DESIGN NOTE #031

AUTHOR: GLENN RICHMOND
KEYWORDS: INTERRUPT, TIMERS

Creating Non-blocking Timers in AVR

When writing code that requires efficient handling of tasks and a quick return to the main processing loop, it is important to be able to generate and handle Non-blocking Timers. One such example of where they would be used is in a communications protocol, where a packet timeout is desirable and/or flashing of the LEDs to indicate received data without blocking the processing loop is essential. The code below is a good way of providing as many Timers as is required. The code consists of a set_timer function for allocating a timer to the buffer, a process_timers function, which checks for the expiration of any timers, and an on_timer function, which handles the processing of timers when a timer occurs. I also included a kill_timers function, which will remove timers of a particular type from the queue, which I found to be quite useful when using a PACKET_TIMEOUT timer. The set_timer function is shown below:

```
void set_timer(unsigned int duration, char type)
{
char i;
// Disable timers..
//#asm(“cli”);
        for (i = 0; i < TIMER_BUFFER_SIZE; i++)
        {// loop through and check for free spot, then place timer in there...
                if (timer_queue[i].timer_id == TIMER_EMPTY)
                {// place new timer here...
                      timer_queue[i].timer_id = type;
                      timer_queue[i].end_time = duration + global_ms_timer;
                      return;
                }
        }
// Re-enable timers..
//#asm("sei");
}
```

The function simply loops through the timer buffer until it finds a free spot, when it places the timer in the buffer with its determined timout. Note that the interrupts are disabled during the function to avoid deadlock in the code. The process_timers is shown below.

```
void process_timers()
{char i;
    for (i=0;i<TIMER_BUFFER_SIZE;i++)
    {
              if ((timer_queue[i].end_time == global_ms_timer) &&
              (timer_queue[i].timer_id != TIMER_EMPTY))
              {// put into message queue...
                      on_timer(timer_queue[i].timer_id);
                      timer_queue[i].timer_id = TIMER_EMPTY;
                      timer_queue[i].end_time = TIMER_EMPTY;
              }
    }
}
```

The above function will loop through the buffer and perform any tasks that are required should a timer have expired. It is important to select a value of TIMER_BUFFER_SIZE that is appropriate for your application. A larger value will result in more simultaneous timers being able to be stored, though it will take longer to process.

The on_timer function (shown below) performs the actions on the Timer event.

```
void on_timer(char type)
{
      switch (type)
      {
      case TURN_OFF_RX_LIGHT:
          turn_off_rx_light();
      break;

      case TURN_OFF_TX_LIGHT:
          turn_off_tx_light();
      break;
      }
}
```

The kill_timers function (shown below) will clear the Timer Buffer of a particular type of timer. This is useful for a PACKET_TIMEOUT, where the Timer is set when the packet is sent, and killed upon receiving a response (i.e., the Timer event will only occur when there is actually a timeout).

```
void kill_timers(char type)
{char i;
// Disable timer interrupt
//#asm("cli");
  for (i = 0; i < TIMER_BUFFER_SIZE; i++)
  {// loop through and check for timers of this type, then kill them...
    if (timer_queue[i].timer_id == type)
    {// kill timers...
      timer_queue[i].timer_id = TIMER_EMPTY;
      timer_queue[i].end_time = TIMER_EMPTY;
      return;
    }
  }
// Enable Timer Interrupts
//#asm("sei");
}
```

Finally, we get down to what drives the Timer queue, and that's the Timer Compare Interrupt. In my case, I have used the 8515 and the TIMER COMPARE A interrupt. I am running the chip at 8 MHz, and the clock is triggered at 125.00 kHz. The following code will set up the required values in the Output Compare Register A.

```
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x7C;
```

Note that the value of the OCR1A Register is 124 (0x007C) as this corresponds to exactly 1 ms (0-124 = 125 triggers at 125kHz). The next step is to enable the Output Compare A interrupt.

```
TIMSK=0x40;
```

Once the global interrupts are enabled with the following code, the interrupt will occur exactly every 1 ms.

```
#asm("sei")
```

It is important to realise that the processing of the Timer on may vary within that ms, or slightly longer for longer delays, though it is relatively accurate over large timers provided that the processing of the timer isn't too substantial. A good way to ensure this is to use a message queue or to set a flag to indicate that an event has occurred, then handling it in the main loop. The code for the timer interrupt is as follows:

```
// Timer 1 output compare A interrupt service routine
// Should occur every ms...
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here
       global_ms_timer++;
       process_timers();
}
```

Conclusion

The previous code will provide a reliable method of obtaining non-blocking timers, providing that precautions are taken to ensure that the buffer will not overflow and the processing of the buffer occurs within a reasonable time. This method has the primary advantage of not using only one interrupt for as many timers as is required, leaving other timers free for use by the programmer.