Embedded Systems Development: GreenHouse Control System

600085

Assigned Coursework Report

Student IDs: 201603992

Date: 30 October 2019

Due Date: 13 December 2019

1. Software listing

1.1 EEPROM

Header

```
// This is a guard condition so that contents of this file are not included more than once.
#ifndef XC_EEP_H
#define    XC_EEP_H

// DRIVER DESCRIPTION:

// INCLUDES.
#include <xc.h> // include processor files - each processor file is guarded.
#include "utils.h"

// DEFINES.

// PUBLIC VARS.

// PUBLIC METHODS.
char* eep_ReadString(char addr, char strN); // Read a series of 5 chars from EEPROM into a target string.
void eep_WriteString(char addr, char str[]); // Write a series of 5 chars into EEPROM at the address.

// PRIVATE METHODS.
//void eep_WriteChar(char addr, char ch); // Write a char to an address in EEPROM.
//char eep_ReadChar(char addr); // Read a char from an address in EEPROM.

#endif    /* XC_EEP_H */
```

Source

```
#include"eep_driver.h"

void eep_WriteChar(char add, char ch); // Write a char to the eeprom address..
char eep_ReadChar(char add); // Read a char from the eeprom address.

// Storage for up to 2 strings read from eeprom.
char eepStr1[0x06]; // String 6chars long.
char eepStr2[0x06]; // String 6chars long.

char* eep_ReadString(char addr, char strN) {
	char *str;
	if (strN) str = eepStr1; // If strN == 1, store in first array.
	else str = eepStr2; // If strN == 2, store in second array.

	// Set each of the 6 chars in one of the strings.
	str[0x00] = eep_ReadChar(addr);
	str[0x01] = eep_ReadChar(addr + 0x01);
	str[0x02] = eep_ReadChar(addr + 0x02);
	str[0x03] = eep_ReadChar(addr + 0x03);
	str[0x04] = eep_ReadChar(addr + 0x04);
	str[0x05] = eol;

	return str;
}

char eep_ReadChar(char addr) {
	return _EEREG_EEPROM_READ(addr); // Uses native function to read eeprom.
}

void eep_WriteString(char addr, char str[]) {
	// Write 5 characters to eeprom e.g. "HH:MM"
	eep_WriteChar(addr, str[0x00]);
	eep_WriteChar(addr + 0x01, str[0x01]);
	eep_WriteChar(addr + 0x02, str[0x02]);
	eep_WriteChar(addr + 0x03, str[0x03]);
	eep_WriteChar(addr + 0x04, str[0x04]);
}

void eep_WriteChar(char addr, char ch) {
	_EEREG_EEPROM_WRITE(addr, ch); // Use native function to write to eeprom.
}
```

1.2 Key Matrix

Header

```
#include"matrix_driver.h"

uch matrix_Scan(uch row); // Scan port C to detect the button pressed.
uch result;

void matrix_Init() {
	TRISC = 0XF0; //C PORT high 4 bits INPUT,low 4 bits OUTPUT
}

char matrix_GetInput() {
	// Scan each row of the matrix for input.
	if (matrix_Scan(0XF7) == false)
	if (matrix_Scan(0XFB) == false)
	if (matrix_Scan(0XFD) == false)
		if (matrix_Scan(0XFE) == false)
		result = 0xFF; // No input detected.

	switch (result) {
	case 0xe7: return 's'; // Select. //// Bottom right.
	case 0xeb: return 'b'; // Backspace.
	case 0xed: return '.';
	case 0xee: return '0';

	case 0xd7: return '>'; // Down/next.
	case 0xdb: return '3';
	case 0xdd: return '2';
	case 0xde: return '1';

	case 0xb7: return '<'; // Up/previous.
	case 0xbb: return '6';
	case 0xbd: return '5';
	case 0xbe: return '4';

	case 0x77: return 'x'; // Cancel/close/back.
	case 0x7b: return '9';
	case 0x7d: return '8';
	case 0x7e: return '7'; //// Top left.
	}
	return '_'; // Nothing.
}
```

```
uch matrix_Scan(uch row) {
	const uch HALFMASK = 0xF0;

	PORTC = row; //C3 OUTPUT low,the other 3 bits OUTPUT high
	NOP(); //delay
	result = PORTC; //read C PORT
	result &= HALFMASK; //clear low 4 bits

	if (result != HALFMASK) { //judge if high 4 bits all 1(all 1 is no key press)
	result |= (row - HALFMASK); //no,add low 4 bits 0x07 as key scan result
	return true;
	}
	return false;
}
```

Source

```
// This is a guard condition so that the contents of this file are not included more than once.
#ifndef XC_Matrix_H
#define    XC_Matrix_H

// DRIVER DESCRIPTION:
//--Button layout:
//-- 7 8 9 X (where X == cancel/back)
//-- 4 5 6 < (where < == up/previous)
//-- 1 2 3 > (where > == down/next)
//-- 0 . b S (where . == decimal point, b == backspace, and S == select/enter)

// INCLUDES.
#include <xc.h> // include processor files - each processor file is guarded.
#include "utils.h"

// PRIVATE.
// uch result; // Stores key press location from matrix_Scan.

// uch matrix_Scan(uch row); // Get key press location.


// PUBLIC.
void matrix_Init(void); // Initialise ports and pins.
char matrix_GetInput(void); // Scan the 4x4 matrix for user interaction and return input.

#endif
```

1.3 Input Output (IO)

Header

```
// This is a guard condition so that contents of this file are not included more than once.
#ifndef XC_IO_H
#define    XC_IO_H

// DRIVER DESCRIPTION:

// INCLUDES.
#include <xc.h> // include processor files - each processor file is guarded.
#include "utils.h"

// DEFINES.

// PUBLIC VARS.
char *io_Status;

// PUBLIC METHODS.
void io_Init(void); // Initialise component.
void io_TogglePin(uch pinN, char status[]); // Toggle individual pins.
void io_SwitchOff(void); // Switch all pins off.

#endif    /* XC_IO_H */
```

Source

```
#include "io_driver.h"

#define pin0 RB6 // Use pin RE0.
#define pin1 RB7 // Use pin RE2.

void io_Init() {
	// Set pins to off by default.
	io_Status = "Init...";
	pin0 = 0x00; // Off.
	pin1 = 0x00; // Off.
}

void io_TogglePin(uch pinN, char status[]) {
	io_Status = status; // Update status string.

	switch (pinN) { // Toggle selected pin.
	case 0x00: pin0 = !pin0; // boolean NOT to flip.
	break;
	case 0x01: pin1 = !pin1;
	break;
	}
}

void io_SwitchOff() {
	pin0 = 0x00; // Off.
	pin1 = 0x00; // Off.
}
```

2. Application Programme Interface (API) description

2.1 EEP

void eep_WriteChar(char add, char ch);
Writes ch to the given address.

char eep_ReadChar(char add);
Returns a character from a given address in EEPROM memory.

Void eep_WriteString(char addr, char str[]);
Writes the first five characters of the given string at the given memory address.
char* eep_ReadString(char addr, char strN);
Return a string of length 4 with an end of line from the given address in EEPROM memory.

2.2 Matrix

void matrix_Init();
Initializes the key matrix.

char matrix_GetInput();
Gets the character of the currently pressed button, returns '_' if no button is pressed.

uch matrix_Scan();
Checks to see what row is being pressed on the matrix, used by matrix_GetInput to find out the pressed button.

2.3 Input Output (IO)

void io_Init()
Turns off pin1 and pin0 and sets io_status to "Init…"

void io_TogglePin(uch pinN, char status[]);
Sets io_status to status, toggles pin0 or pin1.

void io_SwitchOff()
Turns of pin1 and pin0.

3. Test reports

3.1 EEPROM

Testing of eep_driver, it was unknown if the EEPROM worked as it had not been mentioned by anyone and there was limited information online, this had us creating a new program purefly to test how the EEPROM worked.

ID	Test Description	Steps	Expected	Pass/Fail
1	Reading a character	Set the all of the EEPROM to 0xFA and read from a memory location	0xFA is returned	Pass
2	Reading a character at a specified location	Repeat the steps for test 1. Set one of the memory locations to 0xFD and read from that location	0xFD is returned	Pass
3	Writing a character at a specified location	Write ‘a’ to memory location 0x10, then read the value back out	‘a’ is returned	Pass
4	Writing a string at a specified location	Call eep_WriteString with 0x10 as the address and ‘Test!’ as a char array. Call eep_ReadString with 0x10 as the address and 0 as the strN.	‘Test!’ is returned	Pass

3.2 Key Matrix

7 Input the number 7	8 Input the number 8	9 Input the number 9	X Cancel/back
4 Input the number 4	5 Input the number 5	6 Input the number 6	< Up/previous
1 Input the number 1	2 Input the number 2	3 Input the number 3	> Down/Next
0 Input the number 0	. Input a decimal point	b backspace	S Enter/Select

Figure 1: Button map.

ID	Test Description	Steps	Expected	Pass/Fail
1	Check that '_' is returned when now buttons are pressed	Call matrix_GetInput and don't touch the key matrix	'_' is returned	Pass
2	Check that each button returns the correct value	Call matrix_GetInput and press each button one at a time checking the returned value aligns with Figure 1.	Each button returns its own value and the buttons mirror the layout of Figure 1.	Pass

3.3 Input Output (IO)

ID	Test Description	Steps	Expected	Pass/Fail
1	Test that the pins correctly turn off	Call io_SwitchOff and check that no LEDs are turned on.	All LEDs are turned off	Pass
2	Test that pin 0 turns on correctly	Call io_TogglePin with pinN as 0 and status as “”	This should turn on one of the LEDs	Pass
3	Test that pin 1 turns on correctly	Call io_TogglePin with pinN as 1 and status as “”	This should turn on the other LED	Pass
4	Test that pin 0 turns off correctly	Call io_TogglePin with pinN as 0 and status as “”	This should turn off the first LED that turned on	Pass
6	Test that pin 1 turns off correctly	Call io_TogglePin with pinN as 1 and status as “”	All LEDs should now be off	Pass

4 Evaluation

The EEPROM might not have been the ideal place to store data, when reading the data sheet for the thermistor we later found that it had a scratchpad to store data. We had a lot of problems as the code for reading and writing to the EEPROM was rather large in memory and we ended up having to refactor the code multiple times. However although the EEPROM used up more of our memory it allowed us to make the whole device configurable as we had 256 bytes of storage.

Source control should have been more optimally used, we started with GitHub flow but would never end up merging the code in as after we fixed any bugs in that branch we would also add some more features. This would lead us to use trunk based development with the trunk branch changing every week as new code was piled on top of the old code. Having this many branches lead us to many problems where code would work in an older branch but the code had changed too much since then so it would have been too hard to go back.

The IO driver was written at the very end of the project and meant we had to keep going through all of the other drivers to see what pins were being used as we did not want to have any collisions. It was originally planned that we would just turn on a pin and use a voltmeter to show the voltage change however we were able to use the LEDs underneath the LCD but the device had to be tilted to see them.