Technical Manual

October 9, 2014

Abstract

This document describes how each of the modules (including both software and hardware) work, and how to use them.

Chapter 1

Serial

1.1 Description:

The serial module takes care of all communication (transmit and receive) over the serial UART (rs-232) port.

1.2 How It Works:

The serial module is INTERRUPT DRIVEN. This means that any background code can be running while the module is transmitting and/or receiving data over the serial line.
The module contains two circular buffers: a transmit buffer and a receive buffer. These buffers are NOT accessible by the rest of the program. Rather, the module provides a public function transmit() which takes a string, and places it into the transmit buffer. Anything in this buffer is then transmitted character by character when the transmit ready interrupt fires.
Whenever a character is received over serial it is stored in the received buffer by an interrupt. Again this buffer is NOT accessible to the rest of the program. Rather it provides a number of functions to interact with it. The most commonly used of which is the readString() function, which returns everything in the buffer up to a carriage return (e.g. a line of input entered by the user).
This serial module also allows users the opportunity to remove or change data they have already transmitted. If a backspace is received, then instead of storing it in the receive buffer it will remove the last received character from the buffer if that character is not a Carriage Return, Newline, or Escape operator. This enables a much more more user friendly system as otherwise there would be no way to fix any syntax error without pressing enter, getting an error and starting again. Furthermore, without this feature, if a user did backspace and change an input it would result in completely unexpected behaviour.

1.3 How To Use It:

- Make sure Interrupts.c is included in the project
- Call the configureSerial() function to set up serial
- Call transmit() to begin transmitting something
- Use receiveEmpty() to check if anything has been received
- Use readString() to read everything received up to a carriage return

1.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

1.4.1 configureSerial

Usage:

Used to set up the serial module for use. This function MUST be called before any serial functions will work. It need only be called ONCE at the beginning of the program.

1.4.2 transmit

Usage:

Used to transmit a string over serial UART. NOTE: The string MUST be NULL TERMINATED. The function will continue pushing data onto the transmit buffer until it hits a null terminator, which could fill the buffer and overwrite the given string if it is not properly null terminated.
IMPORTANT: This function CANNOT be passed a literal. E.g. the following code will not work:

```
transmit("Some Message");
```

The literal is stored in ROM, and the pointer argument the function uses cannot access ROM memory. This will result in unexpected behaviour. Instead use the following code:

```
char message[] = "Some Message";
transmit(message);
```

This loads the literal into RAM when the program begins.

Arguments:

A pointer to the start of a null-terminated string to begin transmitting.

Return value:

No return value

1.4.3 transmitted

Usage:

Used to determine if a transmit action has been completed. For example if coordination between serial transmissions and other actions is required then this function must be used.

Arguments:

This function takes no arguments

Return Value:

non-zero (true) if all transmit actions have been completed (i.e. the transmit buffer is empty).

1.4.4 transChar

Usage

Used to transmit a single character over serial. Works exactly the same way as transmit, and will thus not interfere with any other transmit function. Use of this function will remove the necessity of defining a string and null terminating it when only a single character has to be transmitted.

Arguments:

A character to transmit over serial

Return Value:

No return value

1.4.5 receiveEmpty

Usage:

Used to determine if anything has been received over serial.

Arguments:

No Arguments

Return Value:

Non-zero if the the receive buffer is empty (nothing new has been received).

1.4.6 receivePeek

Usage:

Used to inspect the next character received over serial without removing it from the buffer. (e.g. a subsequent receivePeek, or receivePop operation will return the same character).

Arguments:

No arguments

Return Value:

The next character received over serial

1.4.7 receivePop

Usage:

Used to read a character received over serial.

Arguments:

No Arguments

Return Value:

the next character received over serial

1.4.8 receiveCR

Usage:

Used to determine whether a Carriage Return has been detected over serial. Carriage Return is used as a "user input confirm" button.

Arguments:

No Arguments

Return Value:

Non-zero if a carriage return has been detected (i.e. there is a CR in the receive buffer)

1.4.9 readString

Usage:

Used to read an entire string received over serial.

Arguments:

Pointer to memory address to store received characters.
NOTE: To avoid possible memory errors, Make sure to reserve the maximum number of characters that can be returned. The macro BUFFERLENGTH (the length of the receive buffer) is defined in CircularBuffers.h. E.g. the following code is advisable:

```
#include "CircularBuffers.h"
...
char [BUFFERLENGTH] receivedData;        //Reserve the max number of characters
readString(receivedData);                //Read string into address
```

Return Value:

No return value

1.5 Example Code:

```
#include "Common.h"
#include "Serial.h"
#include "CircularBuffers.h"     //Only used for BUFFERLENGTH definition

void main()
{
        char message[] = "Serial Example";
        char received[BUFFERLENGTH];

        //Set up the Serial module for use
        configureSerial();

        //Transmit the message
        transmit(message);

        //Wait for a carriage return
        while(!receiveCR());

        //Read in returned string
        readString(received);
```

```
        //Transmit received string
        transmit(received);

        for (;;)
        {
                //Poll received data
                while (receiveEmpty());

                //Re-transmit received character
                transChar(receivePop());
        }
}
```

Chapter 2

PanTilt

2.1 Description:

The Pan Tilt module is the module which controls the servo actuators which point the sensors in the correct direction. As such it generates the PDM's (Pulse Duration Modulated waves) necessary to run the servo's, and has public functions such as move, which directs the Pan Tilt mechanism to a certain direction.

2.2 How It Works:

2.3 How To Use It:

-

2.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

2.4.1

Usage:

Arguments:

Return Value:

2.5 Example Code:

Chapter 3

Temp

3.1 Description

The Temperature module controls the sampling and calibration of the temperature sensor.

3.2 How It Works:

3.3 How To Use It:

-

3.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

3.4.1

Usage:

Arguments:

Return Value:

3.5 Example Code:

Chapter 4

Tracking

4.1 Description

The tracking module contains the high level tracking and search algorithms used by the system. It then uses the Pan Tilt and Range modules to enact these modules.

4.2 How It Works:

4.3 How To Use It:

-

4.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

4.4.1

Usage:

Arguments:

Return Value:

4.5 Example Code:

Chapter 5

User_Interface

5.1 Description

The user interface module is the equivalent of the serial module for the local user interface. It has a circular buffer and stores all the user inputs to the system to be handled later.

5.2 How It Works:

5.3 How To Use It:

-

5.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

5.4.1

Usage:

Arguments:

Return Value:

5.5 Example Code:

Chapter 6

Range

6.1 Description

The Rang module is responsible for sampling the range from the IR and ultrasonic sensors. It is also used to detect the object as the functions return zero if the object is out of range.

6.2 How It Works:

6.3 How To Use It:

-

6.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

6.4.1

Usage:

Arguments:

Return Value:

6.5 Example Code:

Chapter 7

MenuSystem

7.1 Description

The Menu System Module is responsible for keeping track of the menu state, and parses all user inputs. The module interfaces with the serial and user interface modules to handle all inputs to the system.

7.2 How It Works:

7.3 How To Use It:

-

7.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

7.4.1

Usage:

Arguments:

Return Value:

7.5 Example Code:

Chapter 8

LCD

8.1 Description

The LCD module is responsible for interfacing with the LCD component and displaying LCD data in the local user mode.

8.2 How It Works:

8.3 How To Use It:

-

8.4 List of Public Functions

Below is a definitive list of all the public functions contained in this module, their use, their arguments and their returns.

8.4.1

Usage:

Arguments:

Return Value:

8.5 Example Code: