Morse Code Tutor -
from the ground up

Part 6: The Menu System

Bruce E. Hall, W8BH

This is part 6 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.

Our project has come a long way, and can send Morse several different ways. It is time to present those choices using a menu system, and allow the user to select a choice using the rotary encoder.

Menu system basics.

I have a small confession: programming a menu system, the stuff of full-blown operating systems, sounds a bit overwhelming and out of my league. I put it off as long as I could. But Jack did it, and that gave me enough confidence to try it myself. To my amazement, it wasn't hard at all! By making a few assumptions we can accomplish it with only a handful of routines.

The first assumption is that there is only one menu on the screen, and that it can use the entire screen if necessary. The second assumption is that the menu fonts and colors are fixed. The menu contents are fixed and stored in arrays of strings. The overall menu structure will be a horizontal menu on the top that is always visible, and vertical drop-down submenus that are only visible when in use.

With these assumptions we define multiple constants:

```
// ================================  Menu Constants ===================================
#define DISPLAYWIDTH      320                     // Number of LCD pixels in long-axis
#define DISPLAYHEIGHT     240                     // Number of LCD pixels in short-axis
#define TOPDEADSPACE       30                     // All submenus appear below top line
#define MENUSPACING       100                     // Width in pixels for each menu column
#define ROWSPACING         25                     // Height in pixels for each text row
#define COLSPACING         12                     // Width of each text character
#define MAXCOL    DISPLAYWIDTH/COLSPACING         // Number of characters per row
#define MAXROW   (DISPLAYHEIGHT-TOPDEADSPACE)/ROWSPACING  // Number of text rows on screen
#define FG              GREEN                     // Menu foreground color
#define BG              BLACK                     // Menu background color
#define SELECTFG         BLUE                     // Selected Menu foreground color
#define SELECTBG        WHITE                     // Selected Menu background color
#define TEXTCOLOR      YELLOW                     // Default non-menu text color
```

Some of these constants are not new – we used them in Part 3 when we added the display. Others require a bit of explanation. Since we want to keep the main menu visible at all times, we will make the top of the screen a "keep out" area for text characters. This is defined by TOPDEADSPACE. Notice how

the number of available text rows is decreased by this new keep-out area. The bottom five constants all deal with color: the colors of unselected menus, selected menus, and the character text.

The first thing to consider is how to display a single menu entry on the screen, remembering that it could be in selected or unselected color. Let's write a small helper function for it:

```
void showMenuItem(char *item, int x, int y, int fgColor, int bgColor)
{
  tft.setCursor(x,y);
  tft.setTextColor(fgColor, bgColor);
  tft.print(item);
}
```

The main menu.

Displaying the top menu is just a matter of placing the contents of the menu, stored in menu[], on the top line of the display. Notice that (i * MENUSPACING) is used as the x coordinate and the y coordinate is 0:

```
for (int i = 0; i < itemCount; i++)             // for each item in menu
  showMenuItem(menu[i],i*MENUSPACING,0,FG,BG);  // display it
```

All of the menus are first shown in the nonselected colors (FG/BG). Now, highlight the indexed (selected) menu by displaying it with the 'select' colors:

```
showMenuItem(menu[index],index*MENUSPACING,0,SELECTFG,SELECTBG);
```

Use the encoder to choose which menu is selected. Three (3) things need to happen: deselecting the previous choice, updating the selection index, and selecting the new choice:

```
      showMenuItem(menu[index],index*MENUSPACING,
        0, FG,BG);                                  // deselect current item

      index += dir;                                 // go to next/prev item
      if (index > itemCount-1) index=0;             // dont go beyond last item
      if (index < 0) index = itemCount-1;           // dont go before first item

      showMenuItem(menu[index],index*MENUSPACING,
           0, SELECTFG,SELECTBG);                   // select new item
```

Finally, if the user pressed the button, return the users selection:

```
while (!button_pressed)                         // loop for user input:
{
  int dir = readEncoder();                      // check encoder
  if (dir) {                                    // did it move?
    showMenuItem(menu[index],index*MENUSPACING,
         0, FG,BG);                             // deselect current item
    index += dir;                               // go to next/prev item
    if (index > itemCount-1) index=0;           // dont go beyond last item
    if (index < 0) index = itemCount-1;         // dont go before first item
    showMenuItem(menu[index],index*MENUSPACING,
         0, SELECTFG,SELECTBG);                 // select new item
  }
}
return index;
```

The dropdown menus.

The routine for the dropdown submenus is almost identical. The only change required is to position the entries vertically instead of horizontally. Here is the entire routine, highlighting the small changes:

```
int subMenu(char *menu[], int itemCount)          // Display drop-down menu & return sel.
{
  int index=0, x,y;
  button_pressed = false;                         // reset button flag

  x = menuCol * MENUSPACING;                      // x-coordinate of this menu
  for (int i = 0; i < itemCount; i++)             // for all items in the menu...
  {
     y = TOPDEADSPACE + i*ROWSPACING;             // calculate y coordinate
     showMenuItem(menu[i],x,y,FG,BG);             // and show the item.
  }
  showMenuItem(menu[index],x,TOPDEADSPACE,        // highlight first item
    SELECTFG,SELECTBG);

  while (!button_pressed)                         // exit on button press
  {
    int dir = readEncoder();                      // check for encoder movement
    if (dir)                                      // it moved!
    {
      y = TOPDEADSPACE + index*ROWSPACING;        // calc y-coord of current item
      showMenuItem(menu[index],x,y,FG,BG);        // deselect current item
      index += dir;                               // go to next/prev item
      if (index > itemCount-1) index=0;           // dont go past last item
      if (index < 0) index = itemCount-1;         // dont go before first item
       y = TOPDEADSPACE + index*ROWSPACING;       // calc y-coord of new item
      showMenuItem(menu[index],x,y,
            SELECTFG,SELECTBG);                    // select new item
    }
  }
  return index;
}
```

With both routines in place, getting the user selection is just a matter of getting the main menu selection, then getting the vertical submenu selection. For convenience, I combine the two results into a single number, where the tens-digit is equal to the main menu selection and the ones-digit is the submenu selection.

```
int getMenuSelection()                            // Display menu system & get user
selection
{
  int item;
  eraseMenus();                                   // start with fresh screen
  menuCol = topMenu(mainMenu,ELEMENTS(mainMenu)); // show horiz menu & get user choice
  switch (menuCol)                                // now show menu that user selected:
  {
   case 0:
     item = subMenu(menu0,ELEMENTS(menu0));       // "receive" dropdown menu
     break;
   case 1:
     item = subMenu(menu1,ELEMENTS(menu1));       // "send" dropdown menu
     break;
   case 2:
     item = subMenu(menu2,ELEMENTS(menu2));       // "config" dropdown menu
  }
  return (menuCol*10 + item);                     // return user's selection
}
```

For example, if the user chooses the fourth item (item=3) from the config menu (menuCol=2), the returned result is (2*10)+3 = selection #23.

Putting it all together.

The menu system requires arrays of strings for each menu. Here are some menus:

```
//================================  Menu Variables ==================================
int  menuCol=0, textRow=0, textCol=0;
char *mainMenu[] = {" Receive ", "  Send   ", "  Config "};
char *menu0[]    = {" Letters ", " Numbers ", " Punc    ", " Words   ", " Let-Num ",
                    "Call Sign", " QSO     ", " Exit    "};
char *menu1[]    = {" Practice", " CopyCat ", "Flashcard", " Exit    "};
char *menu2[]    = {" Speed   ", "CharSpeed", " Chk Spd ", " Tone    ", " Dit Pad ",
                    " Defaults", " Exit    "};
```

Now, in the main program loop(), get the menu selection from the user and call the corresponding routine. Any selection that has not been programmed will just be ignored.

```
void loop()
{
  int selection = getMenuSelection();             // get menu selection from user
  eraseMenus(); textRow=0; textCol=0;             // clear screen below menu
  button_pressed = false;                         // reset flag for new presses
  randomSeed(millis());                           // randomize!
  switch(selection)                               // do action requested by user
  {
    case 00: sendLetters(); break;
    case 01: sendNumbers(); break;
    case 03: sendCommonWords(); break;
    case 04: sendMixedChars(); break;
    case 05: sendCallsigns(); break;
    default: ;
  }
```

Part 6 Summary.

Pretty neat, huh? We have a fully functioning device. We display a menu, let the user to choose the Morse practice they want, then run the code associated with that choice. Furthermore, we have the tools needed to create our own menus and create new types of practice. The rest is, as they say, just a "simple matter of programming". Part 7 will flesh out some of the remaining bits and pieces, including how to save user preferences.

Source code is available on my github account.

73, Bruce.