```c
/* ================================================================================
 UPC - EETAC - CSD - http://digsys.upc.edu
Project P9 tutorial on the introduction of microcontrollers.
Dual_MUX4 combinational circuit.

Example of using C language and PIC microcontrollers to solve logic equations or
design combinational blocks. A simple way to introduce the basics of low level
programming (bitwise operations) and code organisation.

================================================================================ */

// This general header file includes the device specific headers like
// "pic18f4520.h". This file contains all the microcontroller declarations
// and the MPLAB X IDE handles it all.
#include <xc.h>

/* ------------------------------------
    Definitions
   ------------------------------------------------------------------------- */

/* ------------------------------------
    Function prototypes
   ------------------------------------------------------------------------- */
void init_system (void);
void read_inputs (void);
void truth_table (void);
void write_outputs (void);

/* ------------------------------------
    Global variables
   ------------------------------------------------------------------------- */
/* Let's use convenient byte-type (char) variables. Each variable is an 8-bit
 * RAM memory position to make simple watching their values. */

static char Var_E;      // Enable pin (E_L) that will be changed to active high.
static char Var_Ch0;
static char Var_Ch1;
static char Var_Ch2;
static char Var_Ch3;
static char Var_Sel;    // Selection of channels
static char Var_Y;      // Output

static char Buffer; // An 8-bit memory position where to store partial results
                    // while processing the data internally in the functions
static char Buffer2;

// ---> So, as you see, this application will require at least 9 bytes of RAM
/* ================================================================================
        Main function
   ================================================================================ */
void main(void){    // Let's make main() the simplest possible.
    init_system();
    while(1) {      /* loop forever reading inputs and calculating outputs*/

        read_inputs();
        truth_table();
        write_outputs();
    }
}

/* ------------------------------------
    Function definitions
   ------------------------------------------------------------------------- */

//*********************************************************************
// Initialise the microcontroller system
//*********************************************************************
void init_system (void){
// Configuration bits

// These are the configuration bits translated to the XC8 Compiler
// Microcontroller general configuration bits. You don't have to change them.
// PIC18F4520 Configuration Bit Settings. They can be generated automatically
```

```
74  // in the MPLAB X IDE
75
76  // CONFIG1H
77  #pragma config OSC = XT       // Oscillator Section bits (XT oscillator)
78  #pragma config FCMEN = OFF
79  // Fail-Safe Clock Monitor E bit (Fail-Safe Clock Monitor disabled)
80  #pragma config IESO = OFF
81  // Internal/External Osc Switch-over bit (Oscillator Switch-over mode disabled)
82
83  // CONFIG2L
84  #pragma config PWRT = ON       // Power-up Timer E bit (PWRT enabled)
85  #pragma config BOREN = SBORDIS
86  // Brown-out Reset E bits (Brown-out Reset enabled in hardware
87  // only (SBOREN is disabled))
88  #pragma config BORV = 3    // Brown Out Reset Voltage bits (Minimum setting)
89
90  // CONFIG2H
91  #pragma config WDT = OFF
92  // Watchdog Timer E bit (WDT disabled (control is on the SWDTEN bit))
93  #pragma config WDTPS = 32768
94  // Watchdog Timer Postscale Sect bits (1:32768)
95
96  // CONFIG3H
97  #pragma config CCP2MX = PORTC
98  // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
99  #pragma config PBADEN = OFF
100 // PORTB A/D E bit (PORTB<4:0> pins are configured
101 // as digital I/O on Reset)
102 #pragma config LPT1OSC = OFF
103 // Low-Power Timer1 Oscillator E bit (Timer1 configured for
104 // higher power operation)
105 #pragma config MCLRE = ON
106 // MCLR Pin E bit (MCLR pin enabled; RE3 input pin disabled)
107
108 // CONFIG4L
109 #pragma config STVREN = ON
110 // Stack Full/Underflow Reset E bit (Stack full/underflow will cause Reset)
111 #pragma config LVP = OFF
112 // Single-Supply ICSP E bit (Single-Supply ICSP disabled)
113 #pragma config XINST = OFF
114 // Extended Instruction Set E bit (Instruction set extension and Indexed
115 // Addressing mode disabled (Legacy mode))
116
117 // CONFIG5L
118 #pragma config CP0 = OFF
119 // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
120 #pragma config CP1 = OFF
121 // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
122 #pragma config CP2 = OFF
123 // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
124 #pragma config CP3 = OFF
125 // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)
126
127 // CONFIG5H
128 #pragma config CPB = OFF
129 // Boot Block Code Protection bit
130 // (Boot block (000000-0007FFh) not code-protected)
131 #pragma config CPD = OFF
132 // Data EEPROM Code Protection bit (Data EEPROM not code-protected)
133
134 // CONFIG6L
135 #pragma config WRT0 = OFF
136 // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
137 #pragma config WRT1 = OFF
138 // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
139 #pragma config WRT2 = OFF
140 // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
141
142 #pragma config WRT3 = OFF
143 // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)
144
145 // CONFIG6H
146 #pragma config WRTC = OFF
```

```
147  // Configuration Register Write Protection bit (Configuration registers
148  // (300000-3000FFh) not write-protected)
149  #pragma config WRTB = OFF
150  // Boot Block Write Protection bit (Boot block (000000-0007FFh)
151  // not write-protected)
152  #pragma config WRTD = OFF
153  // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
154
155  // CONFIG7L
156  #pragma config EBTR0 = OFF
157  // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from
158  // table reads executed in other blocks)
159  #pragma config EBTR1 = OFF
160  // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from
161  // table reads executed in other blocks)
162  #pragma config EBTR2 = OFF
163  // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from table
164  // reads executed in other blocks)
165  #pragma config EBTR3 = OFF
166  // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from table
167  // reads executed in other blocks)
168
169  // CONFIG7H
170  #pragma config EBTRB = OFF
171  // Boot Block Table Read Protection bit (Boot block (000000-0007FFh)
172  // not protected from table reads executed in other blocks)
173
174      /* -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*/
175  // Initialise starts here:
176      PORTA   = 0x00; // Reset all Flip-Flops at PORTA
177      LATA    = 0x00; //Another way to initialise the PORTA
178
179      /* All the PortA is set to be digital I/O. Remember that the PORTA
180       * can be also defined as analogue inputs, and this is why we must
181       * configure this special register ADCON1                          */
182      ADCON1  = 0x0F;
183                              //***********************************************
184      TRISA   = 0b00111000;   // This is for setting the pin direction
185                              // '1' means an input; '0' means an output
186                              // RA4, RA3 and RA5 are inputs
187
                                //***********************************************

188      PORTB   = 0x00; // Reset all Flip-Flops at PORTB
189      LATB    = 0x00;
190      TRISB   = 0x00001011; // RB3, RB1 and RB0 are inputs
191
192      PORTC   = 0x00; // Reset all Flip-Flops at PORTC
193      LATC    = 0x00;
194      TRISC   = 0x11001000; // RC7, RC6 and RC3 are inputs
195
196      PORTD   = 0x00; // Reset all Flip-Flops at PORTD
197      LATD    = 0x00;
198      TRISD   = 0b00001100;   // RD3 and RD2 are inputs
199
200
201      GIE     = 0;    // No interrupts used in this very simple application
202
203      Var_Y = 0x00;   //Output variable cleared
204  }
205
206  /* ============================================================================
207   Scan inputs and process the bits so that we can set our convenient variables
208   many of the variables are "char" because we can easily monitor them using
209   the watch window for debugging purposes.
210    ============================================================================ */
211  void read_inputs (void){
212      // Let's prepare our E_L bit as a byte of memory (Var_E),
213      // so that we'll be able to "watch" it when running the simulation
214
215  // 1) Read all the bits (like LD a parallel 8-bit register)
216      Buffer = PORTC;
217  // 2) Mask the bit of interest ("&" is a bitwise AND)
```

```
    Buffer = Buffer & 0b00001000;

// 3) Change the active level and save
    if (Buffer == 0)
        Var_E = 1;
    else
        Var_E = 0;      // Now it's an Active high variable


    // Let's read/sample/poll all the PORTC to get Ch3(1..0):
    Buffer = PORTA;
    Buffer = Buffer & 0b00011000;    // Now mask the bits of interest
    Var_Ch3 = Buffer >> 3;           // Now shift and save


    // Let's poll all the PORTD to get Ch2(1..0):
    Buffer = PORTD;
    Var_Ch2 = (Buffer & 0b00001100)>> 2; // In a single instruction


    // Let's poll all the PORTC to get Ch1(1..0):
    Buffer = PORTC;
    Var_Ch1 = (Buffer & 0b11000000)>> 6;

    // Let's poll all the PORTB to get Ch0(0) and the PORTA to get Ch0(1)
    Buffer = (PORTB & 0b00001000)>> 3;  // Get the Ch0(0)

    Buffer2= (PORTA & 0b00100000)>> 4;  // Get the Ch0(1))
    Var_Ch0 = Buffer | Buffer2;         // "|" is a bitwise OR


    // Let's read the channel selection variable
    Var_Sel = PORTB & 0b00000011;
}

/* ============================================================================
 The behavioural description of the circuit operation or the data processing
 algorithm.
 When this function is executed, all the variables are in the software domain,
 meaning that they are completely independent of the hardware, thus the
 algorithm is compatible among microcontroller families and pin connexions.
  ============================================================================ */
void truth_table(void){

// This is the combinational circuit truth table described in a behavioural way:

    if (Var_Sel == 0 && Var_E == 1) {
        Var_Y = Var_Ch0;
        }
    else if (Var_Sel == 1 && Var_E == 1){
        Var_Y = Var_Ch1;
        }
    else if (Var_Sel == 2 && Var_E == 1){
        Var_Y = Var_Ch2;
        }
    else if (Var_Sel == 3 && Var_E == 1){
        Var_Y = Var_Ch3;
        }
    else
        Var_Y = 0x00;
}

/* ============================================================================
    Driving the output pins. Let's drive the output ports, converting
    microcontroller variables into electrical signals.
   ============================================================================ */
void write_outputs(void){

    Buffer  = (Var_Y & 0b00000010) << 6  ; // Y(1) in RB7
    Buffer2 = (Var_Y & 0b00000001) << 2  ; // Y(0) in RB2
    PORTB = Buffer | Buffer2;
 // It is better to write the PORT in a single instruction, thus, it will be
 // required to mask and organise the port pins as when reading inputs,
```

```
291     // specially if the same port that you like to write contains already other
292     // outputs.
293   }
294
```