PixBlock - Software

Die PixBlock-Software wurde in C++ für Atmel-AVR-Controller geschrieben. Sie lässt sich als Arduino-Library einsetzen, kann aber auch für andere AVR-Controller verwendet werden.

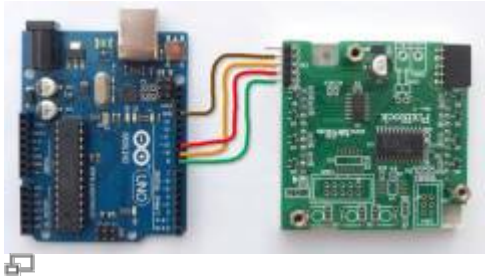
Der Source-Code steht auf Github zur Verfügung. Für den Arduino Uno ist ein einfaches Demo-Programm enthalten. Wer nicht mit der Arduino-Entwicklungsumgebung arbeitet, kopiert sich einfach die Dateien aus „libraries→PixBlock“ in sein Projektverzeichnis. Es werden die in <inttypes.h> definierten Datentypen verwendet. Die Typen „uint8_t“ und „uint16_t“ entsprechen den Arduino-Datentypen „byte“ und „word“.

Konfiguration

Die Konfiguration der Software geschieht im Headerfile „dot_matrix.h“. Hier muss insbesondere angegeben werden, wieviele PixBlocks in X- und Y-Richtung angeschlossen sind.

```
// dot matrix display
#define NUM_BLOCKS_X  3    // number of PixBlocks in horizontal direction
#define NUM_BLOCKS_Y  3    // number of PixBlocks in vertical direction
```

Die PixBlocks bilden eine Kette aus NUM_BLOCKS_X * NUM_BLOCKS_Y Blöcken.



Verbindung mit einem Arduino

Welche Pins zur Ansteuerung verwendet werden, kann ebenfalls festgelegt werden. Standardmäßig sind dies:

< 22em 5em 6em 11em >		
Signal	**Pin (AVR)**	**Pin (Arduino Uno)**

SDI	PB0	D8
CLK	PB1	D9
LATCH	PB2	D10

```
// hardware pins for accessing the dot matrix
#define DM_DATA_DDR   DDRB
#define DM_DATA_PORT  PORTB
#define DM_DATA_BIT   0
#define DM_CLK_DDR    DDRB
#define DM_CLK_PORT   PORTB
#define DM_CLK_BIT    1
#define DM_LATCH_DDR  DDRB
#define DM_LATCH_PORT PORTB
#define DM_LATCH_BIT  2
```

Klassen

Es gibt zwei Klassen, DotMatrix und Ticker. Während sich DotMatrix um die Ansteuerung der PixBlocks kümmert, dient Ticker zur Darstellung von Lauftexten.

```
// global variables
DotMatrix   dm;    // PixBlock
```

Multiplexing

Der PixBlock wird spaltenweise gemultiplext, d. h. es wird in schneller Folge jeweils eine der 8 LED-Spalten angezeigt. Durch den schnellen Wechsel ergibt sich ein stehendes Bild. Über die Einschaltdauer werden pro LED ausserdem vier Helligkeitsstufen realisiert (100 %, 50 %, 25 %, aus). So entstehen neben dem „Aus“-Zustand 15 mögliche Mischfarben.

update()

Nach der Initialisierung der DotMatrix werden alle diese Aufgaben durch die „update“-Methode erledigt. Sie muss dazu periodisch aufgerufen werden. Am besten geschieht dies in einer Interrupt-Routine.

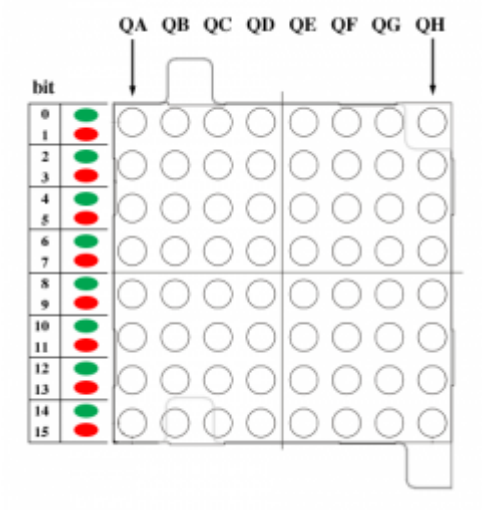
```
// dot matrix system timer
#define DM_REFRESH_FREQ   2500    // dot matrix column refresh rate (Hz)
#define DM_PRESCALER  32    // prescaler division ratio (1, 8, 32 or 64)

#define DM_REFRESH    (uint8_t)(0.5 + F_CPU / ((float)DM_REFRESH_FREQ *
(float)DM_PRESCALER))
#if DM_PRESCALER == 1
#define DM_CS 1
#elif DM_PRESCALER == 8
```

```
#define DM_CS 2
#elif DM_PRESCALER == 32
#define DM_CS 3
#elif DM_PRESCALER == 64
#define DM_CS 4
#else
#error Bad prescaler setting (DM_PRESCALER)
#endif

void setup()
{
    dm.init();

    // timer2 is used as dot matrix system timer
    ASSR = 0;            // use internal clock
    OCR2A = DM_REFRESH;        // set dot matrix refresh time
    TCCR2A = 0;            // normal mode
    TCCR2B = (DM_CS << CS00);
    TIMSK2 = (1 << OCIE2A);

    sei();                // enable interrupts

    // <your setup code goes here>
}

// dot matrix refresh interrupt
ISR(TIMER2_COMPA_vect)
{
    OCR2A += DM_REFRESH;    // setup next interrupt cycle
    sei();
    dm.update();        // do the PixBlock multiplexing
}
```

Bildspeicher

Der Bildinhalt wird in dem Array „screen“, das aus Pixel-Columns besteht, abgelegt. Pro PixBlock werden hierfür 32 Bytes an RAM-Speicher benötigt.

Pixel-Column



Zuordnung von Bits und Leds

Eine Pixel-Column ist eine Datenstruktur, welche eine Spalte aus 8 untereinander liegenden Pixeln repräsentiert. Jedes Pixel besitzt eine rote und eine grüne LED, so dass sich eine Pixel-Column mit einem 16-Bit-Wert beschreiben ließe. Da jedoch jede LED vier Helligkeitsstufen besitzt, verwenden wir zwei 16-Bit-Werte, einen für die höherwertigen Helligkeitsbits und einen für die niederwertigen.

```
typedef struct {
    uint16_t lsb;    // least significant bits
    uint16_t msb;    // most significant bits
} pixcol_t;
```

Hidden Screen

Falls genügend RAM-Speicher vorhanden ist, kann neben dem sichtbaren ein weiterer unsichtbarer Bildspeicher angelegt werden.

```
// dot matrix display
#define ENABLE_HIDDEN_SCREEN  // if defined two screens (visible & hidden)
are implemented
```

selectScreen(VISIBLE / HIDDEN)

swapScreen()

Die beiden Speicher werden über die Konstanten VISIBLE und HIDDEN identifiziert. Mit der Methode „selectScreen“ wählt man aus, auf welchem Bildspeicher zukünftige Befehle arbeiten sollen; „swapScreen“ vetauscht beide Speicher.

```
DotMatrix   dm;

dm.selectScreen(HIDDEN);
dm.swapScreen();
```

clearScreen()

Mit „clearScreen“ wird der aktuell ausgewählte Bildspeicher gelöscht.

Pixel-Ansteuerung

setPixel(x, y, color)

getPixel(x, y)

Einzelne Pixel lassen sich mit „setPixel“ setzen bzw. mit „getPixel“ abfragen. Der Koordinatenursprung liegt in der linken oberen Ecke. Für einige Farben sind in „dot_matrix.h“ entsprechende Konstanten vordefiniert.

```
dm.setPixel(7, 2, ORANGE);
if (dm.getPixel(7, 2, ORANGE) != ORANGE) {
    // this is strange
}
```

Grafikblöcke

displayGraphics(x, y, mode, graphics-pointer, memory-type, length)

Grafikblöcke mit einer Höhe von 8 Pixeln stellt man mit „displayGraphics“ dar. Die ersten beiden Parameter geben die x/y-Koordinate an. Danach folgt der Verknüpfungsmodus. Er kann folgende Werte annehmen:

- OPAQUE: Der Bildinhalt wird durch den Grafikblock überschrieben.
- TRANSPARENT: Schwarze Pixel im Grafikblock werden als transparent betrachtet.
- XOR: Bildinhalt und Grafikblock werden exklusiv-oder verknüpft.

Der nächste Parameter ist ein Zeiger auf eine Reihe von 16-Bit-Werten. Jeweils zwei aufeinanderfolgende Werte stellen LSB und MSB einer Pixel-Column dar. Außerdem muss angegeben werden, auf welche Speicherart (RAM, FLASH oder EEPROM) sich der Zeiger bezieht. Der letzte Parameter teilt mit, wieviele Pixel-Columns der Grafikblock breit ist.

```
// 8 x 8 graphics block predefined in "dot_matrix.h"
const uint16_t PROGMEM rainbow[16] = {
    0xEE20, 0xFA80, 0x7B88, 0xFEA0, 0xDEE2, 0x7FA8, 0x77B8, 0x5FEA,
    0x1DEE, 0x57FA, 0x477B, 0x15FE, 0x11DE, 0x057F, 0x0477, 0x015F
};

// show rainbow
dm.displayGraphics(0, 0, OPAQUE, rainbow, FLASH, 8);
```

pattern2PixCol(pattern, color, pixel-column-pointer)

Mit dieser Methode kann man ein 8-Bit-Pixelmuster in eine entsprechende Pixel-Column wandeln, wobei jedes 1-Bit des ursprünglichen Musters mit der angegebenen Farbe in der Pixel-Column angelegt wird.

Textausgabe

displayText(x, y, mode, text-pointer, memory-type, start-column, length)

Texte können mit „displayText“ angezeigt werden. Nach den x/y-Koordinaten und dem Verknüpfungsmodus (siehe Grafikblöcke) gibt man einen Zeiger auf den Text an und sagt auf welche Speicherart (RAM, FLASH oder EEPROM) sich der Zeiger bezieht. Der Text besteht aus den Zeichencodes und muss mit einer Null abgeschlossen werden (null-terminierter String). Zum Schluss legt man noch fest, welcher Ausschnitt des Strings angezeigt werden soll (Start-Position und Breite des Textausschnitts in Pixeln).

Zeichensätze

Zeichensatz „Special“

Es stehen mehrere Zeichensätze zur Verfügung.

- ascii_de: ASCII-Zeichensatz, deutsche Umlaute und einige Sonderzeichen
- bold: wie ascii_de aber in Fettschrift
- square: ASCII-Zeichensatz, deutsche Umlaute und einige Sonderzeichen, eckiges Aussehen
- iso8859: internationaler europäischer Zeichensatz nach ISO 8859-15
- special: Sonderzeichen und Symbole

In der Datei „fonts.h“ können durch Auskommentieren nicht benötigte Zeichensätze entfernt oder neue eigene Zeichensätze hinzugefügt werden. Es sind maximal 7 Zeichensätze möglich.

Steuerzeichen

Im Text können mit Steuerzeichen die Farbe und der Zeichensatz geändert werden.

- Mit den Codes 1, 2, 3, ... wird der erste, zweite, dritte usw. Zeichensatz ausgewählt. Die Reihenfolge entspricht der Reihenfolge in „fonts.h“.
- Code 16 schaltet in den Invers-Modus und wieder zurück.
- Die Codes 17 - 32 wechseln die Farbe. Bit0..1 bestimmen dabei den Grünanteil, Bit2..3 den Rotanteil.

Konfiguration

In der Datei „dot_matrix.h“ werden verschiedene Standardwerte für die Textausgabe gesetzt.

```
// some character font defaults
#define DEFAULT_FONT      font_ascii_de
#define DEFAULT_CHAR_BASE CHAR_BASE_ASCII_DE    // character base of default
font
#define DEFAULT_FONT_SIZE ( sizeof (DEFAULT_FONT) / sizeof (DEFAULT_FONT[0])
)
#define DEFAULT_COLOR     ORANGE
```

Laufschrift

Die Klasse „Ticker“ erledigt die Darstellung von Laufschriften.

Textpuffer

Für den Ticker muss ein Textpuffer angelegt werden. In ihm wird der Lauftext als null-terminierter String gespeichert. Daher muss seine Größe um eins größer sein als die maximale Anzahl an Zeichen.

```
// global variables
DotMatrix   dm;               // PixBlock
char       ticker[101];           // text buffer for 100 characters +
terminating zero
Ticker      tt(dm, ticker, sizeof(ticker));    // ticker text
```

printChar(character)

Hiermit wird ein einzelnes Zeichen in den Lauftext-Puffer geschrieben. Mit <BS> (Code 8) wird das vorherige Zeichen gelöscht. Die Zeichen <CR> oder <LF> (Codes 13 bzw. 10) löschen den kompletten Textpuffer.

printString(string-pointer)

printString_P(string-pointer)

Mit diesen Methoden wird ein String in den Textpuffer geschrieben. Bei „printString“ liegt der String im RAM, bei „printString_P“ im FLASH-Speicher. In jedem Fall muss der String mit einer abschließenden Null enden.

clearBuffer()

Löscht den kompletten Textpuffer indem es ihn mit Nullen vollschreibt.

Viewport

Üblicherweise wird von dem gesamten Textpuffer nur ein Ausschnitt auf dem PixBlock-Display angezeigt.

setViewport(x, y, width)

Mit „setViewport“ wird deshalb ein Fenster definiert, in dem der aktuelle Ausschnitt des Textpuffers zu sehen ist. Die Koordinaten geben die linke, obere Ecke an. Der Parameter width legt die Breite in Pixeln fest.

clearViewport()

Der Viewport wird gelöscht. Der Inhalt des Textpuffers bleibt hiervon unberührt.

Betrieb

setSpeed(speed, delay)

Mit speed wird die Scroll-Geschwindigkeit festgelegt (größer = schneller). Delay gibt an, wie lange die Pause zwischen zwei Scroll-Durchgängen ist (0 = keine Pause). Beide Parameter haben einen Wertebereich von 0 bis 15.

start()

Startet die Laufschrift.

stop()

Stoppt die Laufschrift.

update()

Die Methode „update“ hält die Laufschrift in Betrieb und sollte regelmäßig aufgerufen werden, z. B. alle 5 Millisekunden. Dies kann in einer Programmschleife passieren oder durch einen Timer-Interrupt getriggert werden.

```
void loop()      // endless loop
{
    wait(5);    // wait 5 ms
    tt.update();    // update ticker text
}
```

PixBlock, Elektronik, Bausatz, Led, Dot-Matrix, Anzeige, Display, Software