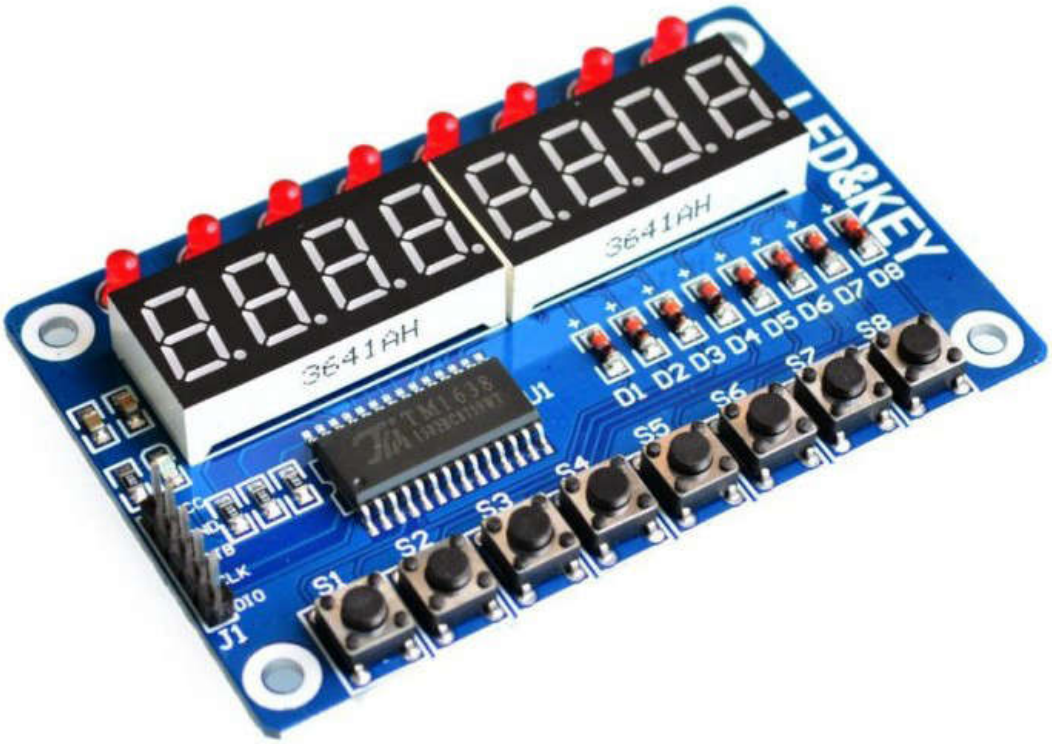
7 SEGMENT MODULE(8DIGIT)



It’s worth noting that you can easily build a nice user interface comprising buttons and digits for your next electronics project with a cheap Chinese display module – the TM1638!

At the heart of the TM1638 module is the TM1638 IC from *Titan Micro Electronics* (www.titanmec.com). It is a clever chip dedicated to light emitting diode display drive/control and equipped with a keypad scan interface. It integrates a microcontroller digital interface, data latch, light emitting diode driver, and keypad scanner.

TM1637 & TM1638 ICs

As you might have noticed, I already did some experiments with its predecessor – the TM1637 IC (https://www.electroschematics.com/universal-digital-thermostat-v1/).

Both the TM1637 and TM1638 special driver chips are commonly used in 7-segment LED displays. However, the former TM1637 IC employs an I2C-like 2-wire (DIO,CLK) bi-directional data communication interface. Both are open-collector terminals with requisite pull-up resistors, but the TM1638 IC adopts an SPI-like 3-wire (DIO,CLK,STB) communication interface. Certainly it's not SPI because the standard SPI's MISO and MOSI lines are merged into one bi-directional DIO line. Since there is no 'start' and 'end' signaling, the digital signal interface becomes more rustic. TM1638 English Datasheet https://retrocip.cz/files/tm1638.pdf. So far so good!

TM1638 IC – In depth

The Chinese description of the module is "8 Bits, 8 keys, 8 LEDs, 8 digital tubes, common cathode

LED digital tube". The module has just 3 interface pins plus 2 pins for power (5VDC) and ground (0V). See the quick specs below:

- Chipset: TM1638
- Supply Voltage: 5VDC
- Display: 8 x LEDs and 8 x 7 Segment Displays
- Input: 8 x Push Button Switches
- Control pins: Data, Clock, Strobe
- Size: 75 x 48mm

One major advantage of the TM1638 module is that it calls for just three I/Os (Data, Clock, Strobe) of the concerned microcontroller for the full play. As mentioned, the interface pins are data, clock, and strobe. The strobe (STB) and clock (CLK) pins are only outputs while the data (DIO) pin works for both input and output. The strobe pin is required when sending data to the module.

While sending data you must set the clock pin to 'low', then you set the data pin, and then set the clock pin back to 'high' to commit the bit value. Perhaps you're already familiar with this method as it is a common way of sending data with shift registers. And hence it's possible to use the standard 'shiftOut' function to send 8 bits of data with just a single line of code.

The module has 4 functions, viz. activate/deactivate the module and initialize the display, write a byte at specific address, write bytes starting from specific address, and read button inputs. The data sent to the board follows a specific protocol. Tthe first byte tells the board what we want to do, and it's followed by zero or more bytes that are arguments for the selected function.

Arguments are sent separately. The strobe pin needs to be set to ‘high’ after sending the command and again to ‘low’ before sending arguments.

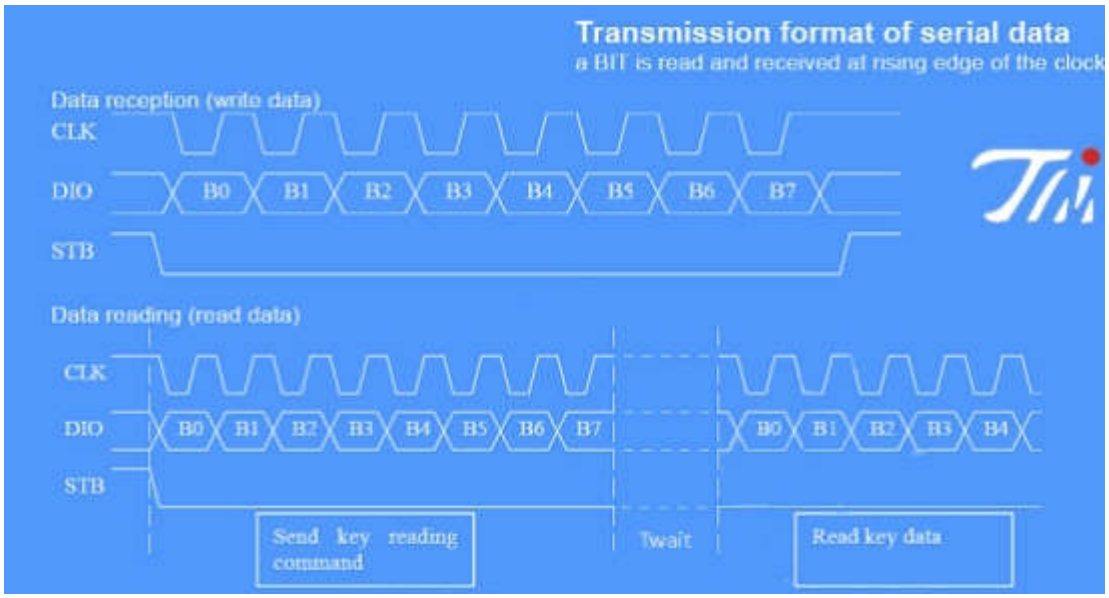
Following is the key features of the 28-pin TM1638 IC as found in its datasheet:

- CMOS technology
- 10 segments × 8 bits display
- Keypad scanning (8 × 3 bits)
- Brightness adjustment circuit (8-level adjustable duty ratio)
- Serial interfaces (CLK, STB, DIO)
- Oscillation mode: RC oscillation
- Built-in power-on reset circuit
- Package type: SOP28

And here is its interface/control pin function (again from the datasheet):

- DIO: Input serial data at rising edge of the clock, starting from lower bits. Output serial data at falling edge of the clock, starting from lower bits. During output, this is a PMOS open drain output
- CLK: Read serial data at rising edge and output data at falling edge
- STB: Initialize the serial interface at falling edge, then wait to receive instructions. The first byte after STB becomes low is considered as an instruction. When an instruction is being processed, other current processes are terminated. When STB is high, CLK is ignored.

Take note, when DIO outputs data, it’s an NMOS open drain output. In order to read the keypad, an external pull-up resistor (1K-10K) must be used. Furthermore, at the falling edge of the clock, DIO controls the operation of NMOS, at which point, the reading will become unstable until the rising edge of the clock.



Run a quick test

An utterly simple way to take a test run of the TM1638 is to connect it with an Arduino board because the module is just a nifty piece of hardware but comes with zero documentation. If you want to learn how your TM1638 module works, you should spend countless hours searching for more information on its background electronics. Better, opt for proven Arduino Libraries as they're most helpful for taking a (layman's) quick test. Hopefully you can make something serious and useful later! This is the download link of one TM1638 Arduino Library by Ricardo Batista I quickly found in GitHub https://github.com/rjbatista/tm1638-library.

My first thought was to use an example sketch from the aforesaid library to test my quick setup. Later I searched for a useful piece of example code without any 'beefy' library and came across a pretty great demo code by MOOZZYK ARDUINO. Since I got the clever code working with my Arduino Uno. I'm sharing the same piece of code below (I still have lots to look at).

Now you have enough resources to breathe life into the display module. First you need to connect the module with your Arduino board. This is how I wired it:

TM1638 Module	**Arduino Uno**
VCC	5V (3.3V minimum)
GND	GND
STB	D7
CLK	D9
DIO	D8

Thereafter you can try this versatile demo code:

```
/* https://github.com/moozzyk/TM1638/blob/master/TM1638_demo/TM1638_demo.ino
*/

const int strobe = 7; // STB to D7

const int clock = 9; // CLK to D9

const int data = 8; // DIO to D8


void sendCommand(uint8_t value)

{

  digitalWrite(strobe, LOW);

  shiftOut(data, clock, LSBFIRST, value);

  digitalWrite(strobe, HIGH);

}
```

```
void reset()
{
  sendCommand(0x40); // Set auto increment mode
  digitalWrite(strobe, LOW);
  shiftOut(data, clock, LSBFIRST, 0xc0);   // Set starting address to 0
  for(uint8_t i = 0; i < 16; i++)
  {
    shiftOut(data, clock, LSBFIRST, 0x00);
  }
  digitalWrite(strobe, HIGH);
}

void setup()
{
  pinMode(strobe, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(data, OUTPUT);

  sendCommand(0x8f);  // Activate with maximum display brightness
  reset();
}

#define COUNTING_MODE 0
#define SCROLL_MODE 1
```

```
#define BUTTON_MODE 2



void loop()

{

  static uint8_t mode = COUNTING_MODE;



  switch(mode)

  {

  case COUNTING_MODE:

    mode += counting();

    break;

  case SCROLL_MODE:

    mode += scroll();

    break;

  case BUTTON_MODE:

    buttons();

    break;

  }



  delay(200);

}

bool counting()

{

                        /*0*/ /*1*/ /*2*/ /*3*/ /*4*/ /*5*/ /*6*/ /*7*/ /*8*/
/*9*/

  uint8_t digits[] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,
0x6f };
```

```
  static uint8_t digit = 0;


  sendCommand(0x40);
  digitalWrite(strobe, LOW);
  shiftOut(data, clock, LSBFIRST, 0xc0);
  for(uint8_t position = 0; position < 8; position++)
  {
    shiftOut(data, clock, LSBFIRST, digits[digit]);
    shiftOut(data, clock, LSBFIRST, 0x00);
  }


  digitalWrite(strobe, HIGH);


  digit = ++digit % 10;
  return digit == 0;
}


bool scroll()
{
  uint8_t scrollText[] =
  {
    /* */ /* */ /* */ /* */ /* */ /* */ /* */ /* */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /*H*/ /*E*/ /*L*/ /*L*/ /*O*/ /*.*/ /*.*/ /*.*/
    0x76, 0x79, 0x38, 0x38, 0x3f, 0x80, 0x80, 0x80,
    /* */ /* */ /* */ /* */ /* */ /* */ /* */ /* */
```

```
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  /*H*/ /*E*/ /*L*/ /*L*/ /*O*/ /*.*/ /*.*/ /*.*/
  0x76, 0x79, 0x38, 0x38, 0x3f, 0x80, 0x80, 0x80,
};

static uint8_t index = 0;
uint8_t scrollLength = sizeof(scrollText);

sendCommand(0x40);
digitalWrite(strobe, LOW);
shiftOut(data, clock, LSBFIRST, 0xc0);

for(int i = 0; i < 8; i++)
{
  uint8_t c = scrollText[(index + i) % scrollLength];

  shiftOut(data, clock, LSBFIRST, c);
  shiftOut(data, clock, LSBFIRST, c != 0 ? 1 : 0);
}

digitalWrite(strobe, HIGH);

index = ++index % (scrollLength << 1);

return index == 0;
```

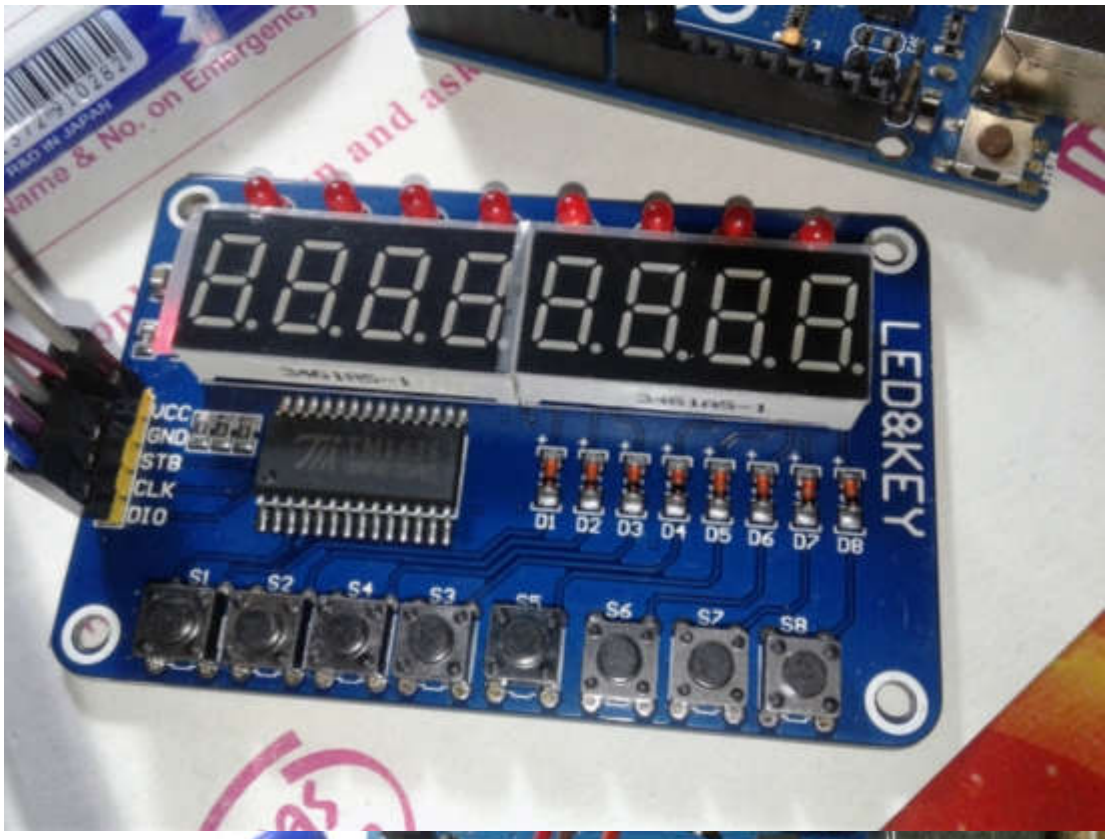
```
}


void buttons()
{
  uint8_t promptText[] =
  {
    /*P*/ /*r*/ /*E*/ /*S*/ /*S*/ /* */ /* */ /* */
    0x73, 0x50, 0x79, 0x6d, 0x6d, 0x00, 0x00, 0x00,
    /* */ /* */ /* */ /* */ /* */ /* */ /* */ /* */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /*b*/ /*u*/ /*t*/ /*t*/ /*o*/ /*n*/ /*S*/ /* */
    0x7c, 0x1c, 0x78, 0x78, 0x5c, 0x54, 0x6d, 0x00,
    /* */ /* */ /* */ /* */ /* */ /* */ /* */ /* */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  };

  static uint8_t block = 0;

  uint8_t textStartPos = (block / 4) << 3;
  for(uint8_t position = 0; position < 8; position++)
  {
    sendCommand(0x44);
    digitalWrite(strobe, LOW);
    shiftOut(data, clock, LSBFIRST, 0xC0 + (position << 1));
    shiftOut(data, clock, LSBFIRST, promptText[textStartPos + position]);
    digitalWrite(strobe, HIGH);
  }
```

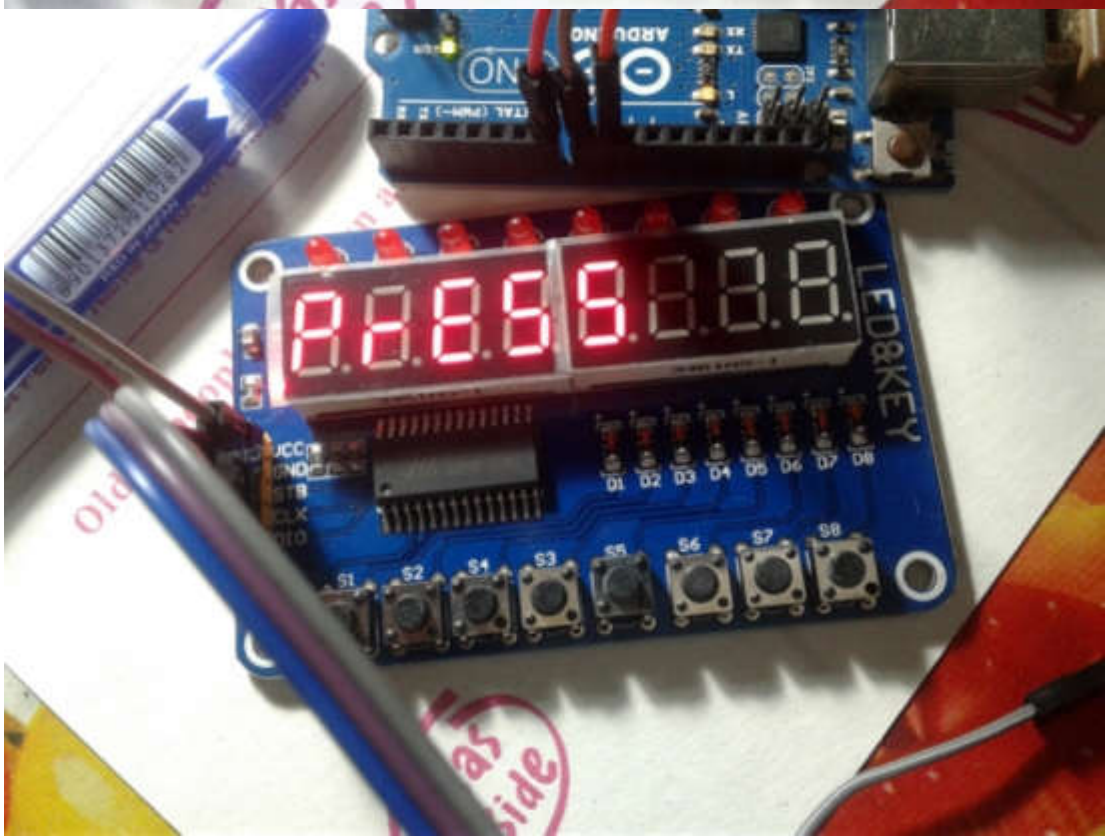
```
  block = (block + 1) % 16;

  uint8_t buttons = readButtons();

  for(uint8_t position = 0; position < 8; position++)
  {
    uint8_t mask = 0x1 << position;

    setLed(buttons & mask ? 1 : 0, position);
  }
}

uint8_t readButtons(void)
{
  uint8_t buttons = 0;
  digitalWrite(strobe, LOW);
  shiftOut(data, clock, LSBFIRST, 0x42);

  pinMode(data, INPUT);

  for (uint8_t i = 0; i < 4; i++)
  {
    uint8_t v = shiftIn(data, clock, LSBFIRST) << i;
    buttons |= v;
```

```
  }


  pinMode(data, OUTPUT);
  digitalWrite(strobe, HIGH);
  return buttons;
}


void setLed(uint8_t value, uint8_t position)
{
  pinMode(data, OUTPUT);


  sendCommand(0x44);
  digitalWrite(strobe, LOW);
  shiftOut(data, clock, LSBFIRST, 0xC1 + (position << 1));
  shiftOut(data, clock, LSBFIRST, value);
  digitalWrite(strobe, HIGH);
}
```

The above code is just a demo version, it runs sequentially through modes until it reaches the last mode (buttons) which will be active for the rest of the time. The demo code may allow you to develop custom codes for your favorite microcontroller to work with TM1638 display modules. Give it a good try!

On a side note, the TM1638 display module is a bit older. Today you can find a vast range of dedicated TM1638 Arduino Libraries. The best way to learn its secrets is to try plain text example sketches (without library inclusions) similar to the one presented here (thanks a lot for your clever work, dear seasoned coder)!


LED&KEY
VCC
GND
STB
CLK
DIO
D1 D2 D3 D4 D5 D6 D7 D8
S1 S2 S4 S3 S5 S6 S7 S8


LED&KEY
D1 D2 D3 D4 D5 D6 D7 D8
S1 S2 S4 S3 S5 S6 S7 S8

You might have noticed that in the demo code “0x8f” is used to activate the board and set the display brightness to maximum level. To write a byte at specific address you need to use “0x44” command, then the address, followed by the value. Likewise, to write values at consecutive addresses you can use “0x40” followed by the starting address, and then followed by the values you want to write. In order to read the buttons, at first you need to send the “0x42” command, then set the data pin as an input pin. Finally, you need to read 4 bytes from the board bit by bit (TM1638 can be read up to four bytes only. Data read is in the order BYTE1 to BYTE4 without skipping any byte). For learning more about these instructions, you may go through the TM1638 English (PDF) datasheet (pages 3 to 10 in particular). Also see the below table.

Command	Arguments	Description
0x8? (1000abbb)	(none)	activate board (bit a), set brightness (bits b)
0x44 (01000100)	0xc? 0x??	write value 0x?? at location 0xc? (single address mode)
0x40 (01000000)	0xc? 0x?? 0x?? 0x??	write values 0x?? starting from location 0xc? (address auto increment mode)
0x42 (01000010)	N/A	read buttons

Good news for Attiny chip lovers!

This is one tinyAVR (ATtiny13, ATtiny25, ATtiny45, ATtiny85, and other) library for TM1638 LED display controller modules https://github.com/lpodkalicki/attiny-tm1638-library/