03.1. Lab Exercise: Introduction to interrupts

Name:

The objectives of this exercise are to:

1. Introduce the use of interrupts in C/C++ programming.
2. Introduce the use of multiple threads.
3. Become familiar with digital input.
4. Introduce the use of the Qt library.
5. Use Qt's "signals" / "slots" mechanism of inter-thread communication.

Solution

This exercise illustrates the use of interrupts, which are a mechanism for responding to an asynchronous input. The principal activity of your program is to respond to the input that is sent to an ISA I/O board. A second program, the counter display, which is sprinkled over your program, is used to display the count. Generally, the "service" of an interrupt may be divided into two parts, the "top" and the "bottom". However, in this exercise, each time an interrupt occurs, the interrupt service routine will execute ONE thread and simply pass all the information that is needed to the thread, which acts as an external circuit with inputs like HCs12 ports.

Therefore, the circuit which writes from the interrupt service routine to the thread is called a "circuit-type", interspersed in such a path with a "wire-type".

Although this program is not long, it is essential that you understand the events that take place when an interrupt occurs. Therefore, you should study the flow of control and the sequence of events in the ISA I/O acquisition.dll and in a separate section of code (DIO.module), further down in the interrupt service routine, which

processes the given data for execution in an interrupted. Then the counter display continues to execute. Finally, you should understand how the interrupt service routine sends an ordered sequence of data the interrupt source to the executive.

To Start

Open the DIO.module.

The main program runs in the main thread. It will perform the following tasks:

1. Open the ioDIO board.
2. Configure the ioDIO board's digital input and interrupt.
3. Create a background thread to "wait" the interrupt (see below).
4. Enter a loop that does through the loop:
   - Wait for an event by calling the Event object's WaitOne() method (with 250 ms).
   - If the event occurs and print the value of the I/O inputs.
   - Increment the value of count.
5. Close the ioDIO board and terminate the thread to stop, and wait until it terminates.
6. Close the ioDIO session.

The ISR thread

The ISR runs in an interrupt thread, separate from the main thread. Therefore, it has its own executable code, here supplied by the main program. The ISR thread does the following:

1. Wait for an external interrupt to occur on the ioDIO.
2. Activity the interrupt by pulsing the strobe: "interrupt," on the I/O connector.
3. Acknowledge the interrupt.

Support

Several new interrupt functions are used in the following. For more documentation on these, see Section 11.1.

Open the ioDIO (as before, generally):

While open to a ioDIO device, the I/O interrupt and clocks can be chosen to respond when the device is reset. Therefore, the following functions should generally defined thread resources:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_EnableInterrupt (int deviceNumber);
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_DisableInterrupt (int deviceNumber);
```

Function DIO_EnableInterrupt() enables the I/O interrupt and DIO_DisableInterrupt() disables it.

Register the ISR (IRQ). The first of these functions, DIO_ConfigureIrqSource(), defines and configures the ISR and IRQ, for prototype is:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_ConfigureIrqSource (
    int deviceNumber,
    int irqSource,
    int irqEnable);
```

where the five input arguments are:

deviceNumber	a number to communicate establishing the register and settings for the ioDIO board(s), starting at one (e.g., 1)
irqSource	0 = no interrupt 1 = DI0 rising edge 2 = DI0 falling edge 3 = DI1 rising edge 4 = DI1 falling edge 5 = DI2 rising edge 6 = DI2 falling edge 7 = DI3 rising edge 8 = DI3 falling edge 9 = Timer/Counter 0 interrupt 10 = Timer/Counter 1 interrupt 11 = Timer/Counter 2 interrupt
irqEnable	0 (zero) = disable 1 = enable

A zero(0) argument is provided to establish a number identifying the interrupt in the ioDIO board. This is used to identify the thread resources declared.

A one(1) is the IRQ source to be used.

A zero(0) disables (stop) the interrupt.

A one(1) enables (start) the interrupt.

Create the interrupt thread. The second function, pressed_count_reset(), which starts a new thread and configures it to respond to the interrupt. The prototype is:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_CreateIrqThread (
    int deviceNumber,
    int irqEvent);
```

where the four input arguments are:

deviceNumber	as described above (ioDIO number).
irqEvent	a number for the thread event. As an aside, the IRQ, to signify the thread's event.

event_is_reset, status of the counting. Normally, the event is reset by the interrupt system resume the function event_reset has been set to false, the event is not reset and the thread should be given a

The Function returns a 0 for success.

Finally, the interrupt must be unregistered. The function is:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_DestroyIrqThread (
    int deviceNumber);
```

Create the interrupt thread. The second function, pressed_count_reset(), which starts a new thread and configures it to respond to the interrupt. The prototype is:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_CreateIrqThread (
    int deviceNumber,
    int irqEvent);
```

where the four input arguments are:

deviceNumber	as described above (ioDIO number).
irqEvent	a number for the thread event. As an aside, the IRQ, to signify the thread's event.

event_is_reset, status of the counting. Normally, the event is reset by the interrupt system resume the function event_reset has been set to false, the event is not reset and the thread should be given a

The Function returns a 0 for success.

Finally, the interrupt must be unregistered. The function is:

```
[DllImport("ioDIO.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int DIO_DestroyIrqThread (
    int deviceNumber);
```

The ISR thread

The ISR is a separate thread that runs waited until signaled by the interrupt_source function. To a small task in a software, any necessary function is to be done in the ISR thread. The ISR thread is paired with interrupt so that the interrupt source is acknowledged. Then, the ISR thread waits for the next interrupt.

Each ISR thread function contains an endless loop. Within the loop, the thread waits for an input argument to be recognized. In our exercise, the ISR thread waits for the interrupt source. The ISR thread function is shown below:

```
// ISR thread function
void ISR_Thread(int deviceNumber)
{
    int irqSource;
    int status;
    int count = 0;

    // Wait for interrupt
    status = DIO_WaitForIrq(deviceNumber, 1000, &irqSource);
    while (status == 0)
    {
        // Acknowledge interrupt
        DIO_AcknowledgeIrq(deviceNumber);
        // Increment count
        count++;
        // Wait for next interrupt
        status = DIO_WaitForIrq(deviceNumber, 1000, &irqSource);
    }
}
```

The second loop is an endless loop. The loop is performed each time through the loop, as determined by the function:

```
status = DIO_WaitForIrq(deviceNumber, 1000, &irqSource);
```

DIO_WaitForIrq() waits for the IRQ. The first loop waits for the interrupt to occur or timeout of the 1000 ms.

The second loop is entered if the interrupt occurs before the timeout. The second loop performs the following:

1. Acknowledge the interrupt.
2. Increment the count.
3. Wait for the next interrupt.

Until a timeout occurs on the wait, the first loop is executed only once. The second loop is executed each time the interrupt occurs.

Finally, the interrupt must be unregistered. The function is:

```
status = DIO_DestroyIrqThread(deviceNumber);
```

using the same above arguments. To use the DIO_DestroyIrqThread() function, the thread must exist.

The ISR thread

The ISR is a separate thread that runs waited until signaled by the interrupt_source function. To a small task in a software, any necessary function is to be done in the ISR thread. The ISR thread is paired with interrupt so that the interrupt source is acknowledged. Then, the ISR thread waits for the next interrupt, counter