CG2271 Real-Time Operating Systems

Tutorial 3 Suggested Solutions

1. There is a new microcontroller from ARM called the Latex M1 that runs at 20MHz. This controller doesn't have PWM capability but it does have a 16-bit Timer module with Interrupt capability which behaves like the Periodic Interrupt Timer in the Cortex M0+. The Timer module is directly clocked by the core clock of 20MHz.

    a. Design the Pseudo Code to generate a 50% Duty Cycle PWM signal using the Timer Interrupt.

Answer:

```
Main
{
  Set PWM_GPIO pin to '0'
  Set Timer Start Value to 0x7FFF.
  Enable Interrupts for Timer Module
  Start Timer
  while(1) {}
}

Timer_ISR
{
    Toggle PWM_GPIO pin
}
```

    b. What is the period of the PWM waveform?

Answer:

20MHz -> 50ns Period
0x7FFF * 50ns = 1.638ms
1.638ms * 2 = 3.277ms (Period of the PWM Waveform)

    c. Another microcontroller Latex M0 doesn't have any Timer module and wishes to still generate a PWM signal with the same period as the Latex M1. Show the Pseudo-Code on how this can be achieved.

Answer:

```
main()
{
   Set PWM_GPIO to '0'
   int counter = 0x7FFF;

   while(1)
   {
      counter--;
      if(counter  == 0)
      {
          Toggle PWM_GPIO
          counter = 0x7FFF;
      }
   }
}
```

2. In your project, you decide to use Serial Interrupts (you don't have a choice anyway…) to capture the data coming in through the Bluetooth interface. The following pseudocode shows a possible implementation.

```
volatile char rx_data; // Global Variable

Serial_ISR
{
  rx_data = Serial_Read_Buffer();
  rx_new_data = 1;
}

Main()
{
      if(rx_new_data == 1)
      {
            rx_new_data = 0;

            if(rx_data == 0x00)
                  move_robot_forward();
            if(rx_data == 0x01)
                  move_robot_right();
            if(rx_data == 0x02)
                  move_robot_left();
            else
                  stop_robot();
      }
      else
            Do_Other_Things();
}
```

a. Describe some issues with the implementation above.

Answer:

The rx_data is not processed immediately by the main(). You need to wait until you complete the "Do_Other_Things()" tasks before you can loop back to check for the flag and process the data.

There is also a possibility that a new ISR can be triggered before the earlier data was read from the rx_data variable. In this case, the older data would be overwritten with new data. Thus, an instruction (data packet) that was sent would have been missed.

b. The global variable rx_data is declared as volatile. How does it affect the behavior of the robot?

Answer:

If the Interrupt is triggered while you are in the midst of the if-else checks, then the value would still be updated. We can potentially ask the robot to perform multiple functions one after another. The actual impact would depend on how the low-level drivers are written.

3. To overcome the challenges earlier, you decide to use the circular queue that you learnt in class. In your application, you have the following code for the Serial_ISR and the Main() routine.

```
//Queue declared with a size of 10 (characters)

Serial_ISR
{
  rx_data = Serial_Read_Buffer();

  if(!Queue_Full())
        Q_Enqueue(rx_data);
}

Main()
{
      if(!Queue_Empty())
      {
           my_data = Queue_Dequeue();

           if(my_data == 0x00)
             move_robot_forward();
           if(my_data == 0x01)
             move_robot_right();
           if(my_data == 0x02)
             move_robot_left();
           else
             stop_robot();
      }
      else
           Do_Other_Things();
}
```

a. What does the new implementation resolve? Are there still things to be concerned about?

Answer:

This helps to solve the issue of overriding the same variable and losing information. However, we still have the issue of the response not being immediate as you still have to wait for the "if" check in the main().

There is still a potential issue of data being lost if the Interrupts happen too frequently and the buffer gets full even before you can start to dequeuer the data.

b. How can we resolve this issue?

Answer:

We can use the ability in ARM to generate an INT request once new data has been updated to the circular queue. This can be done though the SetPending_IRQ() mechanism. So the Serial_ISR cpatures the data and puts it in a buffer. Another ISR processes it (almost immediately).