Beschreibung Source- und Headerdateien

ch32v003_gpio.h

ch32v003.h ist eine Wrapperdatei die dazu dient, mit Projekten anderer MCU's kompatibel zu sein. Mittels dieser werden GPIO's nach einem einheitlichen Schema angesprochen. Hierzu werden Makros deklariert, die nach einem einheitlichen "Schlüssel" aufgebaut sind.

Namensgebung der Makros:

PINNAME_output_init()
PINNAME_set()
PINNAME_clr

PINNAME_input_init()
PINNAME_float_init()
is_PINNAME()

Hierbei gibt PINNAME die Bezeichnung eines GPIO-Pins an. Hierbei gilt:

- output_init: initialisiert einen GPIO als Ausgang
- _set: setzt einen als Ausgang initialisierten Ausgang auf eine logische 1
- _clr: setzt einen als Ausgang initialisierten Ausgang auf eine logische 0

- input_init: initialisiert einen GPIO als Eingang mit internem Pullup-Pulldown Widerstand
- float_init: initialisiert einen GPIO als Eingang ohne internem Pullup-Pulldown Widerstand (float)
- is_: ermittelt den logischen Zustand an einem GPIO

Beispiele:

PC3 als Ausgang

```
PC3_output_init();
PC3_set();              // setzt PC3 auf 1
PC3_clr();              // setzt PC3 auf 0
```

PC4 als Eingang mit Pullup-Pulldown Widerstand

```
PC4_input_init();
level= is_PC4();        // der logische Zustand an GPIO-Pin PC4 wird
                        // der Variable levelzugewiesen
```

uart.h / uart.c

uart.h / .c stellt einfache, rudimentäre Funktionen zum Umgang mit einer seriellen Schnittstelle zur Verfügung.

Anschluesse der seriellen Schnittstelle sind:

PD6 fuer Receive-Data RxD
PD5 fuer Transmit-Data TxD

Mittels eines Define-Schalters in uart.h kann festgelegt werden, ob eine Funktion zum Einlesen eines Integerwertes mit in den Code eingebunden werden soll oder nicht:

```
#define readint_enable     0    // 0 => keine readint-Funktion
                                // 1 => readint-Funktion verfuegbar
```

Funktionen:

uart_init(uint32_t baudrate);

initialisiert serielle Schnittstelle mit Protokoll 8N1

Argumente:
baudrate : einzustellende Baudrate

void uart_putchar(uint8_t ch);

sendet ein Zeichen auf der seriellen Schnittstelle

Argumente:
ch : zu sendendes Zeichen

uint8_t uart_getchar(void);

wartet, bis auf der seriellen Schnittstelle ein Zeichen eingegangen ist und liest dieses ein.

Rückgabe:
gelesenes Zeichen

uint8_t uart_ischar(void);

testet, ob auf der seriellen Schnittstelle ein Zeichen eingegangen ist, liest dieses aber nicht ein und wartet auch nicht auf den Eingang eines Zeichens.

Rückgabe:
1 : es ist ein Zeichen eingetroffen
0 : kein Zeichen verfügbar

int16_t readint(void);

liest einen 16-Bit signed Integer auf der seriellen Schnittstelle ein (allerdings reicht der Eingabebereich nur von -32767 .. +32767).

Korrektur ist mit der Löschtaste nach links moeglich.

readint ist nur verfügbar, wenn in uart.h das entsprechende define aktiviert ist:

```
#define readint_enable     1
```

Beispiel:

```
value= readint();
```

term_colpos.h / term_colpos.c

term_colpos beinhaltet Funktionen und Deklarationen, um die Zeichenausgabe in einem seriellen Terminal hinsichtlich Farben und Ausgabeposition steuern zu können.

Für die Ausgabe in einem seriellen Terminal wird zusätzlich die Software aus my_printf.h / my_printf.c benötigt.

Die Nummerierung der darstellbaren 16 Farben erfolgt nach dem Schema der (ur)alten EGA-Grafikkarte:

```
#define black           0
#define blue            1
#define green           2
#define cyan            3
#define red             4
#define magenta         5
#define brown           6
#define grey            7
#define darkgrey        8
#define lightblue       9
#define lightgreen      10
#define lightcyan       11
#define lightred        12
#define lightmagenta    13
#define yellow          14
#define white           15
```

Innerhalb von term_colpos werden Farbattribute verwendet. Ein Farbattribut bedeutet hier, dass in einem Byte (uint8_t) die Farben für Hintergrund und Vordergrund gespeichert ist.

Hierbei gilt: die oberen 4 Bits (oberes Nibble) repräsentieren die Farbnummer für den Hintergrund, wobei keine Darstellung von intensiven Farben möglich ist.

Die unteren 4 Bits (unteres Nibble) repräsentieren die Textfarbe.

Beispiel:

```
settextattr(0x1e);            // setzt für den Hintergrund die Farbe 1
                              // (blau) mit Textfarbe 15 (0x0e=15=gelb)
```

Funktionen:

void clrscr(void);

löscht den Bildschirminhalt des seriellen Terminals

void gotoxy(uint8_t x, uint8_t y);

Positioniert den Textcursor auf die Position der naechsten Textausgabe im seriellen Terminal.

Argumente:

x,y : Textkoordinate auf die der Cursor positioniert wird. Die Koordinate 1,1 repraesentiert hierbei die linke obere Ecke

void settextattr(uint8_t attr);

setzt die Farben für Hintergrund und Text. Hierbei repräsentieren die oberen 4 Bit die Hintergrund-, die unteren 4 Bit die Textfarbe. Die Farbnummerierung erfolgt nach dem EGA-Farbschema

Argumente:

attr : Farbattribut für Hinter- und Vordergrundfarbe

Makros:

textcolor(col)

legt die Textfarbe nach dem EGA-Farbschema fest

bkcolor(col)

legt die Hintergrundfarbe nach dem EGA-Farbschema fest

my_printf.h / my_printf.c

my_printf ist ein in der Funktionalität reduzierter Ersatz für printf, speziell zur Benutzung in Verbindung mit Mikrocontrollern. Er macht dann Sinn, wenn es darum geht, (Flash)speicher zu sparen und ermöglicht den Einsatz auf Mikrocontrollersystemen ab 2 kByte Flashspeicher.

Ein vom Standard abweichendes Umwandlungszeichen %k ermoeglicht es, Pseudo-Kommazahlen auszugeben.

my_printf benötigt zur Ausgabe irgendwo in den zu einem Programmprojekt gehörenden Dateien (vorzugsweise in der Datei, die die main-Funktion beinhaltet) eine Funktion namens:

```
void my_putchar(char ch);
```

my_printf bedient sich dieser Funktion zur Zeichenausgabe. Soll bspw. ein Programm die Ausgaben auf der seriellen Schnittstelle vornehmen, existiert wahrscheinliche eine Funktion ähnlich

```
void uart_putchar(char ch);
```

Um diese Ausgabefunktion nutzen zu können, kann ein einfaches Programm folgendermassen aussehen:

```
#define printf    my_printf

 /* ----------------------------------------------------------
    my_putchar

    wird von my-printf aufgerufen und hier muss
    eine Zeichenausgabefunktion angegeben sein, auf das
    printf dann schreibt !
    ---------------------------------------------------------- */
 void my_putchar(char ch)
 {
   uart_putchar(ch);
 }

 int main(void)
 {
   uart_init(115200);
   printf("\n\r Hallo Welt\n\r");
   while(1);
 }
```

Umwandlungszeichen my_printf

Zeichen	Datentyp und Darstellung
%d	Integer als dezimale Zahl ausgeben int ist 16-Bit Wert auf 8-Bit Systemen int ist 32-Bit Wert auf 32-Bit Systemen
%k	(Pseudo-Kommazahl) Integerwert mit eingesetztem Kommapunkt ausgeben In der globalen Variable printfkomma wird angegeben, an welcher Position ein Dezimalpunkt ausgegeben wird.
%c	Ausgabe eines char als (Ascii)-Zeichen
%s	Zeichenkette (String) ausgeben
%x	Integer als hexadezimale Zahl ausgeben
%%	Ausgabe des Prozentzeichens

Beispiel:

```
#define printf    my_printf

nt a, a2, b, c;

printf("\n\r ASCII-Zeichen '%c' = dezimal %d = hexadezimal 0x%x\n\r", 'M', 'M', 'M');

a= 42; b= 13;

// Variable a um eine Zehnerstelle nach links,
// um spaeter eine Nachkommastelle anzeigen zu koennen

a2= a*100;
c= a2 / b;

printfkomma= 2;
printf(" %d / %d = %k", a, b, c);
```

Ausgabe:

ASCII-Zeichen 'M' = dezimal 77 = hexadezimal 0x4D
42 / 13 = 3.23