MCES PRODUCTATHON
TITLE: BOMB DIFFUSION SIMULATOR

Description of the Product:

The "Bomb Diffusion Simulator" is a mini project designed to simulate the process of diffusing a bomb using various electronic components and a microcontroller, in this case, the LPC2148. Here's a detailed description of how the project works:

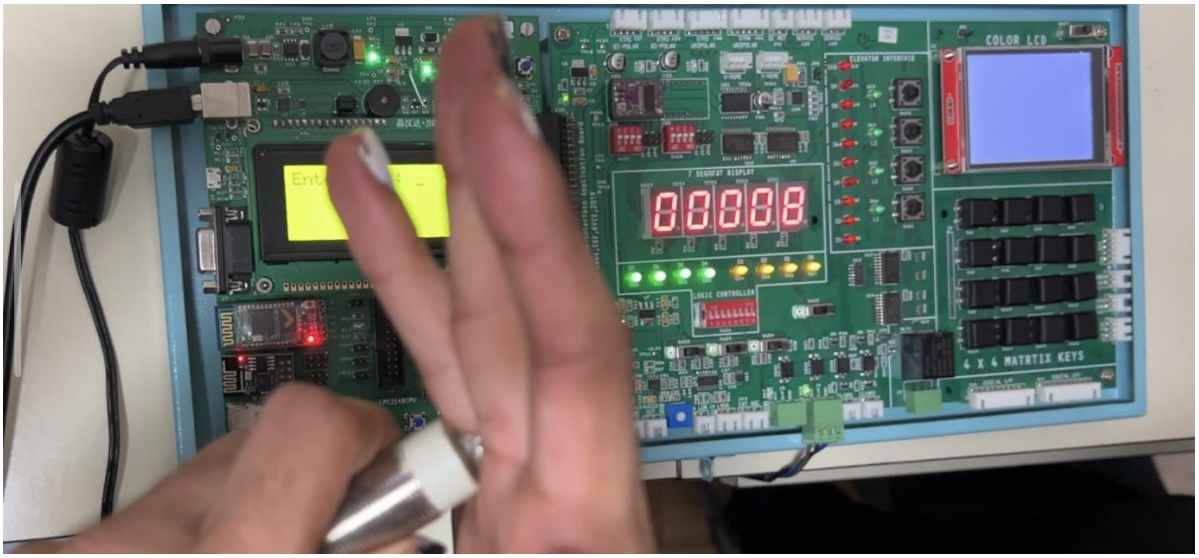
Hardware Components Used:

1. LPC2148 Microcontroller: This serves as the brain of the system and controls all the other components. It handles the logic for bomb diffusion, password entry, countdown timer, and interaction with peripherals.
2. Proximity Sensor: The proximity sensor is used to detect if a person has approached the bomb to diffuse it.
3. LCD Display: An LCD screen is used to provide visual feedback to the user. It initially displays the message "Enter pass:" and later shows either "Bomb Diffused" or "Bomb Exploded" based on the outcome.
4. 4x4 Matrix Keyboard: This keyboard is used for password input. It consists of a 4x4 grid of keys, allowing users to enter a numerical password.
5. Seven-Segment Display: A seven-segment display is used to show the countdown timer. It displays the remaining time the user must diffuse the bomb.

Working of the Product:

1. Initialization: When the system is powered on or reset, the microcontroller initializes all the components and prepares the display and countdown timer.
2. Display Initialization: The LCD display is set up to show the initial message, "Enter pass:"
3. Countdown Timer and Proximity Sensor: The seven-segment display is initialized with a countdown timer set to 10 seconds. This timer starts counting down as soon as the system is ready. This timer runs in the background with the help of Interrupts. While in the foreground, the proximity sensor is continuously detecting for a person to come and diffuse the bomb.
4. Countdown Stop: If the proximity sensor detects a person, the countdown timer is stopped.
5. Password Entry: The user is then prompted to enter a password using the 4x4 matrix keyboard. The entered password is compared to a predefined correct password.
6. Outcome Display: Depending on whether the correct password was entered before the proximity sensor detected a person, the LCD display will show one of two messages:
   If the password is correct and the proximity sensor detects a person, it will display "Bomb diffused!"
   If the password is incorrect and the proximity sensor detects a person, it will display "Bomb exploded!"

Product Images:



Source Code:

```
#include <lpc214x.h>
#include <string.h>
#define CLEAR (IO0SET = 0x000F0000)
#define SET1 IO0CLR = 1 << 16
#define SET2 IO0CLR = 1 << 17
#define SET3 IO0CLR = 1 << 18
#define SET4 IO0CLR = 1 << 19
#define C0(IO1PIN & 1 << 19)
#define C1(IO1PIN & 1 << 18)
#define C2(IO1PIN & 1 << 17)
#define C3(IO1PIN & 1 << 16)
#define LED_ON IO0CLR |= 1 << 31
#define LED_OFF IO0SET |= 1 << 31
#define Check0 if (C0 == 0) { col = 0; break; }
#define Check1 if (C1 == 0) { col = 1; break; }
#define Check2 if (C2 == 0) { col = 2; break; }
#define Check3 if (C3 == 0) { col = 3; break; }
#define ENTER '0'
#define WAIT while (C0 == 0 || C1 == 0 || C2 == 0 || C3 == 0);
#define RS_ON (IO0SET = 1 U << 20)
#define RS_OFF (IO0CLR = 1 U << 20)
#define EN_ON (IO1SET = 1 U << 25)
#define EN_OFF (IO1CLR = 1 U << 25)
unsigned int x = 0;
long int count = 10;
char buf[5];
char ch, keys[5], password[5] = "1234";
unsigned char len = 0;
unsigned int i = 0;

void delay(int x) {
    for (int i = 0; i < x; i++)
        for (int y = 0; y < 10000; y++)
    ;
}
static void delay_us(unsigned int count) {
    unsigned int j = 0, i = 0;
    for (j = 0; j < count; j++) {
        for (i = 0; i < 10; i++)
        ;
    }
}
static void LCD_SendCmdSignals(void) {
    RS_OFF; EN_ON;
    delay_us(100);
    EN_OFF;
}
static void LCD_SendDataSignals(void) {
    RS_ON; EN_ON;
    delay_us(100);
    EN_OFF;
}
static void LCD_SendHigherNibble(unsigned char dataByte) {
    // send the D7,6,5,D4(uppernibble) to P0.16 to P0.19
    IO0CLR = 0X000F0000;
    IO0SET = ((dataByte >> 4) & 0x0f) << 16;
}
static void LCD_CmdWrite(unsigned char cmdByte) {
    LCD_SendHigherNibble(cmdByte);
    LCD_SendCmdSignals();
    cmdByte = cmdByte << 4;
    LCD_SendHigherNibble(cmdByte);
    LCD_SendCmdSignals();
}
static void LCD_DataWrite(unsigned char dataByte) {
    LCD_SendHigherNibble(dataByte);
    LCD_SendDataSignals();
    dataByte = dataByte << 4;
    LCD_SendHigherNibble(dataByte);
    LCD_SendDataSignals();
}
static void LCD_Reset(void) {
    /* LCD reset sequence for 4-bit mode*/
    LCD_SendHigherNibble(0x30);
```

```
    LCD_SendCmdSignals();
    delay(100);
    LCD_SendHigherNibble(0x30);
    LCD_SendCmdSignals();
    delay_us(200);
    LCD_SendHigherNibble(0x30);
    LCD_SendCmdSignals();
    delay_us(200);
    LCD_SendHigherNibble(0x20);
    LCD_SendCmdSignals();
    delay_us(200);
}
static void LCD_Init(void) {
    delay(100);
    LCD_Reset();
    LCD_CmdWrite(0x28u); // Initialize the LCD for 4-bit 5x7 matrix type
    LCD_CmdWrite(0x0Eu); // Display ON cursor ON
    LCD_CmdWrite(0x01u); // Clear the LCD
    LCD_CmdWrite(0x80u); // go to First line First Position
}
void LCD_DisplayString(const char * ptr_string) {
    while (( * ptr_string) != 0)
        LCD_DataWrite( * ptr_string++);
}
unsigned char getAlphaCode(unsigned char alphachar) {
    switch (alphachar) {
    case 'f': return 0x8e;
    case 'i': return 0xf9;
    case 'r': return 0xce;
    case 'e': return 0x86;
    case 'h': return 0x89;
    case 'l': return 0xc7;
    case 'p': return 0x8c;
    case ' ': return 0xff;
    case '1': return 0xf9;
    case '2': return 0xa4;
    case '3': return 0xb0;
    case '4': return 0x99;
    case '5': return 0x92;
    case '6': return 0x82;
    case '7': return 0xf8;
    case '8': return 0x80;
    case '9': return 0x90;
    case '0': return 0xc0;
    default: break;
    }
    return 0xff;
}
void alphadisp7SEG(char * buf) {
    unsigned char i, j;
    unsigned char seg7_data, temp = 0;
    for (i = 0; i < 5; i++) {
        seg7_data = getAlphaCode( * (buf + i));

        for (j = 0; j < 8; j++) {
            temp = seg7_data & 0x80;

            if (temp == 0x80)
                IOSET0 |= 1 << 19;
            else
                IOCLR0 |= 1 << 19;

            IOSET0 |= 1 << 20;
            delay(1);
            IOCLR0 |= 1 << 20;
            seg7_data = seg7_data << 1;
        }
    }

    IOSET0 |= 1 << 30;
    delay(1);
    IOCLR0 |= 1 << 30;
    return;
}
__irq void Timer0_ISR(void) // an ISR program
{
    // 7seg logic
```

```
152.     sprintf(buf, "%05lu", count);
153.     alphadisp7SEG( & buf[0]);
154.     count--;
155.     if (count == 0) {
156.         count = 10;
157.     }
158.     delay(600);
159.     T0IR = 0x01; // clear match0 interrupt, and get ready for the next interrupt
160.     VICVectAddr = 0x00000000; // End of interrupt
161. }
162. char getKey() {
163.     unsigned char lookup_table[4][4] = {
164.         {'0','1','2','3'},
165.         {'4','5','6','7'},
166.         {'8','9','a','b'},
167.         {'c','d','e','f'}
168.     };
169.     unsigned int row = 0, col = 0;
170.     do {
171.         row = 0; CLEAR; SET1;
172.         Check0; Check1; Check2; Check3;
173.
174.         row = 1; CLEAR; SET2;
175.         Check0; Check1; Check2; Check3;
176.
177.         row = 2; CLEAR; SET3;
178.         Check0; Check1; Check2; Check3;
179.
180.         row = 3; CLEAR; SET4;
181.         Check0; Check1; Check2; Check3;
182.     } while (1);
183.
184.     delay(50); // debouncing
185.     WAIT;
186.     delay(50); // debouncing
187.
188.     return lookup_table[row][col];
189. }
190.
191. int readSensor(int sen_no) {
192.     int result = 0;
193.     IO1DIR |= 1 << 24;
194.     IO1CLR |= 1 << 24; // enable sensor logic
195.     switch (sen_no) {
196.     case 1:
197.         result = IO1PIN & (1 << 22);
198.         break;
199.     case 2:
200.         result = IO1PIN & (1 << 23);
201.         break;
202.     default:
203.         result = 0;
204.     }
205.     IO1SET = 1 << 24;
206.     return result;
207. }
208.
209. int main() {
210.     int result = 0;
211.     IO1DIR |= 1 U << 25;
212.     IO0DIR |= 1 U << 31 | 1 U << 19 | 1 U << 20 | 1 U << 30 | 0x00FF0000;
213.
214.     LED_OFF;
215.
216.     LCD_Reset();
217.     LCD_Init();
218.     delay(100);
219.     LCD_CmdWrite(0x80);
220.     LCD_CmdWrite(0x80);
221.
222.     T0TCR = 0x00; // stop the timer, to initialize different registers
223.     T0MCR = 0x0003; // Enable Interrupt and reset timer after match
224.     T0TC = 0x00; // make TC = 0
225.     T0MR0 = 1500000; // generates 1s
226.
227.     // load interrupt related registers , assigning Timer0 to IRQ slot 4
228.     VICVectAddr4 = (unsigned long) Timer0_ISR; // set the timer ISR vector address
```

```
    VICVectCntl4 = 0x0000024; // set the channel
    VICIntEnable = 0x0000010; // enable the timer0 interrupt
    T0TCR = 0x01; // start the timer

    LCD_DisplayString("Enter Pass: ");

    while (1) {
        while ((result = readSensor(2)) != 0) {
            char ch;
            do {
                i = 0;
                VICIntEnable = 0x00000000;
                T0TCR = 0x00; // start the timerdelay(10);
                IO0SET |= 1 U << 16 | 1 U << 17 | 1 U << 18 | 1 U << 19;
                while (1) {
                    if ((ch = getKey()) == ENTER)
                        break;
                    keys[i++] = ch;
                }
                keys[i] = '\0';

                if ((strcmp(keys, password)) == 0) {
                    LCD_CmdWrite(0x80); // go to First line First Position
                    LCD_DisplayString("BOMB DIFFUSED!");
                    LED_ON;
                    delay(2000);
                    return 0;
                }
                LCD_CmdWrite(0x80);
                LCD_DisplayString("BOMB EXPLODED!");

            } while (1);
        }
    }
    return 0;
}
```