8051 Timer Tick Interrupts APNT_105

OVERVIEW

Often it becomes necessary to use a timer in your programs for timing events or just as a simple lap timer. With C51, it is easy to add robust, interrupt-driven timer tick support into your application with little effort.

This application note provides complete source code for a timer tick library that provides you with a 100Hz timer tick.

There are several routines necessary for a set of timer tick functions. They are:

- An interrupt service routine for the timer.
- An initialization routine.
- A routine to get the current tick.
- A routine to delay for a while.

These routines are described in the following sections.

AN INTERRUPT SERVICE ROUTINE FOR THE TIMER

This example makes the following assumptions about the target system. You must change the source code provided if your target is different.

- Timer/Counter 0 is used exclusively for the timer tick.
- The oscillator frequency is 11.0592 MHz.

The following interrupt service routine is compiled using C51 Version 5. This interrupt function is called whenever interrupt 1 (Timer 0) occurs.

```
#include <reg51.h>

#define TIMER0_COUNT 0xDC11           /* 10000h - ((11,059,200 Hz / (12 * FREQ)) - 17) */

static unsigned timer0_tick;                                    /* timer tick variable */

/*------------------------------------------------------------------------------
static void timer0_isr (void);

This function is an interrupt service routine for TIMER 0.  It should never
be called by a C or assembly function.  It will be executed automatically
when TIMER 0 overflows.
------------------------------------------------------------------------------*/
static void timer0_isr (void) interrupt 1 using 1
{
unsigned i;

/*-------------------------------------------------
Stop Timer 0, adjust the timer 0 counter so that
we get another interrupt in 10ms, and restart the
timer.
------------------------------------------------*/
TR0 = 0;                                                          /* stop timer 0 */

i = TIMER0_COUNT + TL0 + (TH0<<8)

TL0 = i;
TH0 = i >> 8;

TR0 = 1;                                                         /* start timer 0 */

/*------------------------------------------------
Increment the timer tick.  This interrupt should
occur approximately every 10ms. So, the resolution
of the timer will be 100Hz not including interrupt
latency.
------------------------------------------------*/
timer0_tick++;
}
```

As you can see, the interrupt code is fairly trivial.

AN INITIALIZATION ROUTINE

The interrupt routine shown above never gets invoked until we initialize the timer and setup the interrupt. The following function does just that.

```
/*------------------------------------------------------------------------------
void timer0_initialize (void);

This function enables TIMER 0.  TIMER 0 generates a synchronous interrupt
once every 100Hz.
------------------------------------------------------------------------------*/
void timer0_initialize (void)
{
EA = 0;                                                     /* disable interrupts */

timer0_tick = 0;

TR0 = 0;                                                          /* stop timer 0 */

TMOD &= ~0x0F;                                          /* clear timer 0 mode bits */
TMOD |= 0x01;                                /* put timer 0 into 16-bit no prescale */

TL0 = (TIMER0_COUNT & 0x00FF);
TH0 = (TIMER0_COUNT >> 8);

PT0 = 0;                                             /* set low priority for timer 0 */
ET0 = 1;                                                 /* enable timer 0 interrupt */

TR0 = 1;                                                         /* start timer 0 */

EA = 1;                                                      /* enable interrupts */
}
```

This initialization routine should be called somewhere in main when you initialize your target hardware.

Note that interrupts are disabled on entry and are enabled on exit. Whenever you initialize and enable peripheral interrupts, you should make sure to disable all interrupts so that the initialization process is not interrupted!

The initialization routine:

- Disables all interrupts.
- Stops Timer 0.
- Sets the mode for 16-bit timer with no prescaler.
- Loads the timer counts.
- Sets the Timer 0 interrupt to low priority.
- Enables the interrupt.
- Starts the timer.
- Enables all interrupts.

After you invoke the initialization routine, your target has a 100Hz timer.

A ROUTINE TO GET THE CURRENT TICK.

The following routine is fairly trivial but very necessary.

```
/*------------------------------------------------------------------------------
unsigned timer0_count (void);

This function returns the current timer0 tick count.
------------------------------------------------------------------------------*/
unsigned timer0_count (void)
{
unsigned t;

EA = 0;
t = timer0_tick;
EA = 1;

return (t);
}
```

You must encapsulate access to the timer tick variable to guarantee that your program does not access this variable at the same time the interrupt is updating it. This function does that by disabling interrupts, copying the timer tick, restoring interrupts, and returning the timer tick copy.

A ROUTINE TO DELAY FOR A WHILE.

Delay functions are useful when waiting for input or when flashing an indicator lamp at 1Hz. The following routine can be used to delay for a specified number of timer ticks. If you call this routine and pass a value of 100, the delay will be approximately 1 second.

```
/*------------------------------------------------------------------------------
void timer0_delay (
  unsigned count);

This function waits for 'count' timer ticks to pass.
------------------------------------------------------------------------------*/
void timer0_delay (
  unsigned count)
{
unsigned start_count;

start_count = timer0_count ();                              /* get the starting count */

while ((timer0_count () - start_count) <= count)            /* wait for count "ticks" */
  {
  }
}
```

The **timer0_delay** function builds on the **timer0_count** function shown earlier. You can continue in this vein creating new functions by adding to tested, working routines.

CONCLUSION

Time tick routines are extremely useful in a number of applications. The routines provided in this application note can be used and reused in any number of applications.