UNIT I:

BLOOM'S LEVEL 2: UNDERSTAND

1. DIFFRENTIATE BETWEEN MICROPROCESSORS AND MICROCONTROLLERS WITH A NEAT BLOCK DIAGRAM.

2. LIST AND EXPLAIN THE FOUR MAJOR DESIGN RULES OF RISC PHILOSOPHY.

3. DIFFERTIATE BETWEEN RISC AND CISC PROCESSORS.

4. LIST AND EXPLAIN IN DETAIL THE ARM DESIGN PHILOSOPHY.

5. JUSTIFY WHY ARM INSTRUCTION SET IS SUITABLE FOR EMBEDDED APPLICATIONS.

6. WITH A NEAT BLOCK DIAGRAM OF AN ARM-BASED EMBEDDED DEVICE, EXPLAIN THE FOLLOWING:

- ARM PROCESSOR
- CONTROLLERS
- PERIPHERALS
- BUS

7. WRITE A NOTE ON THE FOLLOWING:

- ARM BUS TECHNOLOGY
- AMBA BUS PROTOCOL
- MEMORY
- PERIPHERALS

8. WITH A NEAT BLOCK DIAGRAM EXPLAIN THE ARM CORE DATA FLOW MODEL.

9. LIST OUT THE VARIOUS REGISTERS OF ARM 7. COMMENT ON ITS WIDTH, AND SPECIAL PURPOSE OF REGISTERS R13, R14 AND R15.

10. DRAW THE NEAT BLOCK DIAGRAM OF CPSR AND COMMENT ON THE SIGNIFICANCE OF **N, Z, C AND V** FLAGS?

11. LIST THE VARIOUS MODES OF OPERATION OF ARM 7.

12. DEFINE PIPELINE. HOW MANY STAGES OF PPELINE IS AVAILABLE FOR ARM7. ILLUSTRATE THE PIPELINE OPERATION FOR THE FOLLOWING INSTRUCTIONS:

a. ADD R0,R1,R2

b. AND R3,R4,R5

c. SUB R6,R7,R8

BLOOM'S LEVEL 3: APPLY

1. WHICH OF THE FOLLOWING STATEMENTS ARE TRUE WITH RESPECT TO ARM 7 ARCHITECTURE.

a. EACH PROCESSOR MODE IS EITHER PREVILEGED OR NONPREVILEGED.

b. PREVILEGED MODE ALLOWS FULL READ WRITE ACCESS TO THE CPSR.

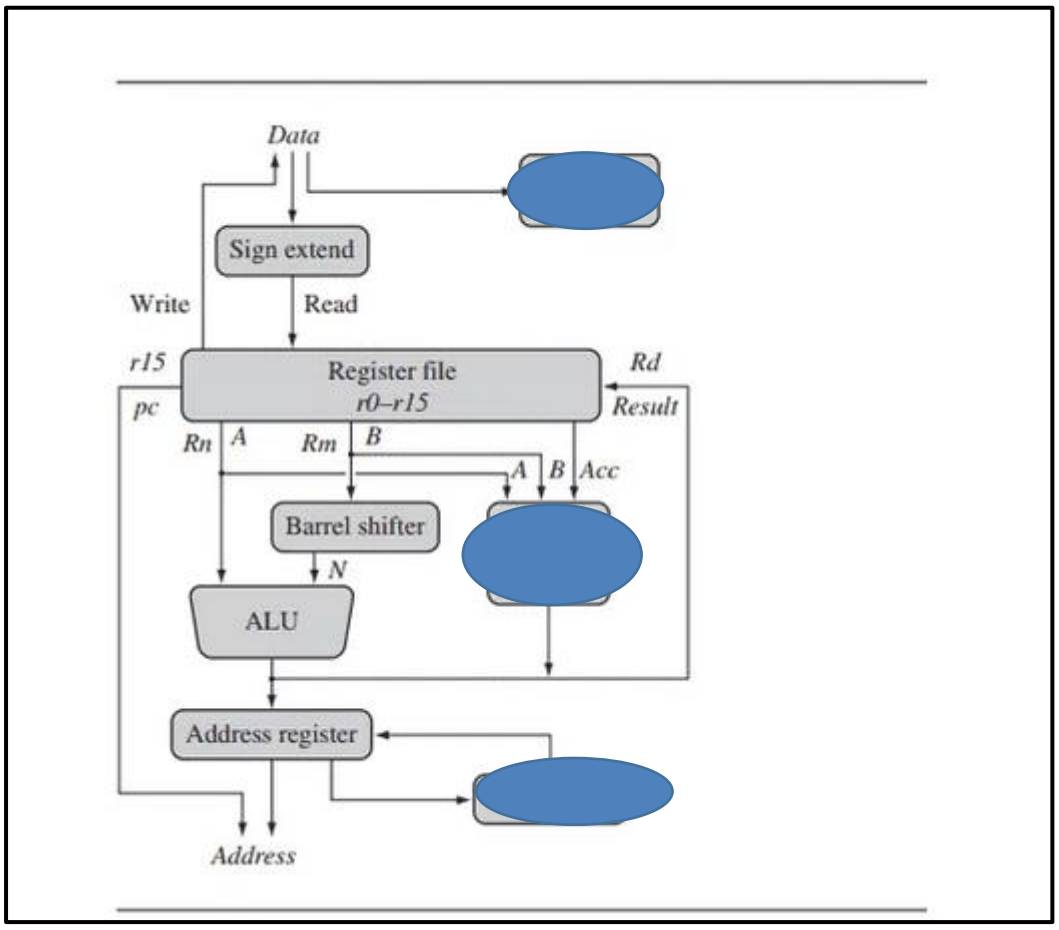
c. THE NEGATIVE FLAG ‘N’ IS SET WHEN BIT 31 OF THE RESULT IS BINARY 1.

d. THE ZERO FLAG ‘Z’ IS USED TO INDICATE EQUALITY.

e. THE CARRY FLAG ‘C’ IS SET WHEN THE RESULT CAUSES AN UNSIGNED CARRY.

f. THE OVERFLOW FLAG ‘V’ IS SET WHEN THE RESULT CAUSES SIGNED OVERFLOW.

BLOOM’S LEVEL 4: ANALYZE: ANALYZE THE ARM CORE DATAFLOW MODEL SHOWN IN FIGURE BELOW AND IDENTIFY THE MASKED BLOCKS AND THEIR SIGNIFICANCE.



UNIT 2:

BLOOM'S LEVEL 2: UNDERSTAND

1. LIST AND EXPLAIN THE VARIOUS DATA TRANSFER INSTRUCTIONS OF ARM7 WITH PROPER SYNTAX AND AN EXAMPLE.

2. WITH A NEAT BLOCK DIAGRAM EXPLAIN THE SIGNIFICANCE OF BARRE SHIFTER AND ALU.

3. LIST AND EXPLAIN THE BASIC 'C' DATA TYPES.

4. LIST AND EXPLAIN THE FOLLOWING INSTRUCTIONS OF ARM7 WITH PROPER SYNTAX AND AN EXAMPLE FOR EACH.

a. SHIFT INSTRUCTIONS
b. ROTATE INSTRUCTIONS
c. ARITHMETIC INSTRUCTIONS
d. LOGICAL INSTRUCTIONS
e. COMPARISON INSTRUCTIONS
f. MULTIPLY INSTRUCTIONS
g. BRANCH INSTRUCTIONS
h. LOAD STORE INSTRUCTIONS
i. SWAP INSTRUCTION
j. PROGRAM STATUS REGISTER INSTRUCTIONS

BLOOM'S LEVEL 3: APPLY

1. DEVELOP AN ASSEMBLY LANGUAGE PROGRAM (ALP) TO PERFORM BLOCK DATA TRANSFER.

2. DEVELOP AN ALP TO GENERATE THE SERIES: 5, 10,15,20,25. HINT: USE MLA INSTRUCTION.

3. DEVELOP AN ALP TO COMPUTE THE FACTORIAL OF A GIVEN NUMBER AND STORE THE RESULT IN RAM LOCATION.

4. DEVELOP AN ALP FIND THE LARGEST NUMBER IN AN ARRAY AND STORE IT IN RAM LOCATION.

5. DEVELOP AN ALP TO ILLUSTRATE THE SIGNIFICANCE OF LOGICAL OPERATIONS.

6. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF SHIFT AND ROTATE INSTRUCTIONS.

7. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF SWAP INSTRUCTION.

```
relay.c | l1.c | counter.c | square.c | Startup.s | tri.c
 1      AREA sw,CODE,READONLY
 2          ENTRY
 3          LDR R2,=BLOCK1
 4          LDR R3,=BLOCK2
 5          LDR R0,[R2]
 6          STR R0,[R3]
 7          MOV R0,#00
 8          MOV R1,#0X8000000F
 9          ;MOV R1,#0X10000002
10          SWP R0,R1,[R3]
11  BLOCK1 DCD 0X12345678
12  L       B L
13      AREA MYDATA,DATA,READWRITE
14  BLOCK2 DCD 0
15      END
```

8. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF LOAD STORE INSTRUCTIONS

```
relay.c | l1.c | counter.c | square.c | Startup.s | tri.c* | block.s
 1      AREA block,CODE,READONLY
 2          ENTRY
 3          MOV R0,#5
 4          LDR R1,=FBLOCK
 5          LDR R2,=SBLOCK
 6  GOTO    LDRH R3,[R1],#1
 7          STRH R3,[R2],#1
 8          SUBS R0,#1
 9          BNE GOTO
10  L       B L
11  FBLOCK DCW 0X1234,0X4567,0X9ABC,0XDEFO,0X9876
12      AREA MYDATA,DATA,READWRITE
13  SBLOCK DCW 0
14      END
```

9. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF PROGRAM STATUS REGISTER INSTRUCTIONS

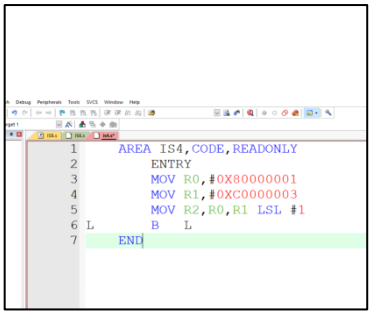
```
l1.c | counter.c | square.c | Startup.s | tri.c* | block.s | sw.s | sw2.s | sw3.s | sw5.s | psr.s
    AREA psr,CODE,READONLY
        ENTRY
        MRS R0, CPSR      // Move the value of CPSR into register R0
        ORR R0, R0, #0x5F // Set the mode bits to User mode (0x10) and enable interrupts
        MSR CPSR, R0      // Move the modified value back to CPSR

        // Your program logic goes here

        // Example: Perform some operations
        MOV R1, #10       // Load the value 10 into register R1
        ADD R2, R1, #5    // Add 5 to the value in R1 and store the result in R2
L       B L
    END
```

BLOOM'S LEVEL 4: ANALYZE:

1. ANALYZE THE GIVEN PIECE OF CODE AND ANSWER THE FOLLOWING:

a. WHAT IS THE CONTENT OF R0,R1 AND R2.

b. COMMENT ON THE STATUS OF NZCV FLAGS AFTER EXECUTING THE LAST INSTRUCTION.



```
1     AREA IS4,CODE,READONLY
2         ENTRY
3         MOV R0,#0X80000001
4         MOV R1,#0XC0000003
5         MOV R2,R0,R1 LSL #1
6 L       B   L
7     END
```

2. ANALYZE THE GIVEN PIECE OF CODES (‘C’ CODE AND COMPILER OUTPUT) AND ANSWER THE FOLLOWING:

```
int checksum_v1(int *data)
{
  char i;
  int sum=0;

  for (i=0; i<64; i++)
  {
    sum += data[i];
  }
  return sum;
}
```

```
checksum_v1
        MOV     r2,r0               ; r2 = data
        MOV     r0,#0               ; sum = 0
        MOV     r1,#0               ; i = 0
checksum_v1_loop
        LDR     r3,[r2,r1,LSL #2]   ; r3 = data[i]
        ADD     r1,r1,#1            ; r1 = i+1
        AND     r1,r1,#0xff         ; i = (char)r1
        CMP     r1,#0x40            ; compare i, 64
        ADD     r0,r3,r0            ; sum += r3
        BCC     checksum_v1_loop    ; if (i<64) loop
        MOV     pc,r14              ; return sum
```

- WHAT IS THE DRAWBACK OF USING CHAR DATA TYPE FOR DECLARING THE LOCAL VARIABLES IN ARM7 ‘C’ PROGRAM?
- IN THE COMPILER OUTPUT HOW CAN WE AVOID THE INSTRUCTION *AND R1,R1,#0XFF*
- WHAT IS THE USE OF BCC INSTRUCTION?
- WHY PC IS UPDATED WITH R14 CONTENT? CAN WE REPLACE R14 BY ANY OTHER REGISTER?

3. ANALYZE THE GIVEN PIECE OF CODE AND ANSWER THE FOLLOWING:

- WHICH DATA TYPE IS USED TO DECLARE THE LOCAL VARIABLE?:int or long int
- WHAT IS THE MODIFICATION THAT IS REQUIRED IN THE COMPILER OUTPUT IF THE VARIABLE SUM IS 16-BIT.

```
checksum_v2
        MOV     r2,r0               ; r2 = data
        MOV     r0,#0               ; sum = 0
        MOV     r1,#0               ; i = 0
checksum_v2_loop
        LDR     r3,[r2,r1,LSL #2]   ; r3 = data[i]
        ADD     r1,r1,#1            ; r1++
        CMP     r1,#0x40            ; compare i, 64
        ADD     r0,r3,r0            ; sum += r3
        BCC     checksum_v2_loop    ; if (i<64) goto loop
        MOV     pc,r14              ; return sum
```

After ADD r0,r3,r0 we should write:

```
MOV     r0,r0,LSL #16
MOV     r0,r0,ASR #16       ; sum = (short)r0
```

4. ANALYZE THE GIVEN PIECE OF CODES (‘C’ CODE AND COMPILER OUTPUT) AND ANSWER THE FOLLOWING:

```
short checksum_v3(short *data)
{
  unsigned int i;
  short sum=0;

  for (i=0; i<64; i++)
  {
    sum = (short)(sum + data[i]);
  }
  return sum;
}
```

```
checksum_v3
        MOV     r2,r0           ; r2 = data
        MOV     r0,#0           ; sum = 0
        MOV     r1,#0           ; i = 0
checksum_v3_loop
        ADD     r3,r2,r1,LSL #1 ; r3 = &data[i]
        LDRH    r3,[r3,#0]      ; r3 = data[i]
        ADD     r1,r1,#1        ; i++
        CMP     r1,#0x40        ; compare i, 64
        ADD     r0,r3,r0        ; r0 = sum + r3
        MOV     r0,r0,LSL #16
        MOV     r0,r0,ASR #16   ; sum = (short)r0
        BCC     checksum_v3_loop ; if (i<64) goto loop
        MOV     pc,r14          ; return sum
```

- HOW CAN WE REDUCE THESE INSTRUCTIONS IN THE COMPILER OUTPUT?
  - ```
    ADD     r3,r2,r1,LSL #1
    MOV     r0,r0,LSL #16
    ```
  - ```
    MOV     r0,r0,ASR #16
    ```
- REWRITE THE ‘C’ CODE TO REDUCE THESE INSTRUCTIONS.

C code:

```
short checksum_v4(short *data)
{
  unsigned int i;
  int sum=0;

  for (i=0; i<64; i++)
  {
    sum += *(data++);
  }
  return (short)sum;
}
```

compiler code:

```
checksum_v4
        MOV     r2,#0           ; sum = 0
        MOV     r1,#0           ; i = 0
checksum_v4_loop
        LDRSH   r3,[r0],#2      ; r3 = *(data++)
        ADD     r1,r1,#1        ; i++
        CMP     r1,#0x40        ; compare i, 64
        ADD     r2,r3,r2        ; sum += r3
        BCC     checksum_v4_loop ; if (sum<64) goto loop
        MOV     r0,r2,LSL #16
        MOV     r0,r0,ASR #16   ; r0 = (short)sum
        MOV     pc,r14          ; return r0
```

UNIT 3:

LEVEL 2:

1. LIST AND EXPLAIN THE VARIOUS REGISTERS OF ARM 7 USED FOR CONFIGURING PORTS AS GPIO, INPUT/OUTPUT AND SET/CLEAR .
2. WHAT VALUE HAS TO BE LOADED INTO THE REGISTERS
   - TO CONFIGURE PORT 0 (P0.0-P0.15) PINS AS INPUT? `IODIR_PORT0`=0X00000000;OR IO0DIR=0X0000;
   - TO CONFIGURE PORT 0 (P0.15-P0.31) PINS AS INPUT? `IODIR_PORT0`=0X0000FFFF;
   - TO CONFIGURE PORT 0 (P0.0-P0.15) PINS AS OUTPUT? `IODIR_PORT0`=0X0000FFFF;

- TO CONFIGURE PORT 0 (P0.15-P0.31) PINS AS OUTPUT?IO0DIR=0XFFFF0000;
- TO CONFIGURE PORT 1 (P1.16-P1.31) PINS AS INPUT? `PINSEL_PORT1`=0X0000FFFF;
- TO CONFIGURE PORT 1 (P1.16-P1.31) PINS AS OUTPUT? `IODIR_PORT1`=0XFFFF0000;
- WHETHER THE PINS P1.0-P1.15 ARE AVAILABLE AS GPIO?NO

3. LIST AND EXPLAIN THE REGISTERS USED TO CONTROL GPIO REGISTERS(IOPIN, IODIR,IOSET,IOCLR)

1. **IOPIN (I/O Pin Register):**
   - **Function:** Reads the current state (high or low) of the individual pins of a GPIO port.
   - **Explanation:** Each bit in the IOPIN register represents the state of the corresponding pin in the GPIO port. If a bit is set (1), it indicates that the corresponding pin is high; if it is cleared (0), the pin is low.
2. **IODIR (I/O Direction Register):**
   - **Function:** Configures the direction of the individual pins (input or output) for a GPIO port.
   - **Explanation:** Each bit in the IODIR register corresponds to a pin in the GPIO port. If a bit is set (1), it configures the corresponding pin as an output; if it is cleared (0), the pin is configured as an input.
3. **IOSET (I/O Set Register):**
   - **Function:** Sets (drives high) the individual output pins of a GPIO port.
   - **Explanation:** Writing a 1 to a bit in the IOSET register sets the corresponding pin to a high logic level (VCC). Other pins' states remain unaffected.
4. **IOCLR (I/O Clear Register):**
   - **Function:** Clears (drives low) the individual output pins of a GPIO port.
   - **Explanation:** Writing a 1 to a bit in the IOCLR register clears the corresponding pin to a low logic level (GND). Other pins' states remain unaffected.

In summary:

- **IOPIN:** Reads the state of GPIO pins.
- **IODIR:** Sets the direction of GPIO pins (input or output).
- **IOSET:** Sets (drives high) the output pins.
- **IOCLR:** Clears (drives low) the output pins.

LEVEL 3:

1. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO BLINK THE LEDS CONNECTED TO PORT 0 PINS(P0.16-P0.23).

```c
#include<LPC21xx.h>
unsigned int delay;
int main()
{
  PINSEL1=0X00000000;//configure P0.16 TO P0.23 as GPIO
  IO0DIR=0XFFFFFFFF;//configure P0.16 TO P0.23 as OUTPUT
  while(1)
  {
    IO0SET=0X00FF0000;//SET PIN_NO 16-23 0F PORT0
    for(delay=0;delay<10000;delay++);
    IO0CLR=0X00FF0000;
    for(delay=0;delay<10000;delay++);
  }
}
```

2. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO IMPLEMENT 8-BIT BINARY COUNTER ON PORT 0 PINS (P0.16-P0.23).

```c
#include<LPC21xx.h>
void delay();
unsigned int count;
int main()
{
  unsigned int comp=0;
  PINSEL1=0X0000000;//configure P0.16 TO P0.23 as GPIO
  IO0DIR=0X00FF0000;//configure P0.16 TO P0.23 as OUTPUT
  while(1)
  {
    for(count=0;count<=0FF;count++){
      comp=(~count);//ensure that after 255 0 should come
      comp=comp & 0x000000FF;//to fetch lower 8 bit
      IOPIN=(comp<<16);
      delay();
  }
}
}

void delay(void){
  unsigned int i;
  for(i=0;i<650;i++);
}
```

3. DEVELOP AN EMBEDDED 'C' PROGRAM TO INTERFACE DAC WITH ARM7 TO GENERATE THE FOLLOWING WAVEFORMS:
   - SQUARE WAVE

relay.c | l1.c | counter.c | square.c | Startup.s

```c
#include<LPC21xx.h>
void delay(void);
int main()
{
  PINSEL0=0X00000000; //P0.0-P0.15 AS GPIO
  PINSEL1=0X00000000; //P0.16-P0.31 AS GPIO
  IO0DIR=0XFFFFFFFF; //P0.0-P0.31 CONFIGURED AS OUTPUT
  while(1)
    {
      IO0PIN=0x00000000;//IO0PIN IS USED TO OBTAIN  STATUS OF PINS
      delay();
      IO0PIN=0XFFFFFFFF;
      delay();
    }
  }
void delay(void)
{
  unsigned int i;
  for(i=0;i<500;i++);
}
```

   - TRIANGULAR WAVE

relay.c | l1.c | counter.c | square.c | Startup.s | tri.c*

```c
#include<LPC21xx.h>
int main()
{
  unsigned long temp=0x00000000;
  unsigned int i;
  PINSEL0=0X00000000; //P0.0-P0.15 AS GPIO
  PINSEL1=0X00000000; //P0.16-P0.31 AS GPIO
  IO0DIR=0XFFFFFFFF; //P0.0-P0.31 CONFIGURED AS OUTPUT
  while(1)
    {
      for(i=0;i!=0xFF;i++)
      {
        temp=i;
        temp=temp<<16;
        IO0PIN=temp;
      }
      for(i=0xFF;i!=0;i--)
    {
      temp=i;
      temp=temp<<16;
      IO0PIN=temp;
    }
  }
}
```

4. DEVELOP AN EMBEDDED 'C' PROGRAM TO INTERFACE THE RELAY WITH ARM7.

relay.c | l1.c | counter.c | square.c | Startup.s | tri.c*

```
#include <LPC21xx.h>
unsigned int i;
int main()
{
  IO0DIR=0x00000600; //set P0.10 as output
  IO0SET=0x00000600; //P0.10-for relay and P0.09 for buzzer is set to HIGH...turning on the relay
  while(1)
  {
    for(i=0;i<1000000;i++);
    IO0SET=0x00000600; //relay on
    for(i=0;i<1000000;i++);
    IO0CLR=0x00000600; //relay off
  }
}
```

5. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO BLINK THE BUILTIN LED CONNECTED TO PIN NUMBER 5 OF ARDUINO UNO.

blink.ino §

```
int LED_BUILTIN=5;
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

6. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE LDR SENSOR CONNECTED TO PIN NUMBER 13 OF ARDUINO UNO.

ldr.ino §

```
//RM3 to RM20
int light_pin=13;
void setup() {
  // put your setup code here, to run once:
  pinMode(light_pin,INPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  int light_data=digitalRead(light_pin);
  if(light_data==1)
  {
    Serial.println("Light not detected!");
  }
  else
  {
    Serial.println("Light detected!");
  }
    delay(1000);
}
```

7. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE BUZZER CONNECTED TO PIN NUMBER 9 OF ARDUINO UNO.

buzzer.ino

```
int buzzer_pin=9;
void setup() {
  // put your setup code here, to run once:
  pinMode(buzzer_pin,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(buzzer_pin,HIGH);
    delay(1000);
    digitalWrite(buzzer_pin,LOW);
}
```