Dear Jari,

Thanks for your C-compiler. I have finally finished the harware and am now writing software. It compiles and links fine. Interrupt routines are all going well. However I do have one problem :

I am using one of the timers on the controller to run an array of virtual timers. When I want a timer I can assign one, or more wisely, accuire one through the function d_timer_get. This function returns the index to a virtual timer in the array. To start this virtual timer I call d_timer_put providing the parameters: timer index, time to action and address to function called after time-out. The code underneath should hopefully clarify the problem.

The question is then:

1)

When setting up a virtual timer, I give as one of the parameters the address of the function I want to be called after time-out. This address is in the program memory, not the data memory. How do I write that in IAR C ? One attempt is shown in the main function underneath. My borland C++ 3.0 accepted it, but IAR C raised an error.

2)

After time-out, how do I call a function as a gosub, not a goto ? Now the address of the function is in data memory, but the function itself is in program memory. The function d_timer_int is called by the timer interrupt routine. The line

```
        dummy = ((*d_timer[next_indx].process)() == 0);
```

will in Borland C++ 3.0 call the function stored in .process. Is this also correct in IAR C ?

I hope dear Jari, that I do not put too much stress on you in answering these questions. If you do not wish to answer, please send a message anyway, so I know it has arrived. I will ofcourse be happy to share future software with you. By the way, have you got an Internet address ?

Happy easter and all the best

73 de Ottar LA9IHA @ LA4O

Example files :

```
---------- ALARM.H ----------------

void alarm (char, char, char);

---------- V_TIMER.H ----------------

#include "bool.h"

typedef struct
{
        int time;
        bool busy;
        bool allocated;
        int (*process)();
} TIMEELEM;

#define max_timers 20
#define MODULE_v_timer 1

---------- V_TIMER.C ----------------

#include "v_timer.h"
#include "alarm.h"

/* Variables */
```

```
TIMEELEM d_timer [max_timers];
bool     timer_started = FALSE;
int      next_time;
char     next_indx;
char     no_of_timers;

void d_timer_init(void)
{
char n;

        timer_started = TRUE;
        no_of_timers  = 0;
        for (n = 0; n > max_timers; n++)
        {
                d_timer[n].busy = FALSE;
        }
        /* SET REGISTERS IN 80C31 TO START TIMER 0 OR 1. DELAY OF 1msec */
}

char d_timer_get (void)
{
char n;
bool overflow;

        n = 0;
        overflow = FALSE;

        while (d_timer[n].busy && !overflow)
        {
                if (n > max_timers)
                        overflow = (n = 0xff);
                n++;
        };
        if (overflow)
                alarm (MODULE_v_timer, 1, 0); /* No more timers ! */
        return (n);
}

void shuffle (void)
{
char n;
        for (n = 0; n < max_timers; n++)
        {
                if (!d_timer[n].busy && (d_timer[n].time < next_time))
                {
                        next_indx = n;
                        next_time = d_timer[n].time;
                };
        };
}

void d_timer_kill (char timer_no)
{
        d_timer[timer_no].busy = FALSE;
        no_of_timers --;
        if (next_indx == timer_no)
                shuffle ();
}

void sub_timers (char exept_timer, int time)
{
char n;
        for (n = 0; n < max_timers; n++)
                if (!d_timer[n].busy && (n != exept_timer))
                        d_timer[n].time = d_timer[n].time - time;
}

void d_timer_put (char timer, int time, int(*proc)())
{
        if (!timer_started)
                d_timer_init ();
        d_timer[timer].time    = time;
        d_timer[timer].process = proc;
        d_timer[timer].busy    = TRUE;
```

```
        no_of_timers ++;
        shuffle ();
}

void d_timer_int (void)
{
        bool dummy;
        next_time --;

        if ((next_time == 0) && (no_of_timers != 0))
        {
                d_timer_kill (next_indx);
                sub_timers(next_indx, next_time);
                shuffle ();

                /* function addr. in d_timer[next_indx].process */
                /* is called
                dummy = ((*d_timer[next_indx].process)() == 0);
        }
}


void test1 (void)
{
        /* Dummy test procedure */
        ;
}
void main(void)
{
        /* Test virtual timer, call d_timer_int directly. Do not wait
           for interrupts */

        d_timer_put(1, 2, test1); /* Call function test1 in 2 msec */
        d_timer_int();                      /* 1 msec passed */
        d_timer_int();                      /* 2 msec passed */
}
```