```
// Include C Libriaries

#include <avr/io.h>
#include <avr/delay.h>
#include <avr/interrupt.h>

// Portdefinitionen

#define IR_OUT_PORT PORTB                       // IR-Empfaenger an Port B, Pin PB0 (PCINT0)
#define IR_IN_PORT PINB                         // IR-Empfaenger an Port B, Pin PB0 (PCINT0)
#define IR_PIN PB0                              // IR-Empfaenger an Port B, Pin PB0 (PCINT0)

// Timerdefinitionen

#define timer_start TCCR0B=(1<<CS02);TCNT0=0  // Vorteiler = 256 --> 1 Count dauert 0,25us x 256 = 64us
                                               // Zaehler = 0
#define timer_stop TCCR0B=0;TCNT0=0            // Zaehler ausschalten
#define timer_value TCNT0                      // aktueller Wert des Zaehlers

// NEC-Code: Dauer der Puls- und Pausephasen --> Anzahl der Counts pro Phase

#define dauer_count 64                         // Vorteiler = 256 --> 1 Count dauert 0,25us x 256 = 64us
#define toleranz 0.2                           // Toleranz für Dauer +/- 20%

#define startbit_puls (9000/dauer_count)       // Start-Bit Puls
#define startbit_pause (4500/dauer_count)      // Start-Bit Pause
#define datenbit_puls (560/dauer_count)        // Datenbit Puls
#define datenbit0_pause (560/dauer_count)      // 0-Bit Pause
#define datenbit1_pause (1690/dauer_count)     // 1-Bit Pause
#define stopbit_puls (560/dauer_count)         // Stop-Bit Puls
#define stopbit_pause (4500/dauer_count)       // Stop-Bit Pause > 4500 us
```

```
// Deklaration Variablen

volatile unsigned long int ir_daten;          // Struktur mit 4 Bytes für die Daten des NEC-Signals
volatile unsigned char ir_bit_count;          // Zaehler für Anzahl der Datenbits (1...32)

volatile unsigned char ir_flag;               // 1 = Daten empfangen
                                              // 0 = keine Daten empfangen / warten auf Signal


volatile unsigned char ir_state;              // Status des Signalempfangs
                                              /*    state 0            warten auf Start-Bit Beginn Puls
                                                    state 1            warten auf Start-Bit Beginn Pause
                                                    state 2            warten auf Datenbit 1 Beginn Puls
                                                    state 3            warten auf Datenbit x Beginn Pause
                                                    state 4            warten auf Datenbit x (Ende-Bit) Beginn Puls
                                                    state 5            warten auf Stop-Bit Beginn Pause
                                                    state 6            testen auf Ende Empfangssignal */

// Ports, Timer und Interrupts initialisieren

void init_devices(void)
{
      cli();                                  // Interrupts deaktivieren

      // Port initialisieren
      IR_OUT_PORT = (1<<IR_PIN);              // PB0 als Input fuer IR-Empfaenger (Port-Bit auf 1 setzen)

      // Timer0 initialisieren
      timer_stop;

      // Interrupts definieren
      GIMSK = (1<<INT0);                      // externer Interrupt durch IR-Signal an Pin PB0 (PCINT0)
      TIMSK = (1<<TOIE0);                     // Timer/Counter0 Overflow Interrupt

      sei();                                  // Interrupts aktivieren
}
```

```
// Interruptroutine für Overflow Timer0

ISR(TIMER0_OVF_vect)
{
	cli();                                  // Interrupts deaktivieren

	timer_stop;                             // Timer anhalten

	ir_flag = 0;                            // ir_flag zuruecksetzen

	if (ir_state = 6) {                     // Status 6 (Warten auf Ende Stop-Bit)
                                                // Overflow nach 256 x 64us = 16,38 ms
		ir_flag = 1;                    // Kennzeichen, dass Befehl erfolgreich empfangen
		ir_state = 0;                   // Status zuruecksetzen
		ir_bit_count = 0;               // Bit-Zaehler zuruecksetzen
		}

	sei();                                  // Interrupts aktivieren
};

// Interruptroutine bei Signalaenderung an PB0 (PCINT0)

ISR(INT0_vect)
{
	// Deklaration Variablen

	unsigned char ir_bit_in;                // aktuelles Datenbit innerhalb des Signals

	// State Machine fuer Steuerung der Signaldecodierung

	switch(ir_state){

		// Warten auf Start-Bit Beginn Puls (= Beginn Start-Bit)
		case 0: timer_start;
			ir_state = 1;
			break;
```

```
// Warten auf Start-Bit Beginn Pause
case 1: timer_stop;
	ir_state = 0;

	if (timer_value>startbit_puls * 0.8 & timer_value<startbit_puls * 1.2) {
		timer_start;
		ir_state = 2;
		break;
	}

// Warten auf Datenbit 1 Beginn Puls (= Ende Start-Bit)
case 2: timer_stop;
	ir_state = 0;

	if (timer_value>startbit_pause * 0.8 & timer_value<startbit_pause * 1.2) {
		timer_start;
		ir_state = 3;

		ir_bit_count = 0;
		ir_daten = 0;
		ir_flag = 0;

		break;
	}

// Warten auf Datenbit x Beginn Pause
case 3: timer_stop;
	ir_state = 0;

	if (timer_value>datenbit_puls * 0.8 & timer_value<datenbit_puls * 1.2) {
		timer_start;
		ir_state = 4;
		break;
	}
```

```
		// Warten auf Datenbit x (oder Stop-Bit) Beginn Puls
		// --> ermitteln, ob 0-Bit oder 1-Bit, und wegschreiben in ir_daten
		case 4: timer_stop;
			ir_state = 0;

			ir_bit_in = 0;                                  // 0-Bit als Default Wert
			if (timer_value>datenbit1_pause * 0.8 & timer_value<datenbit1_pause * 1.2) {
				ir_bit_in = 1;}                         // 1-Bit bei langer Pause
			timer_start;

			ir_bit_count++;                                 // Zaehler für Datenbits hochzaehlen

			if (ir_bit_count == 32) {                       // alle 32 Bits empfangen
				ir_state = 5; }                 // ja, warten auf Stop-Bit
			else {
				ir_state = 3; }                 // nein, warten auf naechstes Daten-Bit

			break;

		// Warten auf Stop-Bit Beginn Pause
		case 5: timer_stop;
			ir_state = 0;

			if (timer_value>stopbit_puls * 0.8 & timer_value<stopbit_puls * 1.2) {
				timer_start;
				ir_state = 6;
				break;
			}

		default: ir_state = 0;
			ir_flag = 0;
			break;
	}
}
```

```
void main(void)
{
	unsigned char NEC_befehl;

	init_devices();

	while(1)
	{
		if (ir_flag == 1)
		{
			// NEC-Signal (Adresse und Befehl) pruefen
			// --> Vergleich mit invertierten Bits und eigener Befehlsliste

			// bei erfolgreicher Pruefung, Befehl ausfuehren

			ir_flag = 0;
			ir_daten = 0;
		}
    }
}
```