8
CONTROLLING HIGH-POWER SHIFT REGISTERS

Many Arduino users rely on the 74HC595 shift register for their projects, as it's popular and easy to use. However, the 74HC595 can handle only a relatively low amount of current, especially when all pins are active: while you can draw 20 mA of current from one output continuously, the entire IC is rated for a draw of only 70 mA through the V_{CC} or GND pin.

If you need 20 mA per output to drive, for instance, eight separate LEDs, you could use only three of the eight pins under recommended operating conditions. While it may be possible to exceed the manufacturer's direction, good electronic designs should consider safety and reliability. A shift register designed to handle higher currents is a better choice.

This chapter shows you how to control higher-powered devices with your Arduino with the TPIC6B595 shift register IC. You'll learn to:

- Experiment with a binary number display
- Use multiple TPIC6B595s to control more than eight high-powered outputs
- Use bright Piranha-style LEDs that are more powerful than regular LEDs

You'll also build a PC-controlled eight-relay board and control giant seven-segment numeric displays.

Introducing the TPIC6B595

The TPIC6B595 is controlled in the same way as the 74HC595 but offers up to 150 mA per output and a 500 mA IC total current draw—just over 60 mA per pin when all pins are used. It can also switch voltages up to 50 V DC. This allows for control of eight higher-current items such as powerful LEDs, relay coils, or mechanical switchgear.

The TPIC6B595 is a *latching shift register*, meaning that as long as power is connected, it will maintain the output status. For example, if you upload a new sketch, the outputs are not affected. If you've powered your circuit from a power supply and not the Arduino, you can reset the Arduino without altering the outputs.

Figure 8-1 shows a TPIC6B595 in a dual in-line package (DIP), through-hole format in a solderless breadboard.

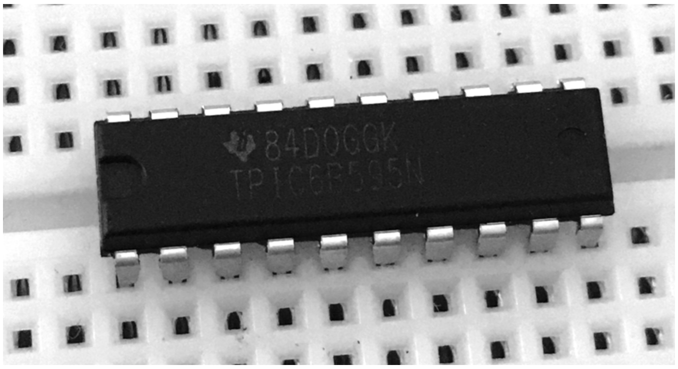


Figure 8-1: A TPIC6B595 shift register

Figure 8-2 shows the TPIC6B595's schematic symbol.

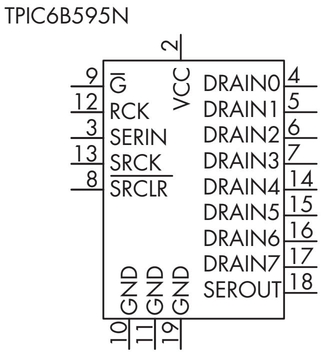


Figure 8-2: The schematic symbol for the TPIC6B595

The eight output pins in the schematic are labeled DRAIN*x*, since the TPIC6B595 has low-side outputs. Like the 2N7000 N-MOSFETs used in previous chapters, each of these outputs controls current entering the pin (as opposed to the 74HC595's high-side outputs, for example, where currents flow out from the eight control pins). This means the devices to be controlled are connected between the power supply and the control pins on the TPIC6B595, which switches the currents' ability to flow from the device to GND.

Consider the schematic in Figure 8-3. Current flows from the 5 V source, through the resistor and LED, and into the TPIC6B595's output pin. When that pin is activated, the current continues to GND, completing the circuit.

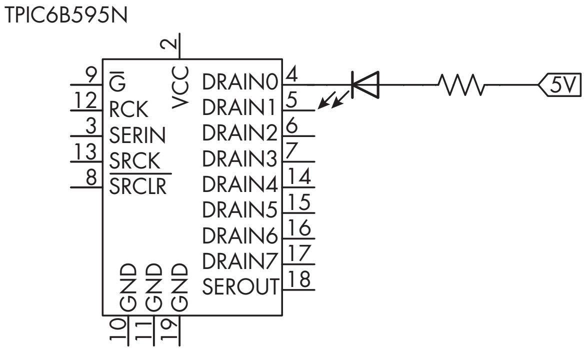


Figure 8-3: An example of LED control with TPIC6B595

Given the TPIC6B595's method of controlling current, the voltage of the devices it controls can be up to 50 V, while the shift register is still operating on 5 V. Conveniently, that means you can control 12 V devices or higher without worrying about level conversion back to the Arduino.

Let's put the TPIC6B595 to the test with a simple project that demonstrates shift register operation.

Project #25: Creating a TPIC6B595 Binary Number Display

This project demonstrates the use of TPIC6B595 outputs while refreshing your knowledge of binary numbers and how they relate to shift register output control. You'll need the following parts:

- Arduino Uno or compatible board and USB cable
- Solderless breadboard
- Various jumper wires
- TPIC6B595 shift register IC
- 0.1 µF capacitor
- Eight LEDs
- Eight 1 kΩ resistors

Assemble the circuit as shown in Figure 8-4.

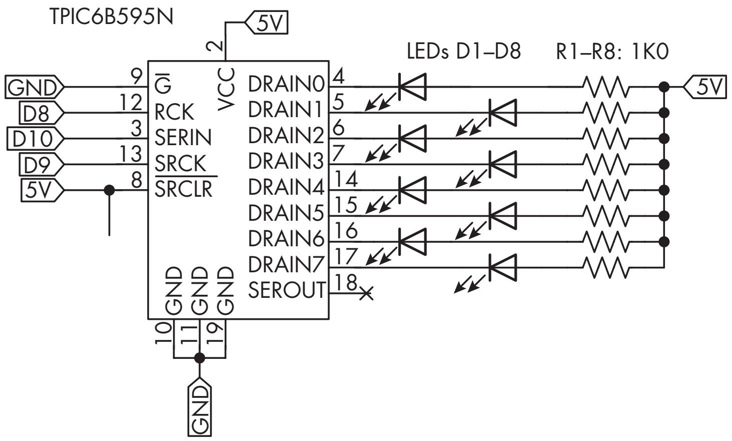
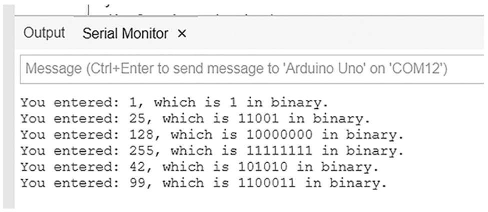


Figure 8-4: The schematic for Project #25

Enter and upload the Project #25 sketch. After a moment, open the Serial Monitor in the IDE and enter a number between 0 and 255, inclusive; then press CTRL-ENTER. The Arduino should respond by displaying the number in binary using the LEDs, as well as in the Serial Monitor, as shown in Figure 8-5. LED 1 will be the least significant bit of the number, which represents 1, and LED 8 will be the most significant bit, which represents 255.


Output
Serial Monitor ×
Message (Ctrl+Enter to send message to 'Arduino Uno' on 'COM12')

```
You entered: 1, which is 1 in binary.
You entered: 25, which is 11001 in binary.
You entered: 128, which is 10000000 in binary.
You entered: 255, which is 11111111 in binary.
You entered: 42, which is 101010 in binary.
You entered: 99, which is 1100011 in binary.
```

Figure 8-5: Example output from Project #25

Let's see how this works:

```
// Project #25 - TPIC6B595 binary number display

❶ #define latch 8 // Latch RCK pin
#define clock 9 // Clock SRCK pin
#define data 10 // Data SERIN pin

void displayBinary(int displayNumber)
{
   digitalWrite(latch, LOW);
   shiftOut(data, clock, MSBFIRST, displayNumber);
   digitalWrite(latch, HIGH);
}
```

```
void setup()
{
 ❷ Serial.begin(9600);
   pinMode(latch, OUTPUT);
   pinMode(clock, OUTPUT);
   pinMode(data, OUTPUT);
}

void loop()
{
   long number = 0;
   long entry = 0;
 ❸ Serial.flush();
   while (Serial.available() == 0) {}
 ❹ while (Serial.available() > 0)
   {
       number = number * 10;
       entry = Serial.read() - '0';
       number = number + entry;
       delay(5);
   }
   displayBinary(number);
   Serial.print("You entered: ");
   Serial.print(number);
   Serial.print(", which is ");
   Serial.print(number, BIN);
   Serial.println(" in binary.");
}
```

The sketch begins by defining the Arduino digital pins used for connecting to the shift register's latch, clock, and data pins, respectively ❶. The custom `displayBinary()` function accepts an integer and sends it to the shift register for output control, using the same method as the 74HC595 shift register mentioned earlier. To send the bits representing the number to the shift register in binary and to activate the pins in the shift register to control the LEDs that will match the number to be displayed in binary, the function uses `MSBFIRST` (most significant bit first).

You can turn shift register outputs on and off with the 8 bits of the number sent to the shift register, as each bit matches an output and a status (1 for HIGH, 0 for LOW). You can also change `MSBFIRST` to `LSBFIRST`, standing for least significant bit first, to see the number "reversed" in binary.

The sketch initializes the Serial Monitor and digital output pins ❷ and then flushes the serial input and waits for the user to enter a number into the Serial Monitor ❸. It then combines the digits of the number entered in the Serial Monitor to make the final number to display ❹. The custom `displayBinary()` function sends that number to the shift register and the Serial Monitor.

You'll put this framework for shift register control to use in the next project.

Project #26: Building a PC-Controlled Relay Board

In this project, you'll build a relay control board with eight single-pole, double-throw (SPDT) relays that you control via a PC or another device with an Arduino-compatible UART. In the future, you might use the technique introduced in this project to control low-voltage lighting or electric door locks, turn speakers on and off, and more.

Each relay in this project is capable of controlling up to 30 V DC at 2 A of current, if you're using the project PCB. If you're using the solderless breadboard, the relays should be used to switch only up to 100 mA or so.

While it's possible to build this project using a solderless breadboard, this requires soldering jumper wires to the relay pins for remotely wiring the relays back to the circuit, as shown in Figure 8-6, as the relay pins don't sit in the breadboard very well. I strongly recommend you use the PCB instead.

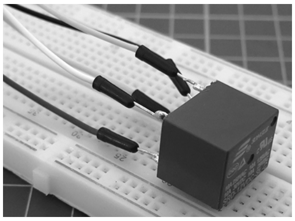


Figure 8-6: A relay with remote wiring for breadboard use

You'll need the following parts for this project:

- Arduino Uno or compatible board and USB cable
- Assorted jumper wires
- 12 V power supply or wall wart with DC plug
- TPIC6B595 shift register
- Twenty-pin IC socket
- 0.1 µF capacitor
- Eight LEDs
- Eight 1 kΩ resistors
- Eight 1N4001 power diodes
- Eight SRD-12VDC-SL-C SPDT relays
- Solderless breadboard or Project #26 PCB

If you're using the PCB, you will also need the following:

- Ten three-way 5.08 mm terminal blocks
- Twenty-pin IC socket
- PCB mount DC socket

Assemble the circuit as shown in Figure 8-7.

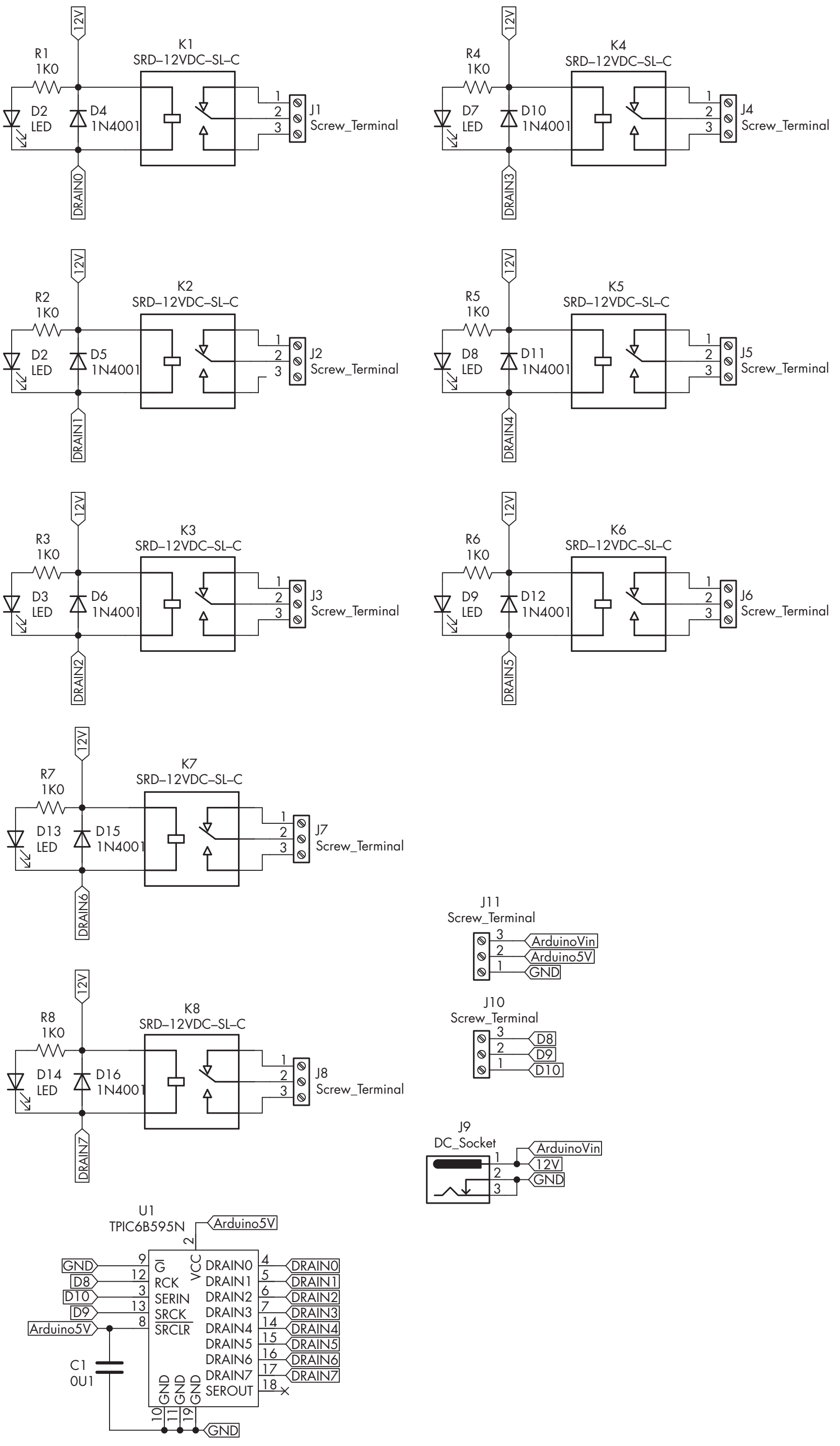


Figure 8-7: The schematic for Project #26

If you're using the PCB, the layout is simple, as shown in Figure 8-8.

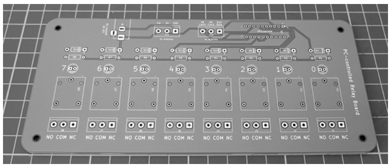


Figure 8-8: The PCB for Project #26

Start by inserting the resistors; then insert the diodes, the LEDs, the IC and DC sockets, and the terminal blocks; and end with the relays. Be sure to insert the shift register correctly—pin 1 is marked on the PCB. Connect to the Arduino via the two terminal blocks next to the shift register, as shown in the schematic and Figure 8-9. The Arduino is powered by the relay board's 12 V supply and returns 5 V to the board to power the shift register.

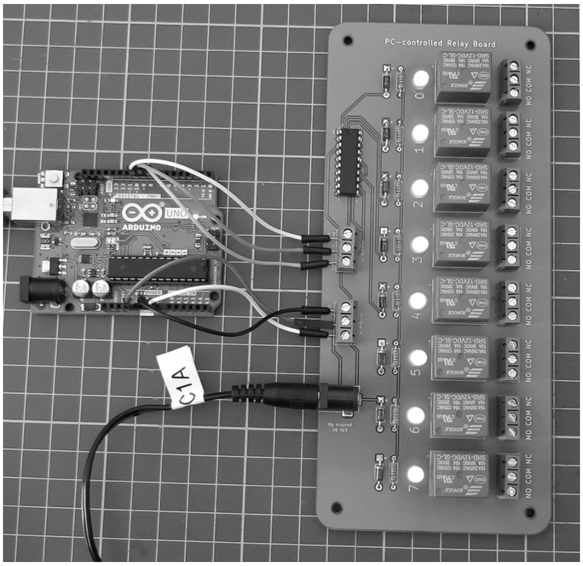
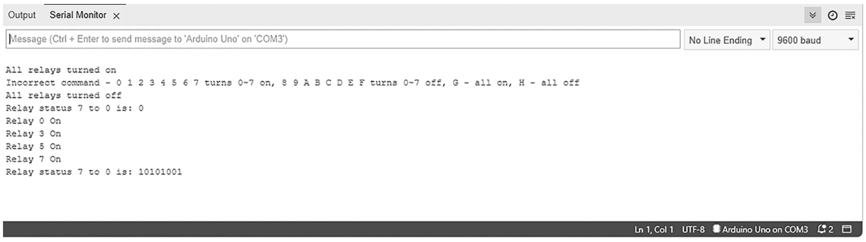


Figure 8-9: The completed hardware for Project #26

Once you've set up the hardware, enter and upload the Project #26 sketch, which controls the relays with various commands. Entering the numbers 0 through 7 using the Serial Monitor or terminal software turns on relays 0 through 7, respectively; entering 8 through F turns off relays 0 to 7; G turns all relays on; and H turns all relays off. Enter **?** to check which relays are on and off based on the commands being entered. The result is returned as a binary number matching the relay order. If the user enters an unrecognized character, the Arduino returns a list of valid commands.

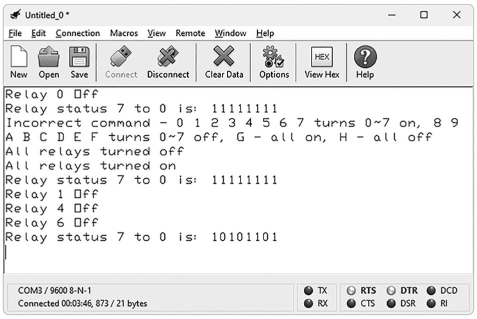
For example, Figure 8-10 shows the output for the commands `G`, then `Q` (which caused the incorrect command message to display), then `H`, and then `0`, `3`, `5`, `7`, and `?`.


Output
Serial Monitor
Message (Ctrl + Enter to send message to 'Arduino Uno' on 'COM3')
No Line Ending
9600 baud
Ln 1, Col 1 UTF-8 Arduino Uno on COM3 2

```
All relays turned on
Incorrect command - 0 1 2 3 4 5 6 7 turns 0~7 on, 8 9 A B C D E F turns 0~7 off, G - all on, H - all off
All relays turned off
Relay status 7 to 0 is: 0
Relay 0 On
Relay 3 On
Relay 5 On
Relay 7 On
Relay status 7 to 0 is: 10101001
```

Figure 8-10: An example of operations in the Serial Monitor

If you're sharing the relay board and Arduino with others, they don't need to run the Arduino IDE for control but can use any terminal software on their PC, Mac, or other computer that supports USB serial. For example, the same operation is possible with Roger Meier's CoolTerm application, available from *http://freeware.the-meiers.org*, as shown in Figure 8-11.


Untitled_0 *
File Edit Connection Macros View Remote Window Help
New Open Save Connect Disconnect Clear Data Options View Hex Help
HEX
COM3 / 9600 8-N-1
Connected 00:03:46, 873 / 21 bytes
TX RX RTS CTS DTR DSR DCD RI

```
Relay 0 Off
Relay status 7 to 0 is: 11111111
Incorrect command - 0 1 2 3 4 5 6 7 turns 0~7 on, 8 9
A B C D E F turns 0~7 off, G - all on, H - all off
All relays turned off
All relays turned on
Relay status 7 to 0 is: 11111111
Relay 1 Off
Relay 4 Off
Relay 6 Off
Relay status 7 to 0 is: 10101101
```

Figure 8-11: Controlling the relays using the CoolTerm application

The control Arduino expects only single characters from the host computer (or other UART), so you can write software for your computer in many environments to control the relays. Search your preferred environment's resources for "plaintext serial over USB" or similar to learn more.

Let's see how this works:

```
// Project #26 - PC-controlled relay board

#define latch 8 // Latch RCK pin
#define clock 9 // Clock SRCK pin
#define data 10 // Data SERIN pin

int relayStatus;

void showStatus()
{
    Serial.print("Relay status 7 to 0 is: ");
    Serial.println(relayStatus,BIN);
}

void waveHello()
{
    int d = 250;
    Serial.println("Hello!");
    allOff();
    for (int a=0; a<8; a++)
    {
        relayOn(a);
        delay(d);
        relayOff(a);
    }
    for (int a=6; a>=0; a--)
    {
        relayOn(a);
        delay(d);
        relayOff(a);
    }
}

void relayOn(int a)
{
    relayStatus = relayStatus|(1<<a);
    sendStatus(relayStatus);
    Serial.print("Relay "); Serial.print(a); Serial.println(" On");
}

void relayOff(int a)
{
    relayStatus = relayStatus^(1<<a);
    sendStatus(relayStatus);
    Serial.print("Relay "); Serial.print(a); Serial.println(" Off");
}

void allOn()
{
    relayStatus = 255;
    sendStatus(relayStatus);
    Serial.println("All relays turned on");
}
```

```
void allOff()
{
    relayStatus = 0;
    sendStatus(relayStatus);
    Serial.println("All relays turned off");
}

void sendStatus(int a)
{
    digitalWrite(latch, LOW);
    shiftOut(data, clock, MSBFIRST, a);
    digitalWrite(latch, HIGH);
}

void setup()
{
    Serial.begin(9600);
    pinMode(latch, OUTPUT);
    pinMode(clock, OUTPUT);
    pinMode(data, OUTPUT);
}

void loop()
{
    char a = 0;
    Serial.flush();
    while (Serial.available() == 0) {}
    while (Serial.available() > 0) ❶
    {
        a = Serial.read();
    }
    switch (a) ❷
    {
        case '0' : relayOn(0); break;
        case '1' : relayOn(1); break;
        case '2' : relayOn(2); break;
        case '3' : relayOn(3); break;
        case '4' : relayOn(4); break;
        case '5' : relayOn(5); break;
        case '6' : relayOn(6); break;
        case '7' : relayOn(7); break;
        case '8' : relayOff(0); break;
        case '9' : relayOff(1); break;
        case 'A' : relayOff(2); break;
        case 'B' : relayOff(3); break;
        case 'C' : relayOff(4); break;
        case 'D' : relayOff(5); break;
        case 'E' : relayOff(6); break;
        case 'F' : relayOff(7); break;
        case 'G' : allOn(); break;
        case 'H' : allOff(); break;
        case 'Z' : waveHello(); break; ❸
        case '?' : showStatus(); break;
        default : Serial.print("Incorrect command - 0 1 2 3 4 5 6 7 turns 0~7 on, "); ❹
        Serial.println("8 9 A B C D E F turns 0~7 off, G - all on, H - all off");
    }
}
```

The sketch first declares the integer `relayStatus`, which holds the status of the relays. Think of this number in binary, with the least significant bit representing relay 0: if this bit is 1, the relay is on, and if it is 0, the relay is off. The custom `showStatus()` function sends the value of `relayStatus` in binary back to the serial interface, so the receiver can see which relays are on or off in their terminal, Serial Monitor, or other software. All the functions that control a relay also send feedback via serial to the user, describing the completed action.

The `relayOn(int a)` function turns on relays, using the bitwise arithmetic operator OR (|) to activate the desired relay without interrupting other relays. The function receives the value `a` and performs a bitwise OR with the `relayStatus` variable; then it updates the relays using the new value of `relayStatus`. For example, if relays 0, 1, 2, and 3 are on, the value of `relayStatus` in binary is currently B00001111. If the user then enters `5` in the Serial Monitor (or terminal software) to turn on relay 5, the program would switch bit 5 to 1, as follows:

```
B00001111 | // Current value of relayStatus
B00100000 = // Perform OR with 1 << 5 — received by the function in a
B00101111   // New value of relayStatus
```

The `sendStatus()` function changes the relays accordingly by updating the shift register outputs.

The `relayOff(int a)` function turns relays off, using the bitwise arithmetic operator XOR (^) to deactivate the desired relay without interrupting other relays. For example, if relays 0, 1, 2, 3, and 5 are on, the value of `relayStatus` in binary is currently B00101111. If you enter `A` in the Serial Monitor or terminal software to turn off relay 3, the program should switch bit 3 to 0, as follows:

```
B00101111 ^ // Current value of relayStatus
B00001000 = // Perform XOR with 1 << 3
B00100111   // New value of relayStatus
```

Again, the `sendStatus()`function changes the relays accordingly by updating the shift register outputs. Two additional functions, `allOn()` and `allOff()`, turn all relays on and off, respectively, by sending 255 (binary `B11111111`) and 0 (binary `B0000000`) to the shift register.

General operation is simple. The Arduino awaits a single character from the serial line ❶. Once a character has arrived, it is matched to a command ❷; the user can activate the simple `waveHello()` function ❸ by pressing `Z` to turn the relays on and off one at a time, for testing and amusement, finally if the character isn't a command, the program sends a quick reference to serial ❹ so the user can learn the command set.

For a challenge, you can modify the sketch so the relay status is saved to the internal EEPROM when changed and the system sets the relays from the EEPROM data after a reset.

Now that you know how to control eight higher-current devices with a single TPIC6B595, I'll show you how to use two or more TPIC6B595s at once.

Using Multiple TPIC6B595s

You can easily use two or more TPIC6B595 shift registers at once to control 16 or more devices, in the same way you would with the 74HC595, but with the ability to handle higher currents. Start by connecting each TPIC6B595's clock lines together, then connecting their latch lines together, then connecting the serial out from the first shift register to the serial in on the second shift register, and finally repeating as required. For example, Figure 8-12 shows the schematic to double the number of LEDs controlled by Project #25.

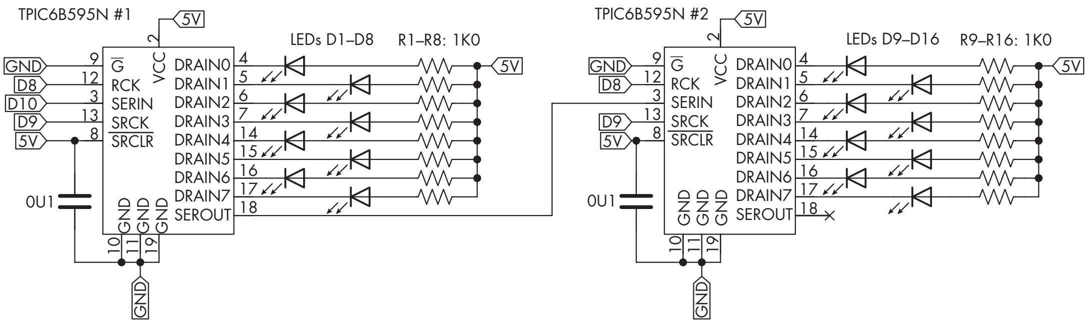


Figure 8-12: The schematic for controlling 16 LEDs with shift registers

Next, send out 2 bytes of data while the latch is low, instead of 1 byte. You'll need to send the byte for the last shift register in the chain first. For example, to send out 2 bytes to the shift registers in Figure 8-12, you would use the following function:

```
void sendData(int a, int b)
{
digitalWrite(latch, LOW);
shiftOut(data, clock, MSBFIRST, b); // For TPIC #2
shiftOut(data, clock, MSBFIRST, a); // For TPIC #1
digitalWrite(latch, HIGH);
}
```

To add one or more additional shift registers, just add more parameters and `shiftOut()` functions, one each for each extra register. You'll use multiple TPIC6B595s in the following project, along with a new type of LED, described in the next section.

Using Piranha-Style LEDs

There's an incredible range of LEDs available on the market, from tiny surface-mount LEDs to those large enough to serve as part of an automotive headlight. One example in between these two is the compact yet very bright Piranha-style LED; its through-hole packaging makes it easy to use. Figure 8-13 shows a pair of Piranha LEDs.

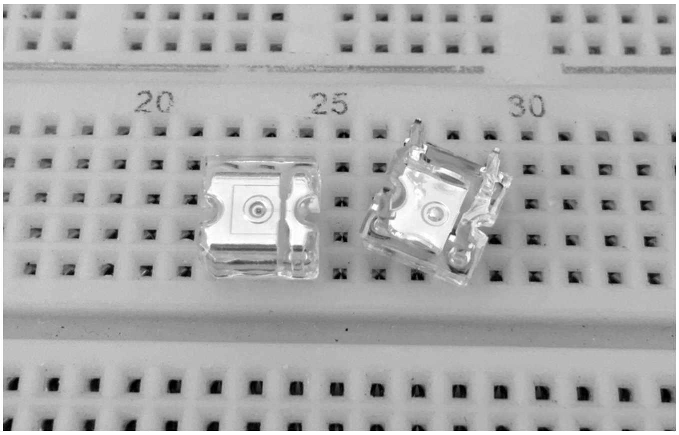


Figure 8-13: Two Piranha LEDs

Each LED has four legs, two anodes, and two cathodes. The two anode pins are electrically connected to each other, as are the two cathode pins. The anode side has the corner cut off at the bottom left, as well as a larger metallic surface compared to the cathode side. Figure 8-14 shows the schematic symbol for Piranha LEDs.

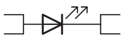


Figure 8-14: The schematic symbol for Piranha-style LEDs

When used alone, these LEDs operate safely at 20 mA of current at around 2 to 2.2 V DC. For the next project, you'll use them in groups of four each, organized in series (rather than parallel to each other). In this configuration, each group of four LEDs will require 9 V DC power and a 47 Ω resistor to maintain the required current and a high brightness.

To calculate the required resistor to use with LEDs in series, you use formula $R = (V_s - V_f) / I_f$, where V_s is the power supply voltage, V_f is the forward voltage (recommended operating voltage) of the LEDs, and I_f is the recommended operating current for the LEDs. For the following project, you'd use 8 V at 20 mA as the LED forward voltage and operating current, with a 9 V supply. This gives you R = (9 V – 8 V) / .02 A, which results in 50 Ω. We don't have a 50 Ω resistor, so 47 Ω is the nearest choice.

Project #27: Creating a Giant Seven-Segment LED Display

Multiple TPIC6B595s work well to drive large numbers of LEDs. In this project, you'll build one or more seven-segment LED displays to create large numerical displays you can use for various purposes. For instance, you might use them to indicate various data generated with an Arduino project, such as temperature, an event count, or time.

To construct a single-digit display, you'll need the following parts. For a larger display, multiply all parts other than the Arduino by the number of digits you'd like to create. This project shows you how to use four digits at once, so multiply by four if you'd like to follow the example exactly. For this project, use the PCB rather than a breadboard.

- An Arduino Uno or compatible board and USB cable
- Assorted jumper wires
- A 9 V power supply or wall wart with DC plug
- A TPIC6B595 shift register
- A 20-pin IC socket
- A 0.1 µF capacitor
- Thirty-two 5 mm Piranha-style LEDs
- Eight 47 Ω resistors
- Four 3-contact 5.08 mm terminal blocks
- A PCB mount DC socket
- The Project #27 PCB

Figure 8-15 shows the project's schematic.

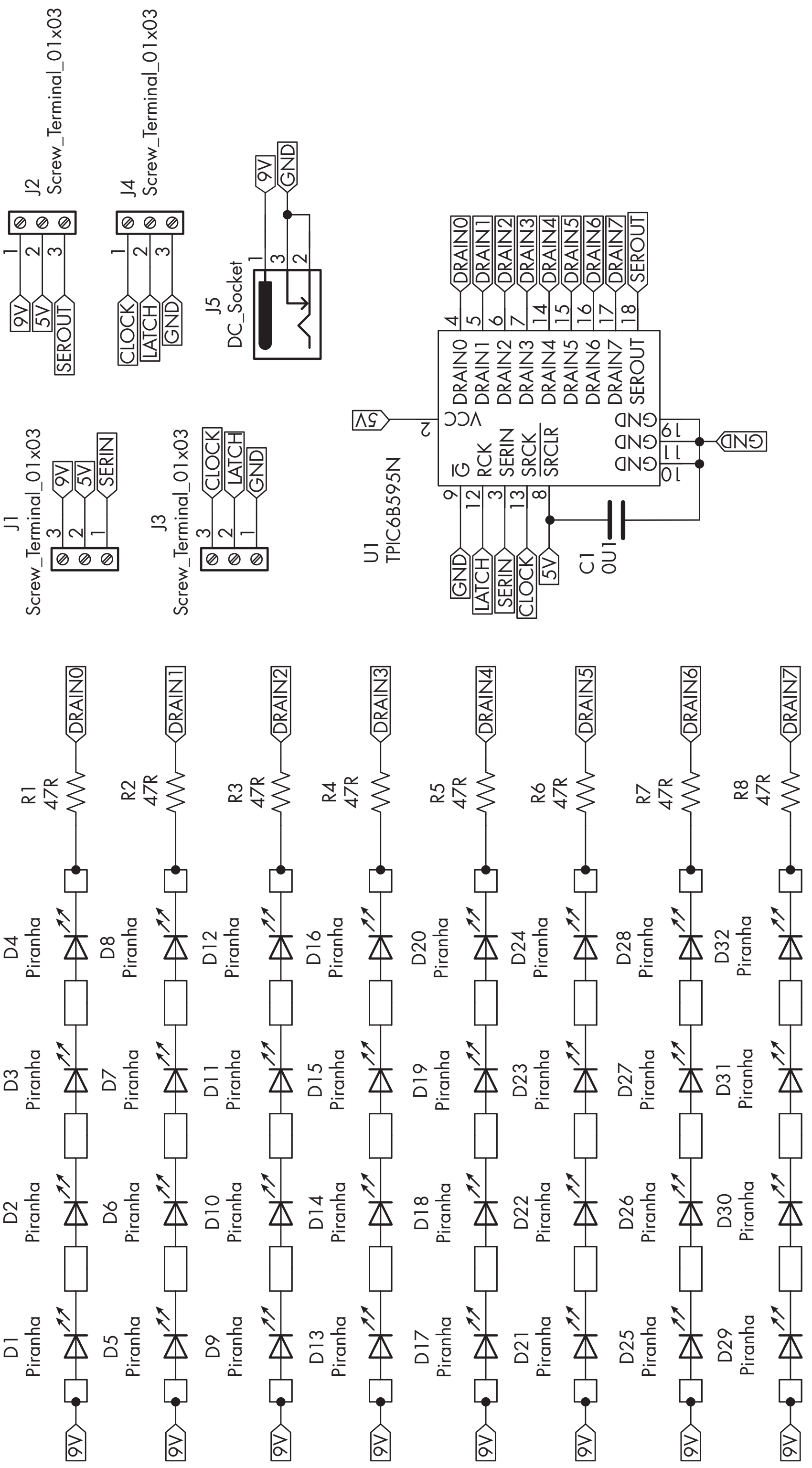


Figure 8-15: The schematic for Project #27

As you can see, the TPIC6B595 has "low-side" outputs. When an output is activated, the current starts from the power supply, goes through the items to be controlled, and then passes through the shift register DRAIN pins and out via GND.

Building a Single-Digit Display

Figure 8-16 shows the simple PCB layout.

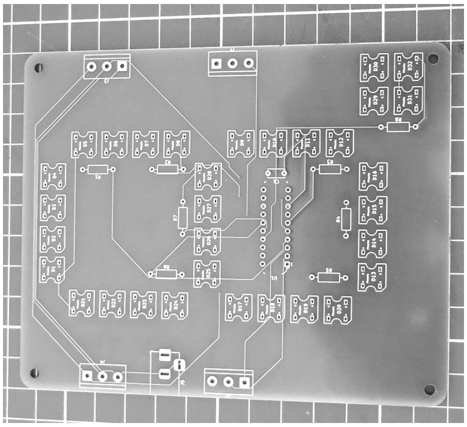


Figure 8-16: The unpopulated PCB for one digit

Always start by connecting the lowest-profile components: the resistors, the IC socket, the LEDs, and, finally, the DC socket and terminal blocks. Don't be tempted to insert all the LEDs at once and then turn the PCB over, as some of the LEDs may come loose or fall out. Instead, solder them in one at a time. Orient all the LEDs with the anodes on the left and cathodes on the right (the left side of the PCB is the one with the DC socket). Once completed, your board should resemble the example shown in Figure 8-17.

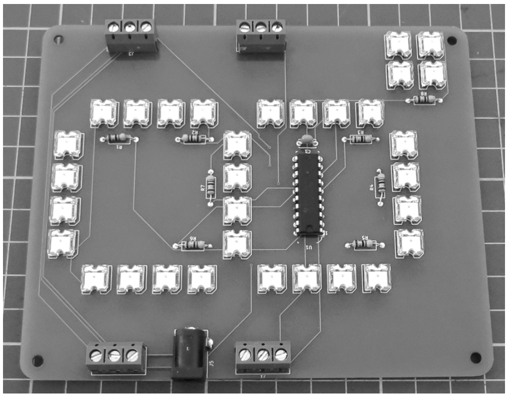


Figure 8-17: The completed PCB for one digit

Once you've set up the hardware, enter and upload the Project #27a sketch that follows, which demonstrates one digit (you can choose to build three more digits in the section "Building a Four Digit Display"). After uploading the sketch, you'll need to connect your Arduino and power supply to the PCB. With the PCB facing upward and the DC socket on your left, make the connections as described in Table 8-1.

Table 8-1 PCB to Arduino Connections

PCB left side	Arduino
Clock	D9
Latch	D8
GND	GND
9 V	Vin
5 V	5 V
Serial In	D10

Finally, connect a 9 V DC power supply to the DC socket on the PCB. The Arduino is powered by the 9 V supply via the Vin pin and feeds back 5 V to the display board to power the shift register. After a moment, the display should count from zero to nine; then again with a decimal point beside each numeral, as shown in Figure 8-18; and finally repeat.

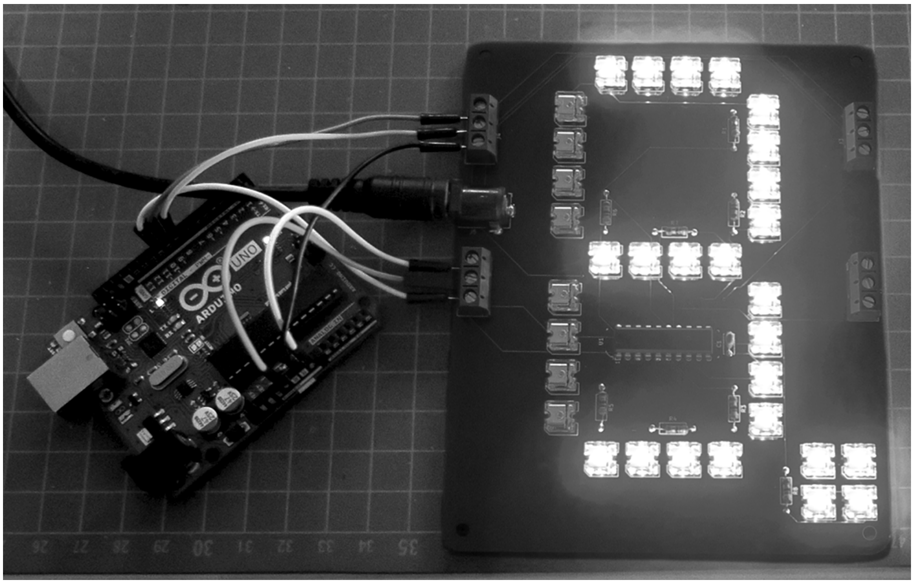


Figure 8-18: Example display board output

Because of the incredible brightness of the LEDs, I set the supply voltage to 7 V DC before taking the photo in Figure 8-18 for a clearer picture. Your display should be much brighter when operating at its designated 9 V DC.

Each segment of the digit consists of 28 Piranha LEDs, plus 4 more for the decimal point, driven by an output on the TPIC6B595. Therefore, you could consider the digit to be made up of seven LEDs, plus one more for the decimal point. These are controlled via the TPIC6B595's outputs in the same method as the relay board described in Project #26.

Let's see how this works:

```
// Project #27a - Single giant 7-segment LED displays

❶ #define latch 8 // Latch RCK pin
#define clock 9 // Clock SRCK pin
#define data 10 // Data SERIN pin

int digits[] = {B00111111,  // 0
              ❷ B00000110,  // 1
                B01011011,  // 2
                B01001111,  // 3
                B01100110,  // 4
                B01101101,  // 5
                B01111101,  // 6
                B00000111,  // 7
                B01111111,  // 8
                B01100111}; // 9

void sendNumber(int a, boolean point)
{
    if (point == false)
    {
        digitalWrite(latch, LOW);
        shiftOut(data, clock, MSBFIRST, digits[a]);
        digitalWrite(latch, HIGH);
    } else if (point==true)
    {
        digitalWrite(latch, LOW);
      ❸ shiftOut(data, clock, MSBFIRST, digits[a]|B10000000);
        digitalWrite(latch, HIGH);
    }
}

void setup()
{
    pinMode(latch, OUTPUT);
    pinMode(clock, OUTPUT);
    pinMode(data, OUTPUT);
}

void loop()
{
  ❹ for (int a = 0; a<10; a++)
    {
        sendNumber(a,false);
        delay(1000);
    }
  ❺ for (int a = 0; a<10; a++)
    {
        sendNumber(a,true);
        delay(1000);
    }
}
```

The sketch first defines the Arduino digital pins used for connecting to the shift register's latch, clock, and data pins ❶. For the array of 8 bytes at ❷, each byte represents the eight outputs that are used to control the segments on the display. For example, the number 1 is represented in binary as `B00000110`, as you want to turn on the second and third segments of the display to display 1.

The custom `sendNumber()` function sends the requisite data to display each digit to the shift register, which then sets the appropriate outputs. This function accepts the digit to display and accepts `true` or `false` as the second parameter, which is used to turn the decimal point on or off. If the decimal point is required, the sketch uses the OR function `|` ❸ (as in Project #7 in Chapter 2) to turn on the most significant bit of the shift register (bit 7), which controls DRAIN7.

The `void setup()` function sets the required digital output pins. Finally, the sketch demonstrates the display by counting from zero to nine and back again ❹ and then again with the decimal point turned on ❺.

You now have a large, impressive numerical LED display that can be seen from quite a distance. If the LEDs are too bright for your liking, you can increase the value of resistors, perhaps to 180 or 270 Ω. Be sure to use 0.25 W (1/4 W) resistor types.

Building a Four-Digit Display

You can build and use multiple display boards for larger numerical projects. In this example, you'll use four boards to display numbers of up to four digits. You might use this to display data or as a large, bright clock.

Construct a second, third, and fourth board just as you did in the previous section, with one difference: these new boards won't need the DC socket, as the first display board will act as your power source. Once the extra boards are assembled, connect them by bridging the terminal blocks on each side to each other. The matching labels on the rear of the display boards, as shown in Figure 8-19, will help you do so.

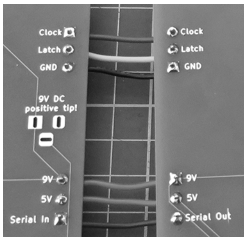


Figure 8-19: The rears of two display boards

Next, enter and upload the Project #27b demonstration sketch. Once the upload completes, connect the Arduino to the leftmost display board and connect the 9 V power supply. After a moment the display boards should show random four-digit numbers, with random placement of the decimal point, as shown in Figure 8-20.

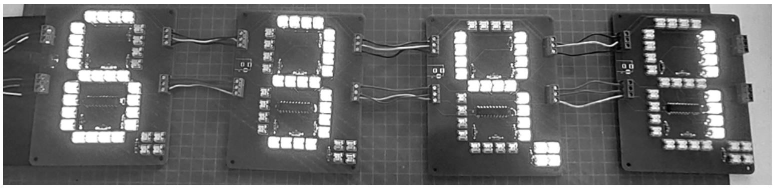


Figure 8-20: The four display boards operating at 7 V DC

Let's see how this works:

```
// Project #27b - Four giant 7-segment LED displays

#define latch 8 // Latch RCK pin ❶
#define clock 9 // Clock SRCK pin
#define data 10 // Data SERIN pin

int digits[] = { B00111111,   // 0 ❷
                 B00000110,   // 1
                 B01011011,   // 2
                 B01001111,   // 3
                 B01100110,   // 4
                 B01101101,   // 5
                 B01111101,   // 6
                 B00000111,   // 7
                 B01111111,   // 8
                 B01100111 }; // 9

void sendNumbers(int numbers[], int dp)
{
    digitalWrite(latch, LOW);
    for (int i = 0; i < 4; i++)
    {
        int dig_idx = numbers[i]; ❸
        if (dp == i) {
        // Display the digit
        shiftOut(data, clock, MSBFIRST, digits[dig_idx] | B10000000); ❹
        } else {
        shiftOut(data, clock, MSBFIRST, digits[dig_idx]);
        }
    }
    digitalWrite(latch, HIGH);
}

void setup()
{
    randomSeed(analogRead(0)); ❺
    pinMode(latch, OUTPUT);
    pinMode(clock, OUTPUT);
    pinMode(data, OUTPUT);
}
```

```
void loop()
{
    int numbers[4] = { random(0, 10), random(0, 10), random(0, 10), random(0, 10) }; ❻
    sendNumbers(numbers, random(0, 3)); ❼
    delay(1000);
}
```

The sketch for four boards has a few differences from that of Project #27a to make using multiple displays easier. Once again, it defines the Arduino digital pins used for connecting to the shift register's latch, clock, and data pins ❶ and defines the array of 8 bytes ❷ that represents the eight outputs used to control the segments on the display.

The `void sendNumbers()` function accepts two parameters: an array of four numbers (one for each display board), and an integer that represents on which boards to display the decimal point. The sketch checks for this at ❸. If the parameter is 1, the required bit for the decimal point display is included in the byte sent to the shift register ❹.

As you're displaying random numbers, the generator is seeded with the analog input data ❺, and then the array for numbers to be displayed is also filled with random numbers ❻ and sent to the display boards ❼. Finally, the sketch includes a delay before displaying more random numbers.

You can display all sorts of numerical data with these boards. You might also add your own characters by adding more elements to the digit arrays. For example, a degree symbol for temperature would be as follows:

```
B01100011 // Binary representation of °
```

For a challenge, pair up four display boards with an Arduino and a real-time clock module to make a large, bright clock. You might add a thermometer as well. I hope you enjoy using these display boards as much as I enjoyed designing them.

Moving On

In this chapter, you learned how to control currents and voltages using the TPIC6B595 shift register, a more capable alternative to the popular 74HC595. You can now control items that draw more current than an Arduino's digital I/O pin can safely handle. You also learned how to use bright Piranha-style LEDs for excellent indicators.

In the next chapter, you'll use MP3 player modules to create digital music players and sound boards for various purposes.