EE443 - Embedded Systems

Experiment 7

Interrupts and Data Flow

Purpose: Eliminate the CPU time wasted in LCD operations. The LCD is a slow device that requires **50 µs** to execute a single display command. Therefore ATmega328 CPU must wait at least **50 µs** before sending another LCD command. The functions provided in **LCDmodule.c** waste CPU time in delay functions to assure proper operation of the LCD. A better solution is to send LCD commands in an ISR invoked by a timer interrupt generated every **50 µs**. The LCD commands are queued in a circular buffer in memory and the ISR sends the commands to LCD one at a time.

Preliminary Work

1. Read the section, "**7.7 Reset and Interrupt Handling**" on page 13 of the ATmega328 datasheet. Answer the following questions:

a) When does ATmega328 disable and enable interrupts during an interrupt call?

b) What does happen if the **`SEI`** instruction is executed in an ISR? Is it possible to respond to another interrupt call while executing an ISR?

c) How are the interrupt priorities determined?

d) What difference does it make, if an interrupt has a higher priority then the other? Can an interrupt with higher priority stop the execution of another interrupt call?

e) What does (or does not) happen if you clear an interrupt flag by writing **0** to the related bit in an SFR?

2. Write the necessary statements initializing Timer-2 SFRs to obtain an interrupt once every **50 µs**. Timer-2 of ATmega328 is very similar to Timer-0. Refer to the information given for the previous experiment to determine the SFR settings for Timer-2.

Use the **Output Compare A** interrupt, and set timer clock to **1 MHz**. Set the timer mode bits, **WGM2[2:0] = 010**, to select the **Clear Timer on Compare Match** (**CTC**) mode.

3. In this experiment, a circular queue will be used to buffer the data that will be sent to the LCD module in the Timer-2 ISR. Write an enqueue function,

```
void EnQueueLCDbuffer(unsigned char  ByteIn)
```

that stores the data given by **`ByteIn`** in the circular queue. Ignore the errors that can be caused by buffer overrun conditions. Use the global declarations and the dequeue function given below for reference. Note that the buffer status is controlled by using the **`Nbyte`** global variable as a counter to simplify the circular queue implementation. Your enqueue function must increment **`Nbyte`** whenever a byte is added to the queue.

```
#define LCDbufferSize    32

// Global variables for data transfer to Timer-2 ISR:
unsigned char  LCDbuffer[LCDbufferSize];
unsigned char  Nbyte = 0;  // number of bytes stored in LCDbuffer

void DeQueueLCDbuffer(void)
/* If there are queued data in LCDbuffer (Nbyte > 0), then sends a single
   byte to the LCD module. If the buffer is empty (Nbyte = 0), then
returns
   without doing anything. */
{
  static unsigned char  *pLCDout = LCDbuffer;
  static unsigned char  Bcount = LCDbufferSize;
  unsigned char  ByteOut;

  if (Nbyte > 0)
// First send the 4 most significant bits of the LCD data:
  { ByteOut = *pLCDout >> 4;
    ----
    ----
// Update the LCDbuffer status:
    Bcount --;        // decrement buffer position counter
    if (Bcount == 0)  // check if reached end of buffer array
    { pLCDout = LCDbuffer;     // go back to the first element
      Bcount = LCDbufferSize;  // reload buffer position counter
    }
    else
      pLCDout ++;  // increment dequeue buffer pointer
    Nbyte --;      // decrement number of stored bytes
  }
  return;
}
```

The complete dequeue function required for this experiment is provided in a separate file (**Exp7dequeue.c**). Copy the dequeue function to your source file for this experiment.

Procedure

1. Create a new project directory and a new AVR project for Experiment 7. Copy the main program and the Timer-1 ISR code you wrote in Experiment 6 and add the files **LCDmodule.c** and **LCDmodule.h** to the new AVR project in Code::Blocks.

In the main program, add the statements that sets an attenuation control variable (i.e. **Atten**) checking the switch inputs as described in Experiment 4.

Timer-1 ISR will not check the switch settings, but it will adjust the PWM output according to the **Atten** value determined in the main program.

Atten should be accessible in the main program and in the Timer-1 ISR. Therefore, it should be a global variable declared outside the main program.

Compile your program and debug if necessary.

2. Open the design file for Experiment 6 in **ISIS**, and save it with a different name in the new project directory. Select the **.hex** file generated by the compiler for Experiment 7 in the **ATMEGA328P** properties window.

Apply a **50 Hz** sinusoidal waveform varying between **0 V** and **5 V** to the **ADC1** input.

Place an oscilloscope (select the **Virtual Instruments** list on the main toolbar) and probe the following nodes:

a) **ADC1** input of ATmega328
b) Filtered PWM signal at the RC low-pass filter output
c) **PD4** output of ATmega328 (time marker from ISR)
d) **PD5** output of ATmega328 (second time marker from ISR)

Test your program looking at the animated simulation results, and debug it, if necessary.

3. In the main program, display the attenuation factor on the second line of LCD every time one of the switches is pressed using the following statements:

```
sprintf(LCDtext, "Atten=%2d", Atten);
LCD_MoveCursor(2, 1);
LCD_WriteString(LCDtext);
```

Compile the program and check the LCD display while pressing the switch buttons several times.

Question 1: What does happen if the Timer-1 ISR displays an ADC result while the main program is using the LCD functions?

Use another global variable to stop LCD access in the Timer-1 ISR while the main program displays the attenuation factor.

Run the program again and check that the LCD display works properly.

4. Add the code you prepared in the preliminary work to initialize Timer-2 SFRs in the main program. Use the following code to check the Timer-2 interrupt timing. Make sure that the pin-5 of port-D is not changed anywhere else in the program.

```
ISR(TIMER2_COMPA_vect)
{
  PORTD |= _BV(PORTD5);   // Set second ISR timing marker.
  PORTD &= ~_BV(PORTD5);  // Clear second ISR timing marker.
}
```

Monitor the timing markers for Timer-1 and Timer-2 ISRs on the oscilloscope. Check the time interval between the Timer-2 interrupts right after the Timer-1 interrupt.

Question 2: If Timer-2 generates an interrupt while the Timer-1 ISR is running then how does this affect the timing of Timer-2 ISR?

5. Write the necessary statements in the Timer-2 ISR to correct the interrupt timing. Check the time interval between the Timer-2 interrupts on the oscilloscope to verify your solution.

Proper timing of Timer-2 interrupt calls is required before attempting to send LCD data through the queue buffer.

6. Add the dequeue function (see **Exp6_Dequeue.c**) provided for this experiment and the enqueue function you wrote in the preliminary work. Call the dequeue

function in Timer-2 ISR to send data to the LCD module one byte at a time. Use the enqueue function to replace the LCD module library functions as follows:

```
      sprintf(LCDtext, "Atten=%2d", Atten);
    // LCD instruction to move cursor to line-2, column-1:
    //  LCD_MoveCursor(2, 1);      // replaced by the following
      EnQueueLCDbuffer(0xC0);
    //  LCD_WriteString(LCDtext);  // replaced by the following
      pText = LCDtext;  // initialize pointer to output text
    // Copy text output to LCDbuffer:
      while (*pText != 0x00)
      { EnQueueLCDbuffer(*pText);
        pText ++;
      };
```

The first queued byte, **0xC0**, is the LCD module instruction to move the cursor to the beginning of the second line. Use the instruction byte, **0x80**, to move the cursor to the first line of LCD before writing ADC data into the queue.

Question 3: What can go wrong if the Timer-2 ISR starts while the main program is calling the enqueue function? What can be done to prevent this error?

7. Eliminate the unnecessary interrupts generated by Timer-2. Disable Timer-2 interrupt when there are no bytes left in the LCD data buffer. Enable Timer-2 interrupt after writing display data into the LCD buffer in the main program and in the Timer-1 ISR.

Monitor the timing markers for interrupt calls on the oscilloscope. Verify that Timer-2 interrupts does not occur regularly while the LCD functions properly.

Question 4: What is the estimated maximum response time for Timer-1 interrupt? In other words, how long does it take to start **ISR(TIMER1_COMPA_vect)** after the Timer-1 interrupt flag, **OCF0A**, is set?

What is the maximum response time for Timer-2 interrupt?

Hint: What does happen if the Timer-1 interrupt is received while the Timer-2 ISR is running?

Question 5: What can be done to reduce the maximum response time for Timer-1 and Timer-2 interrupts? Which global variable in your program may cause errors if the other interrupts are enabled while running an ISR? How can you prevent these errors?