```
// ================================================================================================================================
// Include C Libriaries
// ================================================================================================================================

#include <avr/io.h>
#define F_CPU 4000000UL
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

// ================================================================================================================================
// Definition der Segmente
// ================================================================================================================================

#define SEG_O  (1<<PD5)          // Segment oben
#define SEG_OR (1<<PD4)          // Segment oben rechts
#define SEG_UR (1<<PD3)          // Segment unten rechts
#define SEG_U (1<<PD2)           // Segment unten rechts
#define SEG_UL (1<<PA0)          // Segment unten rechts
#define SEG_OL (1<<PA1)          // Segment unten rechts
#define SEG_M (1<<PD1)           // Segment unten rechts

// ================================================================================================================================
// NEC Code: Dauer der Puls- und Pausephasen [Anzahl der Interrupts pro Halbbit/Phase]
// ================================================================================================================================

#define dauer_count 40           // Dauer zw. 2 Interrupts = Frequenz LED
                                 // 80 kHz => 1/80.000 = 12 us

// Test wegen uint8 bei Empfänger --> max 256 Counts
#define startbit_pulse (9000/dauer_count)                // Startbit Pulse = 225
#define startbit_pause (4500/dauer_count)                // Startbit Pause = 112
#define datenbit_pulse (560/dauer_count)                 // Datenbit Pulse = 14
#define datenbit0_pause (560/dauer_count)                // 0-Bit Pause = 14
#define datenbit1_pause (1690/dauer_count)               // 1-Bit Pause = 42
#define stopbit_pulse (560/dauer_count)                  // Stopbit Pulse = 14
#define stopbit_pause (3000/dauer_count)                 // Stopbit Pause = 78

#define toleranz 0.5                                     // 50% Toleranz für Länge der Phasen

// ================================================================================================================================
// Wiederholte Befehlsfolgen
// ================================================================================================================================

// Sichern der Counts pro Phase und Zuruecksetzen der Variablen
#define phaseReset IR_signal_changed=0;IR_interrupt_count=0;IR_counts_letzte_phase=0
#define initVariablen
modus=0;status_empfang=0;i=0;datenbits=0;IR_signal_changed=0;IR_counter_active=0;IR_interrupt_count=0;
IR_counts_letzte_phase=0;LED_interrupt_count=0

// ================================================================================================================================
// Globale Variablen
// ================================================================================================================================

volatile uint8_t IR_counter_active;              // Flag zum Starten/Stoppen des Counters
volatile uint16_t IR_interrupt_count;            // Zaehler fuer Anzahl Interrupts (= Anzahl LED-Impulse)
volatile uint16_t IR_counts_letzte_phase;        // Puffer fuer Anzahl Interrupts der abgeschlossenen Phase
volatile uint8_t IR_signal_changed;              // Flag bei Signaländerung IR Empfänger (An>Aus>An)

volatile uint16_t LED_interrupt_count;           // Zaehler fuer Anzahl Interrupts (= Dauer LED-Anzeige pro Zahl)

volatile uint8_t ziffer[3];                      // 4 Ziffern fuer aktuell anzuzeigende Zahl
```

```
// Tabelle in Programmspeicher fuer LED-Segmente pro Ziffer
typedef struct {
        uint8_t portA;
        uint8_t portD;
} segmente_t;

static const PROGMEM segmente_t segmente_array[] =
{
{SEG_UL | SEG_OL, SEG_O | SEG_OR | SEG_UR | SEG_U },            // Ziffer 0
{0, SEG_OR | SEG_UR},                                           // Ziffer 1
{SEG_UL, SEG_O | SEG_OR | SEG_U | SEG_M},                       // Ziffer 2
{0, SEG_O | SEG_OR | SEG_UR | SEG_U | SEG_M},                   // Ziffer 3
{SEG_OL, SEG_OR | SEG_UR | SEG_M},                              // Ziffer 4
{SEG_OL, SEG_O | SEG_UR | SEG_U | SEG_M},                       // Ziffer 5
{SEG_UL | SEG_OL, SEG_O | SEG_UR | SEG_U | SEG_M},              // Ziffer 6
{0, SEG_O | SEG_OR | SEG_UR},                                   // Ziffer 7
{SEG_UL | SEG_OL, SEG_O | SEG_OR | SEG_UR | SEG_U | SEG_M},     // Ziffer 8
{SEG_OL, SEG_O | SEG_OR | SEG_UR | SEG_U | SEG_M},              // Ziffer 9
{0, 0},                                                         // aus
};

// ================================================================================================================
// Funktionsprototypen
// ================================================================================================================

void displayZahl(uint16_t zahl, uint16_t dauer);
void aktiviereSegmente(uint8_t ziffer);

// ================================================================================================================
// Hauptprogramm
// ================================================================================================================

int main(void)
{
        // Variablen deklarieren
        uint8_t modus;                  // 0 = IR-Signal empfangen
                                        // 1 = Ergebnisse anzeigen
        uint8_t status_empfang;         // 0 = warten auf Startbit Pulse (Ende)
                                        // 1 = warten auf Startbit Pause (Ende) = erstes Datenbit Pulse (Beginn)
                                        // 2 = warten auf Datenbit Pulse (Ende) bzw. Stopbit Pulse (Ende)
                                        // 3 = warten auf Datenbit Pause (Ende)
                                        // 4 = warten auf Stopbit Pause (Ende = Ueberlauf Counter)

        uint8_t i;                      // Index fuer Array mit Counts pro Phase

        uint32_t datenbits;             // Speicherung der 32 empfangenen Datenbits
        uint8_t datenbyte[3];           // Aufsplitten in 4 Bytes fuer Anzeige

        uint16_t zahl;                  // anzuzeigende Zahl für Display
        uint16_t dauer;                 // Anzeigedauer einer Zahl

        // Variablen initialisieren
        initVariablen;
        dauer = 500;

        // Ports initialisieren
        DDRB = (1<<PB4) | (1<<PB5) | (1<<PB6) | (1<<PB7);                       // Ausgang Ziffern
        DDRA = (1<<PA0) | (1<<PA1);                                             // Ausgang Segmente
        DDRD = (1<<PD0) | (1<<PD1) | (1<<PD2) | (1<<PD3) | (1<<PD4) | (1<<PD5);  // Ausgang Segmente
        DDRB |= (1<<PB2);                                                       // Ausgang Kontroll-LED
        PORTB = 0b00000101;                                                     // Eingang IR LED / Konroll-LED aus
```

```
// Timer initialisieren

// IR-Empfang: 4 MHz / (2*55) ~ 40 kHz (40.000 Interrupts und LED-Toogle pro Sekunde)
TCCR0A=(1<<WGM00);                              // Phasenkorrekter PWM
TCCR0B=(1<<WGM02)|(1<<CS00);                    // Vorteiler 1
OCR0A=55;                                       // Compare-Wert = 55
TCNT0=0;                                        // Zähler auf 0 setzen

// LED-Anzeige: 4 MHz / 256 / 30 ~ 520 Hz
TCCR1A=(1<<WGM10)|(1<<WGM11);                   // Fast PWM
TCCR1B=(1<<WGM12)|(1<<WGM13)|(1<<CS12);         // Vorteiler 256
OCR1A=30;                                       // Compare-Wert = 30
TCNT0=0;

// Interrupts initialisieren
PCMSK = (1<<PCINT0);                            // Interrupt bei Signalaenderung IR LED (Aus>An>Aus)
GIMSK = (1<<PCIE);
TIMSK = (1<<OCIE0A);                            // Interrupt zum Zaehlen der Pulse/Pausen pro Phase
sei();                                          // Interrupts aktivieren

// Endlosschleife
while(1)
{
        // Modus "IR Signal empfangen"
        if (modus==0)
        {
                cli();                          // Interrupts deaktivieren

                switch (status_empfang)
                {
                        // warten auf Startbit Pulse (Ende)
                        case 0: if ((IR_signal_changed==1) &&
                                (IR_counts_letzte_phase>=startbit_pulse*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=startbit_pulse*(1+toleranz)))
                                {
                                        phaseReset;
                                        status_empfang=1;
                                }
                                break;

                        // warten auf Startbit Pause (Ende) = erstes Datenbit Pulse (Beginn)
                        case 1: if ((IR_signal_changed==2) &&
                                (IR_counts_letzte_phase>=startbit_pause*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=startbit_pause*(1+toleranz)))
                                {
                                        phaseReset;
                                        status_empfang=2;
                                }
                                // falls Startbit Pulse erneut empfangen wird
                                else if ((IR_signal_changed==1) &&
                                (IR_counts_letzte_phase>=startbit_pulse*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=startbit_pulse*(1+toleranz)))
                                {
                                        phaseReset;
                                        status_empfang=1;
                                }
                                // Fehler in Signalfolge --> zurueck zu Status 0
                                else if (IR_signal_changed!=0)
                                {
                                        initVariablen;
                                }
                                break;
```

```
                        // warten auf Datenbit Pulse (Ende) bzw. Stopbit Pulse (Ende)
                        case 2: if ((IR_signal_changed==1) &&
                                (IR_counts_letzte_phase>=datenbit_pulse*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=datenbit_pulse*(1+toleranz)))
                                {
                                        phaseReset;
                                        if (i<=31)
                                        {
                                                status_empfang=3;       // Datenbit
                                        }
                                        else
                                        {
                                                status_empfang=4;       // Stopbit
                                        }
                                }
                                // Fehler in Signalfolge --> zurueck zu Status 0
                                else if (IR_signal_changed!=0)
                                {
                                        initVariablen;
                                }
                                break;

                        // warten auf Datenbit Pause (Ende)
                        case 3: if ((IR_signal_changed==2) &&           // 1-Bit
                                (IR_counts_letzte_phase>=datenbit1_pause*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=datenbit1_pause*(1+toleranz)))
                                {
                                        phaseReset;
                                        i++;
                                        datenbits=(datenbits<<1);
                                        datenbits|=(1<<0);
                                        status_empfang=2;
                                }
                                else if ((IR_signal_changed==2) &&      // 0-Bit
                                (IR_counts_letzte_phase>=datenbit0_pause*(1-toleranz)) &&
                                (IR_counts_letzte_phase<=datenbit0_pause*(1+toleranz)))
                                {
                                        phaseReset;
                                        i++;
                                        datenbits=(datenbits<<1);
                                        status_empfang=2;
                                }
                                // Fehler in Signalfolge --> zurueck zu Status 0
                                else if (IR_signal_changed!=0)
                                {
                                        initVariablen;
                                }
                                break;

                                // warten auf Ende des Signals
                                case 4: if (IR_interrupt_count>=2000)
                                {
                                        IR_counter_active=0;    // Counter stoppen
                                        modus=1;                // Empfang beendet, Ergebnisse anzeigen
                                }

                } // Ende switch (status_empfang) ...

                sei();                                  // Interrupts aktivieren

        } // Ende if (modus==0) ...
```

```
                // Modus "Zahl auf Display anzeigen"
                if (modus==1)
                {
                        TIMSK = (1<<OCIE1A);            // Interrupt zur Steuerung LED Mulitplexing
                        sei();                          // Interrupts aktivieren

                        PORTB&=!(1<<PB2);               // Kontroll-LED an
                        _delay_ms(300);
                        PORTB|=(1<<PB2);                // Kontroll-LED aus

                        // Datenbits aufbereiten und anzeigen
                        for (i=0;i<4;i++)
                        {
                                datenbyte[3-i]=datenbits;
                                datenbits=datenbits/256;
                        }

                        for (i=0;i<4;i++)
                        {
                              displayZahl(datenbyte[i],dauer);
                              _delay_ms(500);
                        }

                        PORTB&=!(1<<PB2);               // Kontroll-LED an
                        _delay_ms(300);
                        PORTB|=(1<<PB2);                // Kontroll-LED aus

                        PCMSK = (1<<PCINT0);            // Interrupt bei Signalaenderung IR LED (Aus>An>Aus)
                        GIMSK = (1<<PCIE);
                        TIMSK = (1<<OCIE0A);            // Interrupt zum Zaehlen der Pulse/Pausen pro Phase
                        sei();                          // Interrupts aktivieren

                        // Variablen initialisieren
                        initVariablen;

                } // Ende if(modus==1) ...

        } // Ende while(1) ...

} // Ende main() ...
```

```
// ================================================================================================================================
// Interrupt Service Routinen
// ================================================================================================================================

ISR(TIMER0_COMPA_vect)
{
        if (IR_counter_active){
                IR_interrupt_count++;                   // Interrupts (= Dauer Pulse/Pausen) zaehlen
        }
}

ISR(PCINT_vect)
{
        IR_counts_letzte_phase=IR_interrupt_count;      // Counter sichern
        IR_interrupt_count=0;                           // Counter zuruecksetzen
        IR_counter_active=1;                            // Counter starten

        // Eingang ist High, d. h. kein Empfang am TSOP --> Wechsel An>Aus --> Ende/Dauer Pulse-Phase
        if (PINB & (1<<PB0))
        {
                IR_signal_changed=1;
        }
        // Eingang ist Low, d. h. Empfang am TSOP --> Wechsel Aus>An --> Ende/Dauer Pause-Phase
        else
        {
                IR_signal_changed=2;
        }
}

ISR(TIMER1_COMPA_vect)
{
        uint8_t aktStelle;
        segmente_t segmente;

        if(LED_interrupt_count>0)
        {
                LED_interrupt_count--;

                aktStelle = LED_interrupt_count % 4;

                // alle Ziffern ausschalten
                PORTB |= (1<<PB0);                      // Eingang IR auf 1 setzen
                PORTB &= ~((1<<PB7)|(1<<PB6)|(1<<PB5)|(1<<PB4)); // Ausgang Ziffern auf 0 setzen

                // Segmente aktivieren
                memcpy_P(&segmente, segmente_array + ziffer[aktStelle], sizeof(segmente_t));
                PORTA = segmente.portA;
                PORTD = segmente.portD;

                // Ziffer aktivieren
                PORTB |= (1<<(7-aktStelle));
        }
}
```

```
// ================================================================================================================================
// Helper Funktionen
// ================================================================================================================================

void displayZahl(uint16_t zahl, uint16_t dauer)
{
	// Zahl in Ziffern zerlegen

	// Einser
	ziffer[3] = zahl % 10;
	zahl = zahl / 10;

	// Zehner
	if (zahl > 0)
	{
		ziffer[2] = zahl % 10;
	}
	else
	{
		ziffer[2] = 10;
	}
	zahl = zahl / 10;

	// Hunderter
	if (zahl > 0)
	{
		ziffer[1] = zahl % 10;
	}
	else
	{
		ziffer[1] = 10;
	}
	zahl = zahl / 10;

	// Tausender
	if (zahl > 0)
	{
		ziffer[0] = zahl % 10;
	}
	else
	{
		ziffer[0] = 10;
	}

	cli();
	LED_interrupt_count = dauer;
	sei();

	while(LED_interrupt_count>0);

	// alle Ziffern und Segmente ausschalten
	PORTA = 0;
	PORTB |= (1<<PB0);                                  // Eingang IR auf 1 setzen
	PORTB &= ~((1<<PB7)|(1<<PB6)|(1<<PB5)|(1<<PB4)); // Ausgang Ziffern auf 0 setzen
	PORTD = 0;
}
```