Simpelt program DCC med intyerrupt

/\*

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;

without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details.

\*/

// arbejdet med 20032017

#define DCC\_PIN 4 // Arduino pin for DCC out

 // this pin is connected to "DIRECTION" of LMD18200

#define DCC\_PWM 5 // must be HIGH for signal out

 // connected to "PWM in" of LMD18200

#define DCC\_THERM 7 // thermal warning PIN

#define AN\_SPEED 2 // analog reading for Speed Poti

#define AN\_CURRENT 0 // analog input for current sense reading

//Timer frequency is 2MHz for ( /8 prescale from 16MHz )

#define TIMER\_SHORT 0x8D // 58usec pulse length

#define TIMER\_LONG 0x1B // 116usec pulse length

unsigned char last\_timer=TIMER\_SHORT; // store last timer value

unsigned char flag=0; // used for short or long pulse

unsigned char every\_second\_isr = 0; // pulse up or down

// definitions for state machine

#define PREAMBLE 0

#define SEPERATOR 1

#define SENDBYTE 2

unsigned char state= PREAMBLE;

unsigned char preamble\_count = 16;

unsigned char outbyte = 0;

unsigned char cbit = 0x80;

// variables for throttle

int locoSpeed=0;

int dir;

int last\_locoSpeed=0;

int last\_dir;

int dirPin = 12;

int FPin[] = { 8,9,10,11};

int maxF =3;

int locoAdr=36; // this is the (fixed) address of the loco

// buffer for command

struct Message {

 unsigned char data[7];

 unsigned char len;

} ;

#define MAXMSG 2

// for the time being, use only two messages - the idle msg and the loco Speed msg

struct Message msg[MAXMSG] = {

 { { 0xFF, 0, 0xFF, 0, 0, 0, 0}, 3}, // idle msg

 { { locoAdr, 0x3F, 0, 0, 0, 0, 0}, 4} // locoMsg with 128 speed steps

 }; // loco msg must be filled later with speed and XOR data byte

int msgIndex=0;

int byteIndex=0;

//Setup Timer2.

//Configures the 8-Bit Timer2 to generate an interrupt at the specified frequency.

//Returns the time load value which must be loaded into TCNT2 inside your ISR routine.

void SetupTimer2(){

 //Timer2 Settings: Timer Prescaler /8, mode 0

 //Timmer clock = 16MHz/8 = 2MHz oder 0,5usec

 TCCR2A = 0;

 TCCR2B = 0<<CS22 | 1<<CS21 | 0<<CS20;

 //Timer2 Overflow Interrupt Enable

 TIMSK2 = 1<<TOIE2;

 //load the timer for its first cycle

 TCNT2=TIMER\_SHORT;

}

//Timer2 overflow interrupt vector handler

ISR(TIMER2\_OVF\_vect) {

 //Capture the current timer value TCTN2. This is how much error we have

 //due to interrupt latency and the work in this function

 //Reload the timer and correct for latency.

 // for more info, see http://www.uchobby.com/index.php/2007/11/24/arduino-interrupts/

 unsigned char latency;

 // for every second interupt just toggle signal

 if (every\_second\_isr) {

 digitalWrite(DCC\_PIN,1);

 every\_second\_isr = 0;

 // set timer to last value

 latency=TCNT2;

 TCNT2=latency+last\_timer;

 } else { // != every second interrupt, advance bit or state

 digitalWrite(DCC\_PIN,0);

 every\_second\_isr = 1;

 switch(state) {

 case PREAMBLE:

 flag=1; // short pulse

 preamble\_count--;

 if (preamble\_count == 0) { // advance to next state

 state = SEPERATOR;

 // get next message

 msgIndex++;

 if (msgIndex >= MAXMSG) { msgIndex = 0; }

 byteIndex = 0; //start msg with byte 0

 }

 break;

 case SEPERATOR:

 flag=0; // long pulse

 // then advance to next state

 state = SENDBYTE;

 // goto next byte ...

 cbit = 0x80; // send this bit next time first

 outbyte = msg[msgIndex].data[byteIndex];

 if(msgIndex==0)

 {

 Serial.println("Idle");

 }

 else

 {

 Serial.println(outbyte);

 }

 break;

 case SENDBYTE:

 if (outbyte & cbit) {

 flag = 1; // send short pulse

 } else {

 flag = 0; // send long pulse

 }

 cbit = cbit >> 1;

 if (cbit == 0) { // last bit sent, is there a next byte?

 byteIndex++;

 if (byteIndex >= msg[msgIndex].len) {

 // this was already the XOR byte then advance to preamble

 state = PREAMBLE;

 preamble\_count = 16;

 //Serial.flush();

 //Serial.end();

 //Stream.flush();

 } else {

 // send separtor and advance to next byte

 state = SEPERATOR ;

 }

 }

 break;

 }

 if (flag==0) { // if data==1 then short pulse //jkp vendt om på 0 og 1

 latency=TCNT2;

 TCNT2=latency+TIMER\_SHORT;

 last\_timer=TIMER\_SHORT;

 //Serial.print("0");

 } else { // long pulse

 latency=TCNT2;

 TCNT2=latency+TIMER\_LONG;

 last\_timer=TIMER\_LONG;

 //Serial.print("1");

 }

 }

}

void setup(void) {

 Serial.begin(115200); //

 //Set the pins for DCC to "output".

 pinMode(DCC\_PIN,OUTPUT); // this is for the DCC Signal

 pinMode(DCC\_PWM,OUTPUT); // will be kept high, PWM pin

 digitalWrite(DCC\_PWM,1);

 pinMode(DCC\_THERM, INPUT);

 digitalWrite(DCC\_THERM,1); //enable pull up

 pinMode(dirPin, INPUT);

 digitalWrite(dirPin, 1); //enable pull-up resistor !!

 for (int i=0 ; i<=maxF; i++)

 {

 pinMode(FPin[i], INPUT);

 digitalWrite(FPin[i], 1); //enable pull-up resistor

 }

 read\_locoSpeed\_etc();

 assemble\_dcc\_msg();

 //Start the timer

 SetupTimer2();

}

void loop(void)

{

 delay(200);

 if (read\_locoSpeed\_etc())

 {

 // some reading changed

 // make new dcc message

 assemble\_dcc\_msg();

 }

}

boolean read\_locoSpeed\_etc()

{

 boolean changed = false;

 // read the analog input into a variable:

 // limit range to 0..127

 locoSpeed = (127L \* analogRead(AN\_SPEED))/1023;

 if (locoSpeed != last\_locoSpeed)

 {

 changed = true;

 last\_locoSpeed = locoSpeed;

 }

 dir = digitalRead(dirPin);

 if (dir != last\_dir)

 {

 changed = true;

 last\_dir = dir;

 }

 return changed;

}

void assemble\_dcc\_msg() {

 int i;

 unsigned char data, xdata;

 if (locoSpeed == 1) { // this would result in emergency stop

 locoSpeed = 0;

 }

 // direction info first

 if (dir) { // forward

 data = 0x80;

 } else {

 data = 0;

 }

 data |= locoSpeed;

 // add XOR byte

 xdata = (msg[1].data[0] ^ msg[1].data[1]) ^ data;

 noInterrupts(); // make sure that only "matching" parts of the message are used in ISR

 msg[1].data[2] = data;

 msg[1].data[3] = xdata;

 interrupts();

}