PixBlock - Software

The PixBlock software was written in C++ for Atmel AVR controllers. It serves as an Arduino library, but can also be used for other AVR microcontrollers.

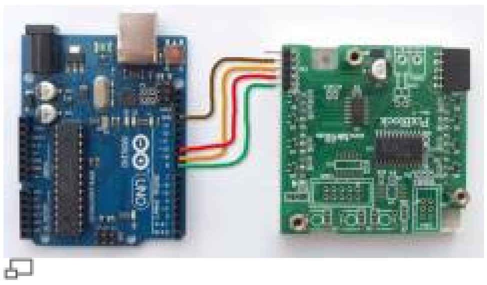
The source code is available on Github. It contains a simple demo program for an Arduino Uno. In case that you use a different non-Arduino development environment, simply copy the files from „libraries→ PixBlock“ to your project directory. The library uses the data types defined in <inttypes.h>. The types „uint8_t“ and „uint16_t“ correspond to the Arduino data types „Byte“ and „Word“.

Configuration

The configuration of the software is done in the header file „dot matrix.h“. Here one must specify how many PixBlocks are arranged in X and Y direction.

```
// 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
```

The PixBlocks form a chain of NUM_BLOCKS_X * NUM_BLOCKS_Y blocks.



Connection to an Arduino board

Which pins are used for control, can also be specified. By default, these are:

< 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
```

Classes

There are two classes, Dot Matrix and Ticker. While Dot Matrix handles the control of the PixBlocks, ticker is used to display scrolling text.

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

Multiplexing

The PixBlock is multiplexed on a column by column basis, i. e. in rapid sequence each of the 8 LED columns is displayed one at a time. Due to the rapid repetition an appearingly still image results. Changing the duty cycle of each LED allows for four brightness levels (100%, 50%, 25%, off) which leads to 16 different colors.

update()

After initialization, all of these tasks are handled by the „update“ method. This method must be called periodically which is best done in an interrupt routine.

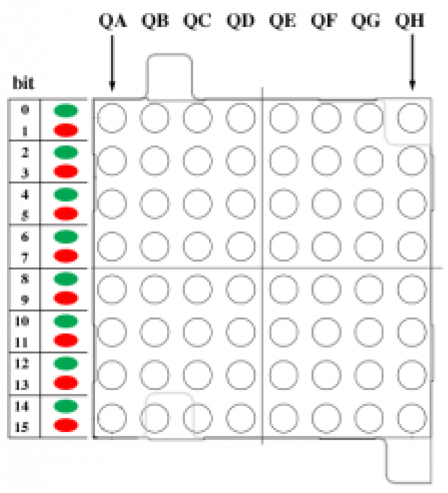
```
// dot matrix system timer
#define DM_REFRESH_FREQ   2500    // dot matrix column refresh rate (Hz)
#define DM_PRESCALER  32    // prescaler division ratio (1, 8, 32 or 64)

#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
}
```

Screen Memory

The image content is stored in the array „screen“ which is composed of pixel columns. For each PixBlock 32 bytes of RAM are required.

Pixel Column



Assignment of bits und leds

A pixel-column is a data structure which represents a column of 8 pixels. Each pixel has a red and a green LED, represented by a 16-bit value. However, since each LED has four brightness levels, we use two 16-bit values, one for the most significant and one for the least significant bits of the led brightness.

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

Hidden Screen

If enough RAM is available, a second invisible screen memory can be provided.

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

selectScreen(VISIBLE / HIDDEN)

swapScreen()

The two memories are identified by the constants VISIBLE and HIDDEN. With the method „Select Screen“ one selects on which screen memory future commands are to operate. „swapScreen“ swaps both memories.

```
DotMatrix   dm;

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

clearScreen()

With „Clear Screen“ the memory of the currently selected screen is erased.

Accessing Pixels

setPixel(x, y, color)

getPixel(x, y)

Individual pixels can be set with „setPixel“ or retrieved with „getPixel“. The origin is in the upper left corner. For some colors corresponding constants are predefined in „dot_matrix.h“.

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

Graphics Blocks

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

„display graphics“ displays a graphics block with a height of 8 pixels. The first two parameters specify the x / y coordinate. After that the drawing mode follows. It can take the following values:

- OPAQUE: The image content is overwritten by the graphics blog.
- TRANSPARENT: Black pixels in the graphics block are considered to be transparent.
- XOR: Image content and graphics block are xor-ed.

The next parameter is a pointer to a series of 16-bit values. Two successive values represent LSB and MSB of a Pixel Column. In addition, one must specify which type of memory (RAM, FLASH or EEPROM) the pointer applies to. The last parameter tells the width of the graphics block.

```
// 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)

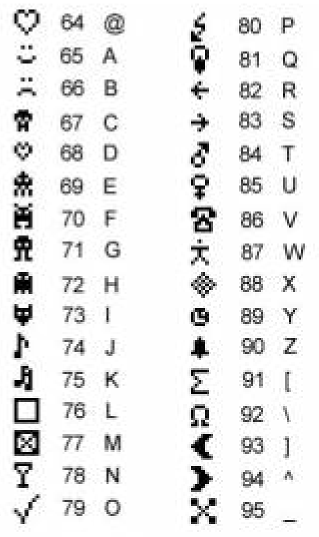
This method transforms an 8-bit pixel pattern into a corresponding pixel column. Each bit that is set in the source pattern is created with the specified color in the pixel column.

Displaying Text

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

Texts can be displayed with „Display Text“. After the x / y coordinates and the drawing mode (see Graphics Blocks) you provide a pointer to the text and specify the type of memory (RAM, FLASH or EEPROM) the pointer applies to. The text consists of the character codes and must be terminated with a zero (null-terminated string). Finally you define which part of the text is displayed.

Character Fonts



64	@	80	P
65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y
74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93	]
78	N	94	^
79	O	95	_

Character font „Special“

There are several fonts available.

- ascii_de: ASCII font, German Umlaute and some special characters
- bold: same as ascii_de but in bold
- square: ASCII font, German Umlaute and some special characters, angular appearance
- iso8859: international European font according to ISO 8859-15
- special: special characters and symbols

In the file „fonts.h“ new custom fonts can be added and unneeded fonts can be removed. There is a maximum of 7 fonts.

Control Characters

Within the text color and font can be changed with control characters.

- Codes 1, 2, 3, …, select the first, second, third, etc. font. The sequence corresponds to the sequence in „fonts.h“.
- Code 16 toggles the inverse mode.
- Codes 17 - 32 change the color. Bits 0..1 determine the amount of green, bits 2..3 the amount of red.

Configuration

The file „dot matrix.h“ defines different default values for the text output.

```
// 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
```

Scrolling Text

Class „Ticker“ does all the mechanics behind scrolling texts.

Text Buffer

First a text buffer must be created for the ticker. Here the scrolling text will be saved as a null-terminated string. Therefore, the buffer size must be greater by one than the maximum number of characters.

```
// 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)

This method puts a single character in the text buffer. With <BS> (Code 8) the previous character will be deleted. <CR> or <LF> (code 13 or 10) delete the entire text buffer.

printString(string-pointer)

printString_P(string-pointer)

These methods write a string to the text buffer. With „print string“ the string is read from RAM while with „printString_P“ it is taken from FLASH memory. In any case the string must terminate with a null character.

clearBuffer()

Clears the text buffer by filling it with zeros.

Viewport

Usually, only a section of the entire text buffer is displayed on the PixBlock screen.

setViewport(x, y, width)

Therefore „setViewport“ defines a viewport in which the current section of the text will appear. The coordinates specify the upper left corner while width defines the width in pixels.

clearViewport()

Clears the viewport. The content of the text buffer is not affected by this method.

Operation

setSpeed(speed, delay)

Parameter speed sets the scrolling speed (larger = faster), parameter delay defines how long the pause between two scroll cycles is (0 = no pause). Both parameters have a value ranging from 0 to 15.

start()

Starts the scrolling.

stop()

Stops the scrolling.

update()

The method „update“ does the actual scrolling and should be called periodically, e. g. every 5 milliseconds. This can be done in a timed program loop or in an interrupt routine which is triggered by a timer.

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

PixBlock, electronics, kit, led, dot-matrix, display, software