```
#include <xc.h>

// CONFIG1L

#pragma config PLLDIV = 1       // PLL Prescaler Selection bits (No prescale (4 MHz oscillator input drives PLL directly))

#pragma config CPUDIV = OSC1_PLL2// System Clock Postscaler Selection bits ([Primary Oscillator Src: /1][96 MHz PLL Src: /2])

#pragma config USBDIV = 1       // USB Clock Selection bit (used in Full-Speed USB mode only; UCFG:FSEN = 1) (USB clock source comes directly from the primary oscillator block with no postscale)


// CONFIG1H

#pragma config FOSC = HSPLL_HS  // Oscillator Selection bits (HS oscillator, PLL enabled (HSPLL))

#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)

#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)


// CONFIG2L

#pragma config PWRT = OFF       // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOR = OFF        // Brown-out Reset Enable bits (Brown-out Reset disabled in hardware and software)

#pragma config BORV = 3         // Brown-out Reset Voltage bits (Minimum setting 2.05V)

#pragma config VREGEN = OFF     // USB Voltage Regulator Enable bit (USB voltage regulator disabled)


// CONFIG2H

#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))

#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)


// CONFIG3H

#pragma config CCP2MX = ON      // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)

#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
```

```
#pragma config LPT1OSC = OFF    // Low-Power Timer 1 Oscillator Enable bit (Timer1 configured for higher power operation)

#pragma config MCLRE = ON       // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin disabled)


// CONFIG4L

#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)

#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)

#pragma config ICPRT = OFF      // Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable bit (ICPORT disabled)

#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))


// CONFIG5L

#pragma config CP0 = OFF        // Code Protection bit (Block 0 (000800-001FFFh) is not code-protected)

#pragma config CP1 = OFF        // Code Protection bit (Block 1 (002000-003FFFh) is not code-protected)

#pragma config CP2 = OFF        // Code Protection bit (Block 2 (004000-005FFFh) is not code-protected)

#pragma config CP3 = OFF        // Code Protection bit (Block 3 (006000-007FFFh) is not code-protected)


// CONFIG5H

#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) is not code-protected)

#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM is not code-protected)


// CONFIG6L

#pragma config WRT0 = OFF       // Write Protection bit (Block 0 (000800-001FFFh) is not write-protected)

#pragma config WRT1 = OFF       // Write Protection bit (Block 1 (002000-003FFFh) is not write-protected)

#pragma config WRT2 = OFF       // Write Protection bit (Block 2 (004000-005FFFh) is not write-protected)
```

```
#pragma config WRT3 = OFF       // Write Protection bit (Block 3 (006000-007FFFh) is not write-protected)


// CONFIG6H

#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) are not write-protected)

#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot block (000000-0007FFh) is not write-protected)

#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM is not write-protected)


// CONFIG7L

#pragma config EBTR0 = OFF      // Table Read Protection bit (Block 0 (000800-001FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR1 = OFF      // Table Read Protection bit (Block 1 (002000-003FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR2 = OFF      // Table Read Protection bit (Block 2 (004000-005FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR3 = OFF      // Table Read Protection bit (Block 3 (006000-007FFFh) is not protected from table reads executed in other blocks)


// CONFIG7H


#define _XTAL_FREQ 20000000 // 20MHz crystal frequency


// Buzzer output

#define BUZZER LATCbits.LATC2


// Raw Switch inputs

#define SW1_RAW PORTBbits.RB0

#define SW2_RAW PORTBbits.RB1

#define SW3_RAW PORTBbits.RB2

#define SW4_RAW PORTBbits.RB3
```

```
// Switch Latch Time (milliseconds)
#define LATCH_TIME 5000
#define WINDOW_TIME 7000

// Variables for latch timers
unsigned long sw1_latch_time = 0;
unsigned long sw2_latch_time = 0;
unsigned long sw3_latch_time = 0;
unsigned long sw4_latch_time = 0;

// Timer for 7-second window
unsigned long window_start_time = 0;
char window_active = 0;

// Function prototypes
unsigned long millis(void);
void update_switch_latches(unsigned long current_time);

void main(void) {
   // Setup
   TRISB = 0x0F; // RB0-RB3 inputs, others outputs
   TRISCbits.TRISC2 = 0; // RC0 output (buzzer)
   LATCbits.LATC2 = 0; // Buzzer OFF initially

   // Timer setup (using Timer0)
   T0CON = 0x87; // Timer0 ON, 8-bit mode, prescaler 256
   TMR0 = 0; // Clear Timer0

   unsigned long current_time;

   while (1) {
```

```
        current_time = millis(); // Get current milliseconds

        update_switch_latches(current_time); // Refresh latched switch states

        // Check latch statuses
        char SW1 = (current_time - sw1_latch_time) < LATCH_TIME;
        char SW2 = (current_time - sw2_latch_time) < LATCH_TIME;
        char SW3 = (current_time - sw3_latch_time) < LATCH_TIME;
        char SW4 = (current_time - sw4_latch_time) < LATCH_TIME;

     if (SW1 && SW3) {
    BUZZER = 1; // Immediate buzzer ON
    window_active = 0; // Cancel any active window
}
else if (SW2 && SW4 && !window_active) {
    // Start 7-second timer
    window_start_time = current_time;
    window_active = 1;
}

if (window_active) {
    if ((current_time - window_start_time) < WINDOW_TIME) {
        if (SW1 && SW3) {
            BUZZER = 1;
            window_active = 0; // Deactivate window
        }
    }
    else {
        window_active = 0; // Timeout expired
    }
}
```

```
if (!SW1 || !SW3 && !window_active) {
   BUZZER = 0; // Otherwise turn buzzer OFF
}
   }
}

// Read current time based on Timer0 overflow
unsigned long millis(void) {
   static unsigned long ms_count = 0;
   static unsigned char last_timer = 0;
   unsigned char current_timer = TMR0;

   if (current_timer < last_timer) {
      // Overflow detected (256 to 0)
      ms_count += 1;
   }

   last_timer = current_timer;

   // Each Timer0 tick is roughly ~13ms at 20MHz with 256 prescaler
   return ms_count * 13;
}

// Update switch latch timers
void update_switch_latches(unsigned long current_time) {
   if (SW1_RAW) sw1_latch_time = current_time;
   if (SW2_RAW) sw2_latch_time = current_time;
   if (SW3_RAW) sw3_latch_time = current_time;
   if (SW4_RAW) sw4_latch_time = current_time;
}
```