# countdown timer with a 7-segment display

7-segment countdown   
By Allan Schwartz, [allans@codevalue.net](mailto:allans@codevalue.net)  
Dec 1, 2016

## Summary

The purpose of this project is to build a countdown timer, comprised of 2 7-segment displays. In doing so, several interesting problems are solved:

* How to interface a 7-segment display display to the Arduino – without using a MAX7219 chip.
* How to write very concise and efficient Arduino C code to drive the 7-segment display, and represent the digit displays in a data structure. This is achieved in just three lines of code!
* Understanding the concept of an Arduino ***setup***() and ***loop***() functions, and the use of time (milliseconds and seconds) in a program.
* In part 2 of this post, I’ll illustrate how to implement this using a higher level of integration.

## Why?

I used to be a manager for an embedded computing development group and this was an example exercise for creating a low-level interface. Usually 7 segment LEDs are hooked up with a common cathode, but this one uses a common anode and the segments are lit when the pin is low, and not lit when the pins are high. It was an exercise for writing code from scratch without using an existing library.

High Level vs Low Level

## Parts

I used an Arduino Uno as the most simple, well-documented microcontroller available.

I know that there have been several recent blog posts using Raspberry Pi’s or the ESP8266 microcontroller, but I wanted to go back to basics.

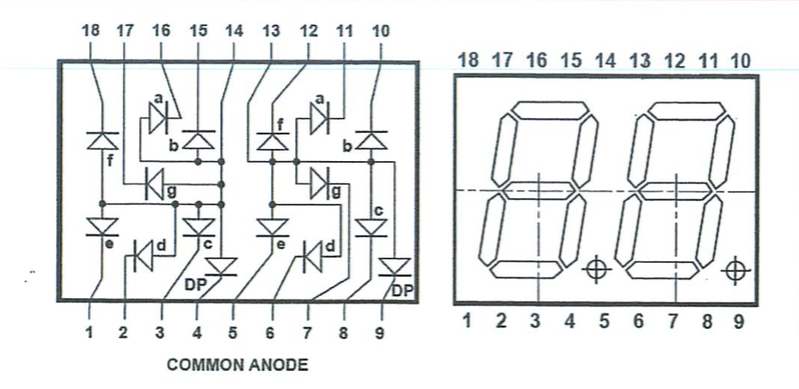
You’ll Need:

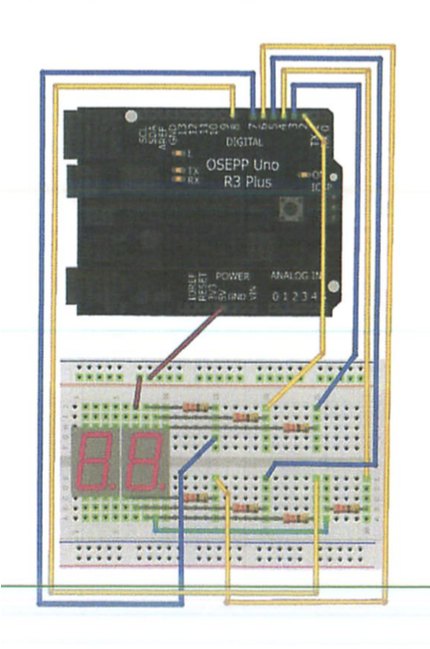
* An Arduino Uno and USB cable
* a Breadboard plus jumper cables
* a 2-digit 7-segment display unit
* plenty of 330 ohm resisters

## Let’s Start

We will be using a 2-digit 7-segment LED for this exercise. Refer to the diagram below for pin numbers.

When 5v is applied to the common anode (pins 13 and 14), and when the cathode or negative side of the LED segment is pulled to ground (through a 330 ohm resistor), the segment is lit. If the cathode is held to 5v (through a 330 ohm resistor) the segment is unlit.





## Programming

Using the above circuit, we display digits 0 through 9 on both the left and right digits.

In the exercise we will be writing an embedded program (Sketch) that displays a countdown timer of the digits counting down for 1 minute, from 60 down to 0. At zero, the display should flash to indicate that the countdown is complete.

***Subproblem one***: how to interface the digital pins. The segment names ‘a, b, c…” is completely standardized on every 7-segment display.



Each of the seven segments can be driven by an Arduino digital pin, using a resistor to limit current flow. Using the following schematic:

<schematic goes here>

remember, that using this wiring, 1 is unlit and 0 is lit.

## Creating digits in just three lines of code

### Most of the sketches available to run a 7-segment display have code for each digit. So, if you need the number “6” the code will light each segment making up the number. This requires a separate function for each digit – which uses up precious Arduino code space.

### // sample code to draw a 6 on the 7-segment display

### void six()

### {

### digitalWrite(a, HIGH);

### digitalWrite(b, LOW);

### digitalWrite(c, HIGH);

### digitalWrite(d, HIGH);

### digitalWrite(e, HIGH);

### digitalWrite(f, HIGH);

### digitalWrite(g, HIGH);

### }

As you can see, creating a function for each digit is a tedious process.

### We can improve on this. Here is how to create, in three lines of code, all 10 digits. Moreover, this can easily be adapted to extend to a full character set.

First,we made a table of ten 7-segment displays glyphs we need to create, and the state of the signals a, b, c, d, e, f and g

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F | G | dp |
|  | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 | off  1 |  |
|  | off  1 | Lit  0 | Lit  0 | off  1 | off  1 | off  1 | off  1 |  |
|  | Lit  0 | Lit  0 | off  1 | Lit  0 | Lit  0 | off  1 | Lit  0 |  |
|  | Lit  0 | Lit  0 | Lit  0 | Lit  0 | off  1 | off  1 | Lit  0 |  |
|  | off  1 | Lit  0 | Lit  0 | off  1 | off  1 | Lit  0 | Lit  0 |  |
|  | Lit  0 | off  1 | Lit  0 | Lit  0 | off  1 | Lit  0 | Lit  0 |  |
|  | Lit  0 | off  1 | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 |  |
|  | Lit  0 | Lit  0 | Lit  0 | off  1 | off  1 | off  1 | off  1 |  |
|  | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 | Lit  0 |  |
|  | Lit  0 | Lit  0 | Lit  0 | off  1 | off  1 | Lit  0 | Lit  0 |  |

Now that we can see this table, it becomes clear how to solve the problem with a simple datastructure, rather than a lot of code. Translating the table above into C code is very straight forward:

const byte numbers[10][7] = { // 0: segment lit, 1: segment unlit

// a b c d e f g

{0, 0, 0, 0, 0, 0, 1}, // 0

{1, 0, 0, 1, 1, 1, 1}, // 1

{0, 0, 1, 0, 0, 1, 0}, // 2

{0, 0, 0, 0, 1, 1, 0}, // 3

{1, 0, 0, 1, 1, 0, 0}, // 4

{0, 1, 0, 0, 1, 0, 0}, // 5

{0, 1, 0, 0, 0, 0, 0}, // 6

{0, 0, 0, 1, 1, 1, 1}, // 7

{0, 0, 0, 0, 0, 0, 0}, // 8

{0, 0, 0, 1, 1, 0, 0}, // 9

};

now, using that data structure ‘numbers’, we write a function which can display any digit (or glyph) we desire. Followed by a function which displays any two-digit number from 0 to 99.

// display the digit {0..9} in either the ones DIGIT or the tens DIGIT

void sevenSegmentDigit(byte digit, byte pin) {

for ( byte i = 0; i < 7; i++, pin++ )

digitalWrite(pin, numbers[digit][i]);

}

const byte DIGIT1\_PIN\_A = 2; // the 1's digit pin a-g are digital pins 2-8

const byte DIGIT10\_PIN\_A = 10; // the 10's digit pin a-g are digital pins 10-17

// display the two-digit number {00..99} in the two 7-segment displays

void sevenSegment\_nn(int nn) {

sevenSegmentDigit((nn / 10) % 10, DIGIT10\_PIN\_A);

sevenSegmentDigit(nn % 10, DIGIT1\_PIN\_A);

}

## Finished Device

