Morse Code Tutor -
from the ground up

Part 4: Add a display

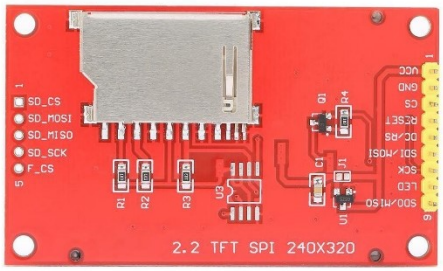
Bruce E. Hall, W8BH

This is part 4 of a series about an inexpensive device that helps you learn Morse Code. It is inspired by Jack Purdum's "Morse Code Tutor", using a Blue Pill microcontroller board and the Arduino IDE.

Now we have all the tools needed to send and receive code, but we don't have any way for the user to check his work. What good is listening to code if the user can't check accuracy? In this part we will add an inexpensive 2.2" TFT LCD display to show the characters that were sent.

The 2.2" display.

This is a 320x240 pixel TFT display on a carrier board, using the ILI9341 driver, and an SPI interface. It is a 3.3V device. Search eBay and Google for "2.2 ILI9341" and you will find many vendors. The current price for the red Chinese no-brands, shown at right, is $6-7 depending on shipping. For a tried-and-true US version, check out the Adafruit #1480 at $25. As a matter of principle, I use the Adafruit software libraries so I support them by purchasing their hardware – at least one of each type that I use. After that I happily try and buy the cheaper alternatives. For this tutorial I will stick to the no-brand Chinese redboards.



My redboard display has 9 pins, already attached to headers, for the LCD and an additional row of 5 holes without headers for the SD card socket. Our project will use the 9 pins with headers.

There are 5 pins on the display that connect to pins on the Blue Pill, and 3 pins that are power/ground related. The following table details the connections:

Display Pin	Label	Connects To:	Function
1	Vcc	Vcc bus (3.3V)	Power
2	Gnd	Gnd bus	Ground
3	CS	Blue Pill, pin PA1	Chip Select
4	RST	Vcc bus (3.3V)	Display Reset
5	DC	Blue Pill, pin PA0	Data/Cmd Line
6	MOSI	Blue Pill, pin PA7	SPI Data
7	SCK	Blue Pill, pin PA5	SPI Clock
8	LED	Vcc bus (3.3V)	LED Backlight Power
9	MISO	(no connection)	SPI Data

Connect the wires and apply power. Make sure the backlight is ON – if not, immediately disconnect and check your wiring. The most common failure at this point is improper wiring.

The code requires a few modifications to accommodate the display. First, we must include the excellent Adafruit graphics and ILI9341 libraries:

```
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"
```

The CS and DC lines must be defined, and a display object created.

```
#define TFT_DC            PA0
#define TFT_CS            PA1

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
```

Some display colors should be defined:

```
#define BLACK          0x0000
#define BLUE           0x001F
#define RED            0xF800
#define GREEN          0x07E0
#define CYAN           0x07FF
#define MAGENTA        0xF81F
#define YELLOW         0xFFE0
#define WHITE          0xFFFF
```

And finally, the display object must be initialized in your setup() routine:

```
void setup() {
  pinMode(LED,OUTPUT);
  pinMode(PADDLE_A, INPUT_PULLUP);                // two paddle inputs, both active low
  pinMode(PADDLE_B, INPUT_PULLUP);
  tft.begin();                                    // initialize screen object
  tft.setRotation(3);                             // landscape mode
  tft.fillScreen(BLACK);                          // start with blank screen
  tft.setTextSize(2);                             // small text but readable
  tft.setTextColor(TEXTCOLOR,BLACK);
  tft.print("Hello from W8BH");                   // for testing only!
}
```

It's time to test the display! Make sure your loop() routine is empty, and upload the code. If all is working correctly, the screen should darken and you should see the test line appear at the top of your

display. If it doesn't, check your wiring connections. It is easy to get the display pins mis-wired, especially since they are labelled on the bottom of the board. You can also try a simple sketch that I wrote which only tests the display.

With a working LCD we can display characters on the screen. The process for displaying on the screen is to specify the location on the screen using tft.setCursor(x,y) and then display the character with tft.print(char). In landscape mode, the horizontal "x" value range is (0,319) and the vertical "y" range is (0,239).

Putting text on the screen.

When dealing with text, it is more convenient to refer to their position in terms of their (col,row) positions rather than the screen's (x,y) coordinates. The number of available rows, MAXROW, is the height of the display divided by the height of each; and the number of available columns, MAXCOL is the width of the display divided by the width of each column. Variable textRow and textCol keep track of the current character position on the display:

```
#define DISPLAYWIDTH      320                    // Number of LCD pixels in long-axis
#define DISPLAYHEIGHT     240                    // Number of LCD pixels in short-axis
#define ROWSPACING         25                    // Height in pixels for each text row
#define COLSPACING         12                    // Width in pixels for each text char
#define MAXCOL   DISPLAYWIDTH/COLSPACING         // Number of characters per row
#define MAXROW   DISPLAYHEIGHT/ROWSPACING        // Number of text-rows per screen

int textRow=0, textCol=0;
```

To place a character on the screen in (col,row) coordinates:

```
void showCharacter(char c, int row, int col)      // display a character at given row &
column
{
  int x = col * COLSPACING;                       // convert column to x coordinate
  int y = row * ROWSPACING;                       // convert row to y coordinate
  tft.setCursor(x,y);                             // position character on screen
  tft.print(c);                                   // and display it
}
```

Now we need a routine to add a character to the display, and increment the textCol/textRow cursor position as we go:

```
void addCharacter(char c)
{
  showCharacter(c,textRow,textCol);               // display char @current col/row posn
  textCol++;                                      // go to next position on current row
  if ((textCol>=MAXCOL) ||                        // are we at end of the row?
     ((c==' ') && (textCol>MAXCOL-7)))            // or at a wordspace near end of row?
  {
    textRow++; textCol=0;                         // yes,advance to beginning of next row
    if (textRow >= MAXROW)                        // but have we run out of rows?
    {
      clearScreen();                              // start at top again
    }
  }
}
```

Now there is enough code to let us add text to the screen, one character at a time, and have to the characters appear left to right, top to bottom, without having to specify the exact location each time. What would the following code do? Try the following.

```
void showBunchOfCharacters()
{
  for (int i=0; i<200; i++)
    addCharacter(randomLetter());
}
```

Integrating display text with Morse.

In part 2 we created a routine called sendCharacter(), which sends Morse code for a given character. All of our sending routines use this for their Morse output. To display the character on the LCD, all we need to do is add a call to our new addCharacter() routine. Nothing more is needed!

```
void sendCharacter(char c) {                  // send a single ASCII character in Morse
  addCharacter(c);                            // display character on LCD
  if (c==32) wordSpace();                     // a space between words
  else sendElements(morse[c-33]);             // send the character
}
```

Part 4 Summary.

We can now send Morse Code and see the corresponding text. In Part 5 we will add a rotary encoder, allowing the user to move between different practice routines.

See my github account for the source code.

73, Bruce.