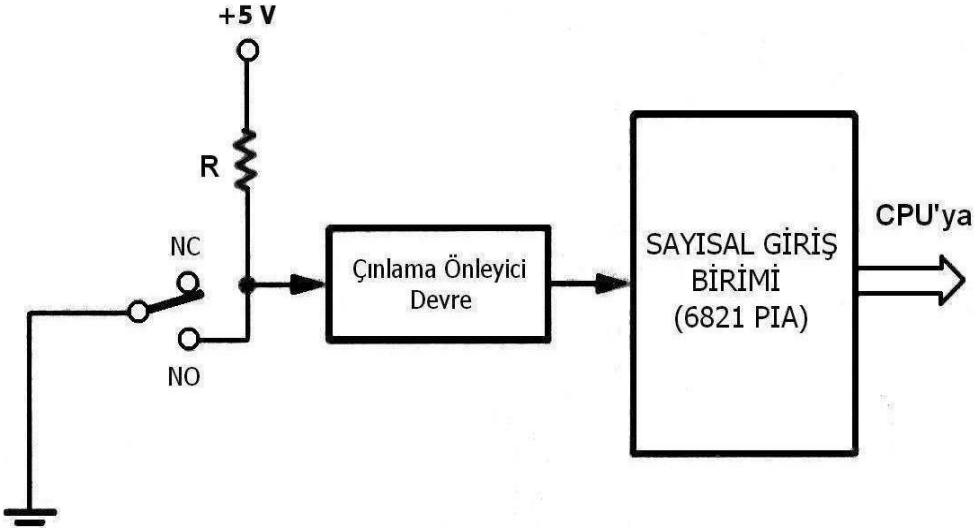
18. MİKROİŞLEMCİ TEMELLİ SİSTEM UYGULAMALARI

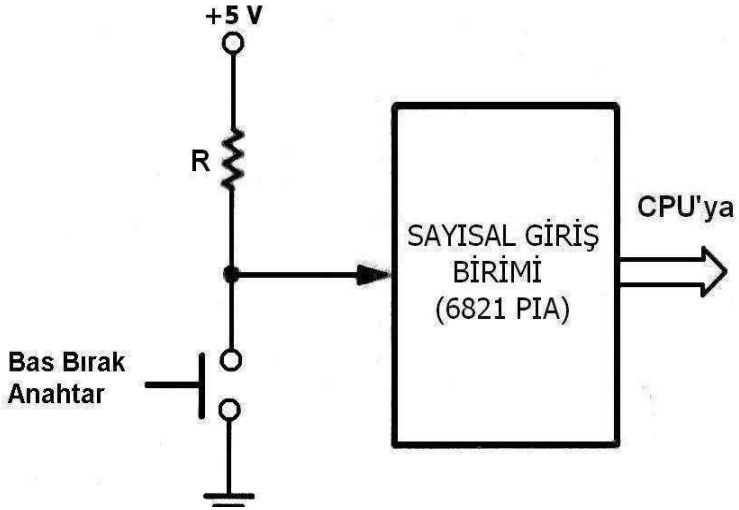
Mikroişlemci temelli sistemler eğitim, güvenlik, ticari, endüstriyel, askeri, sağlık, vs. alanlardaki uygulamalarda yaygın olarak kullanılmaktadır. Bunun sonucunda uygulamaya yönelik olarak mikroişlemci temelli sistemlerin donanım ve yazılımının tasarlanması ve gerçekleştirilmesi çeşitli alanlardaki mühendislik çalışmalarında önemli bir yer tutmaktadır. Endüstriyel uygulamalar, kendisi de mikroişlemci temelli bir sistem olan kişisel bilgisayarlar ile gerçekleştirilebilmektedir.

18.1. Giriş Uygulamaları

Mikroişlemci temelli bir sisteme dışarıdan bilgi girişi yapmanın en basit yöntemi yapılan işe uygun bir anahtar kullanmaktır. Şekil 18-1'de tek kutup çift yollu (Single Pole Double Trough, SPDT) bir anahtar kullanılarak tasarlanmış giriş devresi ve blok diyagramı verilmiştir. Burada anahtar önce normalde kapalı (Normally Close, NC) kontaktadır. Bu durumda +5V'a bağlı olan direnç üzerinden çınlama önleyici devreye lojik "1" seviyesinde gerilim giriş olarak uygulanır. Sonra bu anahtar kapatılmasıyla normalde açık (Normally Open, NO) kontak durumuna geçer.



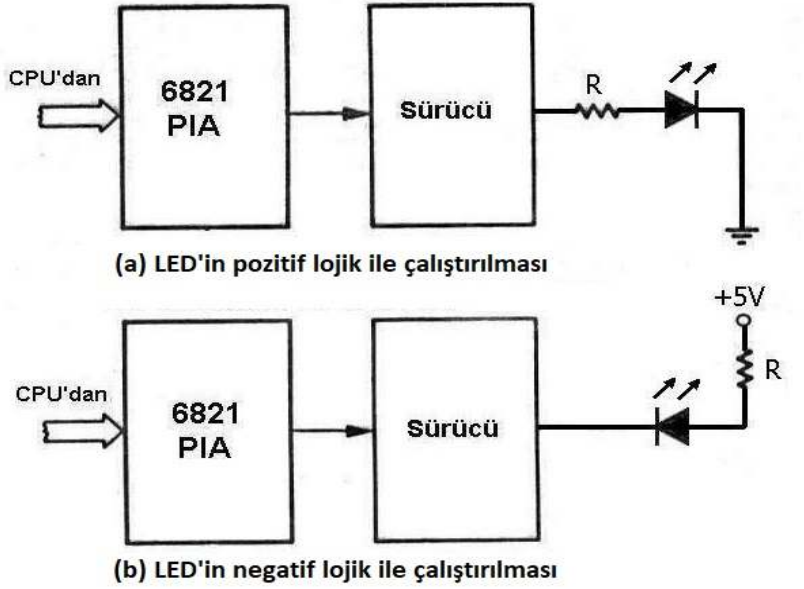
Şekil 18-1 Bir anahtar ile oluşturulmuş fiziksel giriş devresi



Şekil 18-2 Bir bas bırak anahtar ile oluşturulmuş fiziksel giriş devresi

18.2. Çıkış Uygulamaları

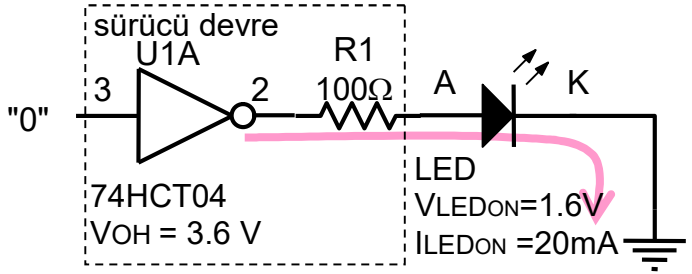
Mikroişlemci temelli bir sistemden dışarıya bilgi çıkışı yapmanın en basit yöntemi yapılan işe uygun bir gösterge kullanmaktır. Bu göstergeler gerçekleştirilen ölçüm, denetlenen süreç hakkında sayısal değerleri gösteren genellikle ekonomik ve uzun ömürlü olması nedeniyle ışık yayan diyot (LED) veya sıvı kristal yapıda olabilir.



Şekil 18-3 LED kullanarak yapılan çıkış uygulamaları

Sürücü çıkışındaki akımı sınırlamak için LED'e seri R direnci kullanılır. Bu direncin hesabı için Şekil 18-4 ve Şekil 18-5'de verilen eşdeğer devreler kullanılabilir.

$$R1 = \frac{V_{OH} - V_{LEDON}}{I_{LEDON}} = \frac{3.6 - 1.6}{20 \cdot 10^{-3}} = 100\Omega$$

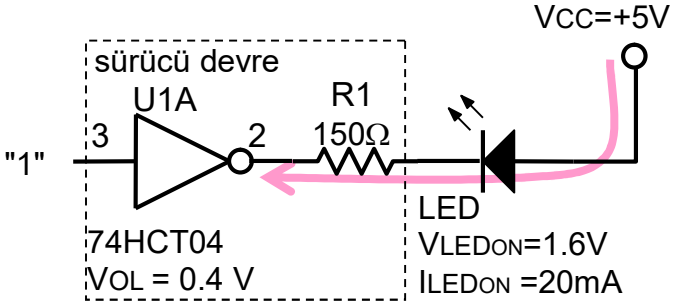


Şekil 18-4 LED'in pozitif lojik ile çalıştırılması için eşdeğer devre

Tablo 18-1 Değişik renkteki LED'lerin DC özellikleri

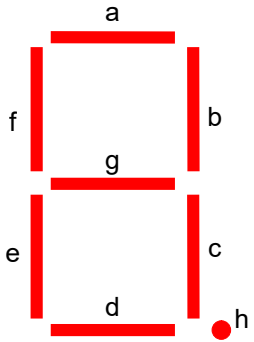
LED	V_{LEDON}	I_{LEDON}
Düşük Akımlı Kırmızı	1.8 V	2 mA
Standart Kırmızı	1.6 V	20 mA
Parlak Kırmızı	2.2 V	20 mA
Standart Sarı	2.2 V	10 mA
Standart Yeşil	2.3 V	10 mA

Şekil 18-5'de verilen devrede LED'in negatif lojik ile sürülmesi durumunda bir eşdeğer devre verilmiştir.

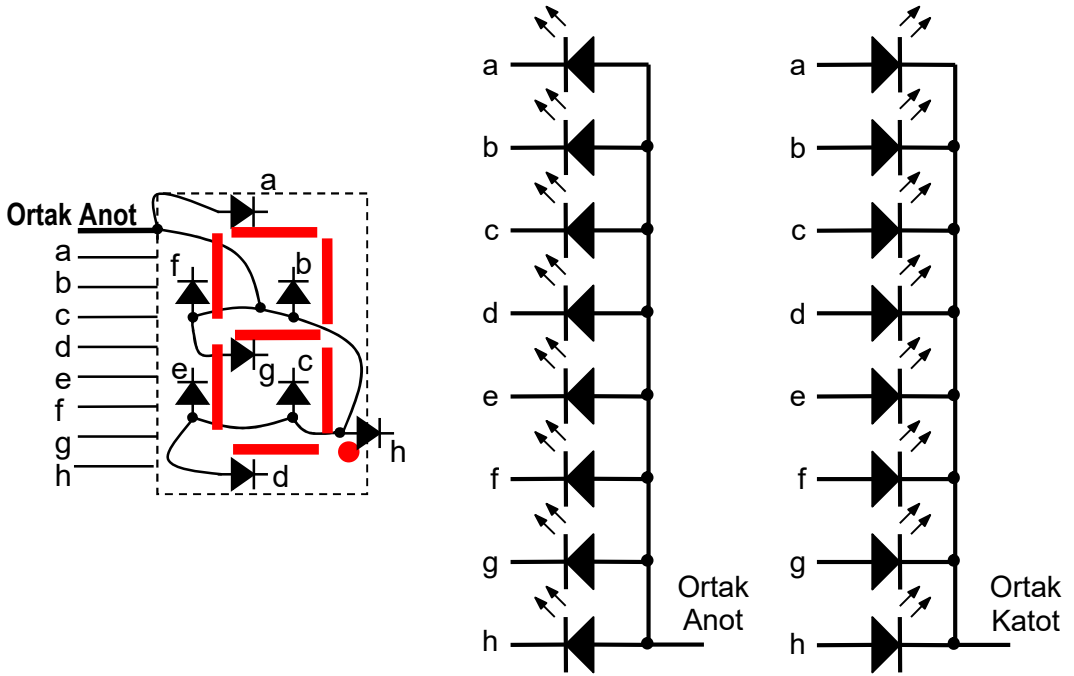


Şekil 18-5 LED'in negatif lojik ile çalıştırılması için eşdeğer devre

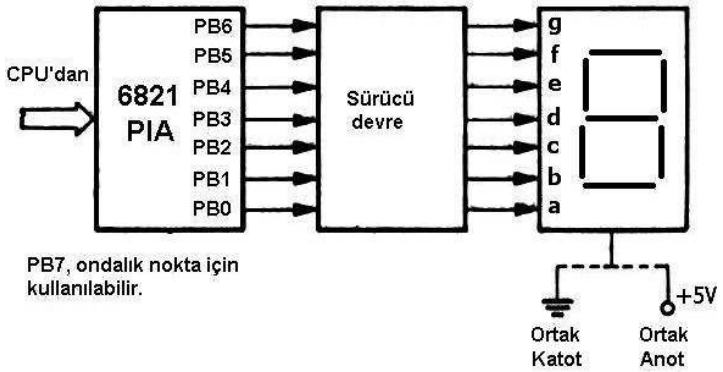
$$R1 = \frac{V_{CC} - V_{OL} - V_{LEDON}}{I_{LEDON}} = \frac{5 - 0.4 - 1.6}{20 \cdot 10^{-3}} = 150\Omega$$



Şekil 18-6 7-parça LED gösterge parça tanımları



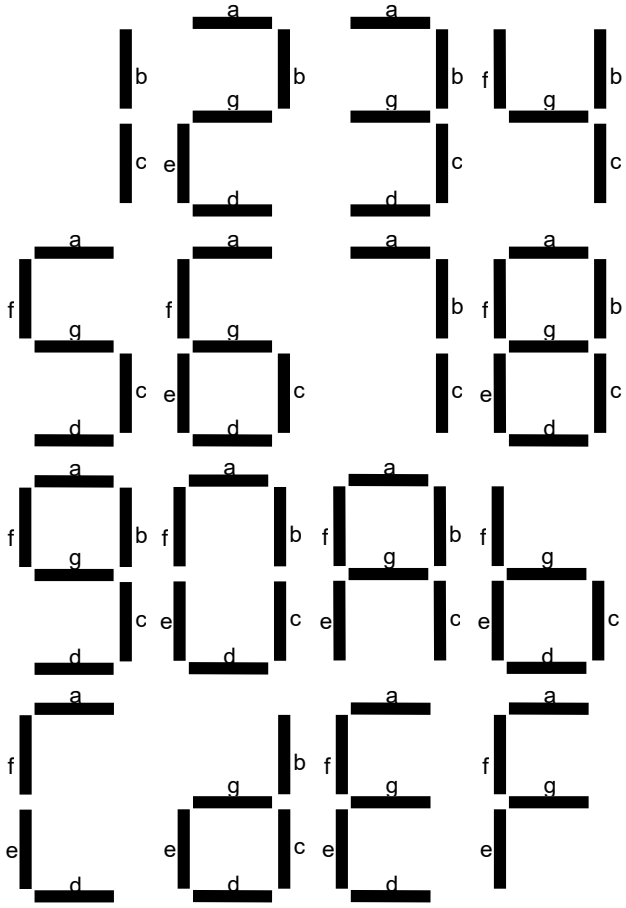
Şekil 18-7 7-parça LED gösterge için Ortak Katot ve Ortak Anot bağlantı şekli



Şekil 18-8 Paralel G/Ç tümleşik devresi ile 7-parça LED gösterge parça bağlantısı

Tablo 18-2 7-parça gösterge için veri dönüşüm tablosu

	Ortak Anot		Ortak Katot	
Sayı	g f e d c b a	hex	g f e d c b a	hex
0	1 0 0 0 0 0 0	40	0 1 1 1 1 1 1	3F
1	1 1 1 1 0 0 1	79	0 0 0 0 1 1 0	06
2	0 1 0 0 1 0 0	29	1 0 1 1 0 1 1	5B
3	0 1 1 0 0 0 0	30	1 0 0 1 1 1 1	4F
4	0 0 1 1 0 0 1	19	1 1 0 0 1 1 0	66
5	0 0 1 0 0 1 1	13	1 1 0 1 1 0 0	6C
6	0 0 0 0 0 1 1	03	1 1 1 1 1 0 0	7C
7	1 1 1 1 0 0 0	78	0 0 0 0 1 1 1	07
8	0 0 0 0 0 0 0	00	1 1 1 1 1 1 1	7F
9	0 0 1 1 0 0 0	18	1 1 0 0 1 1 1	67

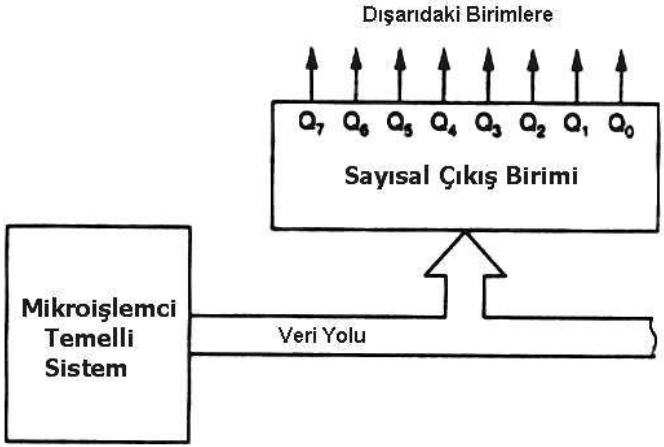


Şekil 18-9 7-parça LED göstergede sayıların şekil ve tanımları

18.3. Zamanlama Uygulamaları

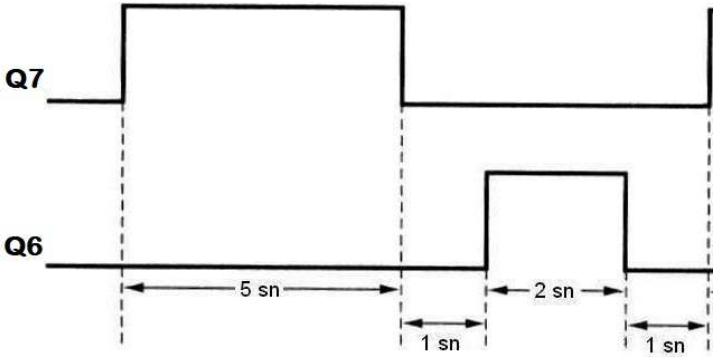
Mikroişlemci temelli sistemlerin doğadaki olayların değişim hızından çok daha hızlı çalışması nedeniyle, giriş/çıkış biriminin giriş işaretlerini algılaması ve ürettiği çıkış işaretleri çok hızlı değişir. Bu nedenle mikroişlemci yazılımında gecikme programları kullanılarak işaretlerin algılama ve üretim hızları yavaşlatılır.

Örnek 18-1 Yazılım ile 1,5ms ve 10ms gecikme sağlayan altprogramlar, 4MHz osilatör hızında çalışan mikroişlemcisinin dilinde tasarlanacaktır. Programı bir çevirici kaynak dosyası biçiminde olacak şekilde gerekli olan bütün tanımlamaları ve her satırındaki komutun açıklamasını yaparak yazınız.



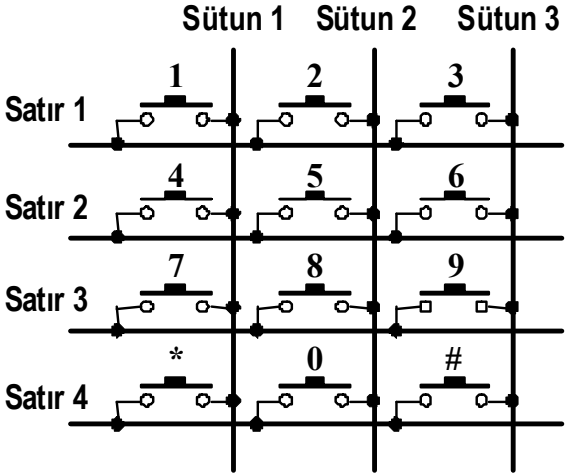
Şekil 18-10 Mikroişlemci yazılımı kullanılarak zamanlama işaretlerinin üretilmesi

Örnek 18-2 Aşağıda verilen iki darbe diyagramını, paralel çıkış birimi kullanarak oluşturan bir zamanlama yazılımı 32,768kHz osilatör hızında çalışan mikroişlemcisi dilinde tasarlanacaktır. Programı bir çevirici kaynak dosyası biçiminde olacak şekilde ve her satırındaki komutun açıklamasını yaparak yazınız



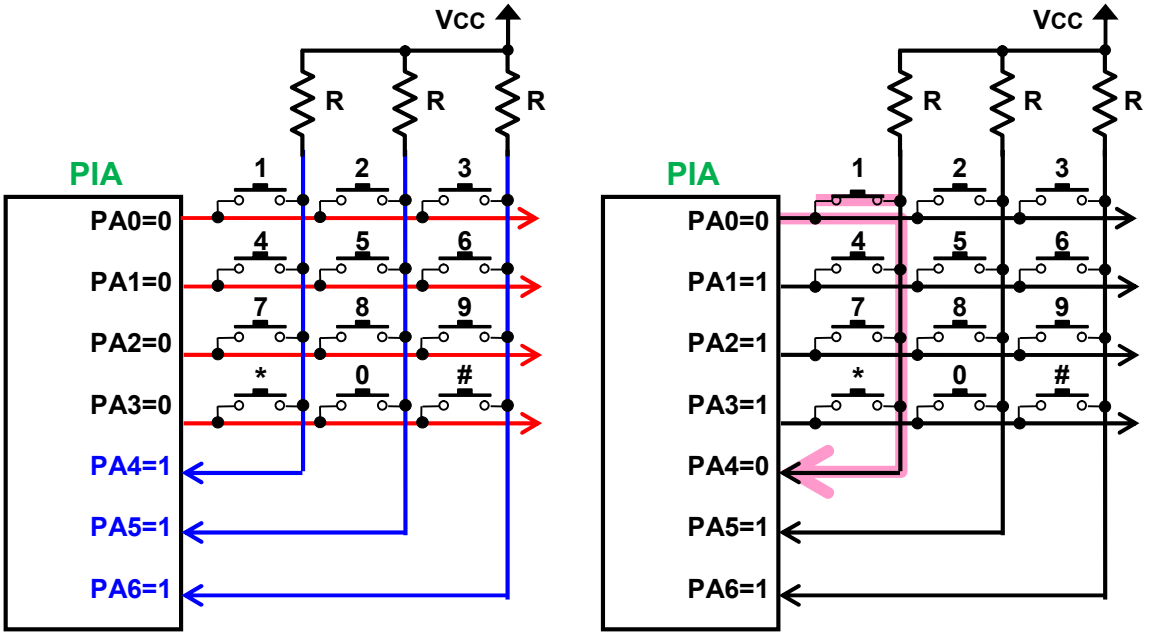
18.4. Tuş Takımı Tarama Uygulaması

Mikroişlemci temelli sistem bir denetim işlevini yerine getiriyorsa, denetlenen büyüklüğün ayarlanması için gerekli değerlerin girişinin yapılması ve girilen değerlerin, değişen çıkış büyüklüğünün kullanıcı tarafından görülmesi için bir gösterge birimine çıkış yapılması gerekir. Tuş takımı olarak ise kolay bulunan ve telefon cihazlarında kullanıldığı için ekonomik olan tuş takımı kullanılır. Bu tuş takımlarında tuşların bağlantısı matris şeklinde yapıldığı için matrisin boyutları verilerek matris tipi tuş takımı şeklinde adlandırılır. Şekil 18-11'de böyle bir 3x4 matris tipi tuş takımının bağlantısı gösterilmiştir.

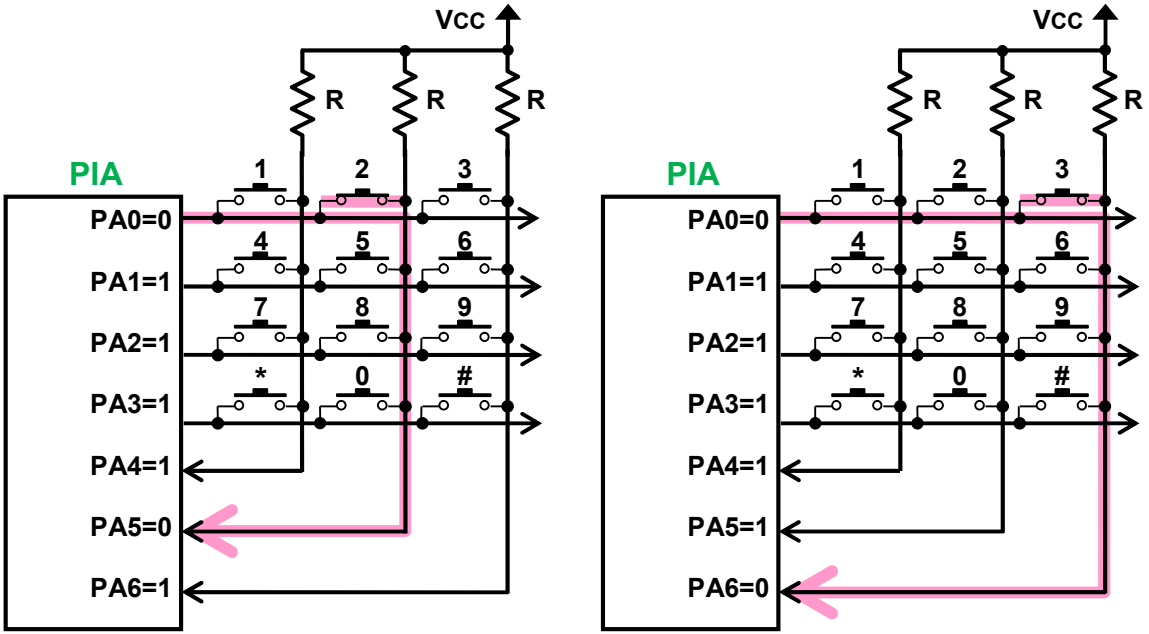


Şekil 18-11 Matris tipi tuş takımı bağlantı diyagramı

Eğer hiçbir tuşa basılmamışsa, sütunlar R direnci ile VCC'ye bağlı olduğu için PA4-PA6 uçlarından lojik "1" okunur.

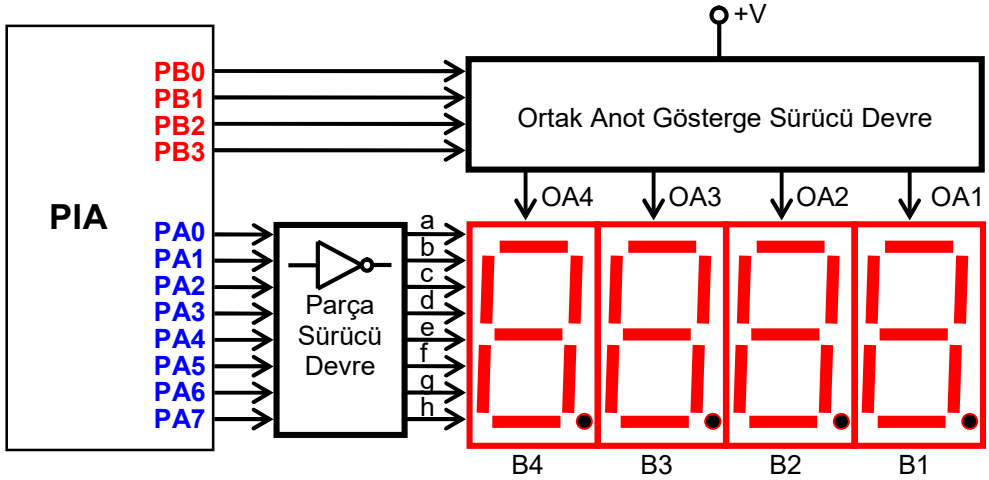


Şekil 18-12 Paralel Giriş/Çıkış tümleşik devresine matris tipi tuş takımı bağlanması.

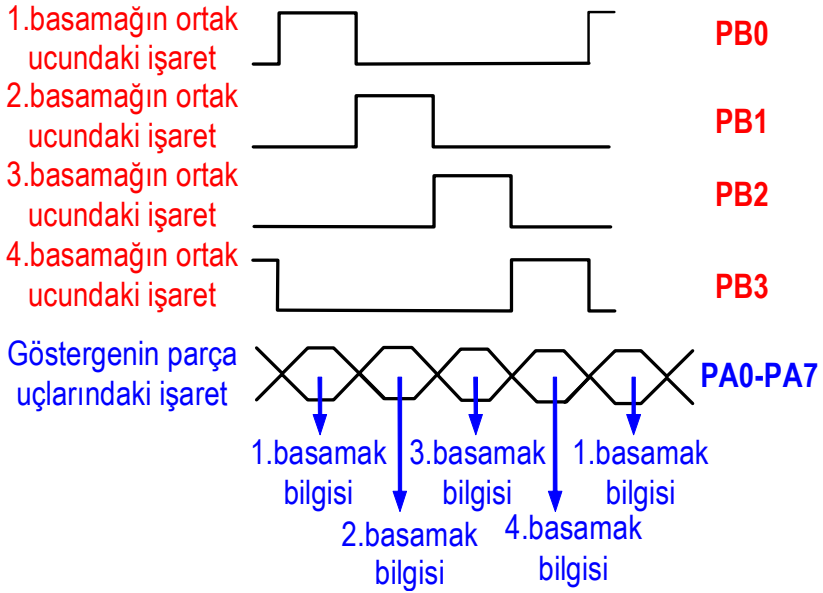


Şekil 18-13 Matris tipi tuş takımında basılan tuşun algılanması.

18.5. Çok Basamaklı Gösterge Uygulamaları

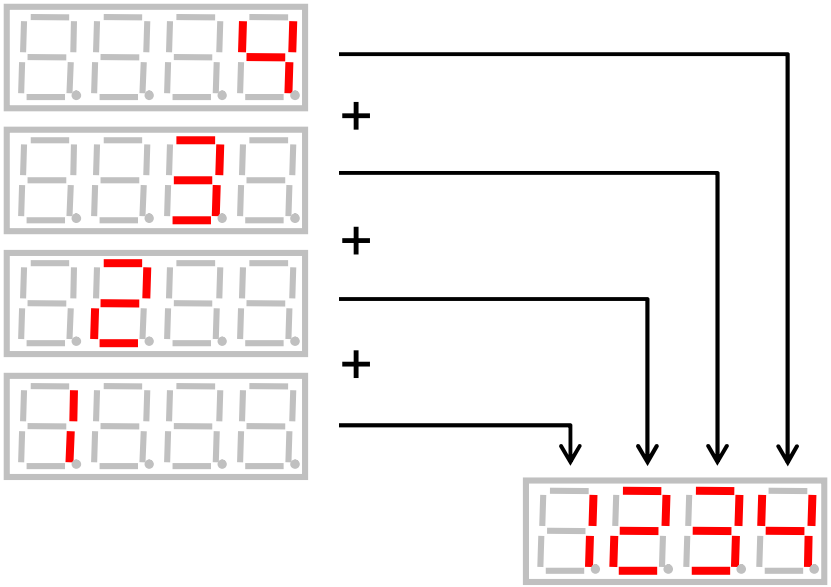


Şekil 18-14 Çok sayıda ortak anot göstergenin bir PIA ile sürülmesi



Şekil 18-15 Çok basamaklı 7-parça göstergelerin dinamik tarama ile sürülmesi

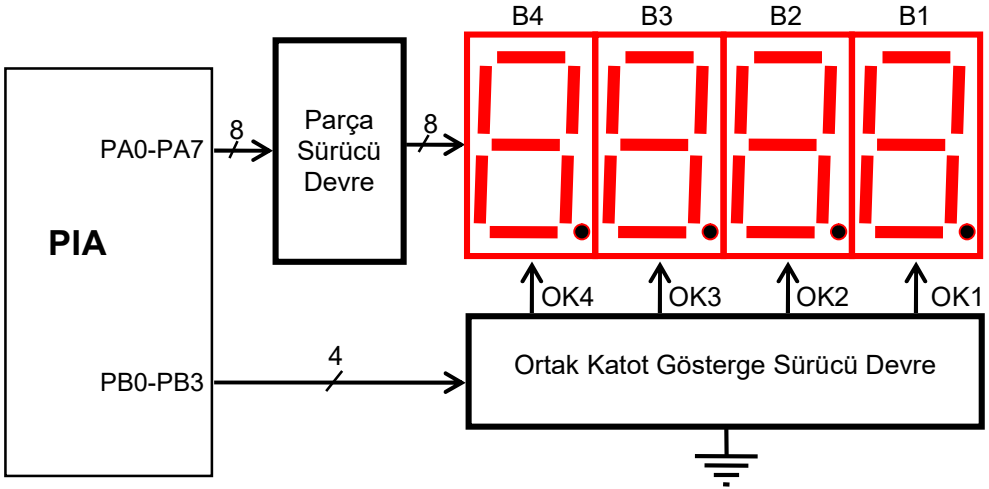
.



Şekil 18-16 Bir 4 basamaklı 7-parça göstergenin dinamik tarama ile sürülmesi

Örnek 18-3 Dinamik gösterge tarama işlemiyle çalışan bir program ile 4 basamaklı ortak katotlu 7-parça LED göstergeye sayısal çıkış işlemi gerçekleştirilecektir. Bunun için bir çıkış birimi sistemini tasarlayınız ve blok diyagramını çiziniz.

Çözüm:



Şekil 18-18 Çok sayıda ortak katot göstergenin bir PIA ile sürülmesi

Summary of Interrupts Structure on Arduino UNO

Interrupt Pins on Arduino UNO

The Arduino UNO supports external interrupts on the following pins:

- Digital Pin 2 → Interrupt 0
- Digital Pin 3 → Interrupt 1

These pins are connected to the INT0 and INT1 interrupt vectors of the ATmega328P.

Features of Interrupts on Arduino UNO

- Asynchronous: Interrupts can occur at any time, independent of the main program flow.
- Fast Response: The processor immediately jumps to the ISR when an interrupt occurs.
- Efficient: Useful for handling time-critical tasks like reading sensors or responding to button presses.

Types of Interrupts in ATmega328P (Arduino UNO)

There are two main categories of interrupts:

1. External Interrupts

- Triggered by a change on external pins (INT0, INT1).
- Can be configured to trigger on:
  - LOW level
  - Any logical change
  - Rising edge
  - Falling edge

2. Internal Interrupts

Triggered by internal events such as:

- Timer Overflows (Timer0, Timer1, Timer2)
- Timer Compare Match
- ADC Conversion Complete
- USART Receive/Transmit
- SPI Transfer Complete
- EEPROM Ready
- Watchdog Timeout
- Pin Change Interrupts (on almost all digital pins)

Example 1.

```
// This example uses the pin change interrupt to blink an LED
// and also demonstrates how to share a variable between
// the interrupt and the main program (volatile).
const byte ledPin = 13;
const byte interruptPin = 2;  // input pin that the interruption
will be attached to
volatile byte state = LOW;  // variable that will be updated in the
ISR

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink,
CHANGE);
}
void blink() {
  state = !state;
}
void loop() {
  digitalWrite(ledPin, state);
}
```

Explanation of the program in Example 1:

1. The interrupt handler. Notice the code "**attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);**". This code defines that the interrupt will come from the Arduino External Interrupt (INT0, pin 2) and defines the function **"blink()"** that will handle the interrupts. It also defines that the interrupt trigger type is "CHANGE".
2. The variable "**state**" is declared as "volatile" as it is used inside the interrupt hander ("**blink**") and the rest of the sketch. Volatile variables are loaded from the RAM, always, instead of the CPU register. Registers contains temporary variable values which may loose consistency when are accessed by interrupt request handlers.
3. Any code in the regular part of the sketch that must not be interrupted **(i.e. "critical code")** is enclosed between the "**noInterrupts();**" and "**interrupts();**" functions. This way, we ensure that variables must be updated reliably, contain reliable values.

1. Kesme işleyicisi. **" attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);"** koduna dikkat edin! Bu kod, kesmenin Arduino Harici Kesme'den (INT0, pin 2) geleceğini ve kesmeleri işleyecek olan **"blink()"** fonksiyonunu tanımlar. Ayrıca kesme uyarım şeklinin "CHANGE" olduğunu da tanımlar.

2. **"state"** değişkeni, kesme işleyicisi **"blink()"** ve programın geri kalanında kullanıldığı için "volatile" olarak bildirilir. Geçici değişkenler her zaman CPU yazmacı yerine RAM'den yüklenir. Yazmaçlar, kesme isteği işleyicileri tarafından erişildiğinde tutarlılığını kaybedebilecek (içeriği değişebilecek) geçici değişken değerleri içerir.
3. Programın normal kısmında kesmeye uğramaması gereken herhangi bir kod **(yani "kritik kod")** "noInterrupts();" ve "interrupts();" işlevleri arasına alınır. Bu şekilde, değişkenlerin güvenilir şekilde güncellenmesini ve güvenilir değerler içermesini sağlarız.

Example 2.

```
#include <TimerOne.h>
// This example uses the timer interrupt to blink an LED
// and also demonstrates how to share a variable between
// the interrupt and the main program.
const int led = LED_BUILTIN;  // the pin with a LED

void setup(void)
{
  pinMode(led, OUTPUT);
  Timer1.initialize(150000);
  Timer1.attachInterrupt(blinkLED); // blinkLED to run every 0.15
seconds
  Serial.begin(9600);
}
// The interrupt will blink the LED, and keep
// track of how many times it has blinked.
int ledState = LOW;
volatile unsigned long blinkCount = 0; // use volatile for shared
variables
void blinkLED(void)
{
  if (ledState == LOW) {
    ledState = HIGH;
    blinkCount = blinkCount + 1;  // increase when LED turns on
  } else {
    ledState = LOW;
  }
  digitalWrite(led, ledState);
}
```

```
// The main program will print the blink count
// to the Arduino Serial Monitor
void loop(void)
{
  unsigned long blinkCopy;  // holds a copy of the blinkCount
  // to read a variable which the interrupt code writes, we
  // must temporarily disable interrupts, to be sure it will
  // not change while we are reading.  To minimize the time
  // with interrupts off, just quickly make a copy, and then
  // use the copy while allowing the interrupt to keep working.
  noInterrupts();
  blinkCopy = blinkCount;
  interrupts();
  Serial.print("blinkCount = ");
  Serial.println(blinkCopy);  delay(100);
}
```

Explanation of the program in Example 2:

1. The interrupt handler. Notice the code "**Timer1.attachInterrupt(blinkLED);**" This code defines that the interrupt will come from the Arduino Internal Interrupt (Timer1) and defines the function **"blinkLED"** that will handle the interrupts.
2. The variable "**blinkCount**" is declared as "volatile" as it is used inside the interrupt hander ("**blinkLED**") and the rest of the sketch. Volatile variables are loaded from the RAM, always, instead of the CPU register. Registers contains temporary variable values which may loose consistency when are accessed by interrupt request handlers.
3. Any code in the regular part of the sketch that must not be interrupted **(i.e. "critical code")** is enclosed between the "**noInterrupts();**" and "**interrupts();**" functions. This way, we ensure that variables must be updated reliably, contain reliable values.

1. Kesme işleyicisi. **"Timer1.attachInterrupt(blinkLED);"** koduna dikkat edin! Bu kod, Arduino'nun dahili kesmelerinden (Timer1) gelen kesmeleri işleyecek olan **"blinkLED"** fonksiyonunu tanımlar.
2. "blinkCount" değişkeni, kesme işleyicisi ("blinkLED") ve programın geri kalanında kullanıldığı için "volatile" olarak bildirilir. Geçici değişkenler her zaman CPU yazmacı yerine RAM'den yüklenir. Yazmaçlar, kesme isteği işleyicileri tarafından erişildiğinde tutarlılığını kaybedebilecek (içeriği değişebilecek) geçici değişken değerleri içerir.
3. Programın normal kısmında kesmeye uğramaması gereken herhangi bir kod **(yani "kritik kod")** "noInterrupts();" ve "interrupts();" işlevleri arasına alınır. Bu şekilde, değişkenlerin güvenilir şekilde güncellenmesini ve güvenilir değerler içermesini sağlarız.

74HC595 4-Digit 7-Segment Display Library for Arduino - DIYables_4Digit7Segment_74HC595

This library is designed for Arduino, ESP32, ESP8266... to control the 74HC595 4-Digit 7-Segment Display LED module, which has 4 dots.

Product Link

[74HC595 4-Digit 7-Segment Display Module]

Features

- **High-Level**: Displaying Integer with the zero-padding option, supporting the negative number
- **High-Level**: Displaying Float with the decimal place, zero-padding options, supporting the negative number
- **Low-Level**: Displaying numeric characters digit-by-digit
- **Low-Level**: Displaying non-numeric characters digit-by-digit, including:
  - - (dash)
  - _ (underscore)
  - ° (degree)
  - C
  - E
  - F
- Low-Level: Displaying dot (decimal place) digit-by-digit

Available Functions

- DIYables_4Digit7Segment_74HC595(int sclk, int rclk, int dio);
- void printInt(int number, bool zero_padding);
- void printFloat(float number, int decimal_place, bool zero_padding);
- void clear();
- void setDot(int pos);
- void setNumber(int pos, int value);
- void setChar(int pos, SegChars value);
- void show();
- void loop();

Available Examples

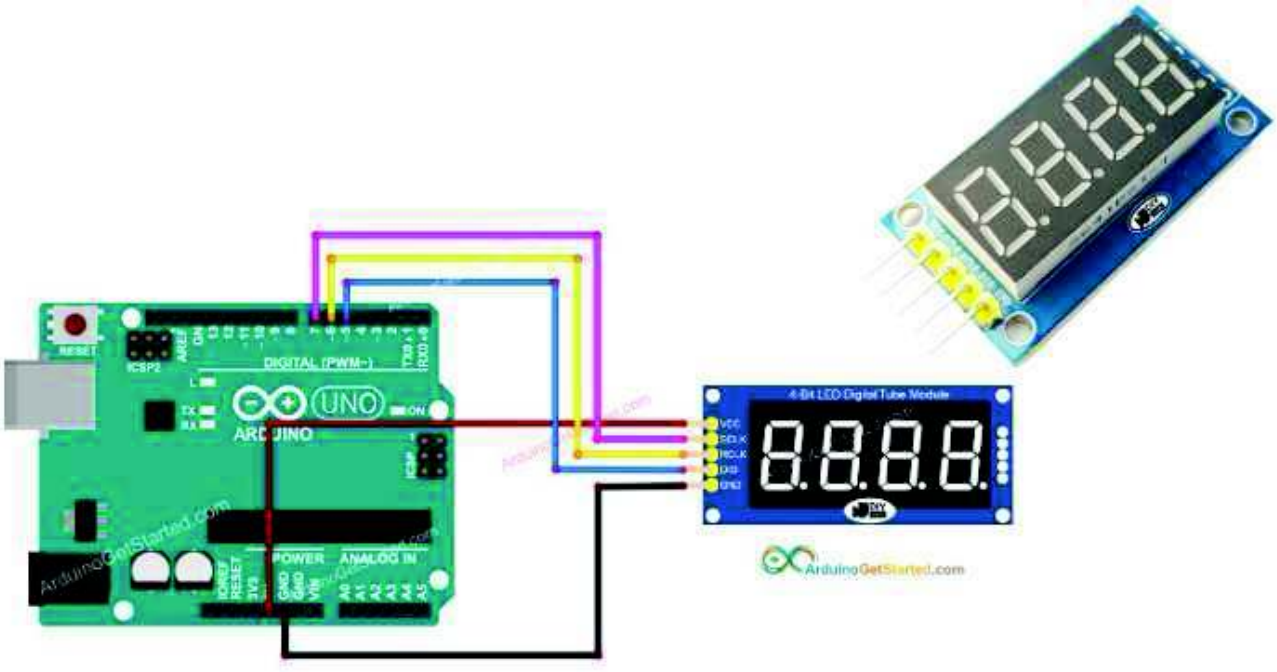
- DisplayInteger
- DisplayFloat
- DisplayTemperature

References

- DIYables_4Digit7Segment_74HC595 Library Reference

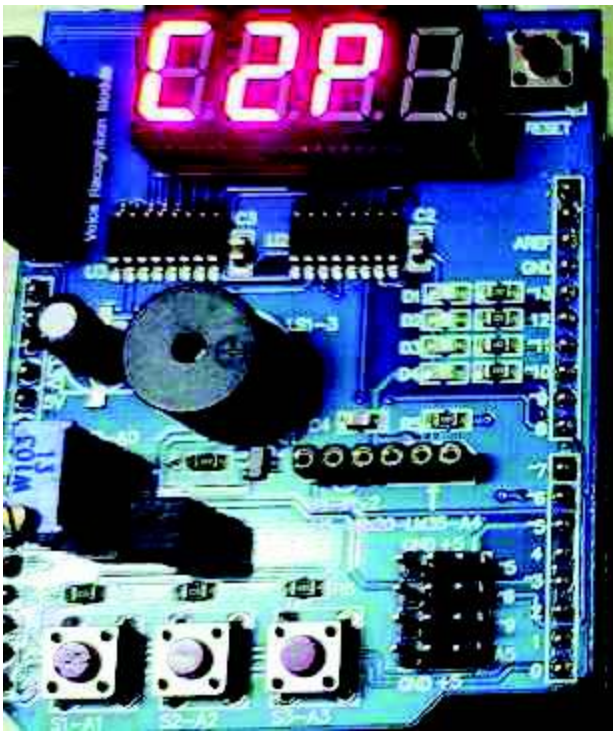
Tutorials

- Arduino - 74HC595 4-Digit 7-Segment Display
- ESP32 - 74HC595 4-Digit 7-Segment Display
- ESP8266 - 74HC595 4-Digit 7-Segment Display



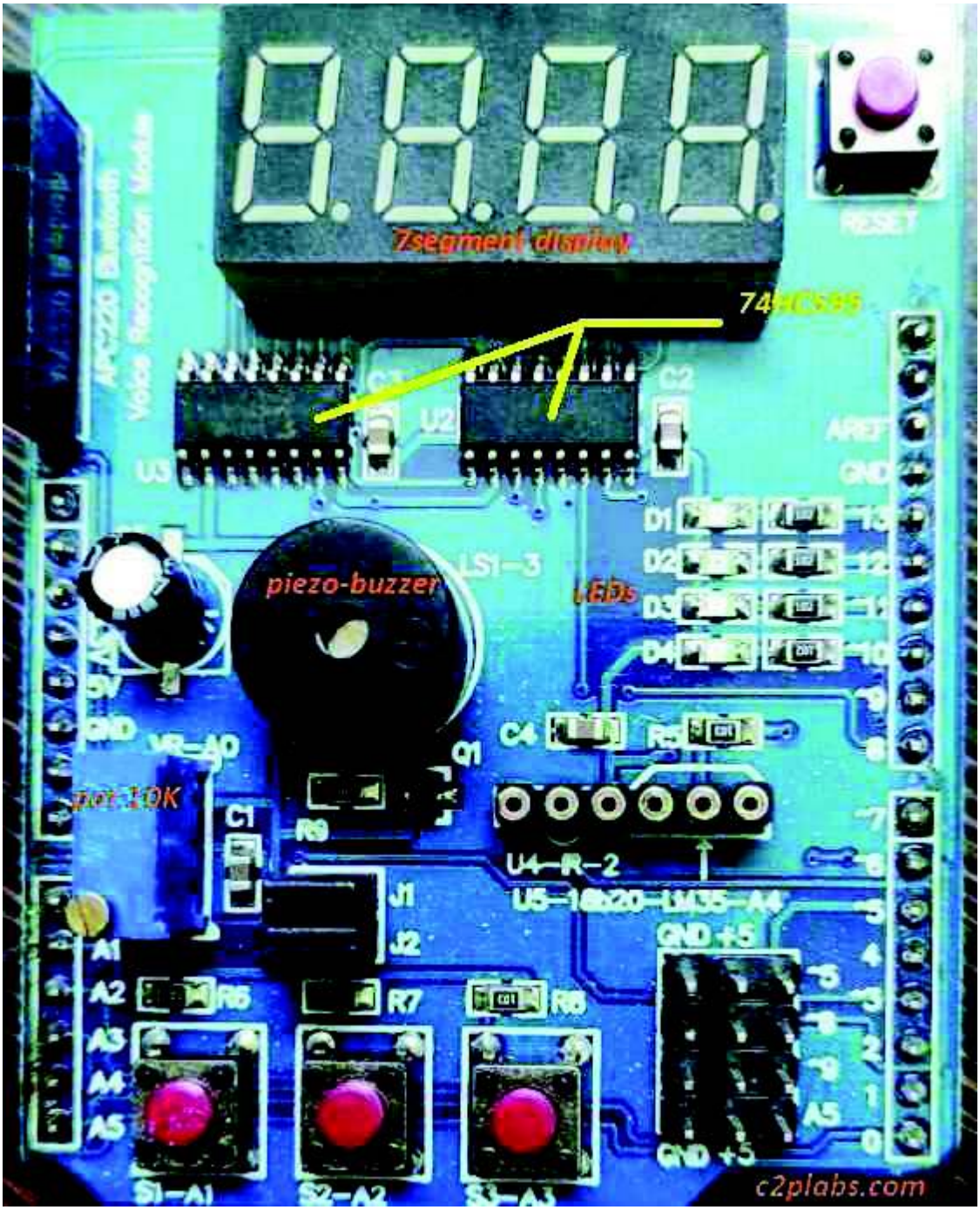
Arduino Multi-Function Shield Seven-Segment Display

BY C2PLABS_ADMIN SEPTEMBER 13, 2020 ARDUINO, TECHNOLOGY



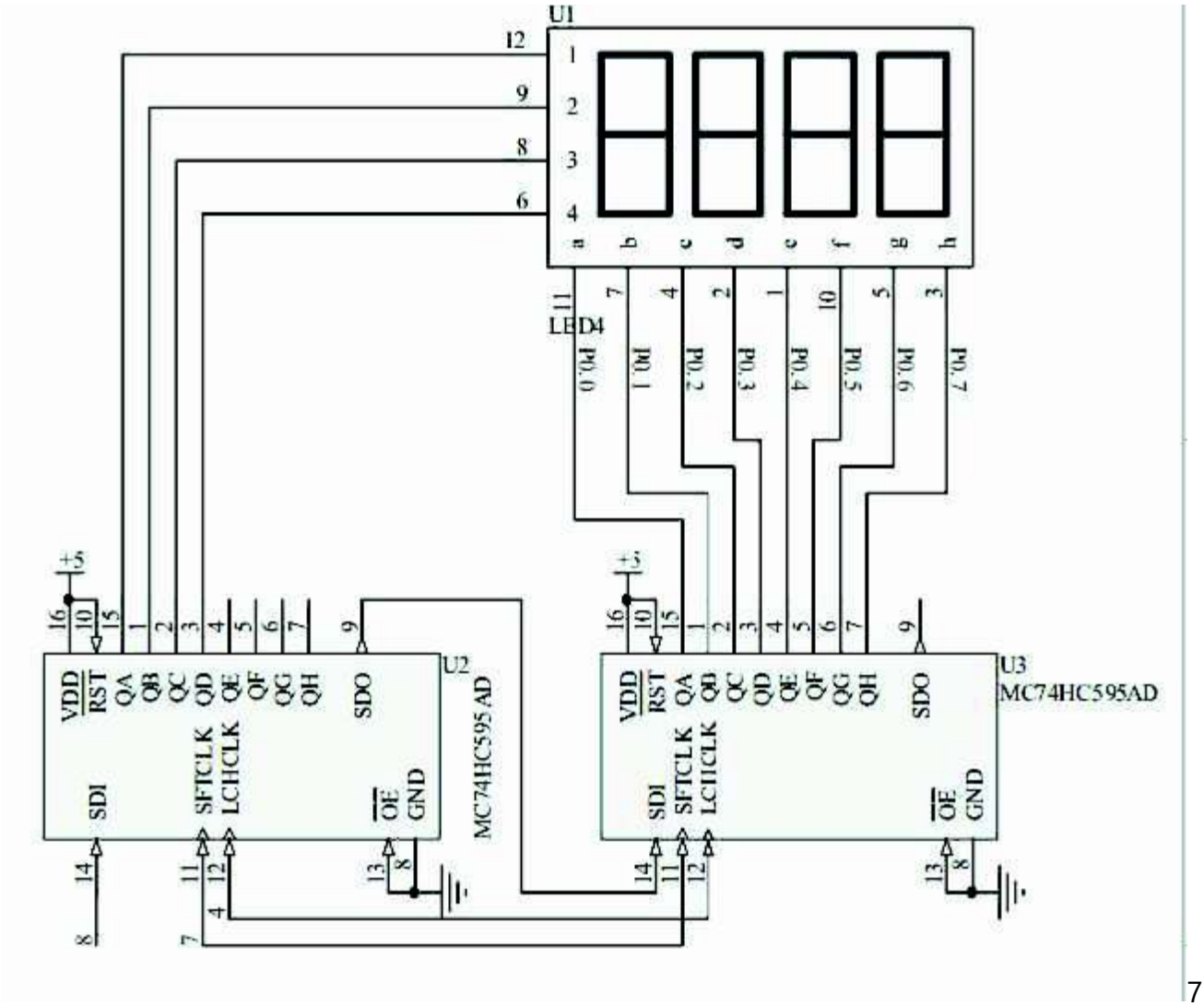
Arduino is a versatile computing platform to work on microcontrollers and embedded systems. It can be used to write programs and run on microcontrollers. These boards comes with on board LED which can be as a hello word Arduino program. To extend the functionality and to interface with external physical world we need do lot of writing with bread board and jumper wires. Multi-function shield is the one comes with rich set of digital input output options to try different input-output devices with Arduino. This comes with the following features.

1. It has four LEDs connected to 10, 11, 12 and 13 pins of Arduino
2. Four digit seven segment display connect through two 74HC595 serial in- parallel out shift registers
3. A piezo-buzzer connect to pin 3, using transistor driver circuit.
4. It also has trim pot(10K) connected to Analog-input pin A0.
5. Three buttons connected



Multi-function shield with labels

In this blog we use 7-segment display to display C2P. The schematic diagram of four digit 7-segment display connected through 74HC595 is shown in the below pic.



7-

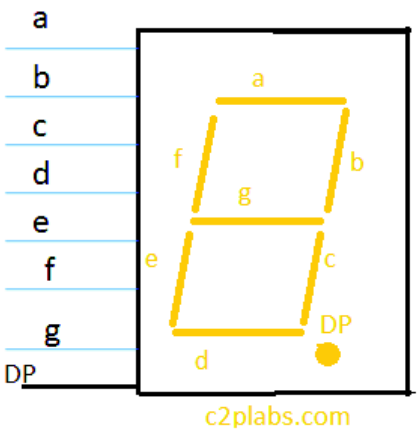
Segment display connections on Multi-Function shield

As shown in the schematic, four digit LED 7-segment display is connected to Arduino using two 74HC595 serial to parallel shift registers in cascade connection. In this way of connection clock and latch pins of shift register is shared same IO pins from Arduino. Pin 7 for CLK and pin 4 for LATCH. SDO of first Shift register (74HC595) is connected to SDI for cascaded one. Pin 8 of Arduino is connected to SDI (Serial Data In) of first shift register.

One of the shift register is connected to select pins to select the digit and other is connected to a-g pins of segments. To display characters on 7-segment first we have to select which digit we want to use and send the hex data of character to be displayed.

Seven Segment display (7-segment display) fundamentals:

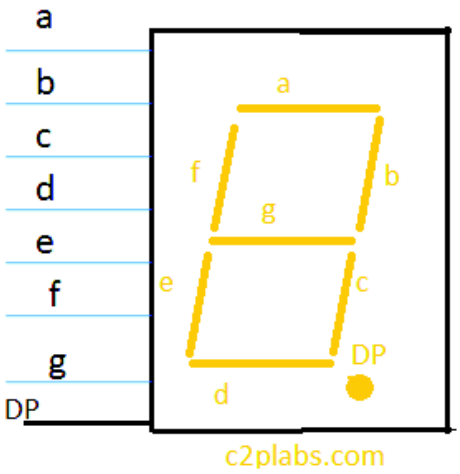
Seven segment displays consists of seven LEDs arranged to form rectangular shape digits and alphabets. Each module also consists of eighth LED which is a dot (**.**).

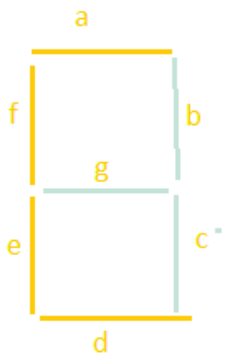


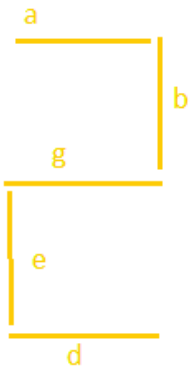
Pin-out of Seven Segment display

Seven segment displays can be of two type's common cathode or common anode. In common cathode module to glow individual LED pin should be made logic HIGH or logic 1. In common anode individual LEDs should be supplied with logic LOW or logic zero. The seven-segment modules used on Arduino multi-function shield is Common Anode (CA) modules.

Truth table for Seven Segment display:







To display a particular character or digit on each segment we have to send appropriate data to segments. Lets discuss about displaying digit 2. To display digit 2, LEDs **a,b,d,e,g** should glow. As the segment used is common anode logic **LOW** zero should be supplied to these segments, and all other **c,f** and **dp** LEDs should be given logic HIGH [1]. The truth table for displaying **zero (0)** and **"C2P"** on seven segment display is shown below.

Display Digit/Letter	DP	g	f	e	d	c	b	a	Hex Byte
Digit-0	1	1	0	0	0	0	0	0	0xC0
Letter-C	1	1	0	0	0	1	1	0	0xC6
Digit 2	1	0	1	0	0	4	0	0	0xA4
Letter-P	1	0	0	0	1	1	0	0	0x8C
OFF	1	1	1	1	1	1	1	1	0xFF

Arduino Program-1:

```
#define LATCH_PIN 4
#define CLK_PIN   7
#define DATA_PIN  8
void SendDataToSegment(byte Segment_no, byte hexValue);
const byte C_HEX  = 0xC6;
const byte TWO_HEX  = 0xA4;
const byte P_HEX  = 0x8C;
byte select_seg1 = 0xF1 ;
byte select_seg2 = 0xF2 ;
byte select_seg3 = 0xF4 ;
byte select_seg4 = 0xF8 ;
const byte SEGMENT_SELECT[] = {0xF1,0xF2,0xF4,0xF8};

void setup ()
{
/* All the pins like latch, clock and data pins are used as output */
  pinMode(LATCH_PIN,OUTPUT);
  pinMode(CLK_PIN,OUTPUT);
  pinMode(DATA_PIN,OUTPUT);
}
/* Main program */
void loop()
{
/* Update the display with the current counter value */
  SendDataToSegment(select_seg1 , C_HEX);
  SendDataToSegment(select_seg2 , TWO_HEX);
  SendDataToSegment(select_seg3 , P_HEX);
}
void SendDataToSegment(byte Segment_no, byte hexValue)
{
  /* Make Latch pin Low */
  digitalWrite(LATCH_PIN,LOW);
  /* Transfer Segmenent data */
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, hexValue);
    /* Transfer Segmenent Number  */
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, Segment_no );
    /* Make Latch pin High so the data appear on Latch parallel pins */
  digitalWrite(LATCH_PIN,HIGH);
}
```

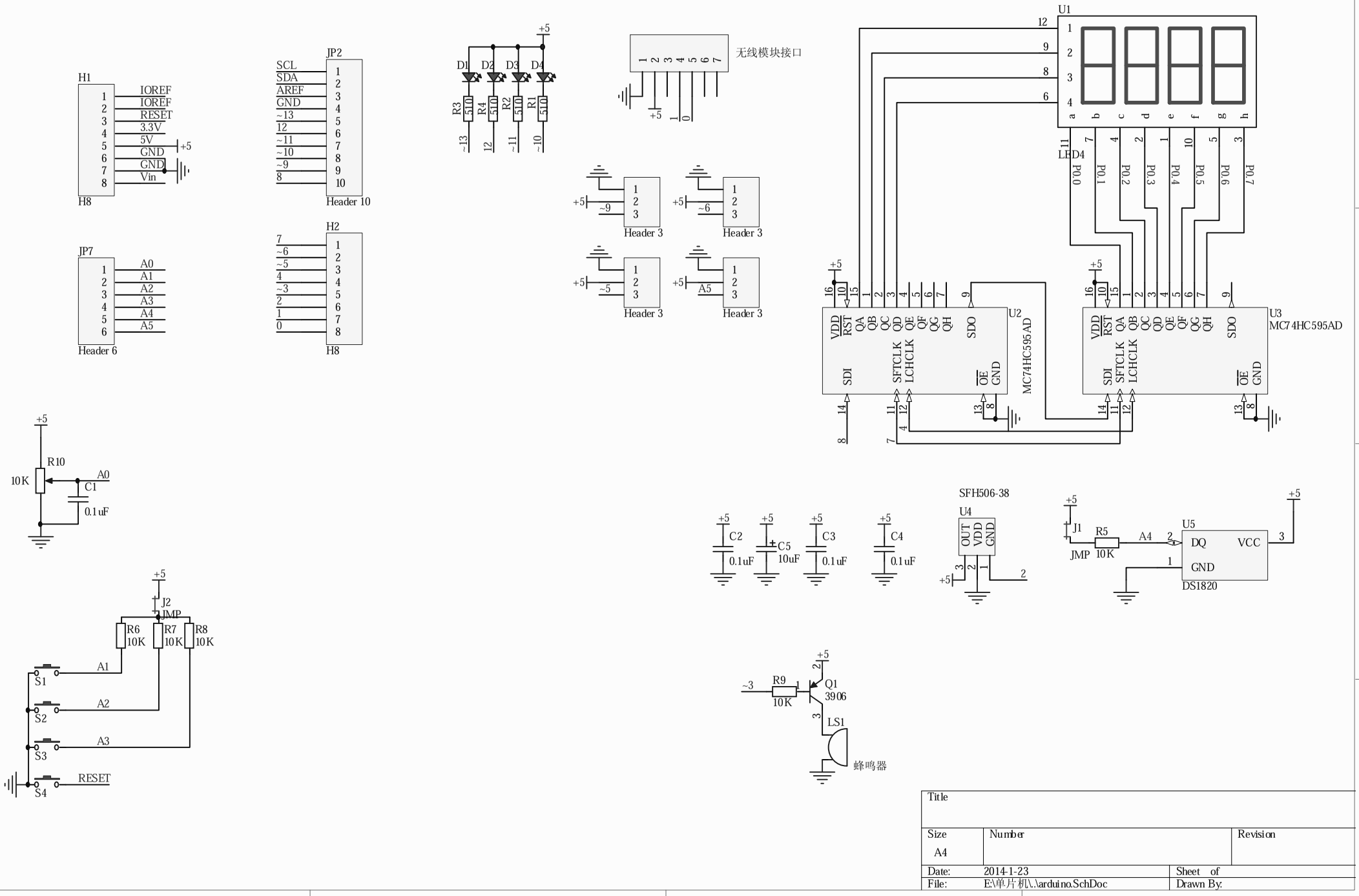
Arduino Program-2:

```
#define LATCH_PIN 4
#define CLK_PIN   7
#define DATA_PIN  8
void SendDataToSegment(byte Segment_no, byte hexValue);
const byte C_HEX  = 0xC6;
const byte TWO_HEX  = 0xA4;
const byte P_HEX  = 0x8C;
const byte SEG_OFF  = 0xFF;
int led1 = 13;
int led2 = 12;
int led3 = 11;
```

```
int led4 = 10;
int buzzer_pin = 3;
byte select_seg1 = 0xF1 ;
byte select_seg2 = 0xF2 ;
byte select_seg3 = 0xF4 ;
byte select_seg4 = 0xF8 ;
#define BUTTON1 A1
#define BUTTON2 A2
#define BUTTON3 A3
const byte SEGMENT_SELECT[] = {0xF1,0xF2,0xF4,0xF8};

void setup ()
{
/* All the pins like latch, clock and data pins are used as output */
Serial.begin(9600);
  pinMode(LATCH_PIN,OUTPUT);
  pinMode(CLK_PIN,OUTPUT);
  pinMode(DATA_PIN,OUTPUT);
  pinMode(buzzer_pin, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  LedSOFF();
}
/* Main program */
void loop()
{
   if(!digitalRead(BUTTON1))
   {
    LedBlink();
    LedSOFF();
   }
   if(!digitalRead(BUTTON2))
   {
       SendDataToSegment(select_seg1 , C_HEX);
       delay(500);
       SendDataToSegment(select_seg2 , TWO_HEX);
       delay(500);
       SendDataToSegment(select_seg3 , P_HEX);
      delay(500);
   }
   if(!digitalRead(BUTTON3))
   {
      analogWrite(buzzer_pin,0);
      delay(1000);
   }
  analogWrite(buzzer_pin,255);
  SendDataToSegment(select_seg1 , SEG_OFF);
  SendDataToSegment(select_seg2 , SEG_OFF);
  SendDataToSegment(select_seg3 , SEG_OFF);
/* Update the display with the current counter value */
}
void LedBlink()
```

```
{
digitalWrite(led1, HIGH);
digitalWrite(led1, LOW);
delay(500);
digitalWrite(led1, HIGH);
digitalWrite(led2, HIGH);
delay(500);
digitalWrite(led2, LOW);
delay(500);
digitalWrite(led2, HIGH);
digitalWrite(led3, HIGH);
delay(500);
digitalWrite(led3, LOW);
delay(500);
digitalWrite(led3, HIGH);
digitalWrite(led4, HIGH);
delay(500);
digitalWrite(led4, LOW);
delay(500);
digitalWrite(led4, HIGH);
delay(500);
}
void LedSOFF()
{
digitalWrite(led1, HIGH);
digitalWrite(led2, HIGH);
digitalWrite(led3, HIGH);
digitalWrite(led4, HIGH);
}
void SendDataToSegment(byte Segment_no, byte hexValue)
{
  /* Sending Active Low signal to 74HC595 latch */
  digitalWrite(LATCH_PIN,LOW);
  /* Transmit serial data to Segment */
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, hexValue);
    /* Select the 7 Segment number  */
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, Segment_no );
    /* Write the data appear on Latch parallel pins */
  digitalWrite(LATCH_PIN,HIGH);
}
```


H1
IOREF
IOREF
RESET
3.3V
5V
GND
GND
Vin
H8
JP2
SCL
SDA
AREF
GND
~13
12
~11
~10
~9
8
Header 10
D1 D2 D3 D4
R3 R4 R2 R1
510
无线模块接口
JP7
A0
A1
A2
A3
A4
A5
Header 6
H2
Header 3
U1
LED4
U2
U3
MC74HC595AD
SFH506-38
U4
U5
DS1820
R10
10K
C1
0.1uF
C2
C3
C4
C5
10uF
R5
JMP
J1
J2
R6
R7
R8
S1
S2
S3
S4
RESET
R9
Q1
3906
LS1
蜂鸣器
Title
Size
A4
Number
Revision
Date:
2014-1-23
Sheet of
File:
E:\单片机\..\arduino.SchDoc
Drawn By:

Interfacing TM1637 4 Digit Seven Segment Display Module with Arduino

Published September 19, 2022 Debashis Das Author

If you are thinking about building your next project around a 4-digit seven-segment display, you need at least 12 connecting pins to drive all the segments properly, that is most of the pins for a microcontroller like Arduino. The workaround for this problem comes in the form of an IC, and as the title of the project suggests it's called TM1673 LED Drive Control. So for this project, we will be using a ready-to-use TM1637 Based 4 Bits Red Digital Tube LED Display Module and will be testing its every feature and also in the end we will give you a working video of the model. In our previous article, we used seven segment displays to build Clock, counter, stopwatch timer, and more. You can check out those, if you are interested.

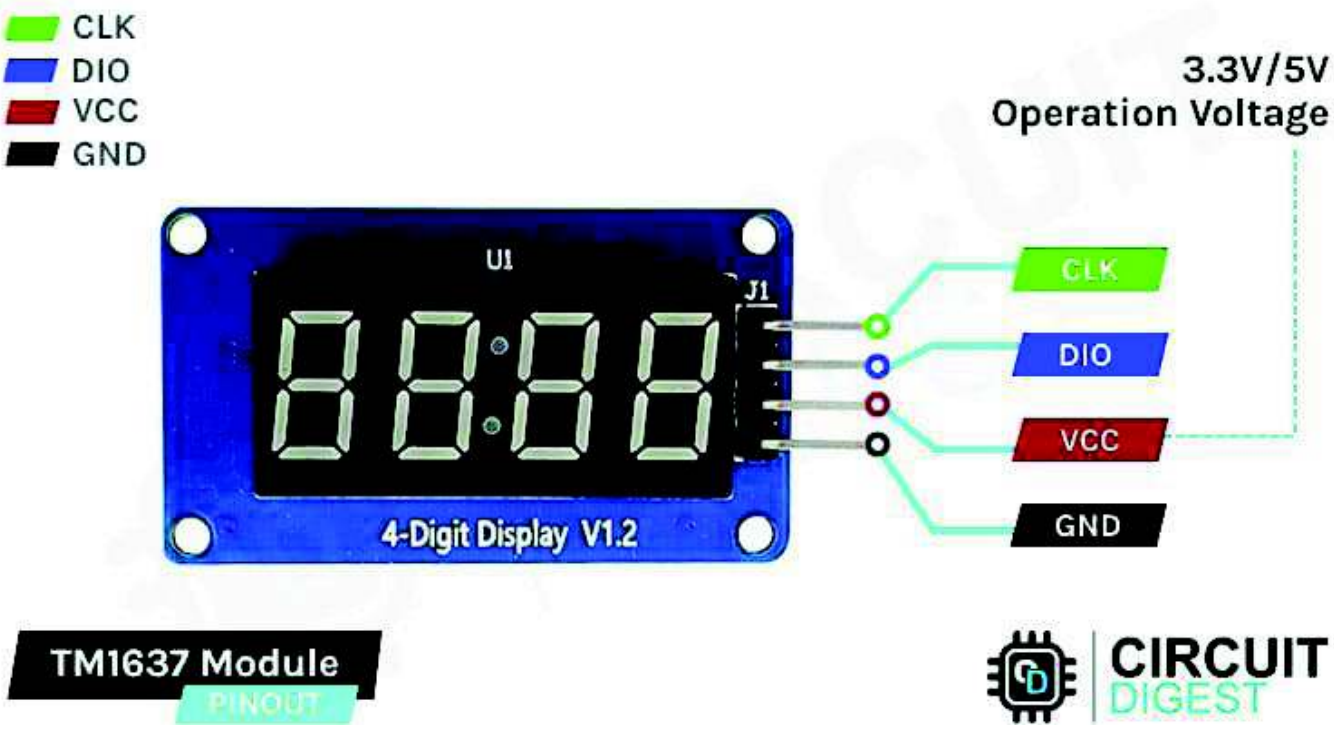
TM1637 IC Basic Features

TM1637 is a kind of **LED** (light-emitting diode display) drive controller special circuit with a keyboard scan interface. It is internally integrated with MCU digital interface, data latch, and LED high-pressure drive. Other than that, this IC also has the capability to control the brightness of the display, which means the display can be **dimmable** if it's required for the project. This device comes in a DIP and SOIC package and can be used for many different applications like display drives of induction cookers, microwave ovens, and small household electrical appliances.

Other than the banner feature, the TM1637 has some interesting features that can make the coding process a lot easier. You can serial upload the display data in the IC, and the IC takes care of refreshing the display, reducing the overhead from the microcontroller. Now if you take a close look at the display module, it features a ‘colon’ that can be used for timer-based projects. The operating voltage for this module is 3.3V to 5V and for the communication process, it uses a two-wire bus rather than the I2C bus so if you are writing a library for this device you need to take note of that. For more information on the IC, you can check the TM1637 7-Segment display driver datasheet IC.

TM1637 Seven Segment Display Module Pinout

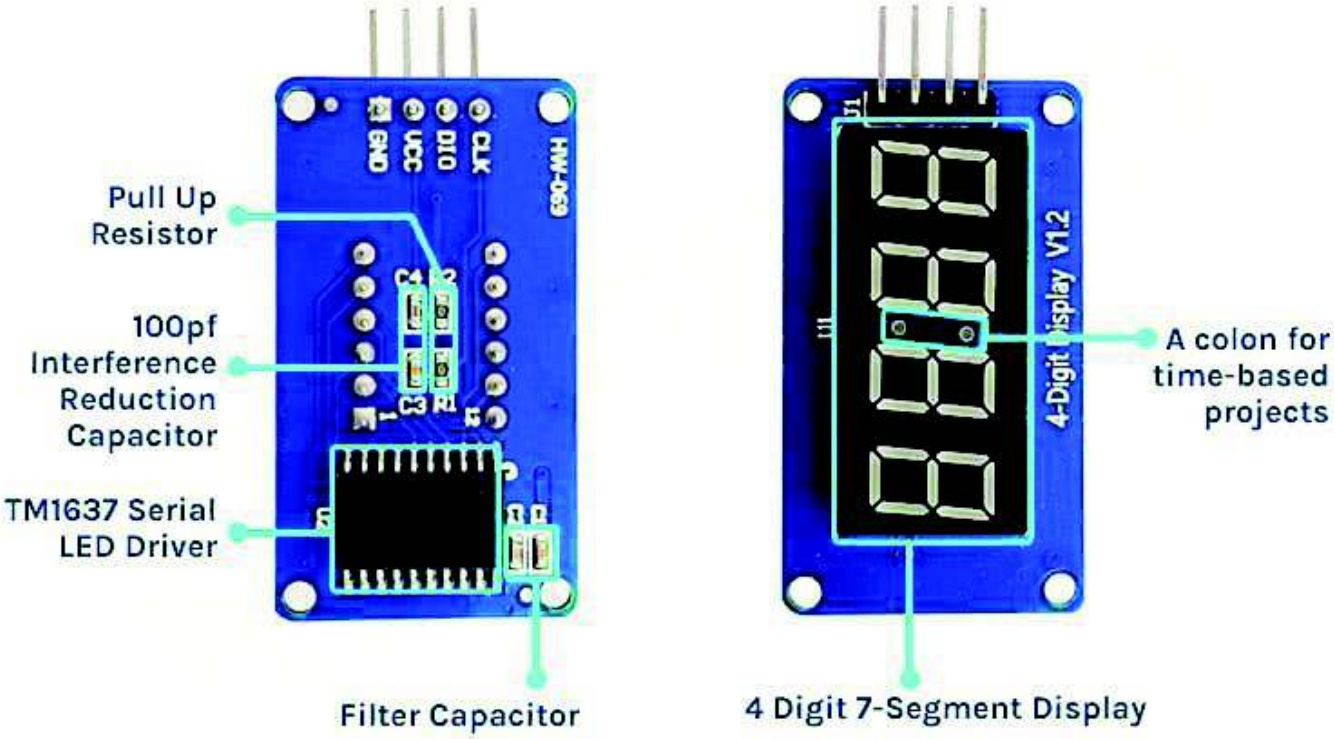
The pinout of the **TM1637 4-digit Seven Segment Display** is shown below. It has 4 pins those are CLK, DIO, VCC, and GND. All the pins of this sensor module are digital, except VCC and Ground, and the device can operate in the 3.3V to 5V voltage range. The Pinout of seven segment display module is shown below:



CLK is a clock input pin. Connect to any digital pin on Arduino Uno.
DIO is the Serial Data I/O pin. Connect to any digital pin on Arduino Uno.
VCC pin supplies power to the module. Connect it to the 3.3V to 5V power supply.
GND is a ground pin.

TM1637 Seven Segment Display Module Parts

The TM1637 module is a low-power, low-cost display module that can be used for many different applications. The **parts marking of the TM1637 4-Digit 7-Segment Display Module** is shown below.



The TM1637 module consists of two parts; the first is a 4-digit 7-segment display and the second one is the TM1637 7-Segment Display Driver IC. The IC supports many functionalities including on-off and brightness control. This IC also has a data queue which means that you can send all the data packets to the IC and the IC will display all the information sequentially, giving headroom to your microcontroller for other tasks.

Commonly Asked Questions

Q. What is TM1637 IC?

TM1637 is a kind of LED (light-emitting diode display) drive control special circuit with a keyboard scan interface and it's internally integrated with MCU digital interface, data latch, LED high-pressure drive, and keyboard scan.

Q. Is TM1637 an I2C?

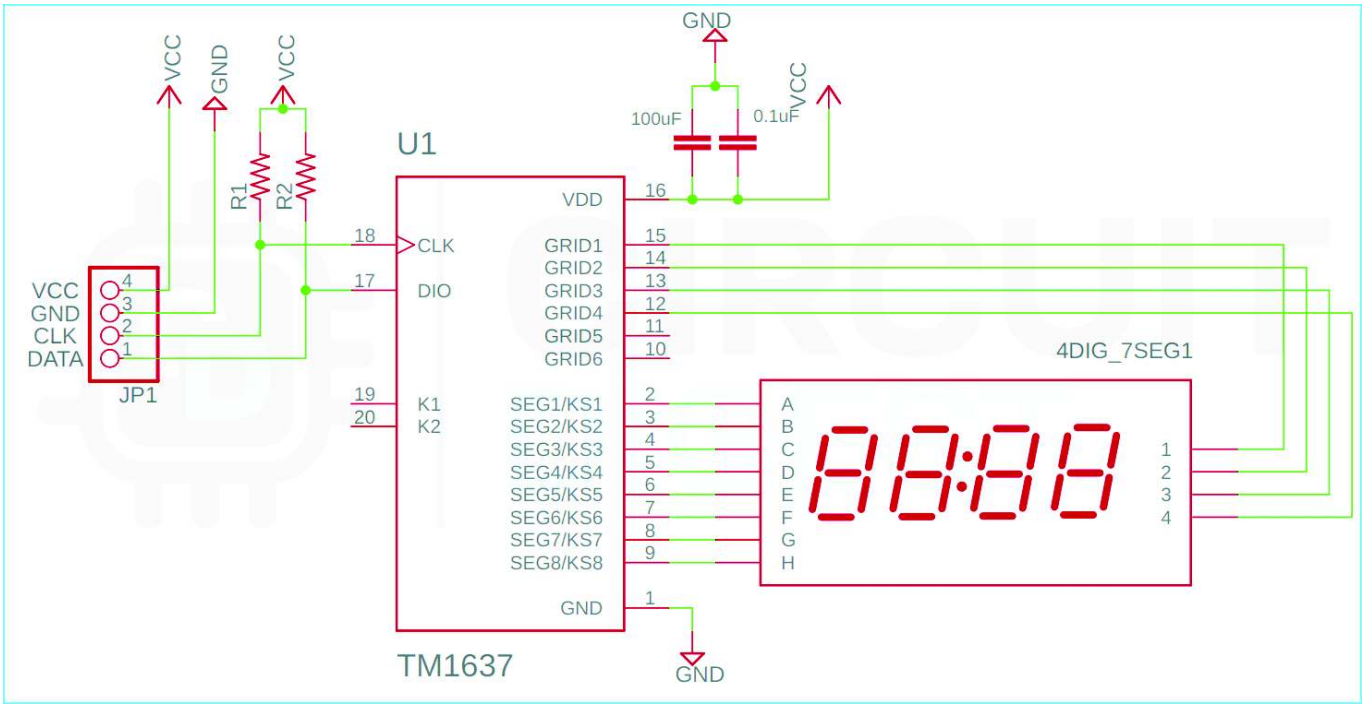
No, the TM1637 does not use I2C, it uses a digital interface so any digital pin of a microcontroller can drive this display without any issues.

Q. How many types of 7-segment displays are there and what is the TM1637 Module Use?

There are two types of LED 7-segment displays: common cathode (CC) and common anode (CA). The difference between the two displays is that the common cathode has all the cathodes of the 7-segments connected directly together and the common anode has all the anodes of the 7-segments connected together. The TM1637 module uses a common cathode display.

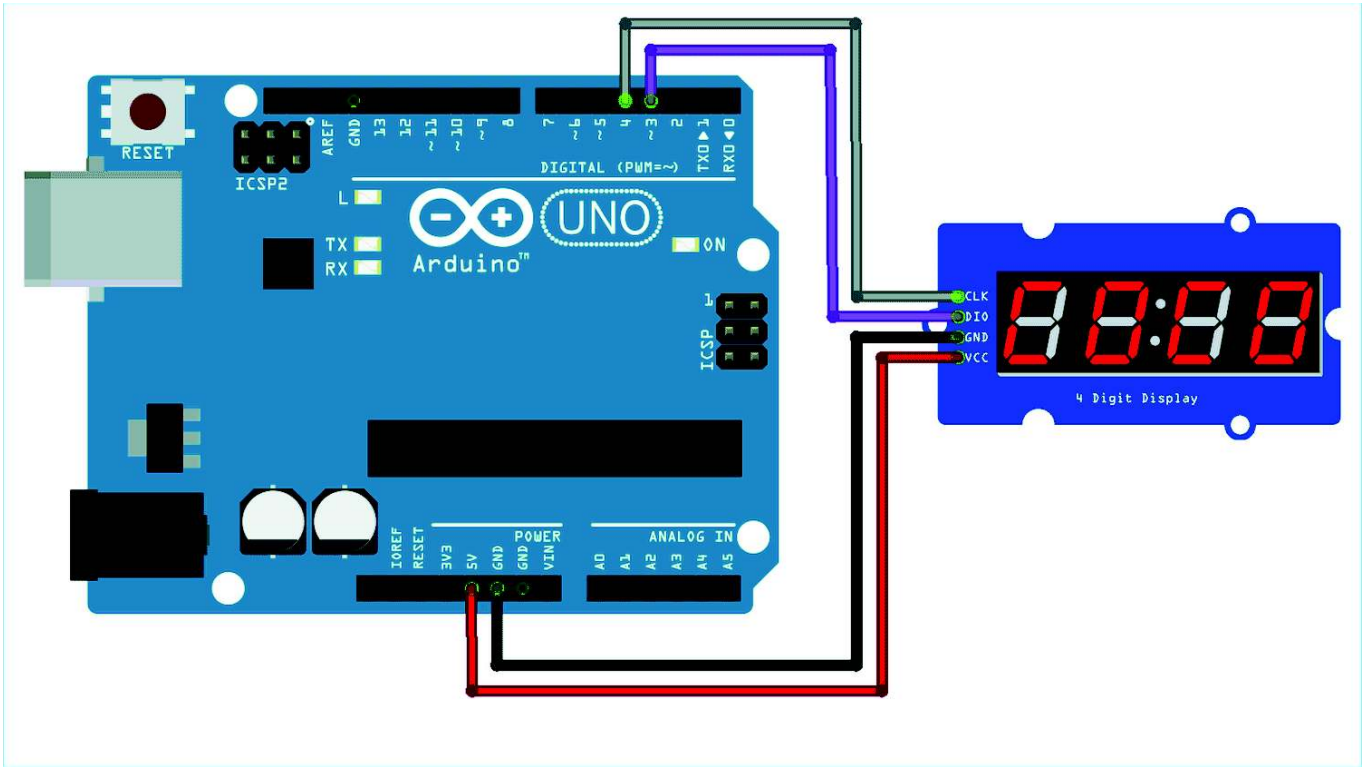
TM1637 4-Digit 7-Segment Display Module Circuit Diagram

The **circuit diagram of the 4-digit 7-segment display** is very simple and needs a couple of components to work properly. The main component of the module is the TM1637 IC and the display module. Other than that, we need 4 capacitors and two resistors and that's all for the components required to build the circuit. The complete schematic diagram of the TM1637 Module is shown below.



Arduino and TM1637 4-Digit 7-Segment Display Circuit Diagram

Now that we have completely understood how a TM1637 seven-segment display driver IC works, we can connect all the required wires to Arduino Uno and write the code to get all the data out from the sensor. The Circuit Diagram of the TM1637 module with Arduino is shown below-



Connecting the TM1637 seven-segment display module with the Arduino is very simple. To communicate with the module we just need to connect the power lines to the 3.3V or 5V pin of the Arduino Uno and we are connecting the data line to the digital pin 3 and digital pin 4 of the Arduino Uno.

Code for Interfacing TM1637 4-Digit 7-Segment Display Module with Arduino

The code to process the display data to the TM1637 display is very simple and easy to understand. We just need to define the pins through which the 7-segment display module is connected to the Arduino and other things will be handled by the Arduino TMS1637 Library. For demonstration purposes, we will be using the example code of the library.

So we start our code by including all the required libraries and we define the CLK and DIO pin. We'll also define the delay between every test.

```
#include <Arduino.h>
#include <TM1637Display.h>
// Module connection pins (Digital Pins)
#define CLK 3
#define DIO 4
// The amount of time (in milliseconds) between tests
#define TEST_DELAY   2000
```

Then we define an array, in this arry the word dOnE is stored.

```
const uint8_t SEG_DONE[] = {
            SEG_B | SEG_C | SEG_D | SEG_E | SEG_G,           // d
            SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F,   // O
            SEG_C | SEG_E | SEG_G,                           // n
            SEG_A | SEG_D | SEG_E | SEG_F | SEG_G            // E
            };
```

Next, we initialize an object name display and pass the CLK and DIO pins in that instance.

```
TM1637Display display(CLK, DIO);
```

Next, we have our setup() function. In the setup() function, we don't need to assign anything because all the major operations happen in the loop section.

```
void setup() {
  // Intialize the object
  myDisplay.begin();
  // Set the brightness of the display (0-15)
  myDisplay.setIntensity(2);
  // Clear the display
  myDisplay.displayClear();
}
```

Next, we have the void loop() function. In the loop() function, the code is written as such that it will go through a set of sequences and display a lot of characters sequentially. We start off our loop() function by defining a variable that will be used for the loop counter later in the code. Next, we have two arrays. The first data array is to light up all the segments and the dots and the second blank array will be used to turn off the display. And the setBrightness() function will be used to set the brightness of the display.

```
  int k;
  uint8_t data[] = { 0xff, 0xff, 0xff, 0xff };
  uint8_t blank[] = { 0x00, 0x00, 0x00, 0x00 };
  display.setBrightness(0x0f);
```

Next, we turn on all the segments and add a predefined delay.

```
 // All segments on
  display.setSegments(data);
  delay(TEST_DELAY)
```

Now the next portion of the code is there to display the 0 1 2 3 digits on the seven-segment display.

```
  // Selectively set different digits
  data[0] = display.encodeDigit(0);
  data[1] = display.encodeDigit(1);
  data[2] = display.encodeDigit(2);
  data[3] = display.encodeDigit(3);
  display.setSegments(data);
  delay(TEST_DELAY);
```

The next portion of the code prints No 23 on the display.

```
  display.clear();
  display.setSegments(data+2, 2, 2);
  delay(TEST_DELAY);
  display.clear();
  display.setSegments(data+2, 2, 1);
  delay(TEST_DELAY);
  display.clear();
  display.setSegments(data+1, 3, 1);
  delay(TEST_DELAY);
```

The next portion of the code displays numbers without leading zeros.

```
// Show decimal numbers with/without leading zeros
  display.showNumberDec(0, false); // Expect: ___0
  delay(TEST_DELAY);
  display.showNumberDec(0, true);  // Expect: 0000
  delay(TEST_DELAY);
            display.showNumberDec(1, false); // Expect: ___1
            delay(TEST_DELAY);
  display.showNumberDec(1, true);  // Expect: 0001
  delay(TEST_DELAY);
  display.showNumberDec(301, false); // Expect: _301
  delay(TEST_DELAY);
  display.showNumberDec(301, true); // Expect: 0301
  delay(TEST_DELAY);
  display.clear();
  display.showNumberDec(14, false, 2, 1); // Expect: _14_
  delay(TEST_DELAY);
  display.clear();
  display.showNumberDec(4, true, 2, 2);  // Expect: 04__
  delay(TEST_DELAY);
  display.showNumberDec(-1, false);  // Expect: __-1
  delay(TEST_DELAY);
  display.showNumberDec(-12);        // Expect: _-12
  delay(TEST_DELAY);
  display.showNumberDec(-999);       // Expect: -999
  delay(TEST_DELAY);
  display.clear();
  display.showNumberDec(-5, false, 3, 0); // Expect: _-5_
```

```
  delay(TEST_DELAY);
  display.showNumberHexEx(0xf1af);         // Expect: f1Af
  delay(TEST_DELAY);
  display.showNumberHexEx(0x2c);           // Expect: __2C
  delay(TEST_DELAY);
  display.showNumberHexEx(0xd1, 0, true); // Expect: 00d1
  delay(TEST_DELAY);
  display.clear();
  display.showNumberHexEx(0xd1, 0, true, 2); // Expect: d1__
  delay(TEST_DELAY);
```

The next portion of the code controls the brightness of the display. If you are looking for how you can control the brightness of the TM1637 display you can refer to this portion of the code.

```
// Brightness Test
  for(k = 0; k < 4; k++)
            data[k] = 0xff;
  for(k = 0; k < 7; k++) {
    display.setBrightness(k);
    display.setSegments(data);
    delay(TEST_DELAY);
  }
```

The next portion of the code does the ON/OFF test.

```
// On/Off test
  for(k = 0; k < 4; k++) {
    display.setBrightness(7, false);  // Turn off
    display.setSegments(data);
    delay(TEST_DELAY);
    display.setBrightness(7, true); // Turn on
    display.setSegments(data);
    delay(TEST_DELAY);
  }
```

Finally, we print the dOnE message that we have defined at the start of the code. If you see the done message in the display, you will know that it has gone through the full loop.