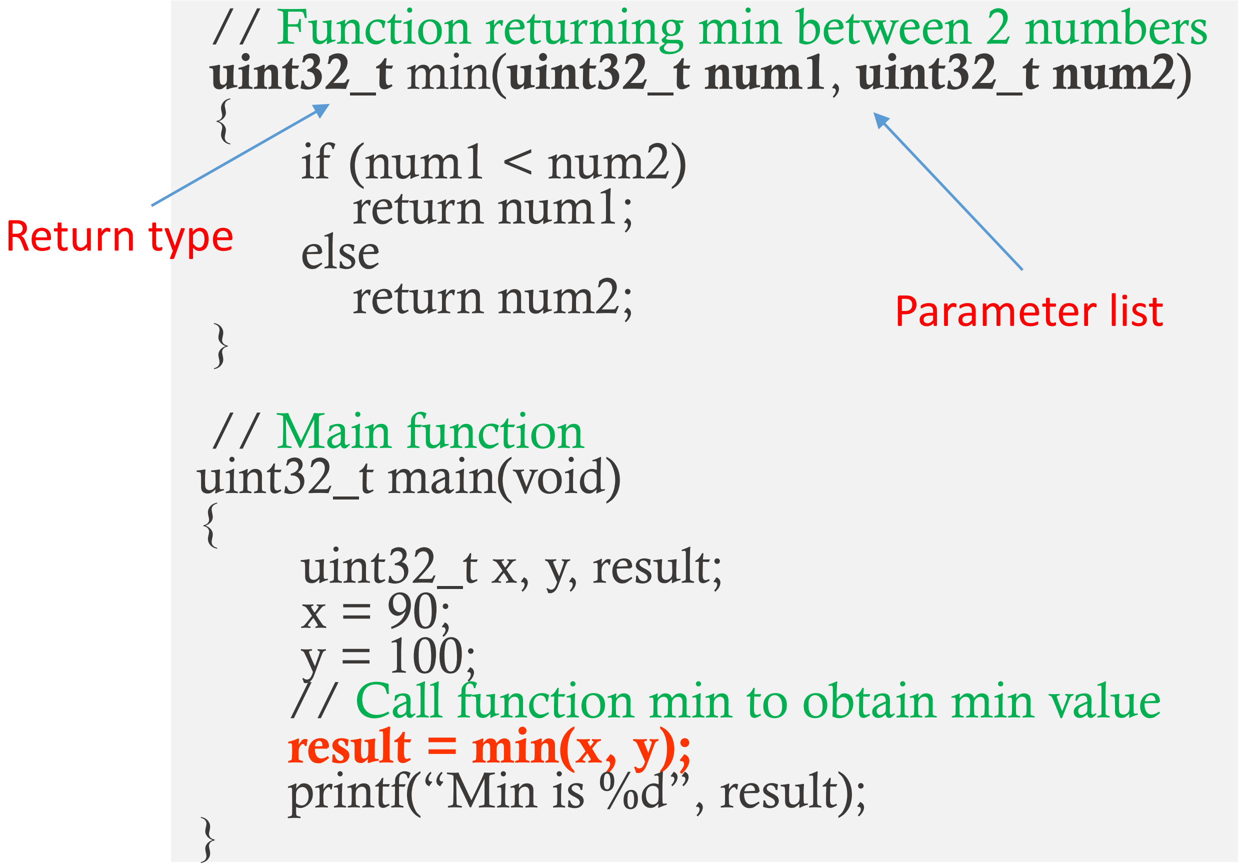
ISR as a Function

- At interrupt, microcontroller stops to run ISR from a new address
    - Similar to a ‘normal’ function call in C program
    - Code ISR as a C function

- ‘Normal’ function vs. ISR
    - ‘Normal’ function call is user planned (programmed)
    - Interrupt is asynchronous
        - Occurrence time may be unpredictable

Coding ISR in C

- Call a ‘normal’ function



```
// Function returning min between 2 numbers
uint32_t min(uint32_t num1, uint32_t num2)
{
        if (num1 < num2)
            return num1;
        else
            return num2;
}

// Main function
uint32_t main(void)
{
        uint32_t x, y, result;
        x = 90;
        y = 100;
       // Call function min to obtain min value
       result = min(x, y);
       printf(“Min is %d”, result);
}
```

Return type

Parameter list

- Use ISR for event-driven processing

```
// ISR (interrupt handler)
void ISR_name(void)
{
       // Body of the ISR
}
         ⋮
// Main function
uint32_t main(void)
{
     uint32_t x, y, result;

     //  Run while waiting for interrupts
     while (1)
     {
            ⋮
     }
}
```

ISR is **NOT** called explicitly anywhere

Background thread keeps running

Useful Tips for ISR Coding

- In most applications, keep ISR short
    - ISR affects normal execution of background thread and fast handling of other interrupts, increasing worst case interrupt latency

    - ISR should only do what is needed at the time of event

- Avoid including in ISR
    - Delay loop
    - Float point operation (need to push & pop additional 18 words if preempted)
    - Operations that halt or hang the system

Where we’re going today

- ISR coding basics
- **Inter-thread communication**
- Shared data problem
- Homework

Inter-thread Communication

- Inter-thread communication through global memory
  - Global variables defined outside of **all** functions

- Global variables
  - **Data**
  - **Binary flag**
  - **Mailbox** (binary flag + data)
  - Circular buffer …

```
// Global variables
 uint32_t variable_name;

// ISR (interrupt handler)
 void ISR_name(void)
 {
        // Body of the ISR
  }

             ⋮

 // Main function
 uint32_t main(void)
 {
        //  Run while waiting for interrupts
        while (1)
        {
             ⋮

         }
 }
```

Shared Data-based Communication

- Example: vending machine
  - Event: a chocolate bar is sold
  - ISR: decrease remaining number of chocolate bars by one
  - Display info when chocolate bars are sold out

```
// Global variables
uint32_t chocolateCnt = 20;

// ISR for the chocolate vending event
void ChocolateIntHandler(void)
{
    chocolateCnt--;
}

// Main function
uint32_t main(void)
{
    while (1)
    {
        if (chocolateCnt == 0)
        {
            // Display sold out information
                        ⋮
        }
        ⋮
    }
}
```

Binary Flag-based Communication

- Shared data-based communication is one way to synchronize threads

- Binary flag is another way
  - Set flag for signaling permission to perform certain operations

  - Remember to clear flag

- ISR sets flag vs. main() sets flag

Binary Flag-based Communication

```
// Global variables
uint16_t flag = 0;

// ISR sets the flag
void ISR_name(void)
{
    flag = 1;              // Set the flag
}
// Main function
uint32_t main(void)
{
    while (1)
    {
        if (flag)
        {
          flag = 0;      // Clear the flag
          // Perform certain operations
        }
        ⋮
    }
}
```

```
// Global variables
uint16_t flag = 0;

// ISR sets the flag
void ISR_name(void)
{
    if (flag)
    {
      flag = 0;      // Clear the flag
      // Perform certain operations
    }
}
// Main function
uint32_t main(void)
{
    while (1)
    {
        ⋮
      flag = 1;        // Set the flag
    }
}
```

Binary Flag-Based Communication

```
// Global variables
 uint32_t medianValue;

// ISR for the push button interrupt
 void PushButtonIntHandler(void)
 {
      // Transmit the median value
      SendMedian(medianValue);
  }

 // main function
 uint32_t main(void)
 {
       while (1)
       {
             ⋮
             // Find the median of a data buffer
             medianValue = FindMedian(& inBuffer);
             ⋮
       }
 }
```

ISR before or during FindMedian() leading to unexpected output

```
// Global variables
 uint32_t medianValue, flag = 0;

// ISR for the push button interrupt
 void PushButtonIntHandler(void)
 {
        flag = 1;           // Set the flag
  }

 // main function
 uint32_t main(void)
 {
       while (1)
       {
             // Find the median of a data buffer
             medianValue = FindMedian(& inBuffer);
             if (flag)
             {
                flag = 0;  // Clear the flag
                SendMedian(medianValue);
             }
       }
 }
```

Median value is transmitted **after** it's computed

Mailbox-Based Communication

- Mailbox consists of a binary flag and shared data
- Exemplary use of mailbox:

```
// Global variables
uint32_t flag = 0, data;

// ISR
void ISR_name(void)
{
        flag = 1;             // Set the flag
        data = GetData(); // Read data
}

// main function
uint32_t main(void)
{
        while (1)
        {
            if (flag)
            {
                flag = 0;       // Clear the flag
                ProcessData(data); // Data processing
            }
        }
}
```

Mailbox

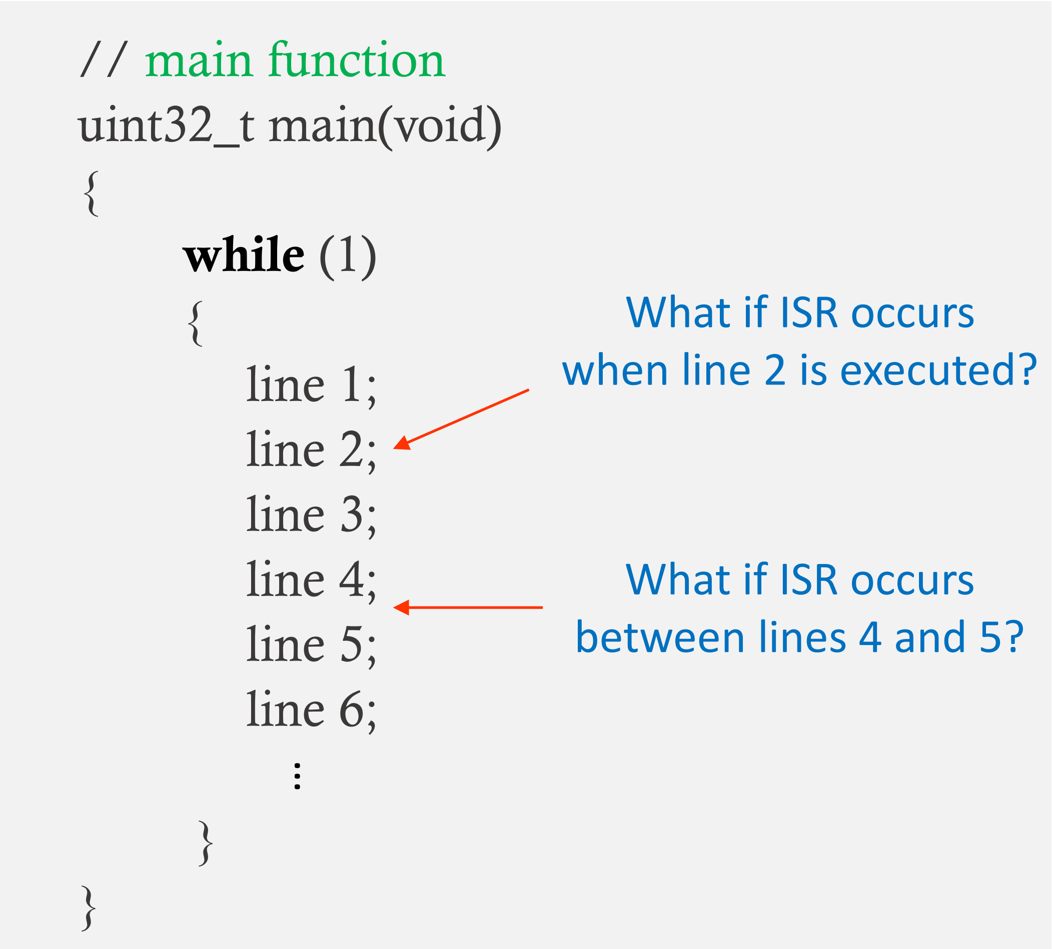
Where we’re going today

- ISR coding basics
- Inter-thread communication
- **Shared data problem**
- Homework

Shared Data Problem

- Global variables are accessed by both foreground and background threads

- Typical scenario for shared data problem
  - Background thread is accessing global variables
  - ISR stops background and changes global variables
  - Inconsistency may occur



```
// main function
uint32_t main(void)
{
        while (1)
        {
            line 1;
            line 2;
            line 3;
            line 4;
            line 5;
            line 6;
              ⋮
        }
}
```

What if ISR occurs when line 2 is executed?

What if ISR occurs between lines 4 and 5?

Critical Section

- Example of shared data problem
  - Frequently occurring events
  - Count number of occurrence
  - Output when pushing button

- Suppose
  - CntHigh = 0x0002
  - CntLow = 0xFFFF
  - EventIntHandler() interrupts SendEventCount()
    - CntHigh pushed
    - CntLow not pushed yet

```
// Global variables
 uint16_t CntLow = 0, CntHigh = 0, flag = 0;

// ISR for event counting
void EventIntHandler(void)
 {
         CntLow ++;       // Count event
         if (CntLow == 0)
            CntHigh ++;
 }

// ISR for the push button interrupt
 void PushButtonIntHandler(void)
 {
         flag = 1;           // Set the flag
 }

 // main function
 uint32_t main(void)
 {
        while (1)
        {
             if (flag)
             {
                 flag = 0;  // Clear the flag
                 SendEventCnt(CntLow, CntHigh);
             }
        }
 }
```

Critical section of the code

Enable/Disable Interrupt

- Critical section of code needs to have undisturbed access to global variables

- Simple way to solve shared data problem
  - Disable interrupts **before** critical section
  - Enable interrupts **after** critical section

```
              ⋮
__disable_interrupt();
SendEventCount(CntLow, CntHigh);
__enable_interrupt();
              ⋮
```

- Enable/Disable peripheral interrupts
  - **Clear/Set PRIMASK (I bit):** __enable_interrupt() & __disable_interrupt()
  - **Set NVIC_ISER0, NVIC_ISER1/Set NVIC_ICER0, NVIC_ICER1**

Homework

- Can static variables be used for inter-thread communication?
  - If so, how? If not, why?

- In the vending machine example, if initially, there are indeed 21 chocolate bars, the sold out information may not be displayed
  - Why? How to fix it?

- Read about the use of NVIC_ISER and NVIC_ICER registers