Phi -menu Software Documentation

Last reviewed on 3/4/2011
John Liu

1. Introduction 2
2. Program details 3
   I. Phi_menu_XXX.pde 3
      - Credits 3
      - Includes 4
      - Core variable definitions and object instantiations 4
      - setup() function 4
      - loop() function 5
   II. FUNCTIONS.pde 6
      - Menu PROGMEM definitions 6
      - User PROGMEM definitions 7
      - User variable definitions and object instantiations 7
      - Utility functions 7
      - The menu 7
      - Menu items 7
   III. ALIUDRuser_interface.pde 9
      - Rendering functions 9
      - Interaction functions 10
   IV. buttons.cpp and buttons.h 11
   V. defs.h 11
      - Shield definitions 11
   VI. PROGMEM.h 12
      - LCD custom character PROGMEM definitions 12
      - Other common keywords PROGMEM definitions 13
3. An example 13
4. Future improvements 16
5. The legal stuff 16

1. Introduction

The phi-menu is an LCD keypad interactive software. It provides a light-weight template for programming the multi-function Phi shields. It provides customizable LCD menus together with functions using the LCD and keypad to collect user inputs such as strings, numbers, Y/N, month, day of the week, time, even Morse code. You can easily develop your own interactive project on Arduino and Phi shields without having to deal with the hassle to either hodgepodge codes from various online authors or write the codes yourself the hard way. From the very start, your project looks like a million dollars.

The phi-menu contains a number of program codes. They are all related in their simple hierarchy. To help understand these codes and use them, here is a diagram:

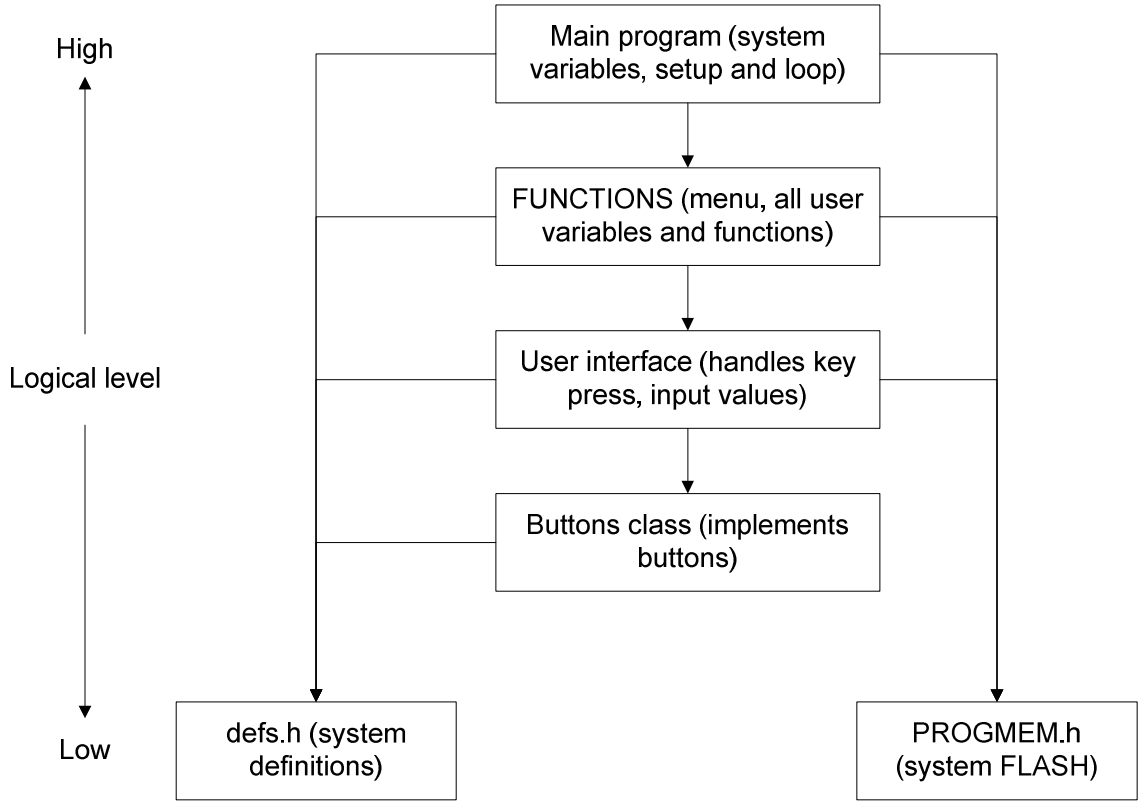

Fig. 1 Structure of the phi-menu

- The main program contains global variables and objects, setup() and loop().
- The "FUNCTIONS.pde" contains all user definable contents, including variables, menu texts, the menu handler do_ menu, and menu functions that are called if the corresponding menu item is selected. See Fig. 2 for details.
- The "ALIUDRuser_interface.pde" contains functions that handle user inputs such as numbers, strings, month, dates, and support functions to display the menu.
- The buttons.h and buttons.cpp define buttons class. You can use them on actual buttons or button-like digital sensors such as Hall Effect switch and photogate.
- The defs.h contains important definitions in order to run all codes, such as which pins are connected to the LCD and which pins are connected to which button.

- The progmem.h contains system static string messages used on the LCD, such as months, days of the week. Please define your own in the FUNCTIONS.pde.

In order to isolate system core and user code, please write all your code and definitions in the FUNCTIONS.pde. Please DONNOT change any other files except for the main pde, where you can add contents to setup. Here are the things to do in order to complete your code:

- ✓ Define some global variables in FUNCTIONS.pde for your parameters
- ✓ Type in your menu texts such as "Display speed" in FUNCTIONS.pde
- ✓ Code each menu functions in FUNCTIONS.pde
- ✓ Modify do_menu() in FUNCTIONS.pde to add or reduce menu items

At the end of this document, in section 3, an example is presented. You can digest the example and go back to section 2 for details of each program. If you prefer, you can try to understand everything in section 2 and examine the example code later. The last section is legal stuff.

2. Program details

The following are details of each program file:

I. Phi_menu_XXX.pde

This is the main program with setup() and loop(). Its sections are described below:

• Credits

This part contains the project description, my contact info, a summary, and a list of functions. Please don't remove this part. If you like this code, pass it on, with its summary and credits. Think what you would do if others pass your code around with missing pieces.

```
/*
Phi-1 shield and phi-menu for Arduino
Test program: Phi-menu car backing/reverse obstacle sensor with sonic ranger v1
This program is fairly long. For a nutshell version of this program to study with, visit
http://liudr.wordpress.com/phi-1-shield/

Programmed by Dr. John Liu
Revision: 03/04/2011
Commercial use without authorization is prohibited.
Find details of the Phi-1 shield, Phi-2 shield or contact Dr. Liu at http://liudr.wordpress.com/phi-1-shield/
All rights reserved.

Summary:
```

This software alerts a driver the distance to an obstacle behind the car for parking or backing safety with both audio and LCD.

Distance sensing: 0m to 6m

List of functions:
** Menu gives you several choices:*
** Display distance is the main function. Any key escapes to main menu.*
** Parameters menu allows you to adjust parameters on frequency of audio beeps and turn audio on and off.*

**/*

- ## Includes

This part contains all the necessary includes. Please don't remove these commands. You may add your include in the end.

```
#include <LiquidCrystal.h>
#include <WProgram.h>
#include <Wire.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include "defs.h"
#include "progmem.h"
#include "buttons.h"
```

- ## Core variable definitions and object instantiations

This part contains all global variables needed to run the programs and all instantiations of objects such as lcd and all buttons. Unless you want to remove certain buttons you should leave this part alone.
User modification: you should not modify this.

```
byte menu_pointer=0;

LiquidCrystal lcd(LCD_RS, LCD_EN,LCD_D4,LCD_D5,LCD_D6,LCD_D7); // Create the lcd object

//These are initializations of the buttons
buttons btn_1(btn_u, LOW);
buttons btn_2(btn_d, LOW);
buttons btn_3(btn_l, LOW);
buttons btn_4(btn_r, LOW);
buttons btn_5(btn_b, LOW);
buttons btn_6(btn_a, LOW);
```

- ## setup() function

The first few lines set up the LCD and loads custom characters. Next, buzzer and LED pins are set up. Unless you want to use the buzzer and LED pins other than driving the buzzer and LED, you should not change this part.

User modification: you should not modify this.

Next part has credit. You're welcome to change this part. Please do your necessary set up after the credits so the code stays well-organized.

```
void setup()
{
  byte ch_buffer[10]; // This buffer is required for custom characters on the LCD.
  lcd.begin(16, 2);

//Programming custom characters. The custom characters will be lost when power is cut.
#ifdef prog_char_now
  for (int i=0;i<8;i++)
  {
    strcpy_P((char*)ch_buffer,(char*)pgm_read_word(&(ch_item[i])));
    lcd.createChar(i, ch_buffer);
  }
#endif

// Set the two channels to output to drive the buzzer and LED.
  pinMode(buzzer,OUTPUT);
  digitalWrite(buzzer,LOW);
  pinMode(led,OUTPUT);
  digitalWrite(led,LOW);

// Display credits
  lcd.clear();
  lcd.print("Sonic ranger V4");
  lcd.setCursor(0,1);
  lcd.print("Dr.Liu 02/28/11");
  wait_on_escape(2000);
  lcd.clear();
  lcd.print("http://liudr.");
  lcd.setCursor(0,1);
  lcd.print("wordpress.com");
  wait_on_escape(2000);
  lcd.clear();

}
```

- **loop() function**

This function is very simple. It only has one line, to run the menu:

```
void loop()
{
  do_menu();
}
```

Since do_menu() doesn't trap execution, the loop() is running the menu non-stop. If you add anything in the loop, it gets executed with the menu. If that something you add traps execution, say waiting for serial communication, the menu will not be responsive.

II. FUNCTIONS.pde

This file contains all your contents, including variables, objects, PROGMEM, #define commands, menu texts, the menu, and all menu functions. Here is a diagram depicting the flow of the code in the file:

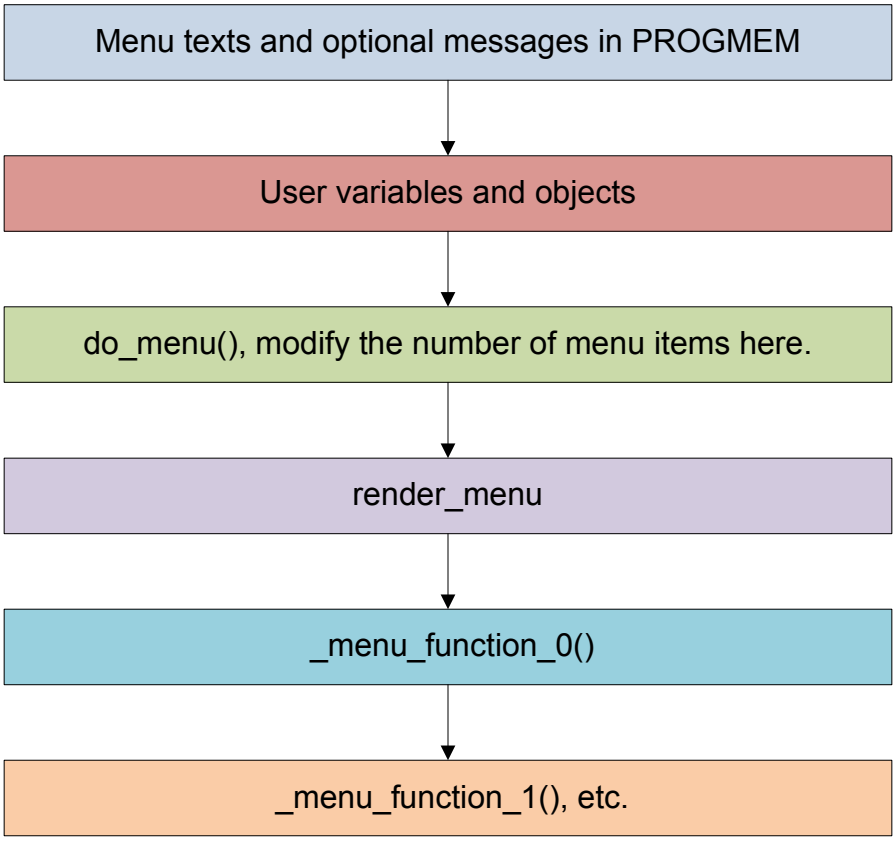

Fig. 2 Structure of the FUNCTIONS.pde

The details of each part is as the following:

- **Menu PROGMEM definitions**

These are defined by you. You write your LCD menu items here
Example:
This program has two menu items, display distance and parameters. You define menu_itemxx[] and put the menu texts in the arrays. Then you list all the menu_itemxx in the menu_item array so the items will display in the menu.

```
PROGMEM prog_char menu_item00[]="Parameters";
PROGMEM prog_char menu_item01[]="Display distance";
PROGMEM const char *menu_item[] = {menu_item00, menu_item01};
```

- **User PROGMEM definitions**

These are optional definitions. You can store all your LCD messages here to save lots of room in SRAM.

- **User variable definitions and object instantiations**

You define your variables and instantiate your objects here in the FUNCTIONS.pde. This way you can debug more easily by not mixing your variables with the system core variables so the program stays organized.

- **Utility functions**

It helps display the menu and other items on the LCD.

void render_menu(int me)
Display the LCD menu item according to the argument [me].
User modification: you should not modify this.

- **The menu**

void do_menu()
This program runs the main menu. It gets called repeatedly by loop(). It handles inputs from the keys, updates the menu or executes a certain menu function when it's selected.
User modification: only a part of this function should be modified to handle your menu items (see below). The rest should stay unchanged.

You may change the content inside this switch. _menu_function_x asAdd or remove _menu_function_x to add or remove amount of menu items.

```
switch (temp1)
  {
    case 0:
    _menu_function_0();
    break;

    case 1:
    _menu_function_1();
    break;
  }
```

- **Menu items**

_menu_function_0()
_menu_function_1()
...
Menu item functions go here. Each function is called when its menu item is selected. They are called only once. This works pretty well if you just to do a series of things in a menu item:

```
void _menu_function_0()
{
 int temp1, temp2;

 delay(100);
 lcd.clear(); // Input on_divider
 lcd.print("On divider:");
 temp1=on_divider;
 temp2=hmi_with_update_3(&temp1, 10, 100, 1, 4, 1, 4, render_number_in_place);
 on_divider=(temp2==-1)?on_divider:temp1;

 delay(100);
 lcd.clear(); // Input off_divider
 lcd.print("Off divider:");
 temp1=off_divider;
 temp2=hmi_with_update_3(&temp1, 4, 50, 1, 4, 1, 4, render_number_in_place);
 off_divider=(temp2==-1)?off_divider:temp1;

 delay(100);
 lcd.clear(); // Input audio_alert
 lcd.print("Audio alert?");
 temp1=audio_alert;
 temp2=hmi_with_update_3(&temp1, 0, 1, 1, 1, 1, 3, render_YN_in_place);
 audio_alert=(temp2==-1)?audio_alert:temp1;
}
```

The content of the code was purposely obscured to show you that three things happen in sequence, each with delay of 0.1 seconds. There is no need to repeat these three things, such as setting values of three parameters.

If you want to do something repeatedly, say displaying some analog readings repeatedly, make sure you have a while loop:

```
void _menu_function_1()
{
 While(1)
 {
 //Do display stuff
 }
}
```

You also need a means of quitting these functions so the program returns to the menu. This is done by calling the following function:

```
void _menu_function_1()
```

```
{
 While(1)
 {
 //Do display stuff
 if (wait_on_escape(50)>0) return; // Return to menu if any key is pressed
 }
}
```

The wait_on_escape() is another utility function defined in ALIUDRuser_interface.pde

The function has the same effect as delay() except that it keeps checking the buttons while waiting and returns any buttons pressed or 0 for no buttons pressed during the waiting. See wait_on_escape() under ALIUDRuser_interface.pde for details.

Almost all sensor measurement functions need some sort of delay so replace your delay() with wait_on_escape() and you can control the flow of the control and returns to menu if a specific key is pressed (wait_on_escape(50)=1 for up button) or if any key is pressed (wait_on_escape(50)>0).

III. ALIUDRuser_interface.pde

This file stores all interface functions that support menu and other functions that need access to button status and render messages on the LCD. Please don't alter the content of this file.

- **Rendering functions**

They help display various information on the LCD.

void msg_lcd(char msg_line)*
Displays strings stored in PROGMEM. Provide the string name [msg_line] stored in PROGMEM to be displayed on the LCD's current cursor position.
User modification: you should not modify this.

void render_number_in_place(int number)
Displays a number on the LCD
User modification: you should not modify this.

void render_00number_in_place(int number)
Displays a two-digit number on the LCD
User modification: you should not modify this.

void render_char_in_place(int number)
Displays a character on the LCD
User modification: you should not modify this.

void render_YN_in_place(int number)
Displays “Yes” or “No” on the LCD
User modification: you should not modify this.

void render_xxx()
Displays information on the LCD
User modification: you should not modify this.

- **Interaction functions**

They facilitate button interactions.

int wait_on_escape(int ref_time)
Returns key pressed or 0 if time expires before any key was pressed. Use this function to wait for users to press a key. You may replace delay() with this function to regulate program flow while checking user input at the same time.
User modification: you should not modify this.

0	Up	1	Down	2	Left
3	Right	4	B	5	A

Table 1. wait_on_escape() return values.

int hmi_with_update_3(int current, int lower, int upper, byte inc, byte column, byte row, byte space, void (*update_function)(int))*
Human machine interaction (HMI) with one update function and input value wrap-around capability.
This function prints the initial value first so the caller doesn't need to.
Function traps until the update is finalized by the left, right, enter button or escape button. Value is updated through the *current pointer instead of return value so the return value is used to carry which functional button was last pushed. Also negative input values are allowed including -1.
Returns buttons pushed so the caller can determine what to do:
Go back to the last slot with left (-3)
Go forward to the next slot with right (-4)
Enter(1)
Escape(-1).

*int hmi_with_update_2(int current, int lower, int upper, byte inc, byte column, byte row, byte space, void (*update_function)(int))*
HMI function with one update function and input value wrap-around capability. This version is older. If you want your user to input a value for a parameter, you use this function and it returns the user input. It handles user button pushes and intermediate values and only returns the final value the user decides on.

Function traps until the update is finalized by the confirm button (B) or forfeited by the esc button (A).
Returns updated value (zero or more) when enter is pressed. Returns -1 if escape is pressed.
You need to first display the current value on the lcd and then call this function to solicit for user input. If -1 is returned, you don't change the variable.
User modification: you should not modify this.

int input_panel(char msg, byte length, byte column, byte row)*
Alphanumerical input panel for texts up to 16 characters.
The total length will not exceed length.
The first letter appears at column, row
The function prints the content of the buffer *msg before polling for input
The function returns number of actual characters
The function returns -1 if the input is cancelled.
The function fills the buffer after the last character with \0 only if the buffer is not filled.
The caller is responsible to fill the character beyond the end of the buffer with \0.
User modification: you should not modify this.

IV. buttons.cpp and buttons.h

This is the buttons class. The various status of a button is modeled as the buttons class. Its details are covered elsewhere.
User modification: you should not modify this.

V. defs.h

This contains all the system definitions such as "#define buzzer 16". The main .pde file includes this file by adding #include "defs.h" to the beginning of the program. Please don't modify the content of this file. Your #define goes in the FUNCTIONS.pde

There is only one type of definitions in this version:

- **Shield definitions**

These definitions lay the ground work of what shield function is using which Arduino pins so later versions of Phi shields will only need to change this section without the need to change user-written codes.
User modification: you should not modify this.
The following applies to the Phi-1 shield:

```
//Phi-1 shield buttons and channel pin assignments
#define buzzer 16
```

```
#define ch1 16
#define ch1_analog 2
#define led 17
#define ch2 17
#define ch2_analog 3
#define btn_u 5
#define btn_d 10
#define btn_l 11
#define btn_r 3
#define btn_b 14
#define btn_a 15

//Phi-1 shield LCD pin setting
#define LCD_RS 8
#define LCD_EN 9
#define LCD_D4 7
#define LCD_D5 6
#define LCD_D6 2
#define LCD_D7 4

//Maximal bytes on your Phi-1 shield eeprom, depending on the chip you use. I'm using 24LC256, with
256KiloBits so 32KiloBytes=32768 bytes.
#define EEPROM_size 32768UL

//#define setRTC // Uncomment this to set the clock time.
```

VI. progmem.h

This file contains all the PROGMEM definitions. Using PROGMEM can save significant SRAM. A different article discusses the details of using PROGMEM.

- **LCD custom character PROGMEM definitions**

These are defined by users like you. If you want an hour glass symbol or a "/s" symbol in one character to save space, you can define as many as 8 characters. Their ASCII codes are 0-7. You will need to use lcd.write() to put the zeroth character on the display. A different article discusses in details how to construct custom characters.
User modification: you may modify this.

```
//Programming custom characters. The custom characters will be lost when power is cut.
//In the character definition, you have to use B100000 instead of B0 since you will need to do strcpy_P,
which terminates at 0 or B0.
PROGMEM prog_char lcd_ch0[]={B110,B100,B100,B110,B100000,B1010,B10101,B10101,0};
PROGMEM prog_char lcd_ch1[]={B1110,B11011,B1010,B11011,B1010,B11011,B1010,B1110,0};
PROGMEM prog_char lcd_ch2[]={B10001,B1110,B11110,B11101,B11011,B10111,B100000,B11111,0};
PROGMEM prog_char lcd_ch3[]={B100000,B11101,B11011,B11101,B11110,B1110,B10001,B11111,0};
PROGMEM prog_char lcd_ch4[]={B11101,B11001,B10101,B1101,B100000,B11101,B11101,B11111,0};
PROGMEM prog_char lcd_ch5[]={B10,B100,B1000,B10011,B100,B10,B1,B110,0};
PROGMEM prog_char lcd_ch6[]={B100000,B100000,B100,B1110,B11111,B100,B100,B100000,0};
PROGMEM prog_char lcd_ch7[]={B100000,B100000,B100,B100,B11111,B1110,B100,B100000,0};
```

```
PROGMEM const char *ch_item[] = {lcd_ch0,lcd_ch1,lcd_ch2,lcd_ch3,lcd_ch4,lcd_ch5,lcd_ch6,lcd_ch7};
```

- **Other common keywords PROGMEM definitions**

These are commonly used keywords for RTC and other devices and can be commented out to save space.
User modification: you should not modify this.

```
PROGMEM prog_char dow_00[]="SUN";
PROGMEM prog_char dow_01[]="MON";
PROGMEM prog_char dow_02[]="TUE";
PROGMEM prog_char dow_03[]="WED";
PROGMEM prog_char dow_04[]="THU";
PROGMEM prog_char dow_05[]="FRI";
PROGMEM prog_char dow_06[]="SAT";
PROGMEM prog_char dow_07[]="M-F";
PROGMEM prog_char dow_08[]="Day of the week";
```

3. An example

The following is an example code for a simple sonic ranger for distance measurement and obstacle avoidance when you back your car. It demonstrates how easy it is to construct a project with the phi-menu. This is a featured project on instructables.com

We want to have an audio alert of the distance of the object so the driver doesn't have to look at the LCD all the time. Then we also want the option to adjust how often the buzzer beeps and to turn the audio alert on and off.

We need to modify menu texts in the PROGMEM area after the highlighted comment:

```
// Messages used in program.
PROGMEM prog_char menu_item00[]="Parameters ";
PROGMEM prog_char menu_item01[]="Display distance";
PROGMEM const char *menu_item[] = {menu_item00, menu_item01};
```

Next, we create variables to store our parameters after the highlighted comment line:

```
//These are related to the specific program
boolean audio_alert=true; // This indicates whether audio beeps are on or off.
int on_divider=40; // The larger this value, the shorter each beep is.
int off_divider=10; // The larger this value, the less frequent the beeps occur.
```

Next, we modify the do_menu to make sure there are two menu items:
The grayed areas are not supposed to be changed.

```
void do_menu()
```

```
{
  //Do not change this
  int temp1;
  temp1=hmi_with_update_2(menu_pointer,0,sizeof(menu_item)/sizeof(char*)-1,1,16,1,0,render_menu);
  menu_pointer=(temp1==-1)?menu_pointer:temp1; // In case escape was triggered, value doesn't change.
  switch (temp1)
  {
    case 0:
    _menu_function_0();
    break;

    case 1:
    _menu_function_1();
    break;

    default:
    break;
  }
}
```

Next, we write the “parameters” function, which asks users to change parameters we just defined in the previous step. From the name of the function we know it is the first item in the menu:

```
void _menu_function_0 ()// Parameters
{
  int para_val, btn_pressed;

  lcd.clear(); // Input on_divider
  lcd.print("On divider:");
  para_val =on_divider;
  btn_pressed =hmi_with_update_3(&para_val, 10, 100, 1, 4, 1, 4, render_number_in_place);
  on_divider=( btn_pressed ==-1)?on_divider: para_val; // In case of escape,, value doesn't change.

  delay(100);
  lcd.clear(); // Input off_divider
  lcd.print("Off divider:");
  para_val=off_divider;
  btn_pressed=hmi_with_update_3(&para_val, 4, 50, 1, 4, 1, 4, render_number_in_place);
  off_divider=(btn_pressed==-1)?off_divider:para_val; // In case of escape, value doesn't change.

  delay(100);
  lcd.clear(); // Input audio_alert
  lcd.print("Audio alert?");
  para_val=audio_alert;
  btn_pressed=hmi_with_update_3(&para_val, 0, 1, 1, 1, 1, 3, render_YN_in_place);
  audio_alert=(btn_pressed==-1)?audio_alert:para_val; // In case of escape, value doesn't change.
}
```

The above function has three blocks of code, each one updating one parameter. The sequence of code used in each block is the following:

- Clear LCD
- Print message

- Assign the parameter's current value to para_val
- hmi_with_update_3 asks the user to update the para_val. The function returns button pressed in btn_pressed.
- If btn_pressed is a cancel button (A), the parameter value is not changed. Otherwise it's changed to the new value stored in para_val.

Then we write the code to sense and display distance. This should be called repeatedly inside of another menu function: menu_function_1(). Here is its content:

```
void _menu_function_1() //Displays sonic ranger result
{
 while (!sonic_ranger(ch2))
 {}; // Keeps running sonic ranger until user presses a key to escape
}
```

The sonic_ranger() function is the work horse. Inside the function, it reads distance by calling sonic_ranger_read()and then calls ranger_display_distance() to display it. Neither called functions are included below. They won't help you unless you also want a sonic ranger project. But some contents in the sonic_ranger function are common features you need in your project:

- Parameters that you defined in the first steps are used in this program to affect how the program runs, such as audio_alert affects whether buzzer turns on.
- You also need delay using wait_on_escape. This prevents Arduino from busy working on its measurement and forgetting to check user inputs. The wait_on_escape handles any key pressed while waiting and returns the key pressed. This also gives us a way to terminate the function and return to the menu, in case a key is pressed while waiting.

```
long sonic_ranger_read(int gate_pin) // Reads and returns the time pulse.
void ranger_display_distance(long duration)
int sonic_ranger(int gate_pin) // Reads and displays distance with audio alert. It returns 0 normally and -1
when user presses a key.
{
 long duration, time;
 float mm;
 time=millis();
 if (audio_alert) tone(buzzer,440);
 duration=sonic_ranger_read(gate_pin);
 mm = (float)(duration) / 5.80;
 ranger_display_distance(duration);
 if (wait_on_escape(int(mm/on_divider))>0) // Wait and then turn off buzzer and return to main menu with
key press
 {
  noTone(buzzer);
  return -1;
 }

 noTone(buzzer);// After wait, turn off buzzer

 if (wait_on_escape(int(mm/off_divider))>0) // Delay after buzzer is off and before next iteration.
```

```
  {
    return -1;
  }

  return 0;
}
```

Now compile and upload! You can see more of this project and program on my blog: http://liudr.wordpress.com/2011/02/12/arduino-parking-sensor/

You can download the code and give it a try!

4. Future improvements

I plan to improve the phi-menu in the following areas:

- Turn the codes into class and encapsulate all variables and functions
- Add multiline message stroll display function support to display longer messages on a 16*2 or larger character display.
- Port the code to work on KS0108-compatible 128*64 dot matrix displays
- Add btn_null and update buttons class to define null button. It will do nothing and stays off. This saves one from deleting buttons objects and sensing codes to free the pin that is shared with a button.

5. The legal stuff

Let's keep this short. The software is defined as any part of the complete collection of the code of the phi-menu system. There is no explicit or implicit warrantee that the software will work. You are free to modify and redistribute the software as long as the credit section is kept. You can add your own credit and modify the summary of the functions.

You are not allowed to use the software for commercial purposes, which include but not limited to selling the software or its modified versions for money directly or selling hardware with the software attached for "free". The software is meant to be shared among interested individuals and educational institutions.

If you like to use the software with your products, commercial licenses can be obtained by contacting the author at http://liudr.wordpress.com