Appendix C

Test Code

Test code was written to test certain components with the microcontroller. Test code was written for the following components:

Test Code for Wireless Module with No LCD Attached ................................................................. 1

Test Code for LCD .......................................................................................................................... 4

Test Code for Wireless Module with LCD Attached ................................................................. 8

Test Code for the Keypad Buttons .............................................................................................. 14

Test Code for the 7-Segment Display ......................................................................................... 16
Test Code for Wireless Module with No LCD Attached

//Wireless test no lcd
/*******************************************************************************
************ Wireless ************
*******************************************************************************/
#define UPDATE   1
#define MAXMESSAGE  6 //Largest message is 4 bytes, +1 for '\0'
#define TIMEOUT   2000
#define D_BAUDRATE 9600

nodebug root int serD_gets(char *string,int max,unsigned long timeout);
/* START FUNCTION DESCRIPTION *********************************************/
init_communications                    <RFMODEM.LIB>
SYNTAX:     void init_communications(void);
DESCRIPTION:  Reads the current configuration of the attached RF modem and
               changes any settings that need to be updated.
RETURN VALUE: none.
END DESCRIPTION ***********************************************************/

void initialize_modem()
{
  char modemMessages[MAXMESSAGE];
  unsigned int update;

  update=0;

  serDopen(D_BAUDRATE);
  serDwrFlush();
  serDrdFlush();

  serDputs("+++");   //Command Mode
  serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
  serDputs("ATSM\r");
  serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
  if(strcmp(modemMessages, "0"))
  {
    serDputs("ATSM0\r"); //Sleep mode IE: 2 seconds cycle, 8 seconds, etc (Page 29 in the
                          manual)
    serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
    update = update;
  }

  serDputs("ATHP\r");
  serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
  if(strcmp(modemMessages, "4")) //Returned in ASCII hex
  {
    serDputs("ATHP4\r"); //Sleep mode IE: 2 seconds cycle, 8 seconds, etc (Page 29 in the
                          manual)
    serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
    update = update;
  }

  serDputs("ATDT\r"); //Set address of modem
  serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
  if(strcmp(modemMessages, "0xF010")) //Returned in ASCII hex
  {
serDputs("ATDTF010\r"); //Sleep mode IE: 2 seconds cycle, 8 seconds, etc (Page 29 in the manual)
serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
update = update;
}

if(update) {
  serDputs("ATWR\r"); //Write changes to long term memory
  serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
}
serDputs("ATCN\r"); //Exit command mode
serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
}

/* START FUNCTION DESCRIPTION **************************************************
serUniv_gets <RFMODEM.LIB>
SYNTAX: int serD_gets(char *string,int max,unsigned long timeout);
DESCRIPTION: Reads characters from the getc function that is passed in via a
pointer until a carriage return is read, max number characters are read, or until
timeout milliseconds transpires between characters after the first one is read.
PARAMETER1: string: Character array into which a null terminated string
is read.
PARAMETER2: max : The maximum number of characters to read into s, not
including the null terminator.
PARAMETER3: timeout: Millisecond wait period to allow between characters
before timing out.
RETURN VALUE: 0 - timed out before CR or max characters read, 1 - no timeout.
END DESCRIPTION ***********************************************************/
nodebug root int serD_gets(char *string,int max,unsigned long timeout)
{
  int index, ch;
  long timeStart;
  index = 0;
  timeStart = MS_TIMER;
  while(max > index && timeout > (MS_TIMER - timeStart))
  {
    ch = serDgetc();
    if(ch == (int)"\r")
    {
      string[index] = '\0';
      return 1;
    }
    else if(ch != -1)
    {
      string[index] = (char)ch;
      index++;
    }
  }
  return 0;
}

/**************************** End Wireless ******************************/

2
fnMsDelay - delay some number of milliseconds
input parameter: int number of milliseconds to delay
return value: none
errors: none
*/
void fnMsDelay ( int iDelay )
{
  unsigned long ul0;
  ul0 = MS_TIMER;    // get current timer value
  while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}

void main()
{
  char modemMessage[MAXMESSAGE];
  while(1)
  {
    fnMsDelay (2500);
    serDputs("The Wireless works");
    serD_gets(modemMessage, MAXMESSAGE,TIMEOUT);
    fnMsDelay (2500);
    serDputs("2nd message");
    serD_gets(modemMessage, MAXMESSAGE,TIMEOUT);
  }
}
Test Code for LCD

/* Modified from:
lcd.c   L. Cicchinelli   Z-World, Inc   April 2000
Copyright Z-World Inc. June 2000   Larry Cicchinelli

Added support for 20x4 line display. Removed 8-bit mode.

Product: Jackrabbit single board computer.

In response to several customer requests, the following sample program was written by the Z-World Technical Support Staff. The sample code has NOT been subjected to standard software testing and validation procedures, and is being made available to users as a convenience only. There is no warranty or guarantee, implied or otherwise.

The hardware and software information below pertains to connecting an LCD which uses the HD44780 controller or an equivalent. This file contains the only documentation for this program.

This file contains information for both a 4 bit and 8 bit interface. The 4 bit interface uses one of the parallel I/O ports while the 8 bit interface uses the external I/O feature of the Rabbit 2000 microprocessor.

**********************************************************************
****************************** 4 Bit Mode ****************************
**********************************************************************

Functions which are unique to the 4 bit interface are named fn4...

This is the initialization sequence for a 2 by 20 character LCD programmed in 4 bit mode (chosen in order to reduce the number of I/O pins):
   delay for power up - allow LCD to reset
   command: 4 bit mode
   command: 2 lines, 5x10 character size, and 4 bit mode again
   command: turn on display and cursor - non blinking
   command: incr address and shift cursor with each character
   note: each command must be followed by a delay to allow the display to respond.

Here is the connection list:
Jackrabbit Pin LCD Signal
   PA0 J3-3  DB4
   PA1 J3-4  DB5
   PA2 J3-5  DB6
   PA3 J3-6  DB7
   PA6 J3-9  RS  Register Select: 0=command, 1=data
   PA7 J3-10 E  normally low: latches on high to low transition

DB0 - DB3 of the LCD are grounded as is the R/W pin. Since we are using 4 bit mode DB0 - DB3 are not used by the controller. We will only be writing so we ground the R/W pin.

Since Port A is used to control the LEDs on the expansion board, they will be controlled by the data values which you program.

VEE is used to control brightness - the simplest thing to do is ground it. If you need to control the brightness you can connect a potentiometer between ground and -5V with the arm going to the VEE pin. Check the specs on the LCD before doing this as some may require a different connection. VCC is +5 and VSS is ground.
fnMsDelay - delay some number of milliseconds
input parameter: int number of milliseconds to delay
return value: none
errors: none
*/
void fnMsDelay ( int iDelay )
{
    unsigned long ul0;
    ul0 = MS_TIMER;    // get current timer value
    while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}

fnUsDelay - delay some number of micro seconds - very approximate!!
about 11usec per iteration with the 7.3MHz crystal. The formula
used here was determined experimentally.
*/
void fnUsDelay ( int iDelay )
{
    int i;
    iDelay /= 11;
    for ( i=0; i<iDelay; i++ );
}

#define DATA  0x40
#define COMMAND 0x00
int iDataFlag;

fn4OneNib - send a single nibble to the LCD
*/
void fn4OneNib ( char cNib )
{
    cNib &= 0x0F;        // remove upper 4 bits
    cNib |= iDataFlag;      // insert data flag for RS
    WrPortI ( PADR, NULL, cNib|0x80 ); // assert E
    WrPortI ( PADR, NULL, cNib );   // remove E
}

#asm
fn4OneNib::
; character is in HL
    ld    a, L          ; get the character
    and  0x0F           ; insure upper nibble is zero
    ld    hl, iDataFlag ; get address of data flag
    or    (hl)          ; insert Data/Command bit
    or    0x80          ; assert E
    ioi   ld    (PADR),a  ; ship data value and E
           and  0x7F          ; remove E
    ioi   ld    (PADR),a  ; ship data value without E
    ret
#endasm
/**/

fn4Byte - send two nibbles to the LCD, upper nibble first
*/
void fn4Byte ( char cNib )
{
    int i;
    i = cNib;    // temp store
    cNib >>= 4;    // put upper nibble into lower 4 bits
    fn4OneNib ( cNib ); // send upper nibble
    fn4OneNib ( i );   // send lower nibble
    fnUsDelay (100);
}

/****************************************************************************
fn4LCD_Init - Initialize the LCD for the following operating parameters:
    4 bit mode, 2 lines, 5x10
    turn on display and cursor: non-blinking
    incr address and shift cursor with each character
*/
void fn4LCD_Init ()
{
    WrPortI ( PADR, NULL, 0 );       // write 0 to port A output register
    WrPortI ( SPCR, NULL, 0x84 );    // make port A output
    iDataFlag = COMMAND;            // show command mode
    fnMsDelay ( 1000 );             // wait for LCD to reset itself
    fn4OneNib ( 3 );                // 4 bit mode
    fnUsDelay (100);                // 4 bit mode
    fn4OneNib ( 3 );                // 4 bit mode
    fnUsDelay (100);                // 4 bit mode
    fn4OneNib ( 3 );                // 4 bit mode
    fnUsDelay (100);                // 4 bit mode
    fn4OneNib ( 2 );                // 4 bit mode
    fnUsDelay (100);
    fn4Byte ( '\B00101100' );       // 4 bit mode, 2 lines, 5x10
    fn4Byte ( '\B00001110' );       // turn on display and cursor: non-blinking
    fn4Byte ( '\B00000110' );       // incr address and shift cursor with each character
}

/****************************************************************************
fn4Display - display a line of text on the LCD
argument: address of null terminated text string
*/
void fn4Display ( char *szp )
{
    iDataFlag = DATA;            // next bytes are data
    while ( *szp ) fn4Byte ( *szp++ );
}

void fn4Clear ( void )
{
    iDataFlag = COMMAND;        // set up for command
    fn4Byte ( 0x01 );           // clear the display
    fnMsDelay (3);              // insure at least 2 msec
}

void fn4Line1 ( void )
{
    iDataFlag = COMMAND;       // set up for command
    fn4Byte ( 0x80 );          // set RAM address to Line 1
    //note: this value may vary for different displays
    fnMsDelay (3);             // insure at least 2 msec
}

void fn4Line2 ( void )
{
    iDataFlag = COMMAND;       // set up for command
    fn4Byte ( 0xC0 );          // set RAM address to Line 2
    //note: this value may vary for different displays
    fnMsDelay (3);             // insure at least 2 msec
}

void fn4Line3 ( void )
{
    iDataFlag = COMMAND;       // set up for command

fn4Byte ( 0x94 );     // set RAM address to Line 3
     //note: this value may vary for different displays
fnMsDelay (3);       // insure at least 2 msec
}

void fn4Line4 ( void )
{  iDataFlag = COMMAND;    // set up for command
  fn4Byte ( 0xD4 );     // set RAM address to Line 4
    //note: this value may vary for different displays
  fnMsDelay (3);       // insure at least 2 msec
}

void main ()
{
    // in the following code, use the fn4... or fn8... as required by the interface
    fn4LCD_Init ();
        // IMPORTANT - the initialization sequence should only
        // be executed once after powerup. Multiple executions
        // will cause the display to malfunction.
    while (1)
    {
        //fn4Clear();
        //fn4Display ( "0123456789" );
        //fnMsDelay (500);
        fn4Clear();
        //fn4Display ( "This is a test!" );
        //fnMsDelay (500);
        fn4Line2 ( );  
        fn4Display ( " 00 : 00 : 00  ");
        fnMsDelay (500);
        fn4Line3 ( );
        fn4Display ( " TIMES UP! ");
        fnMsDelay (500);
        //fn4Line4 ( );
        //fn4Display ( " TIMES UP! ");
        //fnMsDelay (500);
    }
}
Test Code for Wireless Module with LCD Attached

//wireless test w/LCD

/*****************************************************************************/
***********  Wireless  ******************************************
****************************************************************************/

#define UPDATE   1
#define MAXMESSAGE  6 //Largest message is 4 bytes, +1 for '\0'
#define TIMEOUT   2000
#define D_BAUDRATE 9600
nodebug root int serD_gets(char *string,int max,unsigned long timeout);
void fnMsDelay ( int iDelay );

init_communications <RFMODEM.LIB>

SYNTAX: void init_communications(void);

DESCRIPTION: Reads the current configuration of the attached RF modem and
changes any settings that need to be updated.

RETURN VALUE: none.

END DESCRIPTION

void initialize_modem()
{
    char modemMessages[MAXMESSAGE];
    unsigned int update;

    update=0;

    serDopen(D_BAUDRATE);
    serDwrFlush();
    serDrdFlush();

    serDgets("+++");  //Command Mode
    serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
    printf("1: %s \n", modemMessages);
    //fnMsDelay (1000);
    /*serDputs("ATSM\r");
    serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
    printf("2: %s \n", modemMessages);
    if(strcmp(modemMessages, "0"))
    {
        serDputs("ATSM0\r");  //Sleep mode IE: 2 seconds cycle, 8 seconds, etc (Page 29 in the
                                manual)
        serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
        printf("2b: %s \n", modemMessages);
        update = update;
    }*/

    serDputs("ATHP\r");
    serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
    printf("3: %s \n", modemMessages);
    if(strcmp(modemMessages, "4")) //Returned in ASCII hex
    {
        serDputs("ATHP4\r");  //Sleep mode IE: 2 seconds cycle, 8 seconds, etc (Page 29 in the
                                manual)
        serD_gets(modemMessages, MAXMESSAGE,TIMEOUT);
        printf("3b: %s \n", modemMessages);
    }
}
update = update;
}
serDputs("ATDT\r"); //Set address of modem
serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
printf("4: %s \n", modemMessages);
if(strcmp(modemMessages, "F010")) //Returned in ASCII hex
{
    serDgets(modemMessages, MAXMESSAGE, TIMEOUT);
    printf("4b: %s \n", modemMessages);
    update = update;
}
if(update)
{
    serDputs("ATWR\r"); //Write changes to long term memory
    serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
    printf("5: %s \n", modemMessages);
}
serDputs("ATCN\r"); //Exit command mode
serD_gets(modemMessages, MAXMESSAGE, TIMEOUT);
printf("6: %s \n", modemMessages);

/* START FUNCTION DESCRIPTION *********************************************/

int serD_gets(char *string, int max, unsigned long timeout)
{
    int index, ch;
    long timeStart;

    index = 0;
    timeStart = MS_TIMER;
    while(max > index && timeout > (MS_TIMER - timeStart))
    {
        ch = serDgetc();
        if(ch == (int)'')
        {
            string[index] = '\0';
            return 1;
        }
        else if(ch != -1)
string[index] = (char)ch;
    index++;
}
}
return 0;
}

/***************************************************************************/
***********  End Wireless ***********
******************************************************************************/

/* Modified from:
lcd.c     L. Cicchinelli     Z-World, Inc     April 2000
Copyright Z-World Inc. June 2000     Larry Cicchinelli

Added support for 20x4 line display. Removed 8-bit mode.

Product: Jackrabbit single board computer.

In response to several customer requests, the following sample program was written
by the Z-World Technical Support Staff. The sample code has NOT been subjected to
standard software testing and validation procedures, and is being made available to
users as a convenience only. There is no warranty or guarantee, implied or otherwise.

The hardware and software information below pertains to connecting an LCD which uses
the HD44780 controller or an equivalent. This file contains the only documentation for
this program.

This file contains information for both a 4 bit and 8 bit interface. The 4 bit
interface uses one of the parallel I/O ports while the 8 bit interface uses the
external I/O feature of the Rabbit 2000 microprocessor.

******************************************************************************/
****************************** 4 Bit Mode ******************************
******************************************************************************/

Functions which are unique to the 4 bit interface are named fn4...

This is the initialization sequence for a 2 by 20 character LCD programmed in 4
bit mode (chosen in order to reduce the number of I/O pins):
    delay for power up - allow LCD to reset
    command: 4 bit mode
    command: 2 lines, 5x10 character size, and 4 bit mode again
    command: turn on display and cursor - non blinking
    command: incr address and shift cursor with each character
    note: each command must be followed by a delay to allow the display to respond.

Here is the connection list:
Jackrabbit Pin LCD Signal
    PA0 J3-3     DB4
    PA1 J3-4     DB5
    PA2 J3-5     DB6
    PA3 J3-6     DB7
    PA6 J3-9     RS     Register Select: 0=command, 1=data
    PA7 J3-10    E     normally low: latches on high to low transition

DB0 - DB3 of the LCD are grounded as is the R/W pin. Since we are using 4 bit
mode DB0 - DB3 are not used by the controller. We will only be writing so we
ground the R/W pin

Since Port A is used to control the LEDs on the expansion board, they will be
controlled by the data values which you program.
VEE is used to control brightness - the simplest thing to do is ground it. If you need to control the brightness you can connect a potentiometer between ground and -5V with the arm going to the VEE pin. Check the specs on the LCD before doing this as some may require a different connection. VCC is +5 and VSS is ground.

fnMsDelay - delay some number of milliseconds
input parameter: int number of milliseconds to delay
return value: none
errors: none
*/
void fnMsDelay ( int iDelay )
{
    unsigned long ul0;
    ul0 = MS_TIMER;    // get current timer value
    while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}

fnUsDelay - delay some number of micro seconds - very approximate!!
about 11 usec per iteration with the 7.3MHz crystal. The formula used here was determined experimentally.
*/
void fnUsDelay ( int iDelay )
{
    int i;
    iDelay /= 11;
    for ( i=0; i<iDelay; i++ );
}

fn4OneNib - send a single nibble to the LCD
*/
void fn4OneNib ( char cNib )
{
    cNib &= 0x0F;        // remove upper 4 bits
    cNib |= iDataFlag;      // insert data flag for RS
    WrPortI ( PADR, NULL, cNib|0x80 ); // assert E
    WrPortI ( PADR, NULL, cNib );   // remove E
}

fn4OneNib:::
; character is in HL
ld  a, L            ; get the character
and 0x0F           ; insure upper nibble is zero
ld  hl, iDataFlag ; get address of data flag
or  (hl)           ; insert Data/Command bit
or  0x80           ; assert E
ioi  ld  (PADR),a ; ship data value and E
and 0x7F           ; remove E
ioi  ld  (PADR),a ; ship data value without E
ret

#asm
fn4Byte - send two nibbles to the LCD, upper nibble first
*/
void fn4Byte ( char cNib )
{
    int i;
    i = cNib; // temp store
    cNib >>= 4; // put upper nibble into lower 4 bits
    fn4OneNib ( cNib ); // send upper nibble
    fn4OneNib ( i ); // send lower nibble
    fnUsDelay (100);
}

fn4LCD_Init - Initialize the LCD for the following operating parameters:
- 4 bit mode, 2 lines, 5x10
- turn on display and cursor: non-blinking
- incr address and shift cursor with each character
*/
void fn4LCD_Init ()
{
    WrPortI ( PADR, NULL, 0 ); // write 0 to port A output register
    WrPortI ( SPCR, NULL, 0x84 ); // make port A output
    iDataFlag = COMMAND; // show command mode
    fnMsDelay ( 1000 ); // wait for LCD to reset itself
    fn4OneNib ( 3 ); // 4 bit mode
    fnUsDelay (100);
    fn4OneNib ( 3 ); // 4 bit mode
    fnUsDelay (100);
    fn4OneNib ( 3 ); // 4 bit mode
    fnUsDelay (100);
    fn4OneNib ( 2 ); // 4 bit mode
    fnUsDelay (100);
    fn4Byte ( '\B00001110' ); // 4 bit mode, 2 lines, 5x10
    fn4Byte ( '\B00000110' ); // 4 bit mode, 2 lines, 5x10
    fn4Byte ( '\B00101100' ); // turn on display and cursor: non-blinking
    fn4Byte ( '\B00000110' ); // incr address and shift cursor with each character
}

fn4Display - display a line of text on the LCD
argument: address of null terminated text string
*/
void fn4Display ( char *szp )
{
    iDataFlag = DATA; // next bytes are data
    while ( *szp ) fn4Byte ( *szp++ );
}

void fn4Clear ( void )
{
    iDataFlag = COMMAND; // set up for command
    fn4Byte ( 0x01 ); // clear the display
    fnMsDelay (3); // insure at least 2 msec
}

void fn4Line1 ( void )
{
    iDataFlag = COMMAND; // set up for command
    fn4Byte ( 0x80 ); // set RAM address to Line 1
    fnMsDelay (3); // note: this value may vary for different displays
                    // insure at least 2 msec
}

void fn4Line2 ( void )
{
    iDataFlag = COMMAND; // set up for command
    fn4Byte ( 0xC0 ); // set RAM address to Line 2
    fnMsDelay (3); // note: this value may vary for different displays
                    // insure at least 2 msec
}
```c
void fn4Line3 ( void )
{
    iDataFlag = COMMAND;    // set up for command
    fn4Byte ( 0x94 );     // set RAM address to Line 3
                          //note: this value may vary for different displays
    fnMsDelay ( 3);      // insures at least 2 msec
}

void fn4Line4 ( void )
{
    iDataFlag = COMMAND;    // set up for command
    fn4Byte ( 0xD4 );     // set RAM address to Line 4
                          //note: this value may vary for different displays
    fnMsDelay ( 3);      // insures at least 2 msec
}

void main()
{
    char modemMessage[6];
    fn4LCD_Init ();
                 // IMPORTANT - the initialization sequence should only
                 // be executed once after powerup. Multiple executions
                 // will cause the display to malfunction.
    initialize_modem();
    while (1)
    {
        //h = 0;
        modemMessage[0] = 'k';
        fn4Clear();
        fn4Display ("line 1");
        fnMsDelay (500);

        serD_gets(modemMessage, MAXMESSAGE,TIMEOUT);
        printf("D: %s\n",modemMessage);
        if(modemMessage[0] == '2')
        {
            printf("if: %s\n",modemMessage);
            fn4Line2 ();
            fn4Display ( "it works" );
            fnMsDelay (500); }
        fn4Line3 ();
        fn4Display ( "This is line 3!" );
        fnMsDelay (500);
        fn4Line4 ();
        fn4Display ( "This is line 4!" );
        fnMsDelay (500);
    }
}
```
#define UPDATE   1
#define MAXMESSAGE2  4
#define TIMEOUT   2000
#define D_BAUDRATE 9600
#define BUT1 0000
#define BUT2 0001

/* START FUNCTION DESCRIPTION ***************************************************/
serUniv_gets <RFMODEM.LIB>

SYNTAX: int serD_gets(char *string,int max,unsigned long timeout);

DESCRIPTION: Reads characters from the getc function that is passed in via a
pointer until a carriage
return is read, max number characters are read, or until
tmout milliseconds transpires between characters after the
first one is read.

PARAMETER1: string: Character array into which a null terminated string
is read.
PARAMETER2: max : The maximum number of characters to read into s, not
including the null terminator.
PARAMETER3: timeout: Millisecond wait period to allow between characters
before timing out.

RETURN VALUE: 0 - timed out before CR or max characters read, 1 - no timeout.

END DESCRIPTION **************************************************************/

nodebug root int serC_gets(char *string,int max,unsigned long timeout)
{
    int index, ch;
    long timeStart;

    index = 0;
    timeStart = MS_TIMER;
    while(max > index && timeout > (MS_TIMER - timeStart))
    {
        ch = serCgetc();
        if(ch == (int)'')
        {
            string[index] = '\0';
            return 1;
        }
        else if(ch != -1)
        {
            string[index] = (char)ch;
            index++;
        }
    }
    return 0;
}

void main()
{
    int key;
    char keyMessages[MAXMESSAGE2];
    char keyl[MAXMESSAGE2];
    serCopen(D_BAUDRATE);
    serCwrFlush();
    serCrdFlush();
key = -1;
while(1){
    key1[0] = '\\';
    key1[1] = 'x';
    key1[2] = '+';
    key1[3] = '0';
    costate
    {
        waitfordone{cof_serCgets(keyMessages, MAXMESSAGE2, TIMEOUT)};
    }
    //serC_gets(keyMessages, MAXMESSAGE2, TIMEOUT);
    switch (keyMessages)
    {
    case BUT1: key = 1 ;
            break;
    case BUT2: key = 2;
            break;
    default: key = -1;
    }
    if(keyMessages[0]=='0')
    {
        key = 1;
    }
    printf("key pressed: %d\n",key);
}
Test Code for the 7-Segment Display

/*
*********************************************************************************************
************
* Module     : MAX7219.C
* Author     : Randy Rasa
* Description: MAX7219 LED Display Driver Routines
* The Maxim MAX7219 is an LED display driver that can control up to 64 individual LEDs, or eight 7-segment LED digits, or any combination of individual LEDs and digits. It frees the host from the chore of constantly multiplexing the 8 rows and 8 columns. In addition, it takes care of brightness control (16 steps), and implements display test and display blank (shutdown) features.
* The host communicates with the MAX7219 using three signals: DATA, CLK, and LOAD. This modules bit-bangs them, but Motorola's SPI interface (or similar interface from other manufacturers) may also be used to simplify and speed up the data transfer.
* DATA _________|D15|D14|D13|D12|D11|D10|D09|D08|D07|D06|D05|D04|D03|D02|D01|D00|______
* CLK          |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|
* LOAD ______|                                                                  |__|
* Implementation Notes:
* 1. This module was written and tested using an Atmel AT89C2051 microcontroller, with the MAX7219 connected to I/O pins P3.3 (LOAD), P3.4 (CLK), and P3.3 (DATA).
* 2. Macros are provided to simplify control of the DATA, CLK, and LOAD signals. You may also use memory-mapped output ports such as a 74HC374, but you'll need to replace the macros with functions, and use a shadow register to store the state of all the output bits.
* 3. This module was tested with the evaluation version of Hi-Tech C-51, using the small memory model. It should be portable to most other compilers, with minor modifications.
* 4. The MAX7219 is configured for "no decode" mode, rather than "code B" decoding. This allows the program to display more than the 0-9,H,E,L,P that code B provides. However, the "no decode" method requires that each character to be displayed have a corresponding entry in a lookup table, to convert the ascii character to the proper 7-segment code. MAX7219_LookupCode() provides this function, using the MAX7219_Font[] array. If you need to display more than the characters I have provided, simply add them to the table ... *
* *********************************************************************************************
************
*/

/*
*********************************************************************************************
************
* Include Header Files
* #include <8051.h>
* #include <stdlib.h>
*/

#include <8051.h>                                    // microcontroller header file
// include "max7219.h" // MAX7219 header file

/#define REG_DECODE 0x09 // "decode mode" register
/#define REG_INTENSITY 0x0a // "intensity" register
/#define REG_SCAN_LIMIT 0x0b // "scan limit" register
/#define REG_SHUTDOWN 0x0c // "shutdown" register
/#define REG_DISPLAY_TEST 0x0f // "display test" register
/#define INTENSITY_MIN 0x00 // minimum display intensity
/#define INTENSITY_MAX 0x0f // maximum display intensity

/#define DATA_PORT P3 // assume "DATA" is on P3.5
/#define DATA_DDR P3
/#define DATA_BIT 0x20
/#define DATA_0() (DATA_PORT &= ~DATA_BIT)
/#define DATA_1() (DATA_PORT |= DATA_BIT)
/#define CLK_PORT P3 // assume "CLK" is on P3.4
/#define CLK_DDR P3
/#define CLK_BIT 0x10
/#define CLK_0() (CLK_PORT &= ~CLK_BIT)
/#define CLK_1() (CLK_PORT |= CLK_BIT)
/#define LOAD_PORT P3 // assume "LOAD" is on P3.3
/#define LOAD_DDR P3
/#define LOAD_BIT 0x08
/#define LOAD_0() (LOAD_PORT &= ~LOAD_BIT)
/#define LOAD_1() (LOAD_PORT |= LOAD_BIT)

/static void MAX7219_Write (unsigned char reg_number, unsigned char data);
/static void MAX7219_SendByte (unsigned char data);
/static unsigned char MAX7219_LookupCode (char character);
/void MAX7219_ShutdownStart ();
void MAX7219_ShutdownStop ();
void MAX7219_DisplayTestStart ();
void MAX7219_DisplayTestStop ();
void MAX7219_SetBrightness (char brightness);
void MAX7219_Clear ();

// .................................................. Public Functions

(/^
**
* MAX7219_Init()
*
* Description: Initialize MAX7219 module; must be called before any other MAX7219 functions.
* Arguments  : none  
* Returns     : none
**/
void MAX7219_Init ()
{
    //DATA_DDR |= DATA_BIT;                               // configure "DATA" as output
    //CLK_DDR  |= CLK_BIT;                                // configure "CLK"  as output
    //LOAD_DDR |= LOAD_BIT;                               // configure "LOAD" as output
    MAX7219_Write(REG_SCAN_LIMIT, 0x00);                   // set up to scan all eight digits
    MAX7219_Write(REG_DECODE, 0x00);                    // set to "no decode" for all digits
    MAX7219_ShutdownStop();                             // select normal operation (i.e. not
    MAX7219_DisplayTestStop();                          // select normal operation (i.e. not
    MAX7219_Clear();                                    // clear all digits
    MAX7219_SetBrightness(INTENSITY_MAX);               // set to maximum intensity
}

(/^
**
* MAX7219_ShutdownStart()
*
* Description: Shut down the display.
* Arguments  : none  
* Returns     : none
**/
void MAX7219_ShutdownStart ()
{
    MAX7219_Write(REG_SHUTDOWN, 0);                     // put MAX7219 into "shutdown" mode
}

(/^
**
* MAX7219_ShutdownStop()
*
* Description: Take the display out of shutdown mode.
* Arguments  : none  
* Returns     : none
**/
}
void MAX7219_ShutdownStop ()
{
    MAX7219_Write(REG_SHUTDOWN, 1); // put MAX7219 into "normal" mode
}

void MAX7219_DisplayTestStart ()
{
    MAX7219_Write(REG_DISPLAY_TEST, 1); // put MAX7219 into "display test" mode
}

void MAX7219_DisplayTestStop ()
{
    MAX7219_Write(REG_DISPLAY_TEST, 0); // put MAX7219 into "normal" mode
}

void MAX7219_SetBrightness (char brightness)
{
    brightness &= 0x0f; // mask off extra bits
    MAX7219_Write(REG_INTENSITY, brightness); // set brightness
}

void MAX7219_Clear ()
Description: Clear the display (all digits blank)
Arguments : none
Returns   : none
******************************************************************************
************
*/
void MAX7219_Clear ()
{
  char i;
  for (i=0; i < 8; i++)
    MAX7219_Write(i, 0x00);  // turn all segments off
}

************
MAX7219_DisplayChar()
* Description: Display a character on the specified digit.
* Arguments : digit = digit number (0-7)
*             character = character to display (0-9, A-Z)
* Returns   : none
******************************************************************************
************
void MAX7219_DisplayChar (char digit, char character)
{
  MAX7219_Write(digit, MAX7219_LookupCode(character));
}

// ..................................... Private Functions
..............................................

************
LED Segments:
  a  
  ----
  f|   |b
  |   |g |
  ----
  e|   |c
  |   |
  ----  o dp
  d

Register bits:
  bit:  7  6  5  4  3  2  1  0
  dp a b c d e f g
******************************************************************************
************
*/
static const struct {
  char ascii;
  char segs;
} MAX7219_Font[] = {
  { ' ', 0x00},
  { '0', 0x7e},
  { '1', 0x6d},
  { '2', 0x6e},
  { '3', 0x99},
  { '4', 0x33},
  { '5', 0x5f},
  { '6', 0x70},
  { '7', 0x6f},
  { '8', 0x00},
  { '9', 0x7e},
  { 'A', 0x33},
  { 'B', 0x6d},
  { 'C', 0x99},
  { 'D', 0x6e},
  { 'E', 0x5f},
  { 'F', 0x70},
  { 'G', 0x6f},
  { 'H', 0x00},
  { 'I', 0x33},
  { 'J', 0x6d},
  { 'K', 0x99},
  { 'L', 0x6e},
  { 'M', 0x5f},
  { 'N', 0x70},
  { 'O', 0x6f},
  { 'P', 0x00},
  { 'Q', 0x33},
  { 'R', 0x6d},
  { 'S', 0x99},
  { 'T', 0x6e},
  { 'U', 0x5f},
  { 'V', 0x70},
  { 'W', 0x6f},
  { 'X', 0x00},
  { 'Y', 0x33},
  { 'Z', 0x6d},
};
static unsigned char MAX7219_LookupCode (char character)
{ char i;
    for (i = 0; MAX7219_Font[i].ascii; i++)             // scan font table for ascii code
        if (character == MAX7219_Font[i].ascii)
            return MAX7219_Font[i].segs;                    // return segments code
    return 0;                                           // code not found, return null (blank)
}

static void MAX7219_Write (unsigned char reg_number, unsigned char dataout)
{ // take LOAD high to begin
    BitWrPortI(PBDR, &PBDRShadow, 1, 7);
    BitWrPortI(PBDR, &PBDRShadow, 0, 7);
    MAX7219_SendByte(reg_number);                       // write register number to MAX7219
    MAX7219_SendByte(dataout);                          // write data to MAX7219
    BitWrPortI(PBDR, &PBDRShadow, 1, 7);
    BitWrPortI(PBDR, &PBDRShadow, 0, 7);
    BitWrPortI(PBDR, &PBDRShadow, 1, 7);
    BitWrPortI(PBDR, &PBDRShadow, 0, 5);
    // take LOAD low to latch in data
    // take LOAD high to end
}
/*********************************************************************************************/
************
* MAX7219_SendByte()
* Description: Send one byte to the MAX7219
* Arguments  : dataout = data to send
* Returns    : none
*********************************************************************************************/

static void MAX7219_SendByte (unsigned char dataout)
{
char i;
unsigned char mask;
for (i=8; i>0; i--) {
    mask = 1 << (i - 1);                // calculate bitmask
    //CLK_0();                                          // bring CLK low
    BitWrPortI(PADR, &PADRShadow, 0, 5);
    printf(“%s”, ”test”);
    if (dataout & mask)                  // output one data bit
        BitWrPortI(PBDR, &PBDRShadow, 1, 6);
        //DATA_1();                              //  "1"
    } else                                   //  or
    { BitWrPortI(PBDR, &PBDRShadow, 0, 6);
        //DATA_0();                              //  "0"
    } BitWrPortI(PADR, &PADRShadow, 1, 5);
    printf(“%s”, ”test2”);
    //CLK_1();                                          // bring CLK high
    }

void main()
{
    int i;
    WrPortI ( PADR, NULL, 0 );  // write 0 to port A output register
    WrPortI ( SPCR, NULL, 0x84 ); // make port A output
    MAX7219_Init ();
    //MAX7219_DisplayTestStart ();
    //while(1){BitWrPortI(PBDR, &PBDRShadow, 0, 4);} //MAX7219_DisplayChar (0x01, '5'); //
    while(1) 
    { //BitWrPortI(PADR, &PADRShadow, 0, 5);
        BitWrPortI(PADR, &PADRShadow, 1, 5);
        //costate{
        MAX7219_DisplayChar (0x01, '5');
        for(i=0;i<100;i++){//stuff
            //MAX7219_DisplayTestStart ();
            
        }
        MAX7219_DisplayChar (0x01, '8');
        //waitfor(DelaySec(5));
        //MAX7219_DisplayChar (1, 6);
        //waitfor(DelaySec(5));
        
    /i=0;
    }
    }