

```
/* ATB_Multi_14.c Created: 18.03.2015 18:17:52 Author: AS */
```

```
// ori Teile P30 und P 31
```

```
#define F_CPU 16000000UL           // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <avr/io.h>                // Einbindung Datein
#include <avr/interrupt.h>
#include <stdint.h>
#include <util/delay.h>

volatile int16_t led1=0;
volatile int8_t wait;
volatile int8_t flag_1ms;
volatile uint8_t key_state;
volatile uint8_t key_press;
volatile uint8_t key_rpt;

#define KEY_DDR   DDRA             // Datenrichtung A
#define KEY_PORT  PORTA           // Angabe Port A
#define KEY_PIN   PINA            // Angabe PIN A
#define KEY_1     1               // PA 1
#define KEY_2     2               // PA 2
#define KEY_3     3               // PA 3
#define ALL_KEYS  (1<<KEY_1|1<<KEY_2|1<<KEY_3)
#define REPEAT_MASK (1<<KEY_1|1<<KEY_2)
#define REPEAT_START 50           // after 500ms
#define REPEAT_NEXT 20            // every 200ms

ISR (TIMER0_COMPA_vect)           // wait1=1ms,
{
    static uint8_t ct0,ct1,rpt;
    uint8_t i;
    if (flag_1ms)
    {
        // Laufzeit der Tasks >2ms, Fehlersignalisierung PA6, Programm stoppen
        PORTA &= ~(1<<PA6);        // Schaltet Pin
        while(1);                  // Endlosschleife, stoppt Programm
    }
    flag_1ms=1;
    if(wait<=9)                   // bei 9 sind es 10ms
    {
        wait++;
    }
    // erhöht
    else                           // wenn dann ...
    {
        wait=0;                   // setzt wait auf 0
        i=key_state ^~KEY_PIN;
        ct0=~(ct0&i);
        ct1=ct0^(ct1&i);
        i&=ct0&ct1;
        key_state^=i;
        key_press|=key_state&i;
        if((key_state & REPEAT_MASK)==0)
            rpt=REPEAT_START;
```

```

        if(--rpt==0)
        {
            rpt=REPEAT_NEXT;
            key_rpt|=key_state & REPEAT_MASK;
        }
    }
}

uint8_t get_key_press(uint8_t key_mask)
{
    cli();
    key_mask &=key_press;
    key_press^=key_mask;
    sei();
    return key_mask;
}

uint8_t get_key_rpt(uint8_t key_mask)
{
    cli();
    key_mask &=key_rpt;
    key_rpt^=key_mask;
    sei();
    return key_mask;
}

uint8_t get_key_short(uint8_t key_mask)
{
    cli();
    return get_key_press(~key_state & key_mask);
}

uint8_t get_key_long(uint8_t key_mask)
{
    return get_key_press(get_key_rpt(key_mask));
}

void led_blinken1()
{
    led1++;
    if(led1==500)
    {
        PORTA &= ~(1<<PA7);           // Schaltet Pin
    }
    else
    {
        if(led1==1000)
        {
            PORTA |= (1<<PA7);         // Schaltet Pin
            led1=0;
        }
    }
}

void timer_init()

```

```

{
    // Timer 0 konfigurieren
    TCCR0A = 0;
    TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00); // CTC Modus
    TCNT0=1; // Prescaler 64
    OCR0A=249;
    TIMSK0|=(1<<OCIE0A); // Interrupt erlauben
}

int main(void)
{
    timer_init();
    DDRA=0b11110000; // Port A auf Ausgang schalten
    KEY_DDR&=~ALL_KEYS;
    KEY_PORT|=ALL_KEYS;

    PORTA |= (1<<PA4);
    PORTA |= (1<<PA5);
    PORTA |= (1<<PA6);
    sei();
    while(1) // Programmschleife
    {
        if(flag_1ms)
        {
            flag_1ms=0;
            led_blinken1(); // Aufruf Unterprogramm
        }
        // delay zum testen einfügen
        // _delay_ms(2);
        // delay zum testen einfügen

        if(get_key_press(1<<KEY_2)) // nur Taste press
        { // LED an
            PORTA &= ~(1<<PA5);
        }

        if(get_key_press(1<<KEY_3)) // nur Taste press
        { // LED aus
            PORTA |= (1<<PA5);
        }

        if(get_key_short(1<<KEY_1)) // kurz schalten immer zusammen mit long
        {
            PORTA &= ~(1<<PA4); // LED an
        }

        if(get_key_long(1<<KEY_1)) // Lang schalten immer mit short
        {
            PORTA |= (1<<PA4); // LED aus
        }
    }
}

```