```
/*****************************************************************************************
6 DIGIT LED-FREQUENZZÄHLER, Autorange, Messbereich bis 84MHz, Eagle-Layout: TN20150607
Kompilieren mit AVR-Studio (getestet mit V4.17), Fusebits: Low=EFh, High=D9h, Lock=FFh

Für uC ATMEGA48 @24,576MHz (overclocked) und LED-Ansteuerung ohne Treibertransistoren.
Die Messbereiche sind am Dezimalpunkt erkennbar: Mit Dezimalpunkt = "MHz", sonst "Hz"
Die Messfolge wird durch blinkenden Dezimalpunkt rechts von der Einerstelle angezeigt.

Ressourcen: LEDs mit gemeinsamer Kathode => Segment ist an bei Kathode = 0, Anode = 1
PB0 - PB5 Segmentausgänge A - F, PD3 Segmentausgang G, PD2 Dezimalpunkte H, PB6,7 = Quarz
PC0 - PC5 = DIGIT A (einer) - DIGIT F (100000er), Kathoden, aktiv bei Bit = 0
PD4 = Zähleingang vom 1:100 Prescaler 74HC390 => erfasst von Timer0 (8 Bit)
PD5 = Zähleingang (direkt) => erfasst von Timer1 (16 Bit), Timer2 zur Zeitsteuerung

V1.00: 15/06/2015 - Initiale Version, (c) T. Neveling                        1266 Byte
V1.10: 18/06/2015 - 24,576MHz Takt, Korrektur LZB für Digit 6                1274 Byte
*****************************************************************************************/

#include <avr/io.h>                 // bindet auch den Prozessortyp gemäß Makefile ein.
#include <avr/interrupt.h>
#include <util/delay.h>

volatile unsigned long display = 123456;// Auszugebender Messwert: 32 Bit Integer Variable
volatile unsigned char dot;             // Position des Dezimalpunkts 1 = rechts..6 = links
const char SEGMENTE[] = {63,6,91,79,102,109,125,7,127,111};     // Segmentmuster für 0..9

unsigned char digicnt= 0;   // Zählt die Digits des Displays bei der Ausgabe durch
unsigned char lowovf = 0;   // Zählt die Overflows von Timer1 => Eingang ohne Prescaler
unsigned int  highovf= 0;   // Zählt die Overflows von Timer0 => Eingang mit Prescaler
unsigned char sekcnt = 0;   // Leitet aus 10ms-Timer2-Overflow die Messintervalle her
unsigned char blink  = 0;   // Blinkender Punkt in Digit A signalisiert die Messfolge (bit 7)

unsigned char WERT_A = 63;  // 0 Segmentmuster der einzelnen Ziffern: Einer-Stelle
unsigned char WERT_B = 6;   // 1
unsigned char WERT_C = 134; // 1.   Diese werden zum Start mit der
unsigned char WERT_D = 113; // F    Firmware-Version initialisiert.
unsigned char WERT_E = 57;  // C
unsigned char WERT_F = 64;  // -                                    100000er-Stelle

void SEGMENTS_OUT (char pattern)
{
    PORTB = pattern;                    // Segmentmuster für A-F ausgeben
    if (pattern & 0b01000000) PORTD = PORTD | 0b00001000;   // Segment G ist extra
    if (pattern & 0b10000000) PORTD = PORTD | 0b00000100;   // Punkt ist extra
}

void MUXEN (void)                       // Multiplexen der LEDs
{
     delay ms(2);                       // Multiplex-Pause
    PORTB = 0x00;                       // Segmente A-F aus
    PORTD = PORTD & 0b11110011;         // Segmente G, P aus
    PORTC = ~(1<<digicnt);              // nächstes Digit einschalten
    switch (digicnt++)
    {
    case 0:                             // Die Einerstelle immer ausgeben
        SEGMENTS_OUT (WERT_A);
        break;
    case 1:                             // Leading Zero Blanking auf der 10er Stelle
        if (display >= 10) SEGMENTS_OUT (WERT_B);
        break;
    case 2:                             // für die weiteren Digits ebenso verfahren
        if (display >= 100) SEGMENTS_OUT (WERT_C);
        break;
    case 3:
        if (display >= 1000) SEGMENTS_OUT (WERT_D);
        break;
    case 4:
        if (display >= 10000) SEGMENTS_OUT (WERT_E);
        break;
    case 5:
        if ((display >= 100000) || (dot == 6)) SEGMENTS_OUT (WERT_F);
        digicnt = 0;
        break;
    }
}

void ausgeben (void)        // Aufbereiten der einzelnen Digits je nach Messbereich
{
    unsigned long tmp1,tmp2;                                // beide Zähler auslesen
    tmp1 = (((unsigned long) lowovf * 65536) + TCNT1);      // Typecasting verhindern
    tmp2 = (((unsigned long) highovf *  256) + TCNT0);      // K&R S.45, hier wichtig!
```

```
//  Beide uC Zähler, sowie beide Overflow-Zähler für die nächste Messung wieder löschen
    TCNT0   = 0;    // interner uC Zähler 0 ( 8 Bit)
    TCNT1   = 0;    // interner uC Zähler 1 (16 Bit)
    lowovf  = 0;    // globale Zählvariable (T1 Overflow)
    highovf = 0;    // globale Zählvariable (T0 Overflow)
    sei();          // nicht früher, damit 16Bit-Registerzugriffe nicht unterbrochen werden.
/*  Um den Messbereich zu bestimmen, reicht es den 24-Bit Wert T0(8) + highovf(16) zu betrachten.
    Durch die 1/100-Frequenzteilung sind die Limits ebenfalls durch 100 zu teilen. */
    if (tmp2 < 9980)        // entspricht < 1MHz => 000000..999999 Hz       Aufl. 1Hz
    {
        dot = 0;
        display = tmp1;
    }
    else if (tmp2 < 90000)  // entspricht < 9MHz => 1.00000..9.00000 MHz    Aufl. 10Hz
    {
        dot = 6;
        display = tmp1 / 10;
    }
    else                    // alles darüber     => 09.0001..84.0000 MHz    Aufl. 100Hz
    {
        dot = 5;
        display = tmp2;     // nur hier wird der Prescaler benutzt (ohne Umrechnung!)
    }

    WERT A = SEGMENTE[(display % 10)];  // Zuweisen der einzelnen Digits aus display: Einerstelle
    blink = blink ^ 128;                // 1x pro Zyklus Bit 7 toggeln (XOR)
    WERT A |= blink;                    // gleichbedeutend WERT_A = WERT_A | blink;
    WERT B = SEGMENTE[(display % 100 / 10)];
    WERT C = SEGMENTE[(display % 1000 / 100)];
    WERT D = SEGMENTE[(display % 10000 / 1000)];
    WERT E = SEGMENTE[(display % 100000 / 10000)];
    if (dot == 5) WERT E |=128;
    WERT F = SEGMENTE[(display % 1000000 / 100000)];
    if (dot == 6) WERT_F |=128;
}

//#############################################################################
ISR (TIMER2 OVF vect)   // Timer2 Interrupt: 10ms Zeitbasis für die Messungen
//#############################################################################
{
    TCNT2 = 16;                     // Timer 2 Offset für 10ms neu laden (s.u.)
    sekcnt++;
    if (sekcnt == 100)              // nach 1s Messbereich festlegen & ausgeben
    {
        sekcnt = 0;
        ausgeben();
    }
}
//#############################################################################
ISR (TIMER1 OVF vect)   // Timer1 Interrupt: Zähler Direkteingang PD5
//#############################################################################
{ lowovf++; }       // interne 8 Bit Zählvariable inkrementieren

//#############################################################################
ISR (TIMER0 OVF vect)   // Timer0 Interrupt: Zähler Prescaler 1/100 Eingang PD4
//#############################################################################
{ highovf++; }      // interne 16 Bit Zählvariable inkrementieren

int main (void)     // Hauptprogramm mit Initialisierungen
{
    DDRB  = 0x3F;   // Data Direction Register B: PB0..PB5 = 6 Segmentausgänge A-F
    DDRC  = 0x3F;   // Data Direction Register C: PC0..PC5 = 6 Digitausgänge (Kathoden)
    DDRD  = 0x0C;   // Data Direction Register D: PD2,3: Segmente G, P, PD4,5: Signaleingänge

    /*** Alle Timer initialisieren ***/
    TCCR0B = (1<<CS02 | 1<<CS01 | 1<<CS00); // Die Eingänge T0=PD4 und T1=PD5 als Impuls-
    TCCR1B = (1<<CS12 | 1<<CS11 | 1<<CS10); // zähler initialisieren (steigende Flanke)
    TCNT0  = 0;
    TCNT1  = 0;     /* Zählerstände löschen
    Setzen der internen Zeitbasis auf 10ms Overflow-Interrupts (Timer2, 8 Bit):
    Preload = 256 - (fQuarz * tov / Prescaler) = 256 - 24576000 * 0,01 / 1024 = 16 */
    TCCR2B = (1<<CS20 | 1<<CS21 | 1<<CS22); // die unteren 3 Bit für 1/1024 setzen
//  TCNT2 = 16; // dieser Offset wird bei jedem T2 Overflow neu geladen (hier unnnötig)
//  Für alle 3 Timer die (T)imer (O)verflow (I)nterrupts (E)nablen => TOIEx
    TIMSK0 = (1 << TOIE0);
    TIMSK1 = (1 << TOIE1);
    TIMSK2 = (1 << TOIE2);

    sei();              // globale Interrupt-Freigabe
    while(1) MUXEN();   // Main-Schleife zum Multiplexen des Displays
}
```