LAMPIRAN C
PROGRAM PADA MIKROKONTROLER ATMEGA8

```
/* Name: main.c

#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include "usbdrv.h"
#include "oddebug.h"
#define INTERRUPT_REPORT    1
#define CMD_WHO     "LM35 USB Reader"

volatile uchar   timerTimeout;
volatile uchar       timerUpdate;
volatile uchar   ledTimeout;
volatile unsigned int valueLM35;

enum {
    SEND_ENCAPSULATED_COMMAND = 0,
    GET_ENCAPSULATED_RESPONSE,
    SET_COMM_FEATURE,
    GET_COMM_FEATURE,
    CLEAR_COMM_FEATURE,
    SET_LINE_CODING = 0x20,
    GET_LINE_CODING,
    SET_CONTROL_LINE_STATE,
    SEND_BREAK
};

static PROGMEM char configDescrCDC[] = {
    9,
    USBDESCR_CONFIG,
    67,
    0,
    2,
    1,
    0,
#if USB_CFG_IS_SELF_POWERED
    USBATTR_SELFPOWER,
#else
              USBATTR_SELFPOWER,
//    USBATTR_BUSPOWER,
#endif
    USB_CFG_MAX_BUS_POWER/2,
    9,
    USBDESCR_INTERFACE,
    0,
    0,
    USB_CFG_HAVE_INTRIN_ENDPOINT,
    USB_CFG_INTERFACE_CLASS,
    USB_CFG_INTERFACE_SUBCLASS,
    USB_CFG_INTERFACE_PROTOCOL,
    0,
    /* CDC Class-Specific descriptor */
    5,
    0x24,
    0,
    0x10, 0x01,
    4,
    0x24,
    2,
    0x02,
    5,
    0x24,
    6,
    0,
    1,
    5,
    0x24,
    1,
    3,
    1,

    /* Endpoint Descriptor */
    7,
    USBDESCR_ENDPOINT,
    0x80|USB_CFG_EP3_NUMBER,
    0x03,
    8, 0,
    USB_CFG_INTR_POLL_INTERVAL,

    /* Interface Descriptor  */
    9,
    USBDESCR_INTERFACE,
    1,
    0,
    2,
    0x0A,
    0,
    0,
    0,

    /* Endpoint Descriptor */
    7,
    USBDESCR_ENDPOINT,
    0x01,
    0x02,
    8, 0,
    0,

    /* Endpoint Descriptor */
    7,
    USBDESCR_ENDPOINT,
    0x81,
    0x02,
    8, 0,
   0,
};
```

```
uchar usbFunctionDescriptor(usbRequest_t *rq)
{
   if(rq->wValue.bytes[1] == USBDESCR_DEVICE)
           {
      usbMsgPtr = (uchar *)usbDescriptorDevice;
      return usbDescriptorDevice[0];
   }
           else
           {
      usbMsgPtr = (uchar *)configDescrCDC;
      return sizeof(configDescrCDC);
   }
}

static uchar    modeBuffer[7];
static uchar    sendEmptyFrame;
static uchar    intr3Status;

/* ------------------------------------------------------------------------- */
/* ----------------------------- USB interface ----------------------------- */
/* ------------------------------------------------------------------------- */

uchar usbFunctionSetup(uchar data[8])
{
           usbRequest_t    *rq = (void *)data;

   if((rq->bmRequestType & USBRQ_TYPE_MASK) == USBRQ_TYPE_CLASS)
           {

      if( rq->bRequest==GET_LINE_CODING || rq->bRequest==SET_LINE_CODING )
                      {
         return 0xff;

      }

#if USB_CFG_HAVE_INTRIN_ENDPOINT3
      if(rq->bRequest == SET_CONTROL_LINE_STATE)
                      {
        if( intr3Status==0 )
            intr3Status = 2;
      }
#endif
#if 1
      /*  Prepare bulk-in endpoint to respond to early termination   */
      if((rq->bmRequestType & USBRQ_DIR_MASK) == USBRQ_DIR_HOST_TO_DEVICE)
         sendEmptyFrame  = 1;
#endif
   }
   return 0;
}
/*---------------------------------------------------------------------------*/
/* usbFunctionRead                                                          */
/*---------------------------------------------------------------------------*/
```

```
uchar usbFunctionRead( uchar *data, uchar len )
{
   memcpy( data, modeBuffer, 7 );
   return 7;
}

/*------------------------------------------------------------------------*/
/* usbFunctionWrite                                                        */
/*------------------------------------------------------------------------*/

uchar usbFunctionWrite( uchar *data, uchar len )
{
   memcpy( modeBuffer, data, 7 );
   return 1;
}

#define TBUF_SZ     256
#define TBUF_MSK    (TBUF_SZ-1)

static uchar tos, val, val2;
static uchar rcnt, twcnt, trcnt;
static char rbuf[8], tbuf[TBUF_SZ];
static uchar u2h( uchar u )
{
   if( u>9 )
      u    += 7;
   return u+'0';
}

static uchar h2u( uchar h )
{
   h    -= '0';
   if( h>9 )
   h    -= 7;
   return h;
}

static void out_char( uchar c )
{
   tbuf[twcnt++]    = c;
#if TBUF_SZ<256
   twcnt   &= TBUF_MSK;
#endif
}




void uint2ascii (unsigned int val)
{
          uchar digval;
          uchar buffer[6];
          uchar p;
          p = 0;
```

```
        do {
                digval = (uchar) (val % 10);
                val /= 10;

                /* convert to ascii and store */
                buffer[p++] = digval + '0';
        } while (val > 0);
        while (p < 4)
        {
                buffer[p++] = '0';
        }
        p--;

        out_char(buffer[p--]);
        out_char(buffer[p--]);
        out_char(buffer[p--]);
        out_char(buffer[p--]);
}

void usbFunctionWriteOut( uchar *data, uchar len )
{

  /*  postpone receiving next data    */
  usbDisableAllRequests();

  /*    host -> device:  request   */
  do {
    char    c;

    //    delimiter?
    c    = *data++;
    if( c>0x20 )
                   {
      if( 'a'<=c && c<='z' )
        c    -= 0x20;        //    to upper case
                          rbuf[rcnt++]    = c;
      rcnt    &= 7;
      continue;
    }
    if( rcnt==0 )
      continue;
    //    command
    if( rcnt==1 )
                   {
      char           *ptr;
      switch( rbuf[0] )

                          {
      case 'A':   //    about
            ptr = PSTR( CMD_WHO );
            while( (c=pgm_read_byte(ptr++))!=0 )
                                                   {
              out_char(c);
            }
            break;
      case 'T':   //    get lm35 value
                                            uint2ascii (valueLM35);
            break;

      default:   //    error
            out_char( '!' );
      }

      out_char( '\r' );
      out_char( '\n' );
      rcnt    = 0;
      continue;
    }

    //    number
    if( rcnt==2 )
               {
      val2    = val;
      val     = tos;
      tos     = (h2u(rbuf[0])<<4) | h2u(rbuf[1]);
      rcnt    = 0;
      continue;
    }

    //    sfr
    if( rcnt>=4 )
               {
      val2    = val;
      val    = tos;

#if defined (__AVR_ATmega8__) || defined (__AVR_ATmega16__) || !defined PORTC
      tos    = 0x30 + ( 'D' - rbuf[--rcnt] ) * 3;
#else
      tos    = 0x20 + ( rbuf[--rcnt] - 'A' ) * 3;
#endif

      rbuf[rcnt]    = 0;
      if( !strcmp_P(rbuf,PSTR("PIN")) )
        tos    += 0;
      else if( !strcmp_P(rbuf,PSTR("DDR")) )
        tos    += 1;
      else if( !strcmp_P(rbuf,PSTR("PORT")) )
        tos    += 2;
      else
        tos    = 0x20;       //    error
      rcnt    = 0;
    }
  } while(--len);

  usbEnableAllRequests();
}

#if INTERRUPT_REPORT
static uchar   intr_flag[4];
```

```
#define INTR_REG(x)     { intr_flag[x>>3] |= 1<<(x&7); }
#if _AVR_IOM8_H_ || _AVR_IOM16_H_
#define INTR_MIN        4
   ISR( TIMER2_COMP_vect )     INTR_REG(4)
   ISR( TIMER2_OVF_vect )      INTR_REG(5)
   ISR( TIMER1_CAPT_vect )     INTR_REG(6)
   ISR( TIMER1_COMPA_vect )    INTR_REG(7)
   ISR( TIMER1_COMPB_vect )    INTR_REG(8)
   ISR( TIMER1_OVF_vect )      INTR_REG(9)
//  ISR( TIMER0_OVF_vect )      INTR_REG(10)
   ISR( SPI_STC_vect )         INTR_REG(11)
   ISR( USART_RXC_vect )       INTR_REG(12)
   ISR( USART_UDRE_vect )      INTR_REG(13)
   ISR( USART_TXC_vect )       INTR_REG(14)
//  ISR( ADC_vect )             INTR_REG(15)
   ISR( EE_RDY_vect )          INTR_REG(16)
   ISR( ANA_COMP_vect )        INTR_REG(17)
   ISR( TWI_vect )             INTR_REG(18)
#if _AVR_IOM8_H_
#define INTR_MAX        19
   ISR( SPM_RDY_vect )         INTR_REG(19)
#else
#define INTR_MAX        21
   ISR( INT2_vect )            INTR_REG(19)
   ISR( TIMER0_COMP_vect )     INTR_REG(20)
   ISR( SPM_RDY_vect )         INTR_REG(21)
#endif
#endif
#if _AVR_IOMX8_H_
#define INTR_MIN        4
#define INTR_MAX        26
   ISR( PCINT0_vect )          INTR_REG(4)
   ISR( PCINT1_vect )          INTR_REG(5)
   ISR( PCINT2_vect )          INTR_REG(6)
   ISR( WDT_vect )             INTR_REG(7)
   ISR( TIMER2_COMPA_vect )    INTR_REG(8)
   ISR( TIMER2_COMPB_vect )    INTR_REG(9)
   ISR( TIMER2_OVF_vect )      INTR_REG(10)
   ISR( TIMER1_CAPT_vect )     INTR_REG(11)
   ISR( TIMER1_COMPA_vect )    INTR_REG(12)
   ISR( TIMER1_COMPB_vect )    INTR_REG(13)
   ISR( TIMER1_OVF_vect )      INTR_REG(14)
   ISR( TIMER0_COMPA_vect )    INTR_REG(15)
   ISR( TIMER0_COMPB_vect )    INTR_REG(16)
   ISR( TIMER0_OVF_vect )      INTR_REG(17)
   ISR( SPI_STC_vect )         INTR_REG(18)
   ISR( USART_RX_vect )        INTR_REG(19)
   ISR( USART_UDRE_vect )      INTR_REG(20)
   ISR( USART_TX_vect )        INTR_REG(21)
   ISR( ADC_vect )             INTR_REG(22)
   ISR( EE_READY_vect )        INTR_REG(23)
   ISR( ANALOG_COMP_vect )     INTR_REG(24)
   ISR( TWI_vect )             INTR_REG(25)
   ISR( SPM_READY_vect )       INTR_REG(26)
```

```
#endif

static void report_interrupt(void)
{
uchar    i, j;

    for( i=INTR_MIN; i<=INTR_MAX; i++ ) {
        j   = i >> 3;
        if( intr_flag[j]==0 ) {
            i   = ( ++j << 3 ) - 1;
            continue;
        }
        if( intr_flag[j] & 1<<(i&7) ) {
            intr_flag[j] &= ~(1<<(i&7));
            out_char( '\\' );
            out_char( u2h(i>>4) );
            out_char( u2h(i&0x0f) );
            out_char( '\r' );
            out_char( '\n' );
            break;
        }
    }
}
#endif
static void hardwareInit(void)
{
              uchar    i;
    /* activate pull-ups except on USB lines */
    USB_CFG_IOPORT    =
(uchar)~((1<<USB_CFG_DMINUS_BIT)|(1<<USB_CFG_DPLUS_BIT));
              /* all pins input except USB (-> USB reset) */
#ifdef USB_CFG_PULLUP_IOPORT    /* use usbDeviceConnect()/usbDeviceDisconnect() if
available */
    USBDDR    = 0;    /* we do RESET by deactivating pullup */
    usbDeviceDisconnect();
#else
    USBDDR    = (1<<USB_CFG_DMINUS_BIT)|(1<<USB_CFG_DPLUS_BIT);
#endif
    for(i=0;i<20;i++)
              {
        wdt_reset();
        _delay_ms(15);
    }
#ifdef USB_CFG_PULLUP_IOPORT
    usbDeviceConnect();
#else
    USBDDR    = 0;
#endif
}
void adcGetValue()
{
              PORTB = 0x01;
              ledTimeout = 50;

              ADCSRA = (1<<ADEN) | (1<<ADSC) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1);
              ADMUX = (1<<REFS1) | (1<<REFS0);

              valueLM35 = 0;
}

void timer0Init( )
{
              TCCR0 = 0x04;
              TCNT0 = 0x00;
              TIMSK = 0x01;

              timerTimeout = 200;
              timerUpdate = 0;
              ledTimeout = 0;

              valueLM35 = 0;

              DDRB = 0x01;
}
ISR( ADC_vect )
{
              valueLM35 = (unsigned int)ADCL;
              valueLM35 |= (unsigned int)(ADCH << 8);

              ADCSRA = 0;
}
ISR( TIMER0_OVF_vect )
{
              timerTimeout--;
              if ( timerTimeout == 0 )
              {
                          timerUpdate = 1;
                          timerTimeout = 200;
              }

              if ( ledTimeout )
              {
                          ledTimeout--;
                          if( ledTimeout == 0 )
                          {
                                      PORTB = 0x00;
                          }
              }
}

int main(void)
{
    wdt_enable(WDTO_1S);
    odDebugInit();
    hardwareInit();
    usbInit();
    intr3Status = 0;
    sendEmptyFrame  = 0;
    rcnt    = 0;
    twcnt   = 0;
```

```
    trcnt   = 0;
            timer0Init();
    sei();
    for(;;) /* main event loop */
            {
        wdt_reset();
        usbPoll();

        /*    device -> host    */
        if( usbInterruptIsReady() )
                        {
            if( twcnt!=trcnt || sendEmptyFrame )
                        {
                uchar    tlen;
                tlen    = twcnt>=trcnt? (twcnt-trcnt):(TBUF_SZ-trcnt);
                if( tlen>8 )
                    tlen    = 8;
                usbSetInterrupt((uchar *)tbuf+trcnt, tlen);
                trcnt   += tlen;
                trcnt   &= TBUF_MSK;
                sendEmptyFrame = (tlen==8 && twcnt==trcnt)? 1:0;
            }
        }
                        if ( timerUpdate )
                        {
                                adcGetValue();
                                timerUpdate = 0;
                        }

#if INTERRUPT_REPORT
        report_interrupt();
#endif
#if USB_CFG_HAVE_INTRIN_ENDPOINT3

        if(intr3Status != 0 && usbInterruptIsReady3())
                        {
            static uchar serialStateNotification[10] = {0xa1, 0x20, 0, 0, 0, 0, 2, 0, 3, 0};
            if(intr3Status == 2)
                                {
                usbSetInterrupt3(serialStateNotification, 8);
            }
                                else
                                {
                usbSetInterrupt3(serialStateNotification+8, 2);
            }
            intr3Status--;
        }
#endif

    }

    return 0;
}
```