ECE 341 Lab 7

9/10

Zachary DeLuca

March 22nd 2023

1 Introduction

For this lab, we worked with using the RS232 connection between the Cerebot and the computer. This allows us to interact with the Cerebot board by sending serial data and being able to read serial data through the connection. We use the putty terminal emulator in conjunction with the functions and macros defined in the comm files to write and read and display the data to and from the Cerebot. For this particular lab, we will be able to control the speed of the motor from either the input in the Cerebot board buttons or by inputting a custom configuration within the terminal. After each change to the motor, there should be a display on both the LCD screen and in the terminal showing the configuration of the motor and its origin or any errors that have occurred from user error. The code for this lab has been chopped into many parts in order to test each subsystem individually and so each of the listings shows a different file used in the overall project, and each listing will be labeled with what system it is in charge of.

✓

2 Implementation

Since this was provided I wouldn't expect it to be included in report

The first two listings are just the ones given to us in the lab handout, but these are the files that allow the user to talk to the Cerebot from the terminal emulator.

2.1 Listing 7.1: comm.h

```
/********************************* comm.h *********************************
* Author:     Richard Wall
* Date:       August 12, 2013
***************************************************************************/

#ifndef __MX7COMM_H__

#define __MX7COMM_H__
#define _UART1
#endif
```

```
#ifndef __COMM_H__

#define __COMM_M__
#define BACKSPACE     0x08
#define NO_PARITY     0
#define ODD_PARITY    1
#define EVEN_PARITY   2

#endif

#include <stdio.h>
#include <string.h>
void initialize_uart1(unsigned int baud, int parity);
void _mon_putc(char c); /* Called by system to implement "printf" functions */
int putcU1( int c);    /* Send single character to UART */
int getcU1( char *ch); /* Get single character from UART */
int putsU1( const char *s);   /* Send string to UART1 */
int getstrU1( char *s, unsigned int len ); /* Get CR terminated string */

/* End of comm.h */
```

2.2 Listing 7.2: comm.c

```
/****************************** comm.c *********************************
* Author:     Richard Wall
* Date:       August 12, 2013 - Original
*             17 Mar 2017 - removed redundant pointer from getstrU1()
*
* This code also uses the "printf" function on UART Serial Port 1
*
* Cerebot MX7cK requires UART crossover cable if connecting with PmodRS232
* Connector Pin  Pmod Pin   Function
*  2    4  MX7 Rx
*  3    3  MX7 Tx
*  5    5  Gnd
*  6    6  Vcc
***************************************************************************/

#include <plib.h>
#include <stdio.h>           /* Required for printf */
#include "comm.h"
#include "CerebotMX7cK.h" /* Has info regarding the PB clock */

/* initialize_comm FUNCTION DESCRIPTION *************************************
* SYNTAX:        void initialize_comm(unsigned int baud, int parity);
* KEYWORDS:      UART, initialization, parity
* DESCRIPTION:   Initializes UART1 comm port for specified baud rate using
*                the assigned parity
* PARAMETER 1:   integer Baud rate
* PARAMETER 1:   integer (parity, NO_PARITY, ODD_PARITY, or EVEN_PARITY)
* RETURN VALUE:  None
*
* NOTES:         9 bit mode MARK or SPACE parity is not supported
* END DESCRIPTION **********************************************************/
void initialize_uart1(unsigned int baud, int parity)
{
   unsigned int BRG;

   BRG=(unsigned short)(((float)FPB / ((float)4 * (float) baud))-(float)0.5);
   switch(parity)
   {
      case NO_PARITY:
      OpenUART1( (UART_EN | UART_BRGH_FOUR | UART_NO_PAR_8BIT),
      (UART_RX_ENABLE | UART_TX_ENABLE) , BRG );
      break;
      case ODD_PARITY:
      OpenUART1( (UART_EN | UART_BRGH_FOUR | UART_ODD_PAR_8BIT),
      (UART_RX_ENABLE | UART_TX_ENABLE) , BRG );
      break;
      case EVEN_PARITY:
      OpenUART1( (UART_EN | UART_BRGH_FOUR | UART_EVEN_PAR_8BIT),
      (UART_RX_ENABLE | UART_TX_ENABLE) , BRG );
      break;
   }

   printf("\n\nCerebot MX7ck Serial Port 1 ready\r\n");
}

/* _mon_putc FUNCTION DESCRIPTION ******************************************
* SYNTAX:        void _mon_putc(char c);
* KEYWORDS:      printf, console, monitor
* DESCRIPTION:   Sets up serial port to function as console for printf.
*                Used only by system.
* PARAMETER 1:   Character to send to monitor
* RETURN VALUE:  None
* NOTES:         This function will block until space is available
*                in the transmit buffer
* END DESCRIPTION **********************************************************/
void _mon_putc(char c)
{
   while(BusyUART1());
   WriteUART1((unsigned int) c);
} /* End of _mon_putc */

/* putcU1 FUNCTION DESCRIPTION ********************************************
* SYNTAX:        int putcU1( int c);
* KEYWORDS:      UART, character
* DESCRIPTION:   Waits while UART1 is busy (buffer full) and then sends a
*                 single byte to UART1
* PARAMETER 1:    character to send
* RETURN VALUE:   character sent
* NOTES:          This function will block until space is available
*                 in the transmit buffer
* END DESCRIPTION **********************************************************/
int putcU1( int c)
{
   while(BusyUART1());
   WriteUART1((unsigned int) c);
   return c;
} /* End of putU1 */

/* getcU1 FUNCTION DESCRIPTION ********************************************
* SYNTAX:         int getcU1( char *ch);
* KEYWORDS:       character, get
* DESCRIPTION:    Checks for a new character to arrive to the UART1 serial port.
* PARAMETER 1:    character pointer to character variable
* RETURN VALUE:   TRUE = new character received
*                 FALSE = No new character
* NOTES:          This function does not block for no character received
* END DESCRIPTION ********************************************************/
int getcU1( char *ch)
{
   if( !DataRdyUART1()) /* wait for new char to arrive */
   return FALSE;        /* Return new data not available flag */
   else
   {
      *ch = ReadUART1(); /* read the char from receive buffer */
      return TRUE;    /* Return new data available flag */
   }
}/* End of getU1 */

/* putsU1 FUNCTION DESCRIPTION ********************************************
* SYNTAX:         int putsU1( const char *s);
* KEYWORDS:       UART, string
* DESCRIPTION:    Sends a NULL terminates text string to UART1 with
*                 CR and LF appended
* PARAMETER 1:    pointer to text string
* RETURN VALUE:   Logical TRUE
* NOTES:          This function will block until space is available
*                 in the transmit buffer
* END DESCRIPTION **********************************************************/
int putsU1( const char *s)
{
   putsUART1(s);
   putcUART1( '\r');
   putcUART1( '\n');
   return 1;
} /* End of putsU1 */
```

```
/* getstrU1 FUNCTION DESCRIPTION ********************************************
* SYNTAX:         int getstrU1( char *s, unsigned int len );
* KEYWORDS:       string, get, UART
* DESCRIPTION:    This function assembles a line of text until the number of
*                 characters assembled exceed the buffer length or an ASCII
*                 CR control character is received. This function echo each
*                 received character back to the UART. It also implements a
*                 destructive backspace. ASCII LF control characters are
*                 filtered out. The returned string has the CR character
*                 removed and a NULL character appended to terminate the text
*                 string. BACKSPACE defined in comm.h header file.
* PARAMETER 1:    character pointer to string
* PARAMETER 2:    integer maximum string length
* RETURN VALUE:   TRUE = EOL signaled by receiving return character
*                 FALSE = waiting for end of line
* NOTES:          It is presumed that the buffer pointer or the buffer length
*                 does not change after the initial call asking to
*                 receive a new line of text. This function does not block
*                 for no character received. A timeout can be added to this
*                 to free up resource. There is no way to restart the function
*                 after the first call until a EOL has been received. Hence
*                 this function has denial of service security risks.
* END DESCRIPTION **********************************************************/
int getstrU1( char *s, unsigned int len )
{
   static int eol = TRUE; /* End of input string flag*/
   static unsigned int buf_len;
   static char *p;      /* copy of the buffer pointer */
   char ch;              /* Received new character */

   if(eol)               /* Start of new line?           */
   {
      p = s;            /* Copy pointer for backspacing  */
      eol = FALSE;
      buf_len = len - 1; /* Save max buffer length with room for NULL */
   }

   if(!(getcU1(&ch)))    /*  Check for character received */
   {
      return FALSE;     /* Bail out if not */
   }
   else
   {
      *p = ch;      /* Save new character in string buffer */
      putcU1( *p);  /* echo character */
      switch(ch) /* Test for control characters */
      {
         case BACKSPACE:       // defined as 0x08 (cntl-H) in comm.h
         if ( p>s)          /* prevent backing up past the start! */
         {
            putcU1( ' ');   /* overwrite the last character */
            putcU1( BACKSPACE);
            buf_len++;
            p--;            /* back off the pointer */
         }
         break;
         case '\r':              /* carriage return */
         putcU1( '\r');      /* echo character */
         eol = TRUE;         /* Set end of line */
         break;
         case '\n':              /* newline (line feed), not EOL */
         putcU1('\n');       // ignore but echo anyway
         break;              // PuTTY only sends \r for Enter Key
         default:
         p++;               /* increment buffer pointer */
         buf_len--;          /* decrement length counter */
      } // end of switch
   } // end of else

   if( buf_len == 0 || eol)    /* Check for buffer full or end of line */
   {
      *p = '\0';             /* add null terminate the string */
      eol = TRUE;
      return TRUE;            /* Set EOL flag */
   }
   return FALSE;               /* Not EOL */

} /* End of getstr */

/* End of comm.c */
```

The _mon_putc() function has been included in this part of the code in order to redirect the output of the print() functions, as the default output location is the terminal in the ISE, meaning that the putty terminal would never see the outputs or the inputs. With this function we can redefine where we receive and send data to on the computer side.

The next pair of listings is the code defined in previous labs that control the stepper motor. In the steppers C file, the macro ”ent” is used. This macro is found in the lab 7 files and means an external unsigned integer. The extern means that we are using variables from other files, and is essential for passing values from file to file, as different functions from different files will want to change or read values pertaining to the motor and its state.

2.3 Listing 7.3: Stepper.h

```
#ifndef _STEP_H
#define _STEP_H

#include <plib.h>
```

```
#include "CerebotMX7cK.h"

#define sui static unsigned int
#define uint unsigned int
#define ent extern uint

#define T1_PRESCALE 1
#define TOGGLES_PER_SEC 1000
#define T1_TICK (FPB/T1_PRESCALE/TOGGLES_PER_SEC)
#define T1_INTR_RATE 10000

void timer1_interrupt_initialize(void);
uint delayed(uint in);
int stepperState(int dir, int mode);
void stepperPush(int input);


const uint step[8] = {0x0A,0x08,0x09,0x01,0x05,0x04,0x06,0x02};

#endif
```

2.4 Listing 7.4: Stepper.C

```
#include "step.h"

ent literate;
ent shoe;
ent walk;
ent step_delay;
ent dir;
ent mode;
ent stepWait;
ent btnWait;
ent source;
ent error;

void timer1_interrupt_initialize(void)
{
   //configure Timer 1 with internal clock, 1:1 prescale, PR1 for 1 ms period
   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_1, T1_INTR_RATE-1);
   // set up the timer interrupt with a priority of 2, sub priority 0
   mT1SetIntPriority(2); // Group priority range: 1 to 7
   mT1SetIntSubPriority(0); // Subgroup priority range: 0 to 3
   mT1IntEnable(1); // Enable T1 interrupts
}

uint delayed(uint in){
   if(mode == 1){
        in = 600/in;
    }
    else{
        in = 300/in;
    }
    return in;
}

int stepperState(int dir, int mode){
    sui pstate;
    if(dir == 1){
        if (mode ==1){
            switch(pstate){
                case 6:
                pstate = 0;
                break;
                case 7:
                pstate = 1;
                break;
                default:
                pstate+=2;
                break;
            }
        }
        else{
            switch(pstate){
                case 7:
                pstate = 0;
                break;
                default:
                pstate++;
                break;
            }
        }
    }
    /* Direction Switch*/
    else{
        if (mode ==1){
            switch(pstate){
                case 1:
                pstate = 7;
                break;
                case 0:
                pstate = 6;
                break;
                default:
                pstate-=2;
                break;
            }
        }
        else{
         switch(pstate){
            case 0:
            pstate = 7;
            break;
            default:
            pstate--;
            break;
         }
      }
   }
   if (pstate > 7){
      pstate = 0;
   }
   if (pstate < 0){
      pstate = 7;
   }
   return pstate;
}


void stepperPush(int shoe){
   LATBCLR = SM_COILS;
   uint write = step[shoe];
   write = write << 7;
   LATBSET = write;
   //LATBINV = step[shoe] << 2;

}

void __ISR(_TIMER_1_VECTOR, IPL2) Timer1Handler(void)
{
   LATBINV = LEDB;
   if(!stepWait){
      LATBINV = LEDA;
      shoe = stepperState(dir,mode);
      stepperPush(shoe);
      stepWait = step_delay;
   }
   stepWait--;
   mT1ClearIntFlag(); // Macro function to clear the interrupt flag
}
```

The next pair of listings is the code that was made in the previous lab but controls the LCD.

2.5 Listing 7.5: LCDLIB.H

```
#ifndef _LCDLIB_H /* Guard against multiple inclusion */
#define _LCDLIB_H
```

```
#include <plib.h>
#include "CerebotMX7cK.h"

#define T1_PRESCALE 1
#define TOGGLES_PER_SEC 1000
#define T1_TICK (FPB/T1_PRESCALE/TOGGLES_PER_SEC)
#define T1_INTR_RATE 10000

void Timer1_delay(int delay);

void LCDinit();
void writeLCD(int addr, char c);
char readLCD(int addr);
void LCD_puts(char *char_string);
void LCD_putc(char intake);
void cursor_step(char intake);
void clearLCD();
int busyLCD();
void busy();

#endif
```

2.6 Listing 7.6: LCDLIB.C

```
#include "LCDlib.h"

void LCDinit(){
   int cfg1 = PMP_ON|PMP_READ_WRITE_EN|PMP_READ_POL_HI|PMP_WRITE_POL_HI;
   int cfg2 = PMP_DATA_BUS_8 | PMP_MODE_MASTER1 |
   PMP_WAIT_BEG_4 | PMP_WAIT_MID_15 | PMP_WAIT_END_4;
   int cfg3 = PMP_PEN_0; // only PMA0 enabled
   int cfg4 = PMP_INT_OFF; // no interrupts used
   mPMPOpen(cfg1, cfg2, cfg3, cfg4);

   Timer1_delay(50);
   PMPSetAddress(0);
   PMPMasterWrite(0x38);

   Timer1_delay(50);
   PMPSetAddress(0);
   PMPMasterWrite(0x0F);

   Timer1_delay(50);
   PMPSetAddress(0);
   PMPMasterWrite(0x01);
   Timer1_delay(50);
}
```

```
void LCD_puts(char *char_string)
{
   while(*char_string) // Look for end of string NULL character
   {
      LCD_putc(*char_string); // Write character to LCD
      char_string++; // Increment string pointer
   }
} //End of LCD_puts

char readLCD(int addr)
{
   PMPSetAddress(addr); // Set LCD RS control
   mPMPMasterReadByte(); // initiate dummy read sequence
   return mPMPMasterReadByte();// read actual data
} // End of readLCD

void writeLCD(int addr, char c)
{
   busy(); // Wait for LCD to be ready
   PMPSetAddress(addr); // Set LCD RS control
   PMPMasterWrite(c); // initiate write sequence
} // End of writeLCD

void LCD_putc(char intake){
   int location;
   location = readLCD(0);
   if(location > 0xF && location < 0x40){
      PMPSetAddress(0);
      writeLCD(0,0xC0);
      if(intake!=’ ’)
      LCD_putc(intake);
   }
   else if (location > 0x4F){
      writeLCD(0,0x80);
   }
   if(intake == ’\r’){
      if(location < 0xF){
         writeLCD(0,0x80);
      }
      else{
         writeLCD(0,0xC0);
      }
   }
   else if(intake == ’\n’){
      if(location < 0xF){
         writeLCD(0,0x80+location+0x40);
      }
      else{
         writeLCD(0,0x80+location-0x40);
      }
   }

   if(intake != '\r' && intake!='\n'){
      writeLCD(1,intake);
   }
   Timer1_delay(1);

}
void clearLCD(){
   writeLCD(0,0x01);
   writeLCD(0,0x80);
   LCD_putc(' ');
}
int busyLCD(){
   int check;
   check = mIsPMPBusy();
   if (check){
      return 1;
   }
   else{
      return 0;
   }
}
void busy(){
   while(busyLCD());
   //Timer1_delay(50);
}

void Timer1_delay(int delay)
{
   while(delay--)
   {
      while(!mT1GetIntFlag()); // Wait for interrupt flag to be set
      mT1ClearIntFlag(); // Clear the interrupt flag
   }
}
```

The next pair of listings were created in a previous lab and control how the interrupt routines handle the button presses. There was an additional function added so that the displays were updated as well as there was no way of handling displays when the code was written to handle the buttons.

2.7 Listing 7.7: BTN.H

```
#ifndef _BTN_H
#define _BTN_H

#include <plib.h>
#include "CerebotMX7cK.h"
```

```
#define sui static unsigned int
#define uint unsigned int
#define ent extern uint

#define T1_INTR_RATE 10000

int readButtons();
void decodeButtons(int buttons);
void cn_interrupt_initialize(void);

#endif
```

2.8 Listing 7.8: BTN.C

```
#include "btn.h"

ent literate;
ent shoe;
ent walk;
ent step_delay;
ent dir;
ent mode;
ent stepWait;
ent btnWait;
ent source;
ent error;

int readButtons(){
   int mask = (BIT_6 | BIT_7);
   int garbo = PORTG & mask;
   return garbo;
}

void decodeButtons(int buttons){
   int output = 0;
   switch(buttons){
      default:
      dir = 1;
      mode = 0;
      step_delay = delayed(15);
      LATGCLR = LED1 | LED2;
      break;
      case BTN1:
      dir = 1;
      mode = 1;
      step_delay = delayed(15);
      LATGCLR = LED1 | LED2;
      LATGSET=LED1;
      break;
      case BTN2:
      dir = 0;
      mode = 0;
      step_delay = delayed(10);
      LATGCLR = LED1 | LED2;
      LATGSET=LED2;
      break;
      case 0x000000C0:
      dir = 0;
      mode = 1;
      step_delay = delayed(25);
      LATGCLR = LED1 | LED2;
      LATGSET = LED1 | LED2;
      break;

   }
}


void cn_interrupt_initialize(void) // Code that is executed only once
{
   unsigned int dummy; // used to hold PORT read value
   // BTN1 and BTN2 pins set for input by Cerebot header file
   // PORTSetPinsDigitalIn(IOPORT_G, BIT_6 | BIT7); //
   // Enable CN for BTN1 and BTN2
   mCNOpen(CN_ON,(CN8_ENABLE | CN9_ENABLE), 0);
   // Set CN interrupts priority level 1 sub priority level 0
   mCNSetIntPriority(1); // Group priority (1 to 7)
   mCNSetIntSubPriority(0); // Subgroup priority (0 to 3)
   // read port to clear difference
   dummy = PORTReadBits(IOPORT_G, BTN1 | BTN2);
   mCNClearIntFlag(); // Clear CN interrupt flag
   mCNIntEnable(1); // Enable CN interrupts
   // Global interrupts must enabled to complete the initialization.
}



void __ISR(_CHANGE_NOTICE_VECTOR, IPL1) CNIntHandler(void)
{
   LATBSET = LEDC;
   Timer1_delay(20);
   literate = readButtons();
   decodeButtons(literate);
   LATBCLR = LEDC;
   source = 1;
   current_events();
   mCNClearIntFlag(); // Macro function
```

```
}
```

?

This last pair of listings is the main part of the lab written by the student in this week. This is what ties all the subsystems together and lets the whole system work.

2.9 Listing 7.9: Lab7.H

```
#ifndef _LAB_7_H  /* Guard against multiple inclusion */
#define _LAB_7_H

#include <plib.h>
#include "CerebotMX7cK.h"
#include <stdio.h>          /* Required for printf */
#include <string.h>
#include "comm.h"
#include "LCDlib.h"
#define sui static unsigned int
#define uint unsigned int
#define ent extern uint

#define T1_PRESCALE 1
#define TOGGLES_PER_SEC 1000
#define T1_TICK (FPB/T1_PRESCALE/TOGGLES_PER_SEC)
#define T1_INTR_RATE 10000

char num[5];
char M[3], D[5];
int S;

void Cerebot_mx7cK_setup(void); /* Cerebot MX7cK hardware initialization */
void system_init (void); /* hardware initialization */
void current_events();
void decode_putty(char *intake);


#define T1_PRESCALE 1
#define TOGGLES_PER_SEC 1000
#define T1_TICK (FPB/T1_PRESCALE/TOGGLES_PER_SEC)



#endif
```

2.10 Listing 7.10: Lab7.C

```
#include "lab7.h"
#include "lab5h.h"
char str_buf[32];
char rdy_msg[] = "\r\n\r\nUART open and ready for business!!\r\n\r\n";
char rules[] = "Enter in motor control as:";
char rules2[] = "MODE(FS/HS) SPEED(RPM) DIRRECTION(CW or CCW)";
unsigned int count = 0;

uint literate = 0; //Stores the value of the buttons, because it's... well read
uint shoe = 0; // stores state of the stepper
uint walk = 0; // stores value to output to stepper
uint step_delay = 600/30; //delay
uint dir = 1; //0 for CCW and 1 for CW
uint mode = 0; // 0 for half and 1 for full
uint stepWait = 600/30;
uint btnWait = 20;
uint source = 0;
uint error = 0;

int main(){
   OpenTimer1(T1_ON | T1_PS_1_1, (T1_TICK-1));
   system_init (); /* Setup system Hardware. */
   putsU1(rules);
   putsU1(rules2);
   current_events();
   while(1) /* Infinite application loop */
   {
      while(!getstrU1(str_buf, sizeof(str_buf)));
      decode_putty(str_buf);
      putsU1("\r\n");
      putsU1(str_buf);
      //putsU1("\r\n");
      sprintf(str_buf, "Message number %d", count++);
      putsU1(str_buf);
      if(!error){
         current_events();
      }

   } // end while(1)
   return 0;
}

void decode_putty(char *intake){
   mCNIntEnable(FALSE);
   clearLCD();
   LCD_puts(intake);

   error = 0;

   sscanf(intake,"%s %d %s",M,&S,D);
   LCD_puts("\r\nDECODE COMPLETE");


   if(!strcmp(M,"FS")){
      mode = 1;
   }
   else if(!strcmp(M,"HS")){
      mode = 0;
   }
   else{
      clearLCD();
      LCD_puts("Mode ERROR");
      putsU1("\r\nMode ERROR");
      error = 1;
   }
   if(!strcmp(D,"CW")){
      dir = 1;
   }
   else if(!strcmp(D,"CCW")){
      dir = 0;
   }
   else{
      clearLCD();
      LCD_puts("Dir ERROR");
      putsU1("\r\nDir ERROR");
      error = 1;
   }
   if(S >0 && S<51){
      step_delay=delayed(S);
   }
   else{
      clearLCD();
      LCD_puts(" Speed ERROR");
      putsU1("\r\nSpeed ERROR");
      error = 1;
   }

   if(!error && source){
      current_events();
   }
   source = 0;


   mCNIntEnable(TRUE);
}

void system_init(void){
   // Setup processor board
   Cerebot_mx7cK_setup();
   PORTSetPinsDigitalOut(IOPORT_G, LED1|LED2|LED3|LED4);
   PORTSetPinsDigitalOut(IOPORT_B, SM_LEDS);/* Set PmodSTEP LEDs outputs */
   LATBCLR = SM_LEDS;
   LCDinit();

   initialize_uart1(19200, ODD_PARITY);

   INTEnableSystemMultiVectoredInt(); //done only once
   INTEnableInterrupts(); //use as needed

   timer1_interrupt_initialize();
   cn_interrupt_initialize();
}

void current_events(){
   clearLCD();
   putsU1("\r\nConfirmed:");
   char output[16];
   char speed[5];
   sprintf(speed,"%d",((600/(2-mode))/step_delay));
   if(mode == 0){
      M[0] = 'H';
      M[1] = 'S';
      M[2] = '\0';
   }
   else{
      M[0] = 'F';
      M[1] = 'S';
      M[2] = '\0';
   }
   if(dir == 0){
      D[0] = 'C';
      D[1] = 'C';
      D[2] = 'W';
      D[3] = '\0';
   }
   else{
      D[0] = 'C';
      D[1] = 'W';
      D[2] = '\0';
   }
   sprintf(output,"%s %s %s",M, speed, D);
   putsU1(output);
   LCD_puts(output);
   if(source == 0){
      LCD_puts("\r\nMangement");
      putsU1("Management\r\n");
   }
   else{
      LCD_puts("\r\nFloor");
      putsU1("Floor\r\n");
   }
   putsU1("Ready for input:");
}
```

3 Testing and Verification

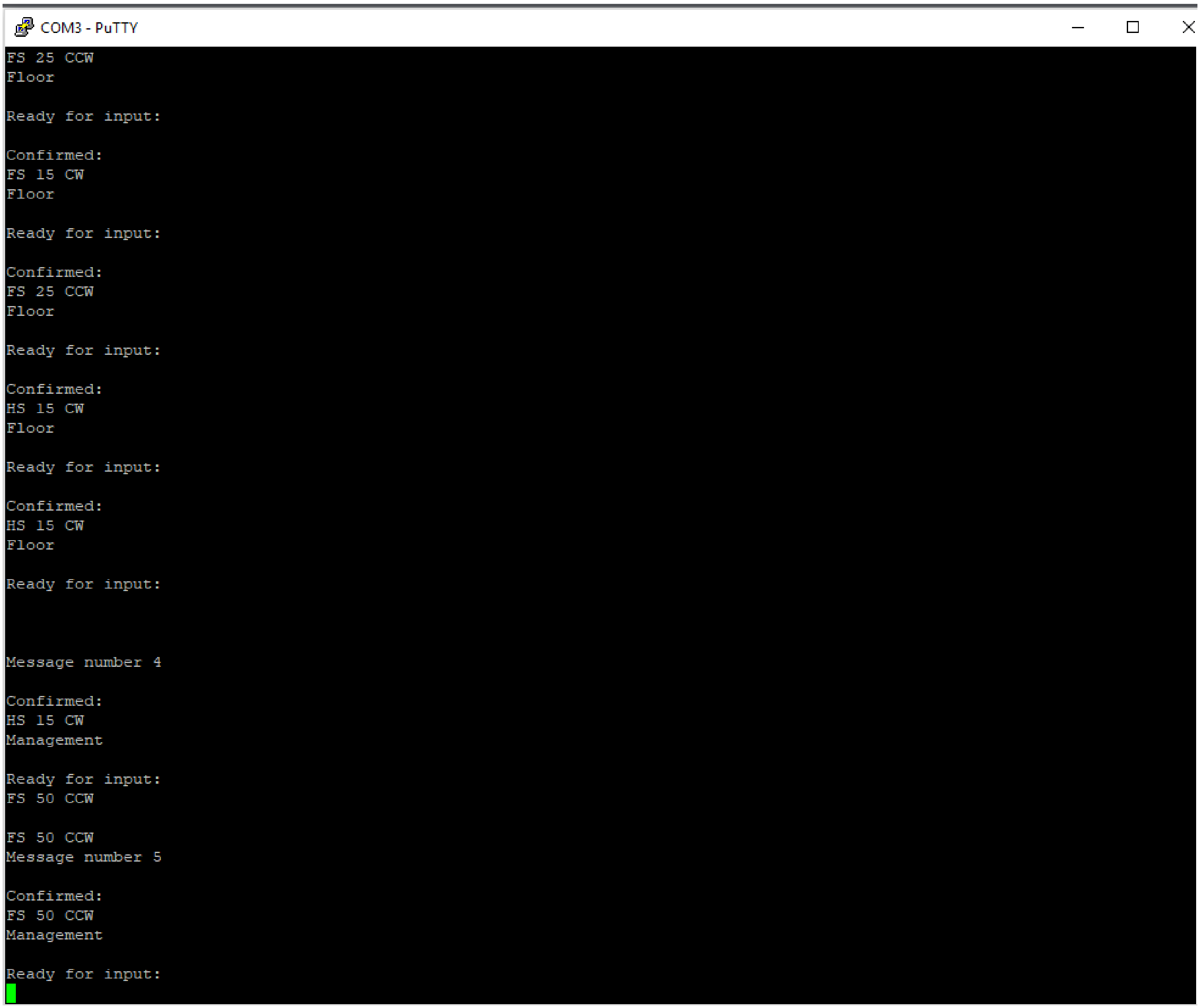
To describe why each subsystem was added, we will run through the list:

Comm Library: This allows us to use the putty terminal and the RS232 communication protocols
LCDLIB: This allows us to display things on the LCD display
BTN: This allows us to control things with the buttons
Stepper: This allowed the data to control the stepper motor

As each subsystem was modular, it allowed for us to remove the other functions from the project and test one at a time. Instead of removing the files during testing, it was often quicker and easier to comment out most of a file or the entire file.

✓

As proof of a working program, a screenshot of the putty window has been provided showing the motor being controlled from both the computer and the buttons, with the message reading from management if it is from the computer and floor if it came from the board buttons.

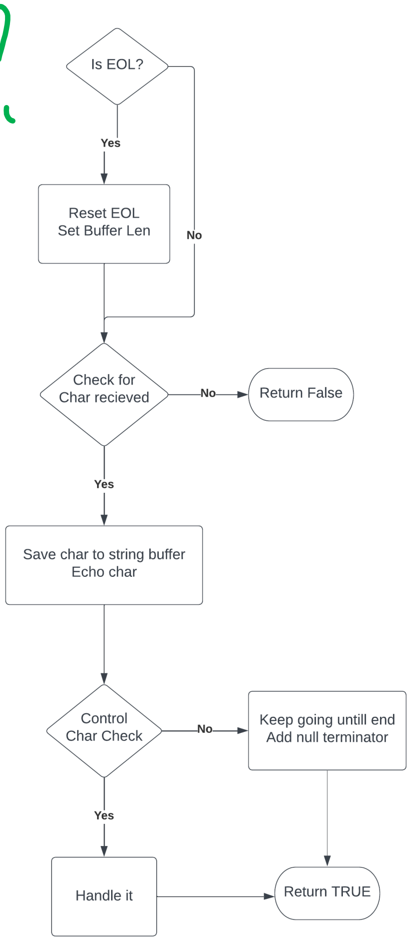


```
FS 25 CCW
Floor

Ready for input:

Confirmed:
FS 15 CW
Floor

Ready for input:

Confirmed:
FS 25 CCW
Floor

Ready for input:

Confirmed:
HS 15 CW
Floor

Ready for input:

Confirmed:
HS 15 CW
Floor

Ready for input:



Message number 4

Confirmed:
HS 15 CW
Management

Ready for input:
FS 50 CCW

FS 50 CCW
Message number 5

Confirmed:
FS 50 CCW
Management

Ready for input:
```

4 getstrU1 CFD

Why duy in the backgrund?

-1



✓

5 Conclusion

In this lab we were able to complete the assigned tasks and added some extra features that allow the user to know when the entered in something outside the expected input. This lab allowed us to use the serial communication of the RS232 port which is preferable to the parallel communication of the previous lab when trying to send information bidirectionally. The serial communication with multiple lines allow us to read and write at the same time as it can receive and transmit at the same time. We can also have greater control of the data transfer speed using standardized baud rates instead of using a handshake protocol like the one that was needed for the LCD controller. The disadvantage of course is that this peer to peer network is stuck with only 2 units communicating, as a master slave setup could be a network of one to many. This lab was also the first that required the code to be chopped up into different libraries. This is beneficial when writing a program with multiple subsystems as the increased complexity allows for error to hide themselves better in the sea of code. By separating each subsystem, it becomes much easier to find and fix errors that pop up in the code.