The format will be same throughout this PDF which is:

1. Objective
2. Brief theory
3. Calculations if any
4. Interfacing block diagram
5. Source code
6. Expected output
7. Sample output (after execution)
8. conclusion

Write-Up **ESIOT**

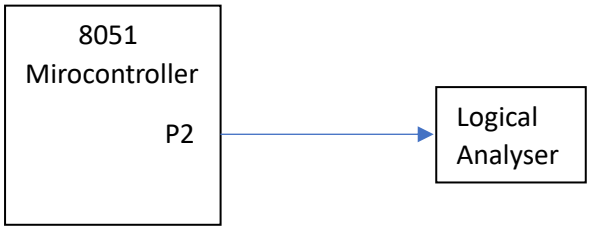
Termwork 1

Objective: The objective of this experiment is to design a 'C' program for the 8051 microcontrollers to implement a MOD-4 (UP/DOWN) counter. The counter will drive LEDs connected to Port 2, and a 1-second delay will be generated using a for loop between each count.

Brief Theory: The 8051 microcontroller is a widely used 8-bit microcontroller with various applications. The MOD-4 counter will count up and down. To achieve the 1-second delay, we will utilize a for loop that iterates a calculated number of times to create the desired time delay.

NO CALCULATIONS

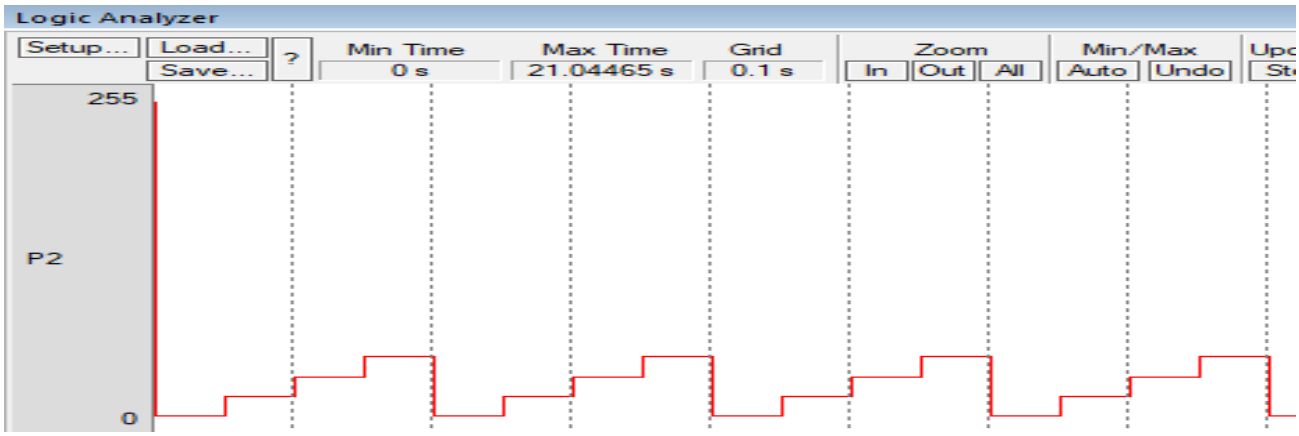
Diagram:



Code:

```
#include "at89c51ed2.h"
void delay(unsigned int);
void main(void){

        while(1){

                P2=0x00;
                delay(1000);
                P2=0x10;
                delay(1000);
                P2=0x20;
                delay(1000);
                P2=0x30;
                delay(1000);

        }
}

void delay(unsigned int itime){

   unsigned int i,j;
   for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);

}
```

Expected output:



Conclusion: The successful implementation of the 8051 'C' program for the MOD-4 (UP/DOWN) counter connected to Port 2 demonstrates the ability to control and generate a 1-second delay using a for loop. This exercise enhances understanding in microcontroller programming and helps build skills in developing simple counting applications

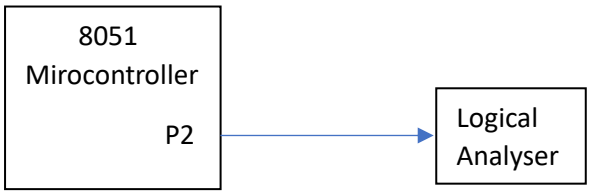
Termwork 2

Objective: The objective of this experiment is to design a 'C' program for the 8051 microcontrollers to implement a MOD-4 (UP/DOWN) counter. The counter will drive LEDs connected to Port 2, and a 0.5-second delay will be generated using a for loop between each count.

Brief Theory: The 8051 microcontroller is a widely used 8-bit microcontroller with various applications. The MOD-4 counter will count up and down. To achieve the 0.5-second delay, we will utilize a for loop that iterates a calculated number of times to create the desired time delay.

NO CALCULATIONS

Diagram:

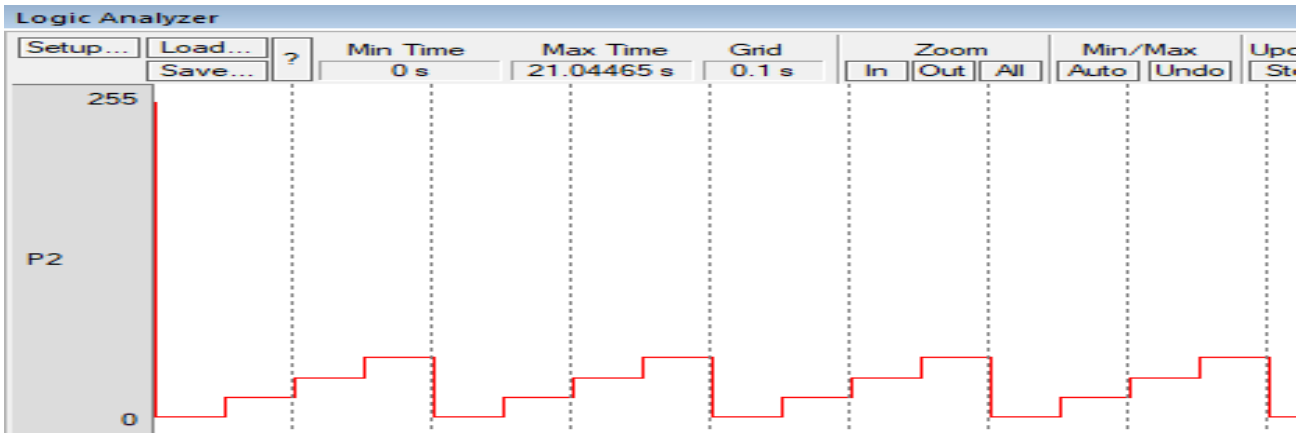


Code:

```
#include "at89c51ed2.h"
void delay(unsigned int);
void main(void){

        while(1){

                P2=0x00;
                delay(500);
                P2=0x10;
                delay(500);
                P2=0x20;
                delay(500);
                P2=0x30;
                delay(500);

        }
}
```

```
void delay(unsigned int itime){

   unsigned int i,j;
  for(i=0;i<itime;i++)
     for(j=0;j<1275;j++);

}
```

Expected output:



Conclusion: The successful implementation of the 8051 'C' program for the MOD-4 (UP/DOWN) counter connected to Port 2 demonstrates the ability to control and generate a 0.5-second delay using a for loop. This exercise enhances understanding in microcontroller programming and helps build skills in developing simple counting applications

Termwork 3

Objective: The objective of this experiment is to design an '8051 C' program to implement a MOD-4 counter, connected to Port 2 of the microcontroller. The counter should increment and display the count using hardware delay. Timer1 in Mode 1 will be utilized to generate a 50 ms delay between each count.

Brief Theory: The 8051 microcontrollers will be programmed in 'C' language to perform the MOD-4 counting operation. Port 2 of the microcontroller will be configured as an output port to drive the LEDs. The program will utilize Timer1 in Mode 1 to generate a delay of 50 ms between each count. The counter will increment from 0 to 3 and then repeat in a continuous loop, effectively creating a MOD-4 counter. The hardware delay ensures that the display is visible and stable between each count.

Calculations:

For Timer1Mode1, TMOD = 0x10

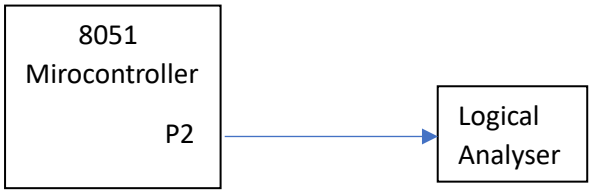
To calculate TH and TL, use

Delay = ((FFFF - YYXX) + 1) x $1.085x10^{-6}$

For 50ms delay,

$50x10^{-3}/1.085x10^{-6}$ = (FFFF - YYXX) + 1
46082 (approx. of 46082.9) = (FFFF - YYXX) + 1
YYXX = FFFF – hex (46081)
YYXX = FFFF – B401
YYXX = **4BFE** (YY = 4B = TH, XX = FE = TL)

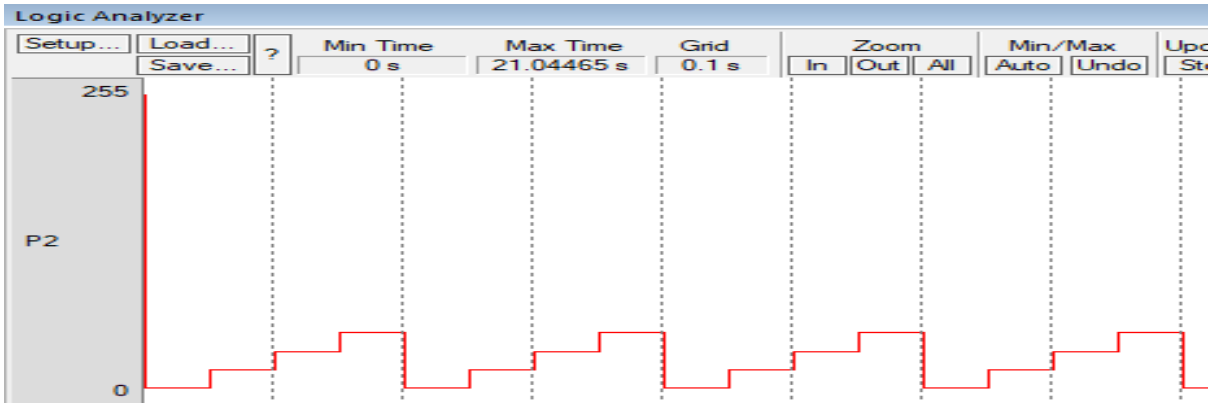
Block diagram:



Code:

```
#include "at89c51ed2.h"
void T1M1delay(void);
void main(void) {

   while(1) {
      P2=0x00;
      T1M1delay();
      P2=0x10;
      T1M1delay();
      P2=0x20;
      T1M1delay();
      P2=0x30;
      T1M1delay();

   }
}

void T1M1Delay(void) {

   TMOD=0X10;    // TIMER 1 MODE 1 (16-BIT MODE)
   TH1=0X4B;      // LOAD TH1 WITH COUNT 0x4B
   TL1=0XFE;      // LOAD TL1 WITH COUNT 0xFE
   TR1=1;         // START TIME
   while(TF1==0);  // WAIT FOR TF1 TO ROLL OVER
   TR1=0;         // TURN OFF T1
   TF1=0;         // CLEAR TF1

}
```

Expected output:



Conclusion: In this experiment, we successfully developed an '8051 C' program to implement a MOD-4 counter connected to Port 2. By employing Timer1 in Mode 1 to generate a 50 ms delay, we achieved smooth counting and display the counts. This project showcases the utilization of hardware delay for creating an efficient counter circuit with the 8051 microcontrollers.

Termwork 4

Objective: The objective of this experiment is to design an '8051 C' program to implement a MOD-4 counter, connected to Port 2 of the microcontroller. The counter should increment and display the count using hardware delay. Timer1 in Mode 2 will be utilized to generate a 25-microsecond delay between each count.

Brief Theory: The 8051 microcontrollers will be programmed in 'C' language to perform the MOD-4 counting operation. Port 2 of the microcontroller will be configured as an output port to drive the LEDs. The program will utilize Timer1 in Mode 2 to generate a delay of 25-microsecond between each count. The counter will increment from 0 to 3 and then repeat in a continuous loop, effectively creating a MOD-4 counter. The hardware delay ensures that the display is visible and stable between each count.

Calculations:

For Timer1Mode2, TMOD = 0x20

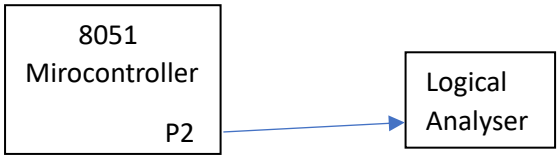
To calculate TH, use

$$\textbf{Delay} = ((FF - YY) + 1) \times 1.085 \times 10^{-6}$$

For 25-microsecond delay,

$25 \times 10^{-6} / 1.085 \times 10^{-6} = (FF - YY) + 1$
23 (approx. of 23.09) = (FF - YY) + 1
YY = FF – hex (22)
YY = FF – B401
YY = **E9** (YY = 4B = TH)

Block diagram:

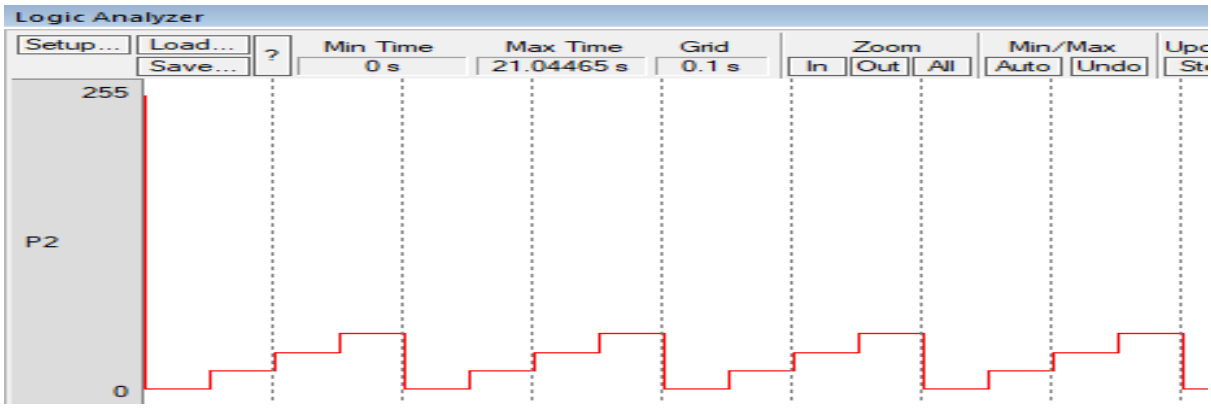


Code:

```
#include "at89c51ed2.h"
void T1M2delay(void);
void main(void)

{

  while(1)

  {

    P2=0x00;
    T1M2delay();
    P2=0x10;
    T1M2delay();
    P2=0x20;
    T1M2delay();
    P2=0x30;
    T1M2delay();

  }

}

void T1M2Delay(void){

  TMOD=0x20;
  TH1=E9;
```

```
    TR1=1;
    while (TF1==0);
    TR1=0;
    TF1=0;

}
```

Expected output:



Conclusion: In this experiment, we successfully developed an '8051 C' program to implement a MOD-4 counter connected to Port 2. By employing Timer1 in Mode 2 to generate a 25-microsecond delay, we achieved smooth counting and display the counts. This project showcases the utilization of hardware delay for creating an efficient counter circuit with the 8051 microcontrollers.

Termwork 5

Objective: The objective of this experiment is to develop an 8051 'C' program that utilizes the DAC 0800 interface to generate two distinct waveforms: a square wave and a triangular wave. The program aims to demonstrate the efficient generation of these waveforms through the digital-to-analog converter.

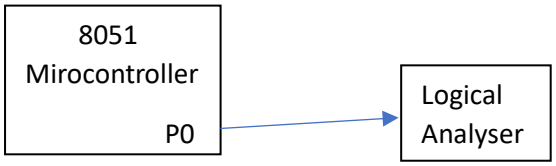
Brief Theory:

1. **Square Wave:** A square wave is a type of periodic waveform characterized by its constant amplitude and equal durations of high and low states. To generate a square wave using the DAC 0800 interface, the 8051 microcontroller outputs a digital signal with alternating high and low values, which the DAC converts to corresponding analog voltage levels, thereby creating the square waveform.

2. **Triangular Wave:** A triangular wave is another periodic waveform, but it varies linearly between its maximum and minimum amplitude values. To generate a triangular wave through the DAC 0800 interface, the 8051 microcontroller provides a gradually increasing and decreasing digital signal, which the DAC converts into an analog voltage, producing the triangular waveform.

NO CALCULATIONS

Block diagram:



Code:

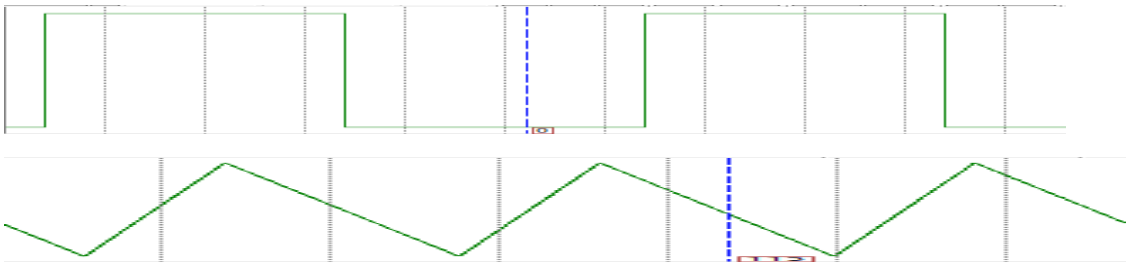
//Code For Square wave

```
#include "at89c51ed2.h"
void delay(unsigned int);
void main (){

   while(1){

      P0 = 0x00;
      delay(200);
      P0 = 0xff;
      delay(200);

   }

}

void delay(unsigned int itime){

   unsigned int i,j;
   for(i=0;i<itime;i++)

      for(j=0;j<1275;j++);

}
```

//Code For Triangular wave

```
#include "at89c51ed2.h"
unsigned char count;
void main (){

   while(1){

      for(count=0; count!=0xff; count++){
         P0=count;

      }

      for(count=0xff; count>0; count--){

         P0=count;

      }

   }

}
```

Expected Output:



Conclusion: The 8051 'C' program successfully demonstrated the generation of two essential waveforms, namely square and triangular, using the DAC 0800 interface. The square wave exhibited consistent amplitude and equal high and low durations, while the triangular wave showcased a linear variation between its peak and trough. This experiment underscores the practical application of DACs in producing precise analog output from digital signals.

Termwork 6

Objective: The objective of this experiment is to develop an 8051 'C' program that interfaces with the DAC 0800 to generate specific waveforms, namely a rectangular wave with a 70% duty cycle and a positive ramp waveform. The program aims to accurately produce these waveforms with the specified time period and duty cycle.

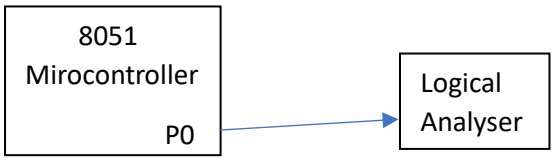
Brief theory:

1. **Rectangular wave** with 70% duty cycle: In a rectangular wave, the signal alternates between two levels - high and low. The duty cycle represents the ratio of time the signal stays at the high level compared to the total time period. To generate a rectangular wave with a 70% duty cycle, the DAC output will be set to the high level for 70% of the specified time period (T=100ms) and low for the remaining 30%.

2. **Positive Ramp:** A positive ramp waveform is a continuously ascending linear signal. To generate this waveform, the DAC output voltage will be varied incrementally at regular intervals during the specified time period (T=100ms). This will result in a smooth and linearly increasing output voltage, creating the positive ramp waveform.

NO CALCULATIONS

Block diagram:



Code:

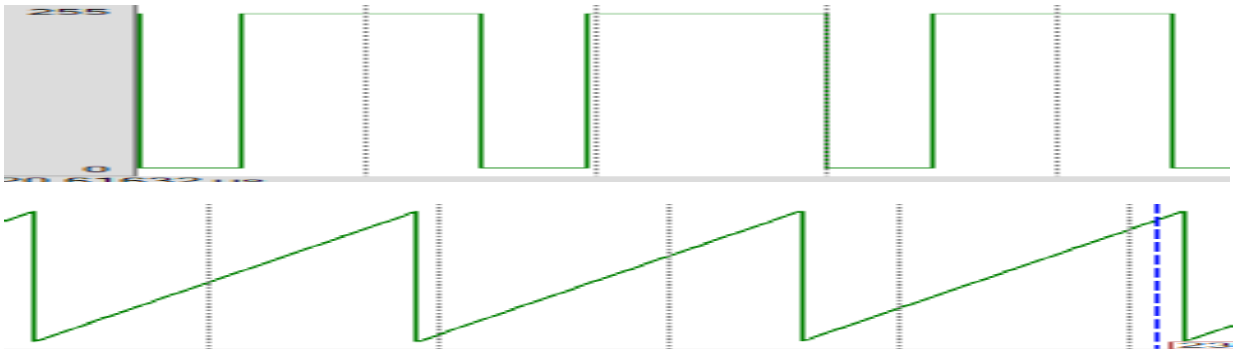
//Code For Rectangular wave

```
#include "at89c51ed2.h"
void delay(unsigned int);
void main (){

  while(1){

    P0 = 0x00;
    delay(70);        // 30 ms off time(70% of 100ms)
    P0 = 0xff;
    delay(30);        // 70 ms on time(100 – 70% of 100ms)

  }

}

void delay(unsigned int itime){

  unsigned int i,j;
  for(i=0;i<itime;i++)
    for(j=0;j<1275;j++);

}
```

//Code For Positive ramp wave

```
#include "at89c51ed2.h"
unsigned char count;
void main (){
  while(1){
    for(count=0; count!=0xff; count++)
      P0=count;
  }
}
```

Expected output:



Conclusion: By implementing the 'C' program for 8051 microcontrollers with DAC 0800 interface, we successfully generated two distinct waveforms - a rectangular wave with a 70% duty cycle and a positive ramp waveform. These waveforms can be utilized in various applications, such as signal generation, testing, and calibration processes. The accuracy of the waveforms demonstrates the effectiveness of the program and its capability to control the DAC output accurately.

Termwork 7

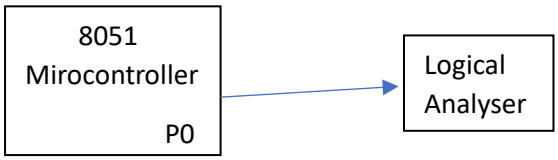
Objective: The objective of this experiment is to develop an 8051 'C' program that utilizes the DAC (Digital-to-Analog Converter) interface to generate two specific waveforms: a square wave and a negative ramp.

Brief Theory:

1. **Square Wave**: A square wave is a type of periodic waveform that alternates between two distinct voltage levels, typically high and low, with equal durations. In this experiment, the 8051 microcontroller will be programmed to output a series of digital values to the DAC, which will convert these digital values into corresponding analog voltages, producing the square wave.
2. **Negative Ramp**: A negative ramp is a waveform that starts at a higher voltage level and linearly decreases in amplitude over time until it reaches a lower voltage level. Similarly to the square wave, the 8051 microcontroller will be utilized to generate a series of digital values for the DAC, resulting in the desired negative ramp waveform.

NO CALCULATIONS

Block diagram:



Code:

//Code For Square wave

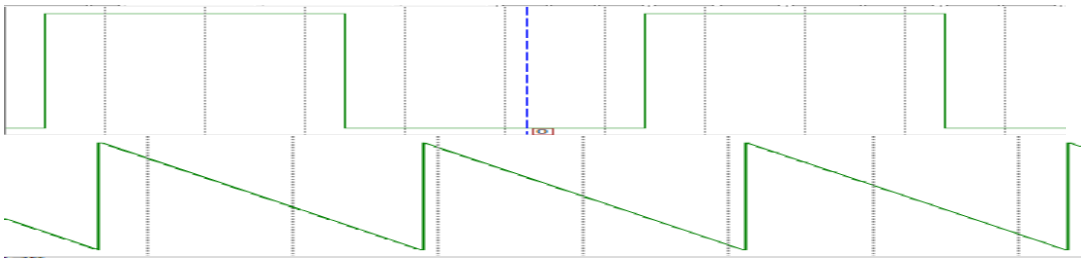
```
#include "at89c51ed2.h"
void delay(unsigned int);
void main (){
  while(1){

    P0 = 0x00;
    delay(200);
    P0 = 0xff;
    delay(200);
  }
}
```

```
void delay(unsigned int itime) {

   unsigned int i,j;
   for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
}
```

//Code For negative ramp wave

```
#include "at89c51ed2.h"
unsigned char count;
void main (){

   while(1){

      for(count=0xff; count>=0; count---)
         P0=count;
    }
}
```

Expected Output:



Conclusion: By implementing the 8051 'C' program and utilizing the DAC interface, we successfully generated two distinct waveforms: a square wave with alternating high and low voltage levels, and a negative ramp waveform with a linear voltage decrease over time. This experiment demonstrates the capability of the 8051 microcontrollers to control external hardware and generate precise waveforms using the DAC interface

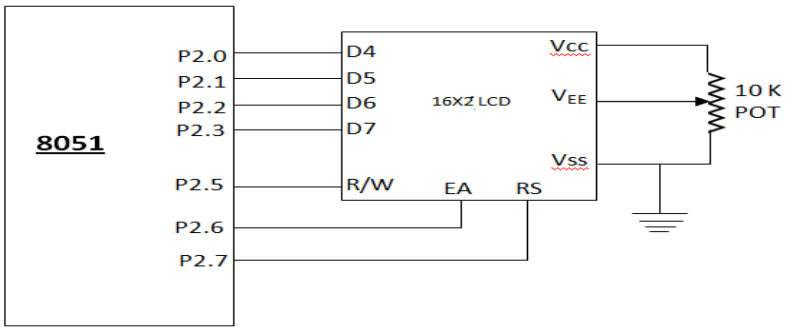
Termwork 8:

Objective: The objective of this experiment is to develop an 8051 'C' program to interface a 2x16 LCD display and display the strings "KLS GIT" and "ESIoT LAB" starting from the 1st position on both lines.

Brief Theory: The 8051 microcontroller is a widely used microcontroller that can be programmed in the 'C' language. The 2x16 LCD display is a common alphanumeric display used to show information in embedded systems. Interfacing the 8051 with the LCD involves connecting the appropriate pins and using 'C' code to send commands and data to the LCD for displaying characters.

NO CALCULATIONS

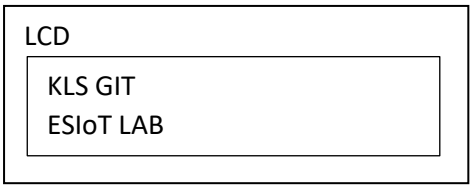
Block Diagram:



Code:

```
#include "at89c51ed2.h"
#include<intrins.h>
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);
unsigned char xdata arr[8] = {"KLS GIT"};            // 8 spaces including the \0 character
unsigned char xdata arr1[10] = {"ESIoT LAB"};        // 10 spaces including the \0 character
unsigned char temp1 = 0x00;
unsigned char temp2;
unsigned int i = 0;
void main(void){

  AUXR = 0x10;
  lcd_init();

  temp1 = 0x80;
  lcd_comm();

  for (i = 0; i < 8; i++){

    temp2 = arr[i];
    lcd_data();

  }

  temp1 = 0xC0;
  lcd_comm();

  for (i = 0; i < 10; i++){

    temp2 = arr1[i];
    lcd_data();

  }
}
```

Expected Output:



Conclusion: In this experiment, we successfully interfaced a 2x16 LCD display with the 8051-microcontroller using 'C' programming. The displayed strings "KLS GIT" and "ESIoT LAB" on the LCD confirmed the proper functionality of the interface. This project demonstrates a fundamental aspect of embedded systems programming and can be extended to create more complex applications.

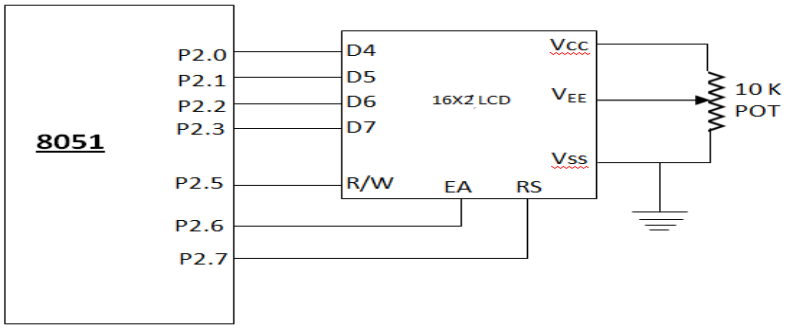
Termwork 9

Objective: The objective of this experiment is to develop an 8051 'C' program to interface a 2x16 LCD display and display the strings "CSE" and "BRANCH" starting from the 6th position on both lines.

Brief Theory: The 8051 microcontroller is a widely used microcontroller that can be programmed in the 'C' language. The 2x16 LCD display is a common alphanumeric display used to show information in embedded systems. Interfacing the 8051 with the LCD involves connecting the appropriate pins and using 'C' code to send commands and data to the LCD for displaying characters.

NO CALCULATIONS

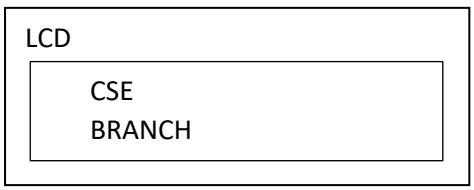
Block Diagram:



Code:

```
#include "at89c51ed2.h"
#include<intrins.h>
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);
unsigned char xdata arr[4] = {"CSE"};
unsigned char xdata arr1[7] = {"BRANCH"};
unsigned char temp1 = 0x00;
unsigned char temp2;
unsigned int i = 0;
void main(void){

   AUXR = 0x10;
   lcd_init();

   temp1 = 0x86;
   lcd_comm();

   for (i = 0; i < 4; i++){

      temp2 = arr[i];
      lcd_data();

   }

   temp1 = 0xC6;
   lcd_comm();

   for (i = 0; i < 7; i++){

      temp2 = arr1[i];
      lcd_data();

   }

}
```

Expected Output:



Conclusion: In this experiment, we successfully interfaced a 2x16 LCD display with the 8051-microcontroller using 'C' programming. The displayed strings "KLS GIT" and "ESIoT LAB" on the LCD confirmed the proper functionality of the interface. This project demonstrates a fundamental aspect of embedded systems programming and can be extended to create more complex applications.

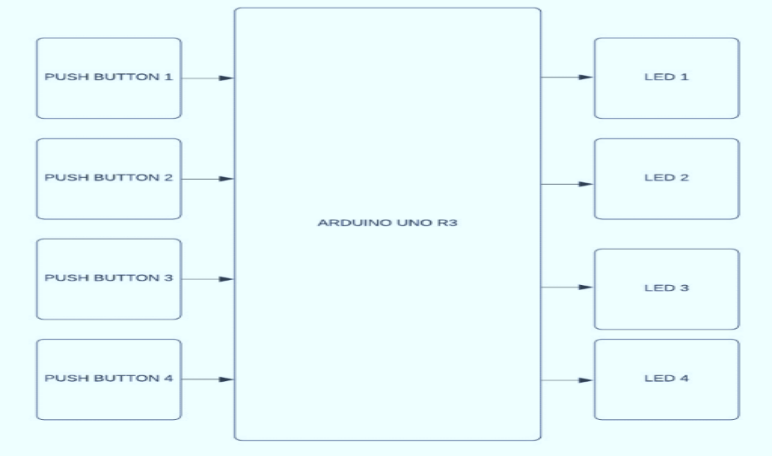
Termwork 10

Objective: The objective of this experiment is to create an embedded 'C' program for an Arduino Single Board Computer (SBC) that enables blinking LEDs upon the pressing of push buttons.

Brief Theory: In this experiment, an embedded 'C' program is utilized to control the LEDs connected to the Arduino SBC. The program monitors the status of the push buttons and triggers the LEDs to blink accordingly. The digital input pins of the Arduino read the button states, and the digital output pins control the LEDs' illumination.

NO CALCULATIONS

Block Diagram



Code:

```
const int ledPin1 = A5;
const int ledPin2 = A4;
const int ledPin3 = A3;
const int ledPin4 = A2;

const int buttonPin1 = 13;
const int buttonPin2 = 12;
const int buttonPin3 = 11;
const int buttonPin4 = 10;

int buttonState1 = LOW;
int buttonState2 = LOW;
int buttonState3 = LOW;
int buttonState4 = LOW;

void setup() {

  Serial.begin(9600);
  pinMode(buttonPin1, INPUT);
  pinMode(buttonPin2, INPUT);
  pinMode(buttonPin3, INPUT);
  pinMode(buttonPin4, INPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
```

```
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);

}

void loop() {

  buttonState1 = digitalRead(buttonPin1);
  buttonState2 = digitalRead(buttonPin2);
  buttonState3 = digitalRead(buttonPin3);
  buttonState4 = digitalRead(buttonPin4);

  if (buttonState1 == HIGH)
    digitalWrite(ledPin1, LOW);
  else
    digitalWrite(ledPin1, HIGH);

  if (buttonState2 == HIGH)
    digitalWrite(ledPin2, LOW);
  else
    digitalWrite(ledPin2, HIGH);

  if (buttonState3 == HIGH)
    digitalWrite(ledPin3, LOW);
  else
    digitalWrite(ledPin3, HIGH);

  if (buttonState4 == HIGH)
    digitalWrite(ledPin4, LOW);
  else
    digitalWrite(ledPin4, HIGH);

  Serial.println(buttonState1);
  Serial.println(buttonState2);
  Serial.println(buttonState3);
  Serial.println(buttonState4);

  delay(10);

}
```

Expected Output:

Conclusion: By successfully developing the embedded 'C' program for the Arduino SBC, the experiment achieves its objective of making the LEDs blink in response to pressing the push buttons. This demonstrates the practical application of embedded programming in simple interfacing tasks, enhancing the understanding of hardware-software interactions.

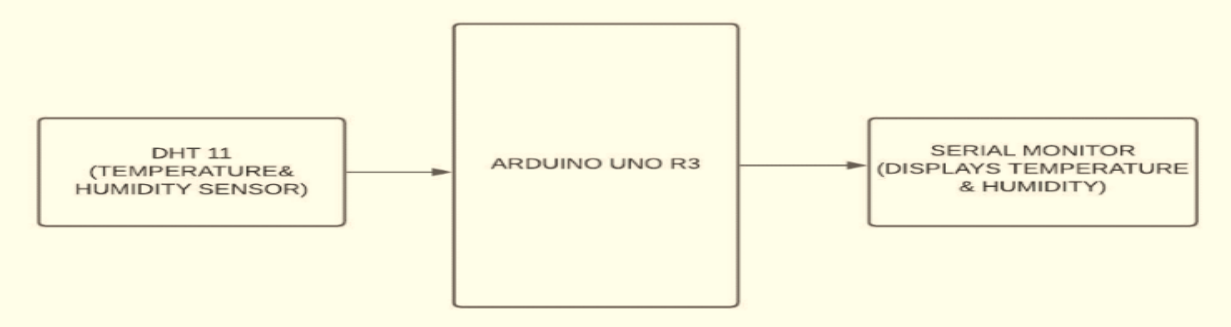
Termwork 11

Objective: The objective of this experiment is to develop an Embedded 'C' program to interface the DHT11 sensor with an Arduino Single Board Computer (SBC) and display the acquired sensor data on the serial monitor. This program will enable real-time monitoring and analysis of temperature and humidity data from the DHT11 sensor.

Brief theory: The DHT11 is a low-cost digital temperature and humidity sensor that uses a capacitive humidity sensor and a thermistor for temperature measurement. It communicates with the Arduino SBC using a one-wire protocol. The 'C' programming language is utilized to develop the firmware for the Arduino board, allowing the device to interact with the DHT11 sensor. The data received from the sensor is then processed and displayed on the serial monitor, providing users with the temperature and humidity readings.

NO CALCULATIONS

Block Diagram



Code:

```
#include <dht.h>
dht DHT;
#define DHT11_PIN 4

void setup() {

   Serial.begin(9600);

}

void loop() {

   int chk = DHT.read11(DHT11_PIN);
   Serial.print("Temperature = ");
   Serial.println(DHT.temperature);
   Serial.print("Humidity = ");
   Serial.println(DHT.humidity);

   delay(2000);

}
```

Expected Output:

```
Humidity = 60.00
Temperature = 30.00
Humidity = 60.00
Temperature = 31.00
Humidity = 81.00
Temperature = 31.00
Humidity = 85.00
Temperature = 31.00
Humidity = 86.00
Temperature = 31.00
Humidity = 84.00
Temperature = 31.00
Humidity = 81.00
Temperature = 31.00
Humidity = 76.00
Temperature = 31.00
Humidity = 73.00
Temperature = 31.00
Humidity = 71.00
```

Conclusion: In conclusion, this experiment successfully demonstrates the interfacing of the DHT11 sensor with an Arduino SBC using an Embedded 'C' program. By following the provided code and setup, users can acquire real-time temperature and humidity data from the DHT11 sensor and monitor it through the Arduino's serial monitor. This project lays the foundation for more advanced applications involving sensor interfacing and data visualization.

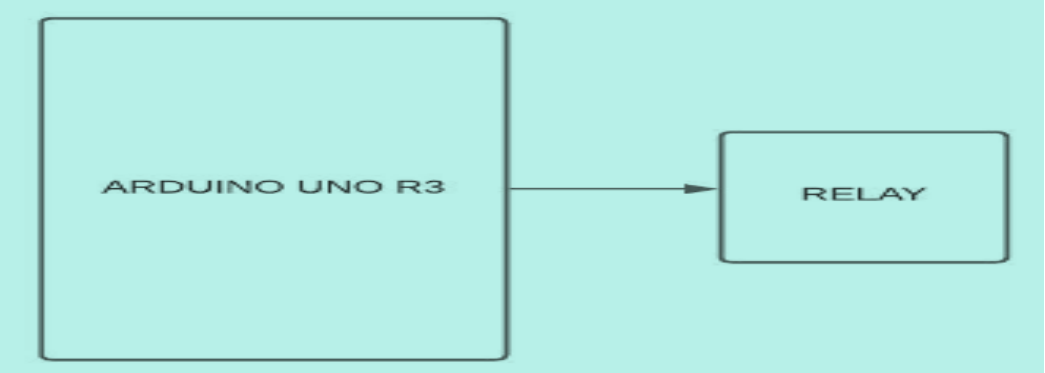
Termwork 12

Objective: The objective of this experiment is to create an embedded 'C' program that can effectively control a relay using an Arduino UNO board. The program aims to demonstrate how to interface and manipulate the relay, enabling users to control external devices and systems through the Arduino platform.

Brief theory: In this experiment, an Arduino UNO is utilized as a microcontroller to manage the relay. The Arduino's GPIO pins are used to communicate with the relay module, which acts as a switch to control the flow of current to external devices. By programming the Arduino in embedded 'C,' the microcontroller can be instructed to toggle the relay state based on user input or specific conditions, thereby controlling the connected external load.

NO CALCULATIONS

Block Diagram



Code:

```
int relayPin = 8;
void setup() {

  Serial.begin(9600);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);

}

void loop() {

  digitalWrite(relayPin, LOW);
  delay(1000);
  digitalWrite(relayPin, HIGH);
  delay(1000);

}
```

Expected Output:

Conclusion: Through the development of the embedded 'C' program on Arduino UNO, we have successfully achieved relay control. The experiment demonstrates the fundamental principles of interfacing hardware components with a microcontroller and showcases the potential applications of using Arduino for automation and control systems. This project serves as a stepping stone for more advanced projects involving embedded systems and smart devices.

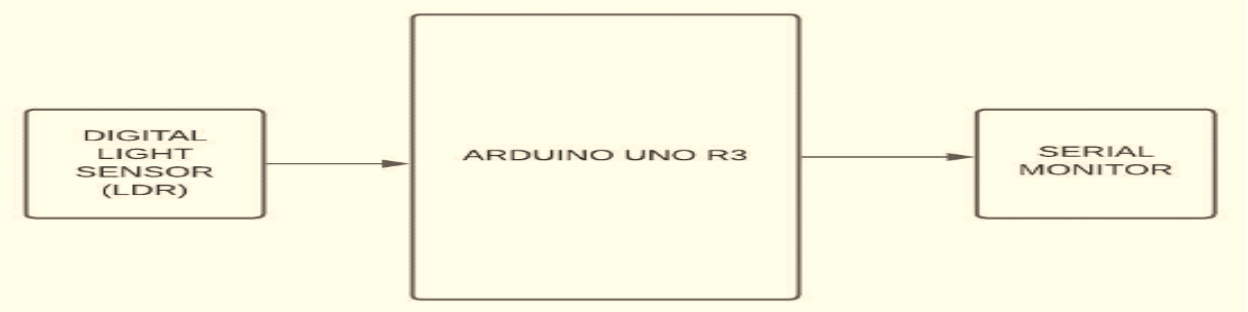
Termwork 13

Objective: The objective of this experiment is to create an Embedded 'C' program that interfaces the Light Dependent Resistor (LDR) sensor with an Arduino Single Board Computer (SBC) and then displays the acquired sensor data on the serial monitor.

Brief Theory: The Light Dependent Resistor (LDR) is a passive electronic component that varies its resistance based on the intensity of light falling on it. The Arduino Single Board Computer (SBC) is a microcontroller-based platform capable of running programs to interact with various electronic components. In this experiment, we will use Embedded 'C' programming to read the analog data from the LDR sensor using the Arduino's Analog-to-Digital Converter (ADC). The acquired data will then be displayed on the Arduino's serial monitor, allowing real-time observation of the light intensity detected by the LDR sensor.

NO CALCULATIONS

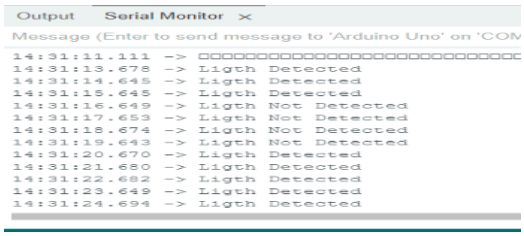
Block Diagram:



Code:

```
int lightPin = 5;
void setup(){

  Serial.begin(9600);
  pinMode(lightPin, INPUT);
}

void loop(){

  int light_data = digitalReal (lightPin);
  if(light_data)
    Serial.println("Light Not Detected!");
  else
    Serial.println("Light Detected!");

  delay(1000);
}
```

Expected Output:



Conclusion: By successfully interfacing the LDR sensor with the Arduino SBC and displaying the acquired data on the serial monitor, we have achieved a functional implementation of a light intensity monitoring system. This experiment showcases the potential of using Embedded 'C' programming and Arduino for sensor interfacing applications. Further improvements and additions to the code could lead to more advanced projects in the field of data acquisition and monitoring.

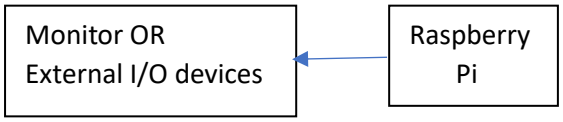
Termwork 14

Objective: The objective of this experiment is to develop a Python program for Raspberry Pi that can control a buzzer, allowing it to be turned "ON" and "OFF" programmatically.

Brief Theory: Raspberry Pi is a popular single-board computer capable of running Python programs. To control the buzzer, we will use a GPIO (General Purpose Input/Output) pin on the Raspberry Pi. GPIO pins can be configured as either input or output, and in this case, we will set the pin as an output to control the buzzer. By sending appropriate signals to the GPIO pin, we can turn the buzzer "ON" and "OFF."

NO CALCULATIONS

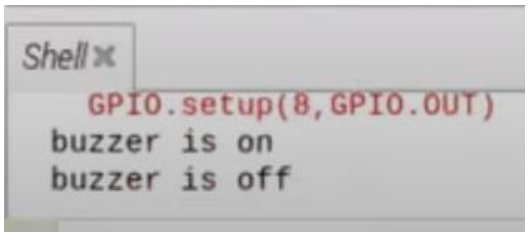
Block Diagram:



Code:

```
import time
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(38, GPIO.OUT)

try:

  while(1):

    GPIO.output(38,0)
    print("Buzzer OFF")
    time.sleep(1)
    GPIO.output(38,1)
    print("Buzzer ON")
    time.sleep(1)

except KeyboardInterrupt:

    GPIO.cleanup()
    exit
```

Expected Output



Conclusion: In conclusion, we have successfully developed a Python program for Raspberry Pi that can control a buzzer. By utilizing the GPIO capabilities of the Raspberry Pi, we can conveniently toggle the buzzer state programmatically. This program can be used in various applications, such as home automation, security systems, or simple interactive projects.

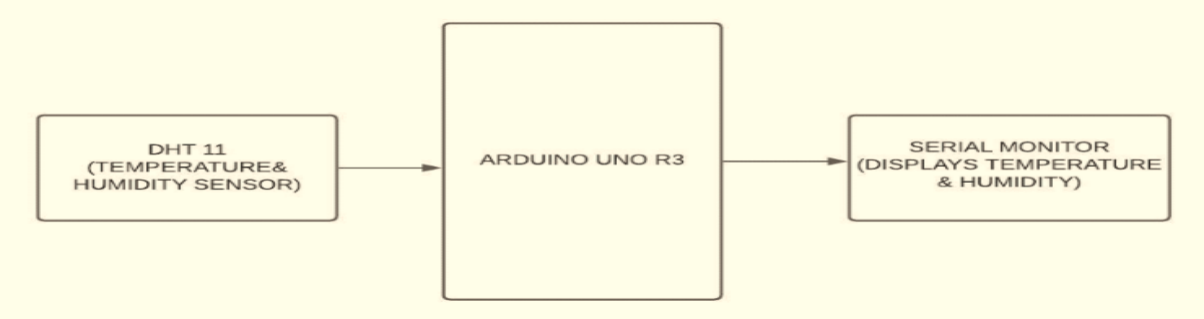
Termwork 15

Objective: The objective of this experiment is to develop an Embedded 'C' program that interfaces the DHT11 sensor with the Arduino Single Board Computer (SBC). The program aims to acquire data from the sensor and display it on the serial monitor. Additionally, the program will control a relay to turn it on when the temperature measured by the DHT11 sensor exceeds 22 degrees Celsius.

Brief theory: The DHT11 sensor is a temperature and humidity sensor that can be interfaced with an Arduino SBC through its digital pins. The sensor provides temperature and humidity readings in a digital format, which can be read by the Arduino using the appropriate 'C' code. The Arduino IDE provides libraries to communicate with the DHT11 sensor easily.

NO CALCULATIONS

Block Diagram:



Code:

```
#include <dht.h>
dht DHT;
const int dht11Pin = 4;
const int relayPin = 8;

void setup() {

  Serial.begin(9600);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);

}

void loop() {

  int chk = DHT.read11(dht11Pin);
  Serial.print("Temperature = ");
  Serial.println(DHT.temperature);

  if(DHT.temperature)
    digitalWrite(relayPin, HIGH);
 else
    digitalWrite(relayPin, LOW);

  delay(1000);

}
```

Expected Output:

Conclusion: In conclusion, the Embedded 'C' program successfully interfaces the DHT11 sensor with the Arduino SBC and displays the acquired temperature and humidity data on the serial monitor. Additionally, it intelligently controls the relay to turn on external devices when the temperature exceeds 22 degrees Celsius. This experiment demonstrates how to efficiently use the Arduino platform for sensor interfacing and data manipulation, enabling various applications in the field of temperature-based automation and control systems.