Ansteuerung eines HT16K33A LED-Treibers mit Mikrocontroller CH32V003

Der HT16K33A Baustein ist ein LED-Treiber für Leuchtdiodenmatrix oder LED-Segmentanzeigen mit I2C Interface.

Hierfür werden 8 Common-Leitungen (active low) für eine LED-Anzeige mit gemeinsamer Kathode geschaltet. Jede Commonleitung schaltet hier bis zu 16 Segmente, sodass die Organisation 8x16 LED‘s beträgt.

Der RAM des Bausteins ist Byteweise angeordnet, so dass alle geraden Adressen die Reihenleitungen 0..7, die ungeraden Adressen die Reihenleitungen 8..15 abbilden.

Die hier vorliegenden ht16k33.h / ht16k33.c kann hier jedoch nur max. 8 Digits 7-Segment-anzeigen bedienen, so dass die Reihen 8..15 brach liegen.

Nachfolgende Tabelle Zeigt den Zusammenhang von Digits der Anzeige zu den RAM-Adressen und Common-Leitungen.

	Treiber RAM-Adresse	**Common-Leitungen**	**Reihen (Segmente)**
Digit 0	0	0	0..7
Digit 8	1	0	8..15
Digit 1	2	1	0..7
Digit 9	3	1	8..15
Digit 2	4	2	0..7
Digit 10	5	2	8..15
Digit 3	6	3	0..7
Digit 11	8	3	8..15
Digit 4	9	4	0..7
Digit 12	9	4	8..15
Digit 5	10	5	0..7
Digit 13	11	5	8..15
Digit 6	12	6	0..7
Digit 14	13	6	8..15
Digit 7	14	7	0..7
Digit 15	15	7	8..15

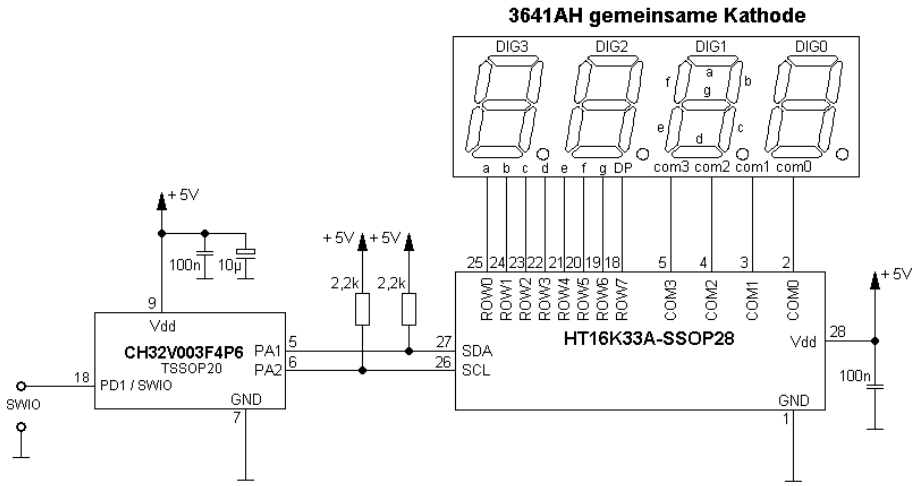
Die I2C-Adresse (8-Bit) lautet 0xE0.

Schaltplan

Der Schaltplan zeigt keine Besonderheiten auf. Da für das Ansprechen des HT16K33 Software-I2C verwendet wird (Bitbanging), sind die Anschlüsse für SDA und SCL per Default NICHT an den Pins, an denen Hardware-I2C ausgegeben wird, sondern an PA1 und PA2 (weil die beim Autor eben gerade frei waren).

In der Datei i2c_sw.h können diese Anschlüsse jedoch den gewünschten Gegebenheiten angepasst werden.

4-Digit 7-Segmentanzeige (CH32V003 µC mit HT16K33A)



Software

Die vorliegende Software baut auf dem Framework CH32FUN von Christian Lohr unter Linux auf. Das gesamte Framework ist unter:

https://github.com/cnlohr/ch32fun

downloadbar.

Zum Übersetzen der vorliegenden Software ist dieses jedoch nicht notwendig, weil die für einen CH32V003 benötigten Dateien aus diesem Framework im Ordner ***include*** vorliegen und nicht die gesamten Dateien aus ***ch32fun*** benötigt werden.

Zudem wurde das Makefile den „***Vorlieben“*** des Autors angepasst:

- der benötigte Compiler riscv-none-elf-gcc muss systemweit verfügbar sein, der Compiler ist downloadbar (Linux) unter:
    - https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack/releases/ (Assets)
    - https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack/releases/download/v14.2.0-3/xpack-riscv-none-elf-gcc-14.2.0-3-linux-arm64.tar.gz (gesamtes Compilerpaket Version 14.2.0.3)

- das Kommandozeilenprogramm ***minichlink*** zum Flashen des CH32V003 Controllers muß ebenfalls systemweit verfügbar sein. ***minichlink*** ist Bestandteil des CH32FUN Frameworks und ist auch dort zu finden. Um als einfacher User den Zugang zu ***minichlink*** und dem USB-Port zu haben, nicht vergessen, die Rules-Datei nach /etc/udev/rules.d (als root) zu kopieren. Hinweis: ein CH32V003 ist nur mit einem
  WCH-LinkE zu flashen, diverse Anbieter verkaufen einen WCH-Link (ohne „E“) und mit diesem ist der Chip nicht programmierbar. Wer keinen WCH-LinkE besitzt oder kaufen möchte oder keinen erhält, kann sich aus einem Arduino / ATmega mit USB2UART-Bridge und einem CH32V003 Chip einen Programmer bauen (eine Anleitung hierfür ist für die Zukunft geplant)

- das Makefile wurde dahingehend geändert, dass ein Aufruf von make nicht mehr eine Quelldatei compiliert und bei fehlerfreiem Compilergang anschließend sofort flasht, sondern make erzeugt nur eine Binär- und Hexdatei. Diese kann mit mit dem Aufruf von make flash dann in den Controller übertragen werden (diese Vorgehensweise entspricht den eingerichteten Menüpunkten der Toolchain des Autors, durch dieses Verhalten kann grundsätzlich „geprüft“ werden, ob ein Programm syntaktisch korrekt ist)

Verzeichnisbaum aller Dateien

```
ht16k33
   |
   |
   +---- ch32fun
   |        |
   |        +---- ch32fun.c                  ( Frameworkdateien CH32FUN )
   |        |
   |        +---- ch32fun.h                  ( dazugehoerender Header )
   |        |
   |        +---- ch32fun.ld                 ( Linkerscript fuer CH32V003 )
   |        |
   |        +---- ch32v003hw.h               ( Hardwaredeklarationen zu CH32V003 / Registeradressen )
   |
   +---- doku
   |        |
   |        +---- ht16k33_b_tssop28.gif      ( Schaltplan )
   |        |
   |        +---- ht16k33a_4digit_7seg.pdf   ( diese Dokumentation hier )
   |
   +---- include
   |        |
   |        +---- ch32v003_gpio.h            ( Headerdatei / Wrapper zum Ansprechen der GPIO-Pins )
   |        |
   |        +---- funconfig.h                ( Konfiguration zum CH32FUN-Framework )
   |        |
   |        +---- ht16k33.h                  ( Headerdatei fuer Funktionen zum LED-Treiber )
   |        |
   |        +---- i2c_sw.h                   ( Header zur Benutzung von Bitbanging I2C-Funktionen )
   |
   +---- src
   |       |
   |       +---- ht16k33.c                  ( Funktionen fuer den LED-Anzeigetreiber )
   |       |
   |       +---- i2c_sw.c                   ( Funktionen fuer I2C mittels Bitbanging )
   |
   |
   +---- Makefile                           ( Datei zum Builden / Compilieren und Flashen )
   |
   +---- ch32v003.mk                        ( funktionaler Teil, wird von Makefile implementiert )
   |
   +---- ht16k33_demo.c                     ( Demoprogramm zum HT16K33A )
```

I2C - Bitbanging

Um einen (sehr) einfachen (ersten) Zugang wurden I2C-Funktionen mittels Bitbanging realisiert. Dieses erleichtert ein Portieren auf ein anderes Microcontrollersystem ungemein, es sind lediglich entsprechende Header zu ändern und das Ansprechen von GPIO-Pins zu implementieren.

Bei diesem Bitbangingfunktionen hier geschieht ein Setzen und Rücksetzen eines GPIO-Pins folgenderweise:

- eine logische 1 wird realisiert, in dem der entsprechende Pin als floatender Eingang (Eingang ohne Pull-Up Widerstand) geschaltet wird. Die logische 1 komm durch den externen „I2C-Pullup Widerstand“ von 2,2 kΩ zustande und ist u.a. deswegen wichtig, damit ein Slave die SDA-Leitung bei einem Acknowledge auf 0 legen kann
- eine logische 0 wird realisiert, in dem der entsprechende Pin als Ausgang geschaltet und eine logische 0 angelegt wird.

Zum Verständnis der I2C-Adresse: Es „streiten“ sich die Gelehrten, ob die Adresse eines I2C-Gerätes nun eine 7-Bit oder eine 8-Bit Adresse besitzt. Technisch gesehen ist dieses sogar klar definiert, es hat eine 7-Bit Adresse, gefolgt von einem read- (logisch1) oder einem write-Bit (logisch 0). Jetzt kommt das große ABER:

Da die 7-Bitadresse um eine Bitposition innerhalb eines Bytes um eine Position nach links verschoben werden muß, kann man innerhalb eines Bytes die Adresse so „interpretieren“, dass das read/write Bit Bestandteil der Adresse ist (was in vielen Fällen den Umgang erleichtert). Am Beispiel des HT16K33A WÄRE die Adresse in Wirklichkeit 0x70, die um eine Stelle nach links geschoben werden muß (was dann 0xE0 ergibt).

0xE0 wäre demzufolge ein Schreibzugriff auf die Adresse 0x70, 0xE1 wäre ein Lesezugrif auf die Adresse 0x70. Innerhalb dieser I2C Bitbangingfunktionen wird die I2C-Adresse als ein 8-Bit Wert begriffen (hinsichtlich des HT16K33A also 0xE0) bei dem das niederwertigste Bit mittels logischer Verknüpfung AND / OR zwecks Zugriff zum Schreiben oder Lesen gelöscht oder gesetzt wird.

i2c_sw.h

Innerhalb dieser Headerdatei kann folgendes eingestellt werden:

Die GPIO-Pins mit denen eine I2C-Kommunikation vorgenommen wird (hier Default PA1 für SDA und PA2 für SCL)

```
#define i2c_sda_hi()      PA1_float_init()
#define i2c_sda_lo()      { PA1_output_init(); PA1_clr(); }
#define i2c_is_sda()      is_PA1()

#define i2c_scl_hi()      PA2_float_init()
#define i2c_scl_lo()      { PA2_output_init(); PA2_clr(); }
```

Manchen I2C-Devices ist selbst das I2C-Bitbanging noch zu schnell (so auch dem HT16K33A) und über die aufgeführten defines kann das Timing verlangsamt werden. Der Wert hinter den #define entspricht etwas mehr als 1µSekunde.

```
#define short_puls     1            // Einheiten fuer einen langen Taktimpuls
#define long_puls      1            // Einheiten fuer einen kurzen Taktimpuls
#define del_wait       1            // Wartezeit fuer garantierten 0 Pegel SCL-Leitung
```

i2c_sw.c

Folgende Funktionen sind in i2c_sw.c definiert:

i2c_master_init

void i2c_master_init(void);

setzt die Pins die für den I2C Bus realisieren auf logisch 1

i2c_sendstart

void i2c_sendstart(void);

erzeugt die Startcondition des I2C Buses

i2c_start

uint8_t i2c_start(uint8_t addr);

erzeugt die Startcondition auf dem I2C Bus und schreibt anschließend eine 8-Bit Deviceadresse auf den Bus

Rückgabe: 0 wenn ein I2C-Slave geantwortet hat (Acknowledge)
1 wenn kein I2C-Slafe geantwortet hat

i2c_startaddr

uint8_t i2c_startaddr(uint8_t addr, uint8_t rwflag);

Diese Funktion ist eine „Kompatibilitätsfunktion“ für eine Startfunktion, bei der die I2C-Deviceadresse als 7-Bit Adresse angegeben wird und das read/write Flag als Argument gesondert mit angegeben wird.

Übergabe: addr : 7-Bitadresse des I2C-Slaves
rwflag : 0 wenn auf den Bus geschrieben werden soll
1 wenn vom Bus gelesen werden sollt

i2c_stop

void i2c_stop(void);

erzeugt die Stopcondition auf dem I2C Bus

i2c_write_nack

void i2c_write_nack(uint8_t data);

schreibt den Wert in ***data*** auf dem I2C Bus OHNE ein Acknowledge einzulesen

i2c_write

uint8_t i2c_write(uint8_t data);

schreibt den in ***data*** Wert auf dem I2C Bus und liest ein Acknowledge ein.

Rückgabe: > 0 wenn Slave ein Acknowledge gegeben hat
== 0 wenn kein Acknowledge vom Slave

i2c_write16

uint8_t i2c_write16(uint16_t data);

schreibt den 16 Bit Wert (2Bytes) in ***data*** auf dem I2C Bus und liest ein Acknowledge ein

Rückgabe: > 0 wenn Slave ein Acknowledge gegeben hat
== 0 wenn kein Acknowledge vom Slave

i2c_read

uint8_t i2c_read(uint8_t ack);

liest ein Byte vom I2c Bus

Übergabe: 1 : nach dem Lesen wird dem Slave ein Acknowledge gesendet
0 : es wird kein Acknowledge gesendet

Rückgabe: vom I2C-Bus gelesenes Byte

HT16K33A

Die Software für den LED-Treiber HT16K33A besteht aus einem Dateipaar ht16k33.h und ht16k33.c

In der Headerdatei sind ausser der Deviceadresse (8-Bit Adresse) keine Angaben zu machen. Der Defaultwert ist hier: 0xE0. Dieser Wert ist nur zu ändern, wenn dem Chip per Verdrahtung eine andere Deviceadresse gegeben wird.

Die Funktionen in ht16k33.c werden mittels eines 8 Byte großen Framebuffers realisiert, der somit maximal 8 7-Segment Digits ansteuern kann. Grundsätzliches Vorgehen beim Umgang mit dem HT16K33 ist zuerst ein Beschreiben des Framebuffers und ggf. einem Setzen des Dezimalpunktes mit anschließendem Transfer zum Chip.

Funktionen für ein 16-Segment Digit sowie ein Einlesen von Tasten sind (noch) nicht realisiert, weil dem Autor zum einen keine 16-Segmentanzeigen zur Verfügung stehen und für mehr als eine 8-stellige Anzeige keine Verwendung gegeben ist. Bei möglichen Funktionen zum Einlesen von Tasten „stört“, dass hier grundsätzlich zusätzlich zum Chip noch Widerstände benötigt werden.

Sollte es einen Bedarf für das Einlesen von Tasten oder mehr als 8-Digit Anzeigen (beim Autor) geben, werden die benötigten Funktionen dann wohl implementiert werden, vorgesehen ist dieses momentan jedoch nicht.

Verfügbare Funktionen in ht16k33.c

ht16k_init

void ht16k_init(void);

initialisiert den Displaytreiber.

Hinweis: ht16k_init initialisiert nur den Treiberchip und nicht die I2C-Schnittstelle. Diese muß vor Nutzung von ht16k_init bereits initialisiert worden sein.

ht16k_wrcmd

uint8_t ht16k_wrcmd(uint8_t val);

schreibt einen einzelnen Wert an die I2C-Adresse des Displaytreibers

Übergabe: val : zu schreibender Wert

Rückgabe: Acknowledge bei erfolgreichem Schreiben

ht16k_wr2ram

uint8_t ht16k_wr2ram(uint8_t val, uint8_t segaddr);

schreibt ein einzelnes Byte val an die Display-RAM-Adresse segaddr.
segaddr ist hierbei die Speicheradresse im HT16K33A Chip

Übergabe: val : zu schreibender Wert
segaddr : Adresse, an der der Wert in val geschickt wird. Zulässige Werte für segaddr sind 0..15 (16 Byte Ram im LED-Treiber)

Rückgabe: Acknowledge

ht16k_wrbuffer

void ht16k_wrbuffer(void);

schreibt den Inhalt des Framebuffers in das RAM des Displays.

Da eine Commonleitung des Treibers 16 ROW-Leitungen betreibt, besteht ein Digit einer Commonleitung aus 16-Bits = 2 Bytes. Aus diesem Grund belegt die Adresse eines Digits im Treiber immer 2 Byte, so dass für ein 7-Segment Display dann auch 2 Adressen benötigt werden. Das höherwertige Byte in Verbindung mit einer max. 8 stelligen 7-Segment Anzeige ist somit unbenutzt.

ToDo:

Framebuffer auf 16-Byte erhöhen und die Software hier so gestalten, dass beim Schreiben des Buffers weitere 8 Digits an die Row-Leitungen 8..15 angeschlossen werden können.

ht16k_setbrightness

uint8_t ht16k_setbrightness(uint8_t val);

stellt die Helligkeit der Anzeige ein

Übergabe: val : einzustellende Helligkeit, erlaubte Werte sind 0x00 (dunkel) .. 0x0f (max. Helligkeit)

Rückgabe: Acknowledge

ht16k_setdez

void ht16k_setdez(int32_t value, uint8_t pos, uint8_t nozero);

zeigt einen 32-Bit Integerwert dezimal auf der 8-stelligen Anzeige an. Hier geschieht zuerst ein Beschreiben des Framebuffers mit einer anschließenden Übertragung des Framebuffers an den Chip.

Übergabe: value : anzuzeigender Wert
pos : Position, ab der der Wert rechtsbündig angezeigt wird
nozero : 1 = führende Nullen werden unterdrückt
0 = werden nicht unterdrückt

ht16k_sethex

void ht16k_sethex(int32_t value, uint8_t pos, uint8_t nozero);

zeigt einen 32-Bit Integerwert hexdezimal auf der 8-stelligen Anzeige an. Hier geschieht zuerst ein Beschreiben des Framebuffers mit einer anschließenden Übertragung des Framebuffers an den Chip.

Übergabe: value : anzuzeigender Wert
pos : Position, ab der der Wert rechtsbündig angezeigt wird
nozero : 1 = führende Nullen werden unterdrückt
0 = werden nicht unterdrückt

ht16k_setdp

void ht16k_setdp(uint8_t pos, uint8_t enable);

setzt oder löscht einen Dezimalpunkt indem ein Dezimalpunkt im Framebuffer eingetragen und dieser dann anschließend an den Chip übertragen wird.

Übergabe: pos : Position, an der ein Dezimalpunkt gesetzt oder gelöscht werden soll.
enable : 1 = Dezimalpunkt wird gesetzt
0 = Dezimalpunkt wird gelöscht

ht16k_puts

void ht16k_puts(char *s, uint8_t anz);

schreibt einen String in die Anzeige. Hierbei sind die Darstellungsmöglichkeiten aufgrund von nur 7-Segmenten begrenzt und manche Zeichen sehen hierdurch etwas „merkwürdig“ aus. Im globalen Array

const uint8_t bmps7asc []

sind die Bitmaps der Zeichen abgebildet und können dort den eigenen Wünschen angepasst werden.

Übergabe *s : Zeiger auf einen zu schreibenden nulterminierten String
anz : Anzahl Digits der Anzeige

Anhang: Beispielprogramm ht16k33_demo.c

```
/* ----------------------------------------------------------------
                            ht16k33_demo.c

     Demo fuer die Verwendung eines HT16K33 LED-Treibers mit I2C-
     Interface.

     Hier fuer Anschluss mit max. 8-Digits

     MCU   :  CH32V003
     Takt  :

     13.05.2025  R. Seelig
   ---------------------------------------------------------------- */

/*
              CH32V003 A4M6                                          CH32V003F4P6
              +-----------+                                        +--------------+
    PC1 / sda |1    C   16| PC0                          PD4 / a7  | 1         20 | PD3 / a4
    PC2 / scl |2    H   15| Vdd                   PD5 / a5 / utx  | 2    C    19 | PD2 / a3
          PC3 |3    3   14| gnd                   PD6 / a6 / urx  | 3    H    18 | PD1 / swio
          PC4 |4    2   13| PA2 / osc0 / a0           nrst / PD7  | 4    3    17 | PC7 / miso
   PC6 / mosi |5    V   12| PA1 / osc1 / a1      PA1 / osc0 / a0  | 5    2    16 | PC6 / mosi
   PC7 / miso |6    0   11| nrst / PD7           PA2 / osc1 / a1  | 6    V    15 | PC5 / sck / scl
   PD1 / swio |7    0   10| PD6 / a6 / urx                   gnd  | 7    0    14 | PC4 / a2
     PD4 / a7 |8    3    9| PD5 / a5 / utx                   PD0  | 8    0    13 | PC3
              +-----------+                                  Vdd  | 9    3    12 | PC2 / scl
                                                             PC0  | 10        11 | PC1 / sda
                                                                  +--------------+
*/

#include <string.h>

#include "ch32fun.h"
#include "ch32v003_gpio.h"

#include "ht16k33.h"


#define laufanz    12

// Laufmuster fuer ein Lauflicht auf einer 4 stelligen 7-Segmentanzeige
// LSB beinhaltet das Bitmap des Digits, MSB die
// Adresse des Digits
uint16_t laufmust[laufanz] =
{
  0x0601, 0x0401, 0x0201, 0x0001, 0x0002, 0x0004,
  0x0008, 0x0208, 0x0408, 0x0608, 0x0610, 0x0620
};

/* ------------------------------------------------------
                            main
   ------------------------------------------------------ */
int main(void)
{
  uint16_t  cx;
  uint8_t   i;
  uint16_t  val;
  uint8_t   lloops;

  SystemInit();
  i2c_master_init();

  ht16k_init();

  while(1)
  {
    // Lauflichtausgabe
    lloops= 0;
    do
    {
      for (i= 0; i< laufanz; i++)
      {
```

```
        // Anzeige loeschen
        for (cx= 0; cx< 4; cx++)
        {
          ht16k_wr2ram(0, cx << 1);
        }

        val= laufmust[i];
        ht16k_wr2ram(val & 0x00ff, val >> 8);
        delay(50);
      }
      lloops++;
    } while(lloops < 2);

    // Hexadezimale Zahl mit "dunkelster" Anzeige ausgeben
    ht16k_setbrightness(0);
    ht16k_sethex(0xabcd,0,0);

    // und diese Aufblenden
    delay(500);
    for (i= 0; i< 15; i++)
    {
      ht16k_setbrightness(i);
      delay(100);
    }
    delay(1000);

    // Zaehler
    for (cx= 990; cx < 1101; cx++)
    {
      // Dezimalzahlenausgabe, rechtsbuendig (pos= 0); Unterdrueckung fuehrender Nullen
      ht16k_setdez(cx,0,1);
      // Anzeige Dezimalpunkt (eine Kommastelle)
      ht16k_setdp(1,1);
      delay(100);
    }
    delay(1000);
  }
}
```