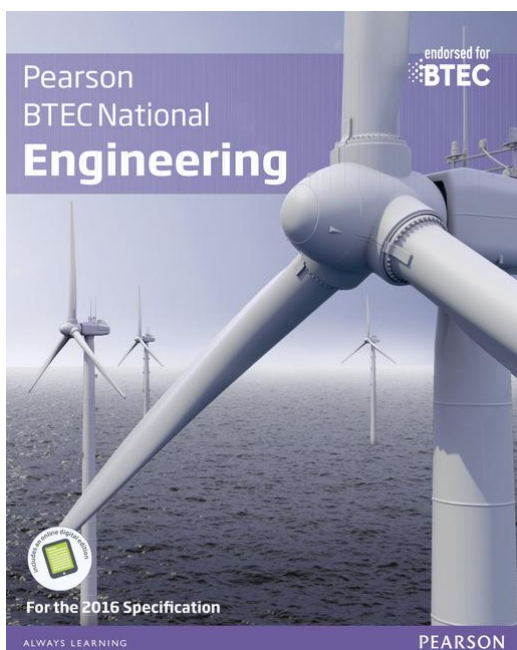


BTEC Level 3 Nationals in Engineering: Unit 6

Your free sample of the student
book: preparation for
assessment

(BTEC National Engineering: Student Book 1 (with ActiveBook),
ISBN: 9781292141008)



Getting ready for assessment

This section has been written to help you to do your best when you take the external practical assessment. Read through it carefully and use it to help you create, develop and build the example projects. Ask your tutor if there is anything you are not sure about.

About the test

The test booklet will contain material for completion of the set task under supervised conditions. You will not receive the booklet until the assessment period. You will complete all the work for your assessment on a computer, using appropriate hardware and software as listed in the unit content. You will use an electronic task book, which will be provided. You will not have access to the internet during the supervised periods.

All your work, a task book and one audio-visual recording must be submitted for assessment on a compact disc (CD). Your centre will provide you with access to suitable audio-visual recording equipment and software to ensure that the footage is recorded in an appropriate file format.

Make sure that your work is backed up securely and is kept until the end of the post-result service period. You must complete your work independently, and your tutor will authenticate it before it is submitted for assessment. Your centre will arrange the supervised assessment.

You must work independently throughout the supervised assessment period, and you should not share your work with other learners. Your tutor may clarify the wording that appears in this task but cannot provide any guidance on how to complete the task.

As the guidelines for assessment can change, you should refer to the official assessment guidance on the Pearson Qualifications website for the latest definitive guidance.

The assessment will be in the form of:

- a scenario
- a client brief
- planning activity
- analysis of brief
- system design
- assembly and programming
- system testing and analysis.

Preparing for the test

This unit is assessed under supervised conditions. Your preparation should include:

- understanding the format of the specification
- knowing how to locate datasheets and conduct other research
- completing the example assessment tasks below
- constructing test schedules
- analysing test schedules
- practice in circuit construction and testing
- construction and analysis of commented code
- review of previous tests, including:
 - booklet
 - task
 - marked solutions.

Hints and tips for the supervised assessment

- Make sure you fully understand the client requirements.
- Planning is vital, so review your proposed solution and plan it thoroughly. Include milestones and allow time at the end to provide the all-important analysis.
- Focus on the scenario and client brief, ensuring that you achieve the minimum requirements first. There will be time for enhancements later.

Sample solutions

We will now look at three worked examples, where solutions are developed for analogue-to-digital conversion with output to LEDs and LCD, a simple I²C and a keypad interface.

Pseudocode is provided for the first example, circuit schematics for the first two examples, and C code for all three examples. The rest of the development process is left to you.

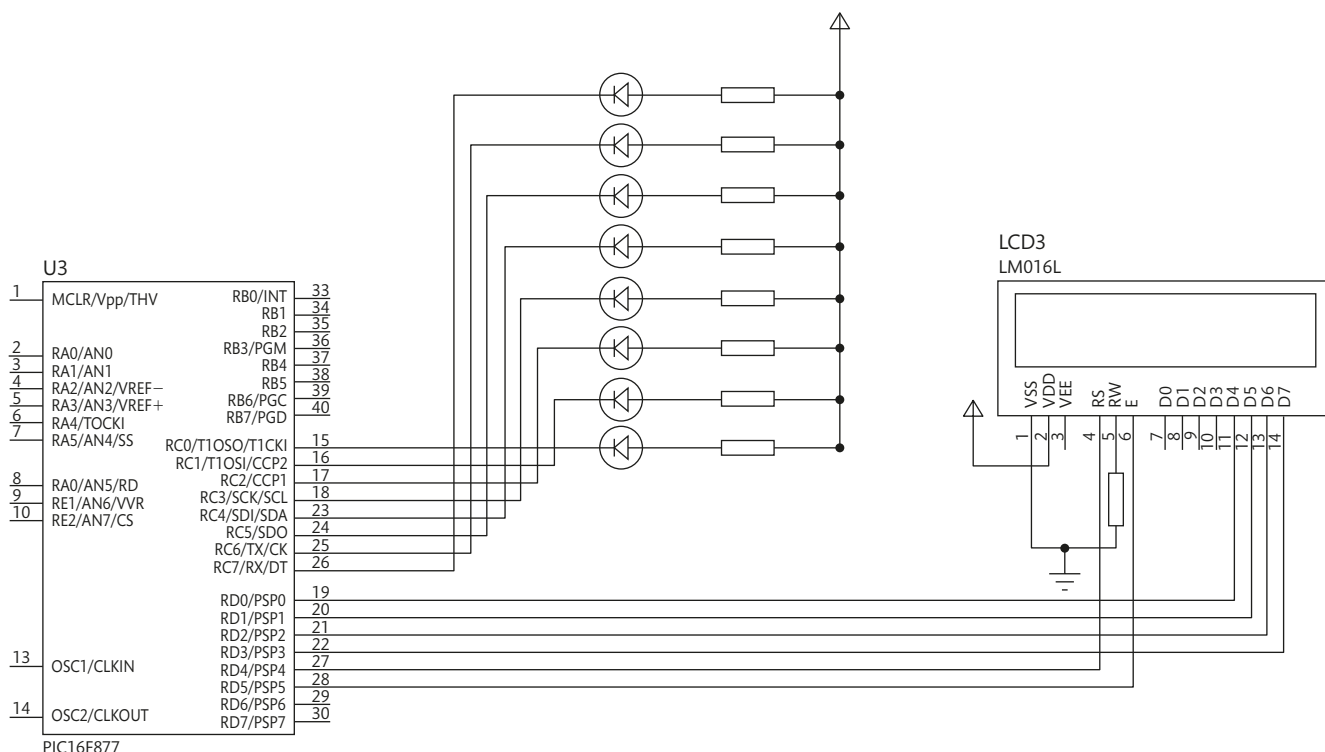
You should consider why the particular inputs and outputs have been chosen and how they meet the requirements of the project. Record your reasons and consider how you can put together a detailed justification.

Create a log of how you develop each solution, breaking it down into stages and recording all testing that you carry out. Many learners forget that every time you download a program and run it to check that it functions as expected, you are testing. It is not just the end tests that count.

Worked example A

ADC input with LED/LCD output

- 1 Build the circuit, referring to the diagrams in **Figures 6.18** and **6.19**.



► **Figure 6.18** Schematic of a microcontroller with both LCD and LED outputs


```

ELSE
    IF value is > 300 light bit Rx.0, Rx.1, Rx.2 connected to LEDs
    ELSE
        IF value is > 400 light bit Rx.0, Rx.1,Rx.2,Rx.3
        ELSE
            IF value is > 500 light bit Rx.0 to Rx.4
            ELSE
                IF value is > 600 light bit Rx.0 to Rx.5
                ELSE
                    IF value is > 700 light bit Rx.0 to Rx.6
                    ELSE
                        IF value is > 800 light bit Rx.0 to Rx.7
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF
        ENDIF
    ENDIF
ENDIF
END

```

```

BEGIN ADC value output to LCD
SET LCD
CALCULATE ADC output and convert to ASCII for output as a character
IF ADC >= 700
    OUTPUT ADC value plus IT IS NIGHT TIME
    IF ADC >= 500
        OUTPUT ADC value plus IT IS MIDDAY
        IF ADC >= 300
            OUTPUT ADC value plus IT IS MORNING
        ENDIF
    ENDIF
ENDIF
END

```

- 3** Study the following code. It is quite complex and covers many aspects. Try to step through the code and notice the links and jumps from one function to another. The code has been annotated as much as possible to help you understand its structure. Many of the functions have been created in line with the PIC16F877 user manual.

```

// Title: Analogue-to-digital conversion
// Description: ADC with an output on an array of LEDs and on an LCD
// Author: Alan Serplus / Paul H Stewart
// Date: 09/02/xxxx
// Versions 1.3a

// Include header files for Micro
#include <pic16f877.h>
#include <htc.h>
// end includes

// define constants
const unsigned char channel_0 = 0b10000001; // set the binary value for each channel to a name - there
// are 8 possible channels
const unsigned char channel_1 = 0b10001001; // each channel is listed in the manual and include only
// those needed in the program

// ADC mode select, i.e. number of channels, assign reference voltages etc.
const unsigned char mode_0 = 0b10000000; // set the ADC mode - there are 16 in total
const unsigned char mode_1 = 0b10000001; // include only those needed in the program
// end define constants

// defines
#define LCD_port PORTD // set port D and name within code as LCD_port
#define LCD_TRIS TRISD // rename TRISD as LCD_TRIS for ease of reference
#define LCD_RS RD5 // name pin RD5 as LCD_RS, i.e. LCD register select line
#define LCD_E RD4 // name pin RD4 as LCD_E, i.e. LCD enable signal
// end defines

// define constants
const char line1 = 0x80; // LCD constants - when line1 is used the LCD will be set to line 1
const char line2 = 0xC0; // LCD constants
const char line3 = 0x94; // LCD constants
const char line4 = 0xD4; // LCD constants
const char LCD_enable = 0x10; // set enable line
// end constants

// declare function prototypes
void output_bar(int ADC_result);
void display_result(int ADC_result);
void config_ADC(unsigned char ADC_mode); // set the ADC mode using constants above
int start_ADC(unsigned char channel_num); // associated with ADC routines
void init_LCD(void);
void clock_inst(void);
void LCD_function(unsigned char function);
void output_char_string(unsigned char line_num, const char output_string[ ]);
void output_char(unsigned char character);
void clock_LCD(void);
void delay_ms(unsigned int length); // millisecond delay routine
void delay_100us(unsigned int length); // 100 microsecond delay routine
// end function declares

void main()
{
    int ADC_result; // declare memory called ADC_result
    TRISC = 0; // set port C as an output
    PORTC = 0x00; // clear port C
    // LCD CODE
    init_LCD(); // initialise LCD routine
    LCD_function(0x0C); // function to turn LCD on
    // ADC code

```

```

config_ADC(mode_0); // first select mode of operation
                    // mode_0 - configure RE2, RE1, RE0, RA5, RA3, RA2, RA1, RA0 as analogue inputs
while (1)
{
    ADC_result = start_ADC(channel_0); // call conversion routine - i/p signal on channel 0 (RA0)
                                        // pass resulting returned int into variable ADC_result
    output_bar(ADC_result);             // call function to display ADC result sending
                                        // parameter ADC_result
    display_result(ADC_result);
}
}

void config_ADC(unsigned char ADC_mode)
{
    ADCON1 = ADC_mode; // select mode of operation
}

int start_ADC(unsigned char channel_num)
{
    unsigned int ADC_result; // declare variable that will be returned
    ADCON0 = channel_num;   // select channel number and turn ADC on
    ADON = 0x01;           // start conversion
    while (ADON){};       // wait for end of conversion
    ADC_result = ADRESH*256 + ADRESL; // convert result to a 16-bit number:
                                        // ADC_result = (upper byte x 256) + lower byte
    return ADC_result;     // return 10-bit result (0 to 1023) in 16-bit format (ADC_result)
}

// BINARY OUTPUT

void output_bar(int ADC_result) // function output_bar takes parameter ADC_result
{
    unsigned char bar;
    bar = 0b00000000; // reset bar height to 0

    if(ADC_result>100) // test if input light level greater than 100
    {
        bar = 0b11111110; // YES - set bar height (or 0xFE)
    }
    if(ADC_result>200) // test if input light level greater than 200
    {
        bar = 0b11111100; // YES - set new bar height
    }
    if(ADC_result>300) // test if input light level greater than 300
    {
        bar = 0b11111000; // YES - set new bar height
    }
    if(ADC_result>400) // test if input light level greater than 400
    {
        bar = 0b11110000; // YES - set new bar height
    }
    if(ADC_result>500) // test if input light level greater than 500
    {
        bar = 0b11100000; // YES - set new bar height
    }
    if(ADC_result>600) // test if input light level greater than 600

```

```

    {
        bar = 0b11000000;
    }
    if(ADC_result>700)           // test if input light level greater than 700
    {
        bar = 0b10000000;       // YES - set new bar height
    }
    if(ADC_result>800)           // test if input light level greater than 800
    {
        bar = 0b00000000;       // YES - set maximum bar height
    }
    PORTC = bar;                 // regardless of value output bar value to port B
}

/*****
 * Function to initialise LCD. Consider further research on methods
 *****/
void init_LCD()
{
    LCD_TRIS = 0;                // configure LCD port as an output port
    LCD_port = 0;                // clear port lines

    delay_ms(20);                // 20ms delay

    LCD_port = (0x03|LCD_enable); // #1 control sequence
    clock_inst();

    delay_ms(5);                 // delay 5ms

    LCD_port = (0x03|LCD_enable); // #2 control sequence
    clock_inst();

    delay_ms(1);                 // delay 1ms

    LCD_port = (0x03|LCD_enable); // #3 control sequence
    clock_inst();

    LCD_port = (0x02|LCD_enable); // #4 control sequence
    clock_inst();

    LCD_port = (0x02|LCD_enable); // #5 control sequence
    clock_inst();

    LCD_port = (0x08|LCD_enable); // #6 control sequence
    clock_inst();

/*****
 * function set command " 0 0 0 0 1 D C B "
 * D = 1 LCD on
 * C = 1 cursor on
 * B = 1 blink on
 *****/
    LCD_port = (0x00|LCD_enable); // #7 control sequence - switch LCD off
    clock_inst();
    LCD_port = (0x08|LCD_enable); // #8 control sequence

/*****
 * Clear display - Control word = 0x01
 *****/
    LCD_port = (0x00|LCD_enable); // #9 control sequence - clear display
    clock_inst();

```



```

    LCD_port = (0x01|LCD_enable);           // #10 control sequence
    clock_inst();

/*****
 * Entry mode setting
 * Entry mode command " 0 0 0 0 0 1 ID S "
 * if ID = 0 no increment (during write and read operation)
 * if S = 0 no shift
 *****/
    LCD_port = (0x00|LCD_enable);           // #11 control sequence
    clock_inst();
    LCD_port = (0x06|LCD_enable);           // #12 control sequence
    clock_inst();
}

// end of initialisation

void clock_inst()
{
    LCD_E = 0;
    delay_ms(2);
    LCD_E = 1;
    delay_ms(2);
}

/*****
 * Outputs a control character to an LCD - high nibble first *
 *****/
void LCD_function(unsigned char function)
{
    LCD_port = ((function>>4)|LCD_enable); // shift high nibble to lower nibble and set RS=0 and LCD_E=1
    clock_inst();                          // clock high nibble of control character

    LCD_port = ((function&0x0F)|LCD_enable); // mask high nibble and set RS=0 and LCD_E=1
    clock_inst();                          // clock low nibble of control character
}

// LCD OUTPUT CODE
void display_result(int ADC_result)
{
    int remainder;
    unsigned char thous, hunds, tens, units; // declare multiple variables
    thous = (ADC_result/1000) + 0x30;       // calculate number of thousands and convert to ascii (add
                                            // 0x30)
    remainder = ADC_result%1000;           // calculate remainder
    hunds = (remainder/100) + 0x30;         // calculate number of hundreds and convert to ascii (add
                                            // 0x30)
    remainder = ADC_result%100;             // calculate remainder
    tens = (remainder/10) + 0x30;          // calculate number of tens and convert to ascii (add 0x30)
    units = ADC_result%10 + 0x30;          // calculate number of units and convert to ascii (add 0x30)

    if(hunds == 0x37)
    {
        output_char_string(line2,"IT'S NIGHT TIME "); // LCD will display if ADC_result is 700 to 799
    }
    if(hunds == 0x35)
    {
        output_char_string(line2,"IT'S AFTERNOON "); // LCD will display if ADC_result is 500 to 599
    }
    if(hunds == 0x33)
    {
        output_char_string(line2,"IT'S MORNING "); // LCD will display if ADC_result is 300 to 399
    }
}

```

```

    }
}

/*****
 * Outputs a character string to specified LCD line number *
 *****/
void output_char_string(unsigned char line_num, const char output_string[])
{
    unsigned char i, character;    // declare to character-sized memory locations named i and character

    LCD_function(line_num);        // send cursor to start of specified line number

    for(i=0;output_string[i] != 0;i++)    // output character string until string terminator
    {
        character = output_string[i];
        output_char(character);
    }
}

/*****
 * Outputs a character to an LCD - high nibble first as in 4-bit mode *
 *****/
void output_char(unsigned char character)
{
    LCD_port = ((character>>4)|0x30);    // get character from character string
                                           // output high nibble first with RS=1 and LCD_E=1
    clock_LCD();                          // clock high nibble of character

    LCD_port = ((character&0x0F)|0x30);    // get character from character string
                                           // output low nibble with RS=1 and LCD_E=1
    clock_LCD();                          // clock low nibble of character
}

void clock_LCD()
{
    LCD_E = 0;                            // falling edge clocks data into LCD
    delay_100us(1);                       // 0.1ms delay
}

/*****
 * Generic delay routine using Timer 0 clocked at 250kHz, i.e. a period of 4 microseconds *
 * - number of loops specified by length *
 * - each loop waits until Timer 0 = 250, i.e. a delay of 1ms *
 *****/
void delay_ms(unsigned int length)
{
    unsigned int i;
    OPTION_REG = 0x01;                    // prescaler = /4 i.e. TMR0 clocked at 250kHz
    for (i=0;i<length;i++)
    {
        TMR0 = 0;                          // generates a 1ms delay
        while (TMR0<250) {};
    }
}

/*****
 * Generic delay routine using Timer 0 clocked at 250kHz, i.e. a period of 4 microseconds *
 * - number of loops specified by length *
 * - each loop waits until Timer 0 = 25, i.e. a delay of 0.1ms *
 *****/

```

```

void delay_100us(unsigned int length)
{
    unsigned int i;
    OPTION_REG = 0x01;           // prescaler = /4 i.e. TMR0 clocked at 250kHz
    for (i=0;i<length;i++)
    {
        TMR0 = 0;                // generates a 0.1ms delay
        while (TMR0<25){};
    }
}

```

4 Collate the documentation, including images, circuit diagrams and analysis.



PAUSE POINT

It may take you several sessions to get to grips with coding in C. Once you have achieved this, you should be able to see multiple applications for this type of system. Revisit the code and attempt to simplify or shorten some aspects of it, as each line of code uses up precious internal memory.

Hint

To further rationalise the code, the multiple 'if' statements can be reduced using the 'switch' decision statement.

Extend

Research header files that have pre-written common routines to reduce the complexity of the code.

Worked example B

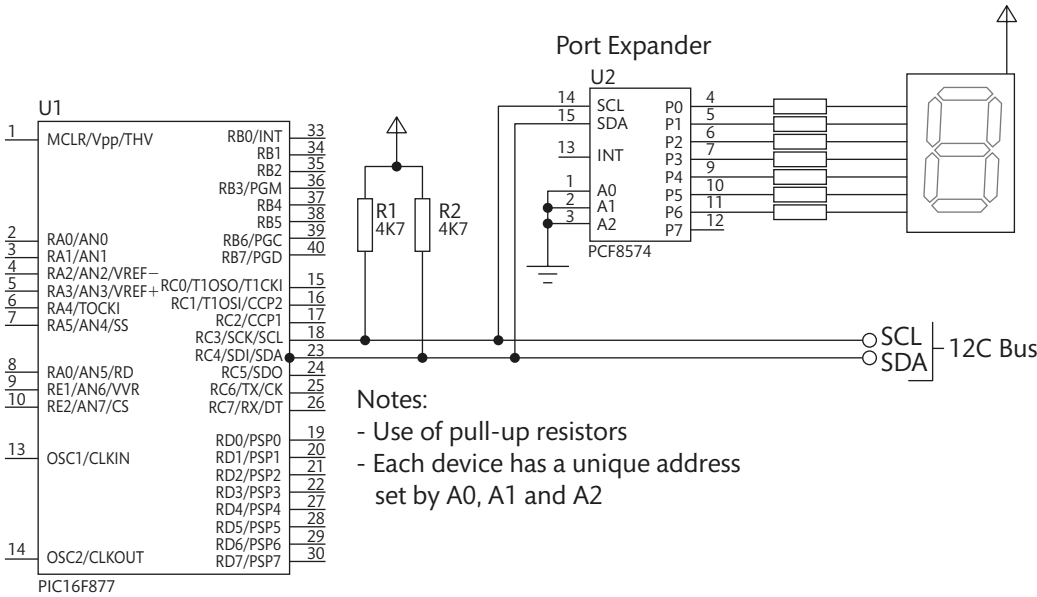
Simple I²C

The fundamental communications procedure for all I²C operations is as follows.

- One IC that wants to talk to another must:
 - wait until it sees no activity on the I²C bus – SDA and SCL are both high, i.e. the bus is 'free'.
 - put a message on the bus that says 'it is mine' – I have **started** to use the bus. All other ICs then **listen** to the bus data to see whether they might be the one who will be called up (addressed).
 - provide on the **clock** (SCL) wire a clock signal – this will be used by all the ICs as the reference time at which each bit of **data** on the data wire (SDA) will be valid and can be used. The data on the data wire (SDA) must be valid at the time the clock wire (SCL) switches from 'low' to 'high' voltage.
 - put out in serial form the unique binary 'address' (name) of the IC that it wants to communicate with.
 - put a message (one bit) on the bus announcing whether it wants to **send** or **receive** data from the other chip. (There is no read/write wire in this system.)
 - ask the other IC to **acknowledge** (using one bit) that it recognised its address and is ready to communicate.
- After the other IC acknowledges that all is OK, data can be transferred.
- The first IC sends or receives as many 8-bit words of data as it wants. After every 8-bit data word, the sending IC expects the receiving IC to acknowledge that the transfer is going OK.

- When all the data has finished transferring, the first chip must free up the bus – it does this by a special message called ‘STOP’. This is just one bit of information transferred by a special ‘wiggling’ of the SDA/SCL wires of the bus.

1 Build the circuit, referring to **Figure 6.20**.



► **Figure 6.20** Schematic for basic communications using an I²C interface

2 Design the program using pseudocode.

3 Study the following code.

```
// Title: Simple I2C sample protocol
// Description: Example of send instruction on I2C bus
// Author: Paul H Stewart
// Date: 09/02/xxxx
// Versions 1.2b

// Include header files for Micro
#include <pic16f877.h>
#include <htc.h>
// end includes

// define constants
const unsigned char BCD_to_7seg[10]           // set up an array for segment codes as connected in circuit
                                           // diagram
      = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F}; // these can vary depending on
                                                                    // order of connection

// end define constants

// defines
```

```

// end defines

// define constants
// end constants

// declare function prototypes
void init_iic(void);
void i2c_write(unsigned char iic_data, unsigned char slave_address);
void delay_ms(unsigned in lengthms)
// end function declares

void main()
{
    unsigned char slave_address, iic_data, unit;

    init_iic();    // call function to initialise iic bus

    while(1)
    {
        for (unit = 0; unit<10; unit++)
        {
            slave_address = 0x40;                // set up slave address
            iic_data = ~BCD_to_7seg[unit];
            i2c_write(iic_data, slave_address);  // call function to output data
            delay_ms(1000);
        }
    }
}

/*****
 * Function to initialise iic bus *
 *****/
void init_iic(void)
{
    TRISC3 = 1;                // configure SCL pin as an input
    TRISC4 = 1;                // configure SDA pin as an input

    SSPADD = 0x28;            // set baud rate = 100kHz, fosc = 4MHz - see manuals for i2c and PIC
    SSPCON = 0b00101000;      // synchronous Serial Port Enable Bit SSPEN = 1
                                // sSPM3:SSPM0 = 1000 - iic master mode
                                // clock = Fosc/(4*(SSAPD+1)) = 100KHz

    RC3 = 1;                  // ensure SCL high
    RC4 = 1;                  // ensure SDA high
    delay_ms(10);
}

/*****
 * Function to send a character over the iic bus *
 *****/
void i2c_write(unsigned char iic_data, unsigned char slave_address)
{
    SEN = 1;                  // set start enable bit
    while (SSPIF == 0) {};    // wait for SSPIF to be set
    SSPIF = 0;                // clear SSPIF

    SSPBUF = slave_address;   // set up slave address
    while (SSPIF == 0){};    // wait for SSPIF to be set
    SSPIF = 0;                // clear SSPIF

    SSPBUF = iic_data;        // transmit data
    while (SSPIF == 0){};    // wait for SSPIF to be set
}

```

```

    SSPIF = 0;           // clear SSPIF

    PEN = 1;            // generate a STOP condition
                        // by setting Stop Enable Bit
    while (SSPIF == 0){ // wait for SSPIF to be set
    SSPIF = 0;         // clear SSPIF
    }

void delay_ms(unsigned in lengthms)
{
    // see previous example for code
}

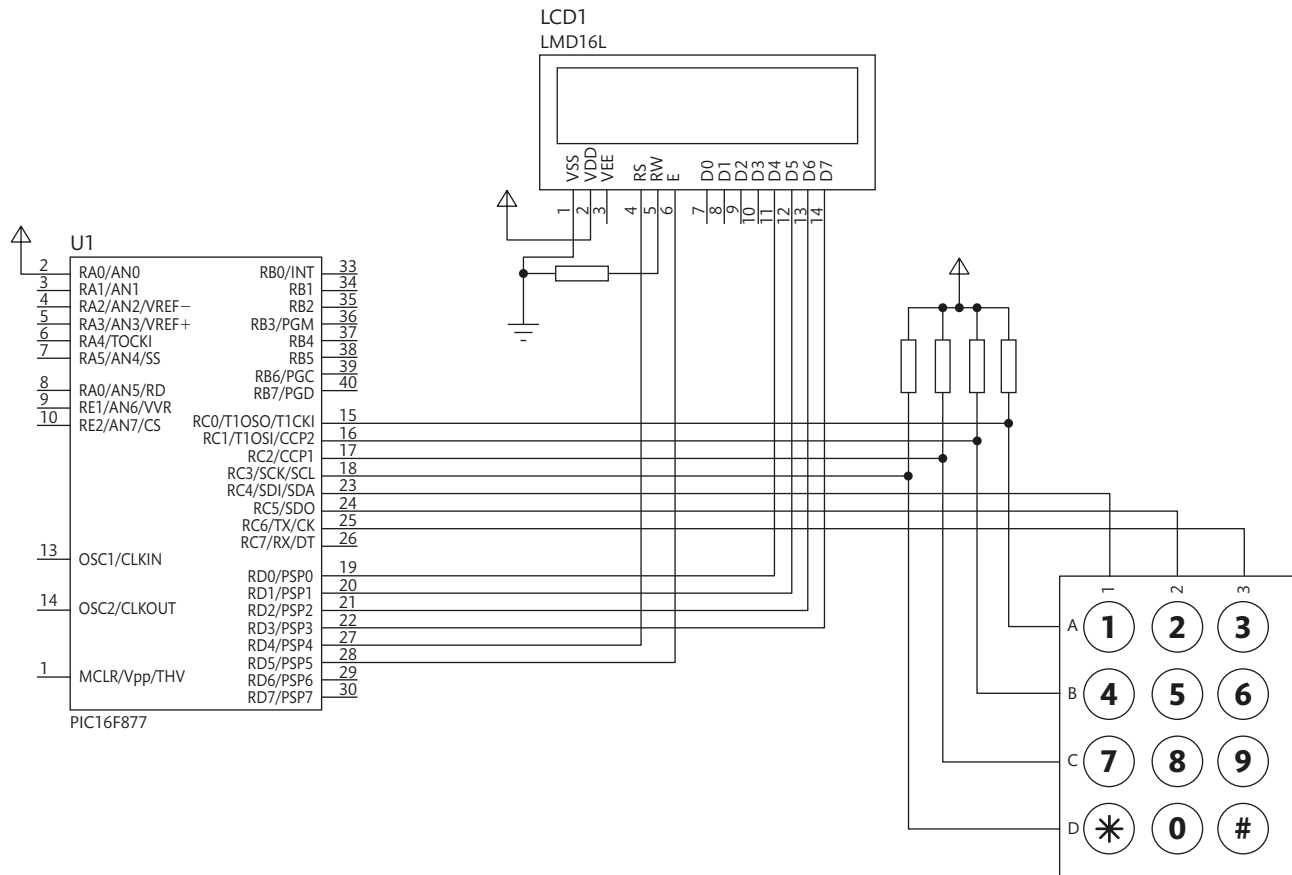
```

Worked example C

Keypad interface

This system allows the integration of relays controlling magnetic locks or other security devices.

Study the circuit schematic (**Figure 6.21**) and code that have been provided, and write the pseudocode (or alternative code development plan) that should have been used to design this code. Working out the connection between the plan and the code will make the coding easier to construct for other projects.



► **Figure 6.21** Schematic of keypad interface

Code for Worked example C:

```

// Title: Keypad interface and scanning
// Description: Example to show how to scan keypads and link to LCD or other devices
// Author: Alan Serplus
// Date: 09/02/xxxx
// Versions 1.0

// Include header files for Micro
#include <pic16f877.h>
#include <htc.h>
// end includes

// defines
#define col_1 RC4 // set pin 4 of port C as column 1
#define col_2 RC5
#define col_3 RC6
// end defines

// define constants
unsigned char keyvalue; // set global variable keyvalue as a character (8-bit)
char port_value;
// end constants

// declare function prototypes
void keypad_scan(void);
void init_LCD(void);
void LCD_function(unsigned char function);
void init_LCD(void);
void clock_inst(void);
void output_char_string(unsigned char line_num, const char output_string[ ]);
void output_char(unsigned char character);
void clock_LCD(void);
void delay_ms(unsigned int length); // millisecond delay routine
void delay_100us(unsigned int length); // 100 microsecond delay routine
// end function declares

void main()
{
    TRISC = 0x0F; // set port C as part input and part output RC.4; RC.5, RC.6, RC.7 are
                // outputs col 1 to 3
    keyvalue = 0; // clear keyvalue
    init_LCD(); // function to initialise LCD
    LCD_function(0x0C); // function to turn LCD on
    output_char_string(line1, "* Keypad Test *"); // output character string
    output_char_string(line2, "Keyvalue is: "); // output character string

    /* To output individual characters:
       First, move cursor to character position using LCD_function() function
       Second, output character using output_char() function
       NB: Cursor position automatically increments after a character has been written to the LCD. */

    while (1) // never-ending loop
    {
        keypad_scan();
        if (keyvalue!=0) // test if keyvalue has changed
        { // YES - output new keyvalue
            LCD_function(0xCD); // move cursor to line 2, position 13
            output_char(keyvalue); // output keyvalue to line 2, position 13
            keyvalue = 0; // reset keyvalue to 0
            delay_ms(500); // wait 0.5 seconds
        }
    }
}

```

```

}
}

void keypad_scan(void)
{
    // SCAN COLUMN 1
    col_1 = 0; col_2 = 1; col_3 = 1; // select column 1, i.e. col 1 = 0
    port_value = PORTC; // variable port_value takes on port C pin values
    port_value = (~port_value & 0xFF); // invert the values from port C
    port_value = (port_value&0x0F); // mask off upper nibble, only monitor lower nibble
    // now test if a col 1 button pressed

    switch(port_value)
    {
        case 1: keyvalue = 0x31; // key pressed is '1' (ascii value: 0x31)
                break; // quit block
        case 2: keyvalue = 0x34; // key pressed is '4' (ascii value: 0x34)
                break; // quit block
        case 4: keyvalue = 0x37; // key pressed is '7' (ascii value: 0x37)
                break; // quit block
        case 8: keyvalue = 0x2A; // key pressed is '*' (ascii value: 0x2A)
                break; // quit block
    }

    // SCAN COLUMN 2
    col_1 = 1; col_2 = 0; col_3 = 1; // select column 2, i.e. col 2 = 0
    port_value = PORTC; // variable port_value takes on port C pin values
    port_value = (~port_value & 0xFF); // invert the values from port C
    port_value = (port_value&0x0F); // mask off upper nibble
    // now test if a col 1 button pressed

    switch(port_value)
    {
        case 1: keyvalue = 0x32; // key pressed is '2' (ascii value: 0x32)
                break; // quit block
        case 2: keyvalue = 0x35; // key pressed is '5' (ascii value: 0x35)
                break; // quit block
        case 4: keyvalue = 0x38; // key pressed is '8' (ascii value: 0x38)
                break; // quit block
        case 8: keyvalue = 0x20; // key pressed is '0' (ascii value: 0x20)
                break; // quit block
    }

    // SCAN COLUMN 3
    col_1 = 1; col_2 = 1; col_3 = 0; // select column 3, i.e. col 3 = 0
    port_value = PORTC; // variable port_value takes on port C pin values
    port_value = (~port_value & 0xFF); // invert the values from port C
    port_value = (port_value&0x0F); // mask off upper nibble
    // now test if a col 1 button pressed

    switch(port_value)
    {
        case 1: keyvalue = 0x33; // key pressed is '3' (ascii value: 0x33)
                break; // quit block
        case 2: keyvalue = 0x36; // key pressed is '6' (ascii value: 0x36)
                break; // quit block
        case 4: keyvalue = 0x39; // key pressed is '9' (ascii value: 0x39)
                break; // quit block
        case 8: keyvalue = 0x23; // key pressed is '#' (ascii value: 0x23)
                break; // quit block
    }
}
}

```



```
void init_LCD()
{
    // see Worked example A for code
}

void LCD_function(unsigned char function)
{
    // see Worked example A for code
}

void clock_inst()
{
    // see Worked example A for code
}

void output_char_string(unsigned char line_num, const char output_string[ ])
{
    // see Worked example A for code
}

void output_char(unsigned char character)
{
    // see Worked example A for code
}

void clock_LCD()
{
    // see Worked example A for code
}

void delay_ms(unsigned int length)
{
    // see Worked example A for code
}

void delay_100us(unsigned int length)
{
    // see Worked example A for code
}
```

Think about it

- ▶ You have now completed several projects, analysed the inputs and outputs, and developed coding and testing schedules. Don't stop there!
- ▶ Look for systems to practise your skills on in the world around you. Think of potential microcontroller systems that could be used to automate or manage the system.
- ▶ Try any available sample assessments and assessments from previous years to hone your skills in all aspects of this unit.
- ▶ You will not have access to your work outside the supervised periods, so speed and accuracy are vital during the assessment – the more you practise, the faster and more accurately you will be able to work.