Binary Operation Challenge Alarm Clock

ABDUL BAQUAR, ABHIJIT SAHOO, ABHISHEK CHOUDHARY, ADITYA RAJ, HARSHA

MS RAMAIAH INSTITUTE OF TECHNOLOGY

ABSTRACT

Waking up on time is a struggle most people face every day. Turning off the alarm or hitting the snooze button is only one touch away on our phones. This project, with the help of 8051 microcontroller and the principles of logic design, aims to design and realize an alarm clock which can only be turned off by solving a binary operation challenge. The challenges require the use of pen and paper to solve, and are designed to offer flexibility in terms of complexity to the end user. Ensuring the end user is mentally active before turning off the alarm is the objective of this project.

GROUP MEMBERS

Name	USN
Abdul Baquar	1MS18EE001
Abhijit Sahoo	1MS18EE002
Abhishek Choudhary	1MS18EE003
Aditya Raj	1MS18EE005
Harsha	

I. Introduction

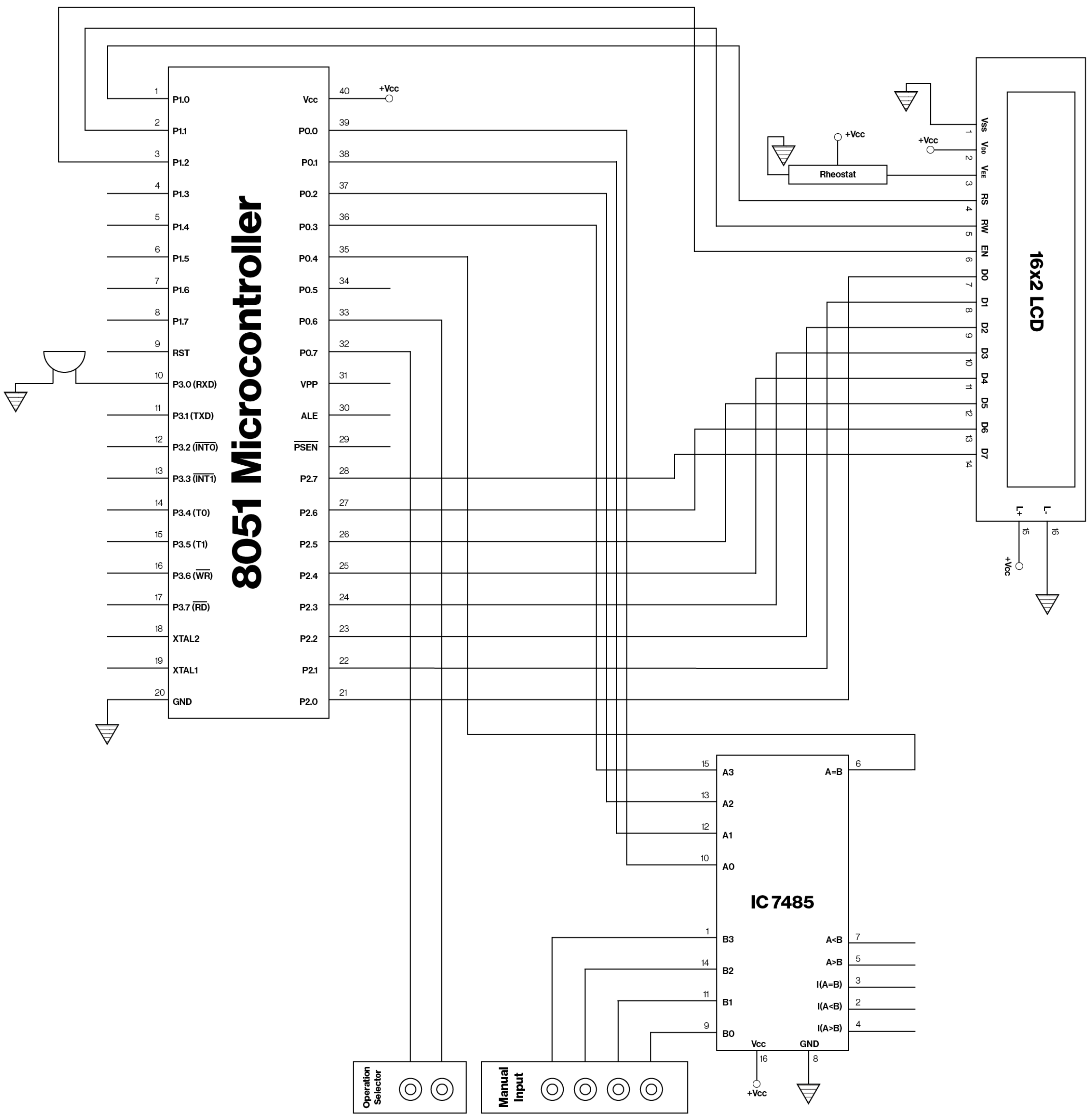
Designed to require the full attention of the end user, we aim to design a better alarm clock. The project has five major components:

i. 8051 microcontroller
ii. random number generator
iii. 2-bit operation selector
iv. 4-bit comparator
v. buzzer.

8051 microcontroller displays a number between 0 and 9 with the help of random number generator implemented using C libraries. The number is displayed on a 16x2 LCD. Simultaneously, user selects their preferred operation using a 2-bit operation selector; currently *00* is 1's compliment, *01* is excess3, *10* is binary to gray code conversion, and *11* is gray to binary code conversion. These operations are hard-coded. Using the output from random number generator and 2-bit operation selector, 8051 microcontroller performs the code conversion internally.

The output of the aforementioned operation is then fed to a 4-bit comparator circuit, this is input 'A' of comparator. The input 'B' of comparator circuit is controlled by the user with the help of toggle switches. If equal, the user has successfully completed the challenge, and the alarm buzzer is turned off.

II. Circuit Diagram



III. Code

```
#include<reg51.h>
#include<stdlib.h>

// LCD Control Variables
sbit rs = P1^0;
sbit rw = P1^1;
sbit en = P1^2;

// LCD Functions
void lcdcmd(unsigned char command);
void lcddat(unsigned char writeData);
void delay();           // fixed delay

// I/O Variables
sbit comp3 = P0^3;      // MSB output of processed data [OUT]
sbit comp2 = P0^2;
sbit comp1 = P0^1;
sbit comp0 = P0^0;

sbit opsHi = P0^7;      //MSB of operation selector [IN]
sbit opsLo = P0^6;

sbit comparison = P0^4;
sbit buzz  = P3^0;      // Buzzer

sbit temp3 = P3^7;
sbit temp2 = P3^6;
sbit temp1 = P3^5;
sbit temp0 = P3^4;
```

```
unsigned char randomNum, rnb;   // rnb = random number backup

// Main
void main() {
    P2 = 0x00;          // init data lines to 0
    comparison = 0;    // init compr test to 0
    // Random number generator
    randomNum = rand() % 10;    // Mod 10 reduces the out: 0-9



    if (opsHi == 0 && opsLo == 1){
        // Assigning bits
        // Adding 3 before binary conversion for excess3 special case
        randomNum = randomNum + 3;
    }

    // Assigning bits
    temp0 = randomNum % 2;         // 9 ->          [1]
    randomNum = randomNum / 2;     // 9/2 -> 4
    temp1 = randomNum % 2;         // 4 ->          [0]
    randomNum = randomNum / 2;     // 4/2 -> 2
    temp2 = randomNum % 2;         // 2 ->          [0]
    randomNum = randomNum / 2;     // 2/2 ->        [1]
    temp3 = randomNum;

    /*     ====  Operations   ===       */
    // Level 0: 1s C
        if (opsHi == 0 && opsLo == 0) {
            comp3 =~ temp3;
            comp2 =~ temp2;
            comp1 =~ temp1;
            comp0 =~ temp0;
        }


    // Level 1: E3
        if (opsHi == 0 && opsLo == 1) {
            comp3 = temp3;
            comp2 = temp2;
            comp1 = temp1;
            comp0 = temp0;
        }

    // Level 2: Binary to gray conversion
        if (opsHi == 1 && opsLo == 0) {
            comp3 = temp3;
            comp2 = temp3 ^ temp2;
            comp1 = temp2 ^ temp1;
            comp0 = temp1 ^ temp0;
        }

    // Level 3: Gray to binary conversion
        if (opsHi == 1 && opsLo == 0) {
            comp3 = temp3;
            comp2 = comp3 ^ temp2;
            comp1 = comp2 ^ temp1;
            comp0 = comp1 ^ temp0;
        }

    // Loop for LCD output and buzzer [quit on compr = 1]
    while(1) {
        // Buzzer
        buzz = 1;
        delay();
```

```
        buzz = 0;
        delay();

        // LCD Output
                // LCD init
        lcdcmd(0x38);       // 5x7 crystal matrix
        delay();
        lcdcmd(0x01);       // Clear screen
        delay();
        lcdcmd(0x10);       // Cursor blinking
        delay();
        lcdcmd(0x0C);       // Display ON
        delay();

                // Passing data
        lcddat(rnb);
        delay();

        // Break condition when user enters correct input
        if(comparison == 1) {
            break;
        }
    }
}


// LCD Commands
void lcdcmd(unsigned char command) {
    P2 = command;
    rs = 0;         // operation mode
    rw = 0;         // operation mode
    en = 1;         // pulse generation
    delay();
    en = 0;
}

void lcddat(unsigned char writeData) {
    P2 = writeData;
    rs = 1;         // write mode
    rw = 0;         // write mode
    en = 1;         // pulse generation
    delay();
    en = 0;
}

// Delay
void delay() {
    unsigned int i;
    for(i = 0; i < 12000; i++);
}
```

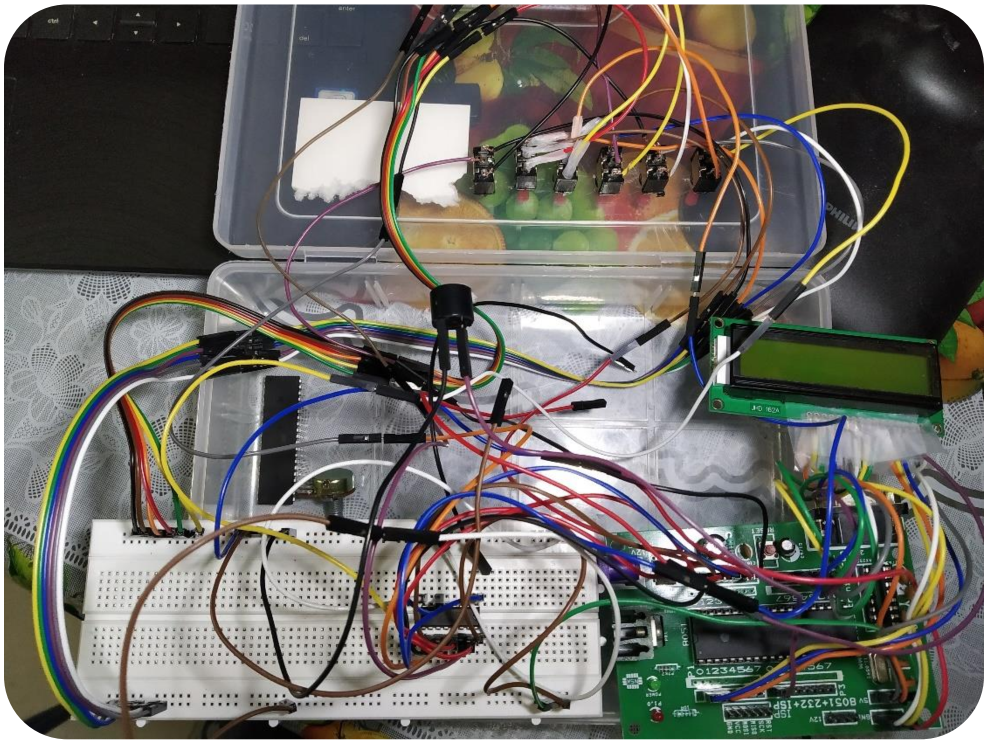
Code hosted on GitHub.

Scan the QR Code using your phone.

https://github.com/Az-21/blaze/blob/master/project013-binaryChallenge/clockBuzzer.c

IV. Conclusion

Designed to require the full attention of the end user, we have created a hardware extension for our alarm clocks. Once triggered, it will turn off when the end user completes the binary operation challenge.



V. Future Scope

Since our project works as an extension, with little modification, we can make it compatible with both analog and digital clocks. Compatibility with digital clocks is easy as it involves tapping an 'alarm enable' trigger and modifying the code around it. To make it compatible with analog clocks, we can install two proximity sensors; when both minute hand and hour hand are in correct place, both proximity sensors are activated, and this triggers our alarm extension.

Other improvements include miniaturization, cell powered circuit, and the use of sticky I/O switches for user input.