/*
 * File:   timers.h
 * Author: mdunne
 *
 * Software module to enable a bank of software timers with a resolution time of
 * one msecond for each. The timers can be individually started, stopped, expired, etc.
 * 
 * NOTE: Module uses TIMER1 for its interrupts.
 * 
 * TIMERS_TEST (in the .c file) conditionally compiles the test harness for the code.
 * Make sure it is commented out for module usage.
 * 
 * Created on November 15, 2011, 9:54 AM
 */

/*******************************************************************************
* PUBLIC #DEFINES                                                          *
******************************************************************************/
#ifndef timers_H
#define timers_H
#ifndef SUCCESS
#define SUCCESS 0
#define ERROR -1
#endif
#define TIMER_ACTIVE 1
#define TIMER_EXPIRED 1
#define TIMER_NOT_ACTIVE 0
#define TIMER_NOT_EXPIRED 0

/*******************************************************************************
* PUBLIC FUNCTION PROTOTYPES                                                *
******************************************************************************/

/**
 * Function: TIMERS_Init
 * 
 * @param None
 * 
 * @return None
 * 
 * @remark Initializes the timer module
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
void TIMERS_Init(void);

/**
 * Function: StartTimer
 * 
 * @param Num, the number of the timer to manipulate
 * 
 * @return ERROR or SUCCESS
 * 
 * @remark simply sets the active flag in TMR_ActiveFlags to resart a
 * stopped timer.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char StartTimer(unsigned char Num);

/**
 * Function: SetTimer
 * 
 * @param Num, the number of the timer to manipulate
 * 
 * @param NewTime, the number of milliseconds to be counted
 * 
 * @return ERROR or SUCCESS
 * 
 * @remark sets the NewTime into the chosen timer and clears any previous
 * event flag and sets the timer active to begin counting.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char SetTimer(unsigned char Num, unsigned int NewTime);

/**
 * Function: StopTimer
 * 
 * @param Num, the number of the timer to manipulate
 * 
 * @return ERROR or SUCCESS
 * 
 * @remark simply clears the bit in TimerActiveFlags associated with this
 * timer. This will cause it to stop counting.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char StopTimer(unsigned char Num);

/**
 * Function: InitTimer
 * 
 * @param Num, the number of the timer to manipulate
 * 
 * @param NewTime, the number of milliseconds to be counted
 * 
 * @return ERROR or SUCCESS
 * 
 * @remark sets the NewTime into the chosen timer and clears any previous
 * event flag and sets the timer active to begin counting.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char InitTimer(unsigned char Num, unsigned int NewTime);

/**
 * Function: IsTimerActive
 * 
 * @param Num, the number of the timer to check
 * 
 * @return TIMER_ACTIVE, TIMER_NOT_ACTIVE or ERROR
 * 
 * @remark used to determine if a timer is currently counting.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char IsTimerActive(unsigned char Num);

/**
 * Function: IsTimerExpired
 * 
 * @param Num, the number of the timer to check
 * 
 * @return TIMER_NOT_EXPIRED, TIMER_EXPIRED or ERROR
 * 
 * @remark used to determine if a timer is currently expired.
 * 
 * @author Max Dunne
 * 
 * @date 2011.11.15 */
char IsTimerExpired(unsigned char Num);

/**
 * Function: ClearTimerExpired
 * 
 * @param Num, the number of the timer to manipulate
 * 
 * @return ERROR or SUCCESS
 */

/* @remark simply clears the appropriate bit in Event Flags to show that
   * the event has been serviced.
   * @author Max Dunne
   * @date 2011.11.15 */
char ClearTimerExpired(unsigned char Num);

/**
 * Function: GetTime
 * @param None
 * @return FreeRunningTimer, the current value of the module variable FreeRunningTimer
 * @remark Provides the ability to grab a snapshot time as an alternative to using
 * the library timers. Can be used to determine how long between 2 events.
 * @author Max Dunne
 * @date 2011.11.15 */
unsigned int GetTime(void);

#endif