BC759x 7seg LED Display Library For Arduino

Index

Overview

BC759x series 7-segment LED display driver + keyboard interface chip provides a unified UART single-line LED display interface, this driver library can be applied to the following chips:

BC7595 -- 48 segments (six 7-segment numeric displays with decimal points) + 48-key keyboard matrix chip

BC7591 -- 256 segments (32 7-segment numeric displays with decimal points) + 96-key keyboard matrix chip

This driver library is compatible with all Arduino devices, and can be used with both hardware serial ports and software serial ports.

Each instruction of the BC759x consists of 2 bytes, the first byte is the instruction and the second byte is the data. The driver library provides a basic function, sendCmd(), which can be used to send arbitrary instructions to the BC759x. At the same time, the driver library provides several upper-layer functions that wrap several of the most commonly used functions in use. The upper-layer functions are as follows.

`clear()` – Clear display content and blink status

`displayDec()` – Display values in decimal

`displayHex()` – Display values in hexadecimal

`displayFloat()` - Display float point values

`digitBlink()` – Digitwise blink control

For those functions that can be done with a single BC759x instruction, although they are frequently used - such as individual LED on/off control - are not implemented as upper-level functions, as this does not simplify their use for the user.

For detailed information about the instructions of the BC759x chip, please refer to the datasheets of the corresponding chip.

Library Installation

The driver library is very easy to install, just search the name of the library or the chip number in the Arduino Library Manager.

Or if you prefer to download the zip file and start from there: In the Arduino IDE menu, select

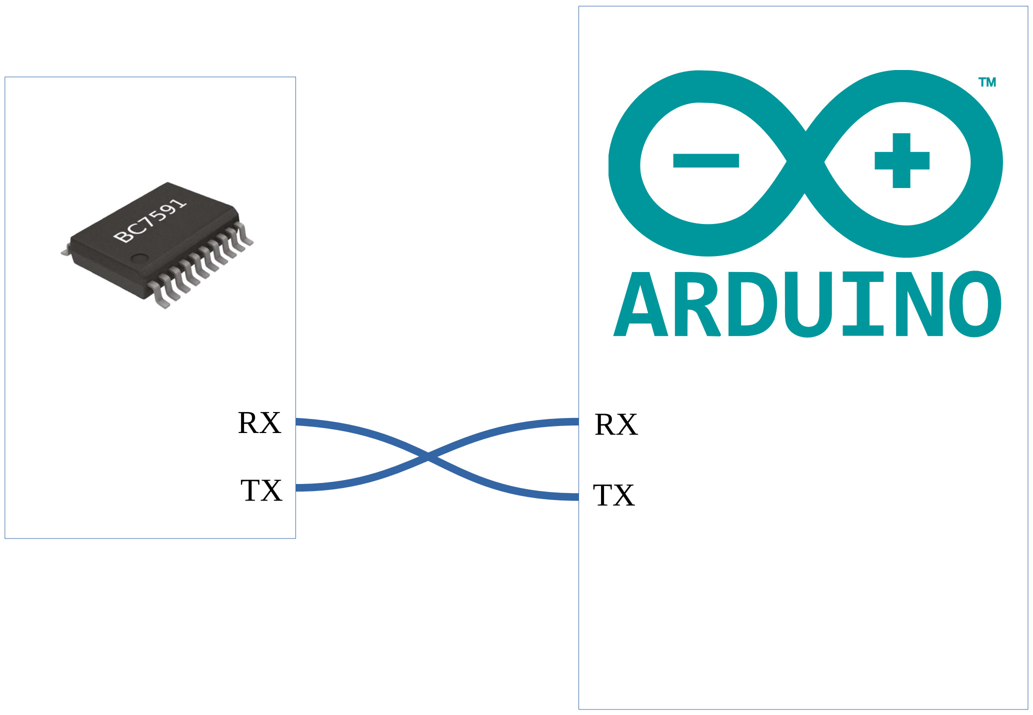
"Sketch --> Include Library --> Add .ZIP Library..."

Select "uart_7seg_display.zip", then it's done.

After installation, you will see the UART_7Seg_Display library in the "Sketch --> Include Library" menu, which means it is ready to use.

Hardware Connection

The BC759x chip uses serial communication, and the communication uses 2 pins TX and RX, where TX on chip is the output of the keyboard interface of the BC759x, and the display driver actually uses the on chip RX pin only. If you use the display function of the BC759x only(without using its keyboard interface), the only pin you need to connect is the RX plus the power supply. As the display and keyboard are usually used together in most cases, both RX and TX will be connected. To connect, the TX/RX pins on the chip are cross-linked to the RX/TX pins of the Arduino's serial port.



Use Of The Library

Create Instance

The interface of the BC759x chip is a serial port, so the use of this driver library must also rely on the Arduino's serial port. The serial port can be either Arduino's hardware serial port (e.g. Serial, Serial1, Serial2, etc.) or software serial port (Software Serial).

For models like the Arduino UNO, there is only one hardware serial port, and that serial port is already used for communication with the computer, so it is appropriate to use the software serial port, otherwise there will be conflicts with the IDE (unless the program is downloaded to the hardware using an external programmer that does not occupy the Arduino's serial port).

Before use, the first step should be to include the library. After loading, the following statement appears in the Sketch editing window.

```
#include <UART_7Seg_Display.h>
```

then, for easy understanding and using, define an instant named `LED_SERIAL`:

```
#define   LED_SERIAL     Serial1
```

Below the statements above, and before setup(), create the instance of the UART_7Seg_Display library, e.g.

```
UART_7Seg_Display        Disp(LED_SERIAL);
```

Where `UART_7Seg_Display` is the name of this driver class, `Disp` is the name given to the instance by the user, it can be anything, as long as it conforms to the Arduino variable naming rules.

If you are using SoftwareSerial, you'll need 2 extra steps. The first step is to include the SoftwareSerial library, which is one of the standard libraries for Arduino.

```
#include <SoftwareSerial.h>
```

The second step requires the creation of an instance of the software serial prior to the creation of an instance of UART_7Seg_Display.

```
SoftwareSerial swSerial(11, 12);
```

Where `swSerial` is the name given to the instance by the user, which can be anything that matches the Arduino variable naming rules. In parentheses the 11, 12 are the RX and TX pins used by the software serial port. See the Arduino software serial port library for more information.

The definition of the LED_SERIAL should also be changed to:

```
#define   LED_SERIAL      swSerial
```

setup()

This library itself does not require any initialization to be used, but the serial port must be properly initialized to connect to the BC759x chip. The serial port must be initialized to baud rate 9600.

`LED_SERIAL.begin(9600);`

Although it can be used without initialization, because different PCB hardware designs may have different PCB layouts, it may be necessary to set the direction of the display in the initialization. If the library is set to display the opposite direction of the actual board, when the value is displayed, the position of the high and low digits will be reversed, for example, when displaying decimal numbers, the ones may be displayed on the leftmost side.

The library has two functions to set the display direction, which are

`setDispLowDigOnLeft()` and `setDispLowDigOnRight()` , respectively, represent the smaller digit number on the board is the 7-segment on the left side or right side. Users can call the corresponding function according to the design of the PCB, for example.

`Disp.setDispLowDigOnLeft();    //` Notify the library of the smaller digit number is on the left

If not set, the default setting is smaller digit number on the right.

In addition, the clear function `clear()` is often used in the initialization section to clear all display content and blink attributes after a reset.

`Disp.clear();`

Display Content

The BC759x chip has internal registers where the display content is written and will remain displayed until it is replaced by new content. Therefore the program only needs to call the library when the display content is to be updated.

The library provides 3 functions related to display.

Display in decimal -- `displayDec()`

Display in hexadecimal -- `displayHex()`

Display float point value -- `displayFloat()`

Blink by digit -- `digitBlink()`

For more information on the use of these three functions, see the function references section later.

Only the three display-related functions above are provided, as other common operations can generally be completed by a BC727x instructions, encapsulated into a function does not simplify the user program. The following are some common scenarios:

Display Special Characters

Some special characters, such as "L", "H", "P", "-", etc., can be implemented by writing directly to the display register, using the BC759x direct register write instruction DIRECT_WT. For example, to display "L":

`sendCmd(DIRECT_WT|Pos, 0x38);` // Where `Pos` is the display position and `0x38` is the character map of "L". For detailed instructions of BC759x, you can refer to the datasheet of BC759x.

Display A Decimal Point

The numeric display function, whether it is a decimal display or a hexadecimal display, will not affect the status of the decimal point. To control the status of the decimal point, you can use the segment addressing instruction SEG_OFF/SEG_ON of the BC759x. For example, to light up the decimal point in the 3rd position:

`sendCmd(SEG_ON, 0x1f);`   // Where `0x1f` is the segment address of the decimal point of the 3rd digit.

Other Special Controls

Any control of the BC759x chip can be done by using the `sendCmd()` function, such as controlling the blinking speed, adjusting the display brightness, etc. For more information about the commands of the BC759x chip, please refer to the datasheet of the BC759x.

Function Reference

sendCmd() : Send Command

Format:

```
sendCmd(Cmd, Data);
```

This function can be used to send any command to the BC759x. The library only provides a few upper-level common display functions. If the user needs to have full control of the BC759x, this function needs to be called. There are two parameters, Cmd is the command to be sent and Data is the data to be sent. In "bc_led_disp.h" header file all the commands for the BC759x are listed. For detailed descriptions of the instructions for specific BC759x chips, please refer to the datasheets of the selected BC759x chip. All other functions provided by the library are implemented by calling this function.

Predefined Commands:

The following commands have already been defined in library and are ready to be used:

• DIRECT_WT	Direct write to display register
• COL_WRITE	Column write (BC7591 only)
• BLINK_WT_CLR	Segment blink clear
• BLINK_WT_SET	Segment blink set
• SHIFT_H_WT	Column insert and shift high(right) (BC7591 only)
• ROTATE_R	Rotate right(high) (BC7591 only)
• ROTATE_L	Rotate left(low) (BC7591 only)
• SHIFT_L_WT	Column insert and shift low(left) (BC7591 only)
• DECODE_WT	Decoded write
• QTR_WT_BOT	Quarter line write at bottom (BC7591 only)
• QTR_INS_BOT	Quarter line insert at bottom (BC7591 only)
• QTR_WT_TOP	Quarter line write at top (BC7591 only)
• WRITE_EXT	Direct write at extended position
• QTR_INS_TOP	Quarter line insert at top (BC7591 only)
• DECODE_EXT	Decoded write at extended position
• SEG_OFF	Segment OFF
• COORD_OFF	Coordinate OFF (BC7591 only)
• SEG_ON	Segment ON
• COORD_ON	Coordinate ON (BC7591 only)
• BLINK_DIG_CTL	Digit blink control
• GLOBAL_CTL	Global control
• WRITE_ALL	Write to all
• BLINK_SPEED	Blink speed
• DIM_CTL	Dim control
• RESET_H	Reset (as Cmd)
• RESET_L	Reset (as Data)
• UART_SEND_0_H	UART send 0 (as Cmd)
• UART_SEND_0_L	UART send 0 (as Data)

clear() : Clear display and blink status

Format:

```
clear();
```

This function is used to clear all display and blink properties.

displayDec() : Display Values In Decimal

Format:

```
displayDec(Val, Pos, Width);
```

This function displays the input value in decimal. Only unsigned values are accepted, and the display of negative values requires the user to write the '-' sign manually. If the higher significant digits are out of the display range of the chip, the excess will not be displayed and no error message will be given.

`Val` is the value to be displayed, and the range is 0 to 4,294,967,295.

`Pos` is the position of the display. The display position is based on the lowest digit, `Pos` value is the DIG number of the character where the lowest significant digit is located, the value range is 0-31.

`Width` is the display width, the valid value range is 0-255. In the binary representation, the lower 7 bits are the display width value, and the 7th bit is the control bit of whether to display leading '0's or not. For example, if the `Width` is set to 5 and the `Val` value to be displayed is 132, the display result will be "␣␣132" (␣ stands for blank), if the `Width` value is 133 (5| 0x80, the highest bit in the binary representation of 5 set to '1'), then the result will be "00132". The valid range of the display width is 1-32. When the display width is set smaller than the width of the actual value, the exceeding part will not be displayed, and when the display width `Width` is larger than the width of the actual value, the exceeding part will be displayed blank or '0' according to the bit7 of `Width`. If the display position of the digit is beyond the display range of the actual chip, the excess part will be ignored.

`setDispLowDigOnLeft()` and `setDispLowDigOnRight()` will affect the direction of the digit display. When `setDispLowDigOnRight()` is called, the lowest digit is displayed at the `Pos` position, and the higher digits of the displayed value will be displayed at the higher DIG position in order. And when using `setDispLowDigOnLeft()`, the lowest digit will be displayed at `Pos` position, and the more significant digits will be displayed at positions smaller than `Pos` in order. The library defaults to the state of `setDispLowDigOnRight()`.

displayHex() : Display Values In Hexadecimal

Format:

```
displayHex(Val, Pos, Width);
```

This function displays the input value in hexadecimal. `Pos` and `Width` have the same meaning as in `displayDec()` above, but the difference is that for hexadecimal display, when `Width` is set to exceed the width of the actual value, the excess part will be displayed as 0, instead of the blank in decimal display. For example, if you input 0xA5, set `Width` to 5, then the result will be 000A5

displayFloat() : Display Float Point Values

Format:

```
displayFloat(Val, Pos, Width, Frac);
```

This function displays a float point value. `Pos` and `Width` have the same meaning as in `displayDec()` above, `Frac` is the width of the fraction part. `Width` is the whole display width. The function doesn't deal with the decimal point and negative sign, user must use proper command to turn on/off decimal point and negative sign.

digitBlink() : Digit Blink Control

Format:

```
digitBlink(Digit, OnOff);
```

Control blinking by digit. This function controls the blinking property of one display digit at a time. The blinking property of the 16th - 31st digit, if it's set directly by the user through the sendCmd() function, may be lost if then changed by using this function. The `Digit` value range is 0-31; `OnOff` is the set state, 1=blinking, 0=no blinking.

Examples

The following Sketch uses Serial1 as display port, displays counts from 0-999 in both decimal and hexdecimal format and a float point number starting form 1.23 increased by 0.01 each step. The display changed every second.

```
#include <UART_7Seg_Display.h>


#define LED_SERIAL Serial1      // By default this code uses Serial1 as LED display port
//  If you are using Arduino with only 1 Serial (such as UNO), you may want to disable
//  the above line and use the following setting:
// #include <SoftwareSerial.h>
// #define LED_SERIAL    swSerial
// SoftwareSerial swSerial(11, 12);       // creating SoftwareSerial instance, using pin 11 as Rx, 12 as Tx (Rx not used in this example)


UART_7Seg_Display Disp(LED_SERIAL);    // creating display driver instance



uint16_t cnt=0;
float x=1.23;

void setup()
{
     Serial.begin(115200);      // Initializing printing serial port
    LED_SERIAL.begin(9600);    // Initializing LED display serial port
    Disp.clear();              // Clear any display contents
}

void loop()
{
    Serial.print("Disaplying in decimal : ");
    Serial.println(cnt, DEC);
    Disp.displayDec(cnt, 0, 3);  // Display cnt as decimal number on DIG0-DIG2 (3 digits wide)

    delay(1000);

    Serial.print("Disaplying in hexdecimal : ");

    Serial.println(cnt, HEX);

    Disp.displayHex(cnt, 0, 4);  // Display cnt as hexdecimal on DIG0-DIG3 (4 digits wide)

    delay(1000);

    Serial.print("Disaplying in float : ");

    Serial.println(x,2);

    Disp.displayFloat(x, 0, 3, 2);  // Display x as float number on DIG0-DIG2, 2 digits after decimal point

    Disp.sendCmd(SEG_ON, 0x17);  // Turn on the decimal point on DIG2 (segment address 0x17, see datasheets)

    delay(1000);

    Disp.sendCmd(SEG_OFF, 0x17); // Turn off the decimal point on DIG2

    cnt = cnt + 1;

    if (cnt == 1000)

    {

        cnt = 0;

    }

    x = x+0.01;

}
```