ME 477 Embedded Computing
Laboratory Experiment #3
Low-Level Character I/O

Objectives

In this exercise you will gain experience with:

1. The keypad and LCD display.
2. Code requirements for character I/O of a custom embedded computing application.
3. On-line debugging techniques.

Introduction

In this lab you will write the lowest-level routines for character I/O for our keypad and LCD display. They include the `putchar_lcd()` function, and the `getkey()` function called from `getchar_keypad()` in lab #2.

Pre-Laboratory Preparation

Part #1: Character Output—The function `putchar_lcd()` puts a single character on the LCD display. The character may be any in the ASCII code or any of the escape sequences described in Lab #1 (\f, \v, \n, \b). The prototype of the `putchar_lcd()` function is

```
int  putchar_lcd(int value);
```

where the input argument (`value`) is the character to be sent to the display. If the input value is in the range [0, 255] then the returned value is also equal to the input value. If the input value is outside that range then an error is indicated by returning `EOF`.

Your version of `putchar_lcd()` will replace that in the `me477LIbrary`. Calls to `putchar_lcd()` might be

`ch = putchar_lcd('m');` or `putchar_lcd('\n');`

Serial data is sent to the LCD display through a Universal Asynchronous Receiver/Transmitter (UART). Write the `putchar_lcd()` to perform four functions:

1. Initialize the UART the *first* time that `putchar_lcd()` is called.
2. Send a character to the display or send a decimal code to the display to implement an escape sequence.
3. Check for the success of the UART write.
4. Return the `EOF` error code, if appropriate. Otherwise, return the character to the calling program.

Background: The UART must be initialized **once** before any data is passed to the display. It is initialized through the `Uart_Open()` function that sets appropriate myRIO control registers to define the operation of the UART. To communicate properly with our lcd display the UART must be initialized as follows:

```
uart.name = "ASRL2::INSTR"; // UART on Connector B
uart.defaultRM = 0;         // def. resource manager
uart.session = 0;           // session referencet
status = Uart_Open( &uart,  // port information
                    19200,  // baud rate
                    8,      // no. of data bits
                    Uart_StopBits1_0, // 1 stop bit
                    Uart_ParityNone); // No parity
```

where `uart` (type: static `MyRio_Uart`) is a port information structure, and the returned value is `status` (type: `NiFpga_Status`). The macros `Uart_StopBits1_0` and `Uart_ParityNone` are defined in `UART.h`. You must `#include UART.h` in your code.

Perform this UART initialization just once, and immediately return `EOF` from `putchar_lcd()` if `status` is less than the `VI_SUCCESS` macro.

Escape sequences, received as the argument of `putchar_lcd()`, control the cursor position and the function of the LCD display. They are implemented by sending specialized codes according to the following table:

Function	*Escape* Sequence	*Decimal Codes*
Backlight & Clear LCD	`\f`	`12, 17`
Cursor left, 1 space	`\b`	`8`
Cursor to Start line-0	`\v`	`128`
Cursor to Start next line	`\n`	`13`

Arguments of `putchar_lcd()`, in the range of 0 to 127, are sent to the display where they are interpreted as the corresponding ASCII characters. Other arguments, in the range 128 to 255 are used for special control functions of this display.

Both escape sequences and ASCII characters are sent to the display using the `Uart_Write()` function. A typical call would be:

```
status = Uart_Write( &uart,  // port information
                     writeS, // data array
                     nData); // no. of data codes
```

where `uart` is the port information structure defined during the initialization, `writeS` (type: `uint8_t`) is an array containing the data to be written, and `nData` (type: `size_t`) indicates the number of elements in `writeS`.

Again, return `EOF` if `status` is less than the `VI_SUCCESS`. Under normal operation (no errors), return the input character to the calling program.

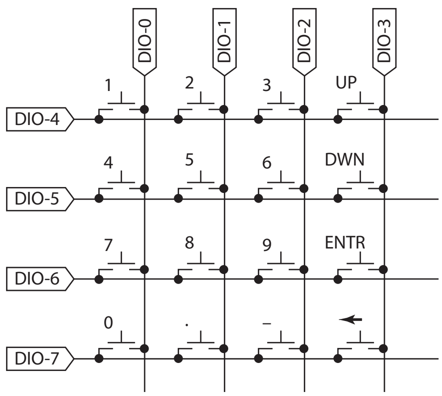
Part #2: Keypad Input—You will write the `getkey()` function waits for a key to be depressed on the keypad, and returns the character code corresponding to that key. The prototype of the `getkey()` function is

```
char  getkey(void);
```

Your version of `getkey()` will replace that in the C library. A call to `getkey()` might be:

```
key = getkey();
```

The keypad is a matrix of switches. When pressed, each switch uniquely connects a row conductor to a column conductor. The row and column conductors are connected to eight digital I/O channels of connector-B (`DIO-0 - DIO-7`)—of the myRiO as shown in the figure.



Each channel may be programmed to operate as either a digital input or an output. As an output, the channel operates with low output impedance as it asserts either a high or a low voltage at its terminal. Programmed as an input, the channel has high input impedance ("Hi-Z mode") as it detects either a high or a low voltage.

How will we detect if a key is depressed? Briefly, this is accomplished by driving one column to low voltage (false), with the other columns channels in Hi-Z mode. Then, all of the rows are scanned (detected). If a row is found to be low, the key connecting that row to the driven column must be depressed. This procedure is repeated for each column. The entire process is repeated until a key is found.

Essential to this scheme is that a 10K "pull-up" resistor is connected between each channel and the high voltage. So, unless a row is connected (through a key) to a low-impedance, low-voltage column, it will always read high.

Strategy—Here is one strategy for `getkey()`:

```
- initialize the 8 digital channels
- while a low bit has not been detected {
    - for each column  {
       - set all columns to Hi-Z
       - write ith column low
           - for each row  {
              - read the jth row
              - if the bit is low {
                 - break out of the row loop
                }
             }
           - if the bit is low {
                 - break out of the column loop, too }
           - wait for xxx ms
      }
  }
- wait for the jth row to go back up
- look up the (i,j) key  & return the key code
```

Background

Channel Initialization

The `MyRio_Dio` structure, defined in DIO.h, identifies the control registers and the bit to read or write for a channel.

```
typedef struct { uint32_t dir;   // direction register
                 uint32_t out;   // output value register
                 uint32_t in;    // input value register
                 uint8_t bit;    // Bit to modify
} MyRio_Dio;
```

Declare an array of `MyRio_Dio` structures, one element for each of the 8 necessary channels. In a loop initialize the channels as follows:

```
MyRio_Dio Ch[8];

for (i=0; i<8; i++) {
    Ch[i].dir = DIOB_70DIR;
    Ch[i].out = DIOB_70OUT;
    Ch[i].in = DIOB_70IN;
    Ch[i].bit = i;
}
```

Again, the symbols shown are defined in DIO.h.

Channel I/O

Input—Digital channel read function prototype:

```
NiFpga_Bool Dio_ReadBit(MyRio_Dio* channel);
```

For example, a typical call might be:

```
bit = Dio_ReadBit(&Ch[row+4]));
```

Note: `In addition to reading the bit, Dio_ReadBit()` sets the channel to Hi-Z mode.

Output—Digital channel write function prototype:

```
void Dio_WriteBit(MyRio_Dio* channel, NiFpga_Bool value);
```

For example, a typical call might be:

```
Dio_WriteBit(&Ch[col], NiFpga_False);
```

The data type `NiFpga_Bool` may take values of either `NiFpga_True` (high), or `NiFpga_False` (low)

Key Code

The key code returned by `getkey()` is determined by the indices of a key code table. The key code table can be stored in a statically declared 4×4 array of characters.

```
char table[4][4] = { {'1','2','3', UP},
                     {'4','5','6', DN},
                     {'7','8','9',ENT},
                     {'0','.','-',DEL}  };
```

For example, if the detected row was 1, and the column was 2, then the value of `table[1][2]` is the character `'6'`.

The symbols `UP, DN, ENT, DEL` are defined in me477.h

Wait

The xxx ms time delay will be determined by executing a delay-interval routine. The "wait" function below is suggested. It executes in a small fraction of a second. In next week's lab we will calculate and measure its precise duration.

```
/*---------------------------------------
Function wait
      Purpose:      waits for xxx  ms.
      Parameters:   none
      Returns:      none
*----------------------------------------*/
void    wait(void) {
        uint32_t i;

        i = 417000;
        while(i>0){
                i--;
        }
        return;
}
```

Main Function—Write a main function that tests your versions of `putchar_lcd()` and `getkey()` it should:

1. Make at least one individual call to each of `putchar_lcd()` and `getkey()`. Be sure to test the value-out-of-range error returned by `putchar_lcd()`.
2. Collect an entire string using `fgets_keypad()` (which automatically calls `getkey()`).
3. Write an entire string using `printf_lcd()` (which automatically calls `putchar_lcd()`). Be sure to test all four escape sequences. (\f, \v, \n, \b)

Laboratory Procedure

Test and debug your program.