Microcontroller Systems and Interfacing

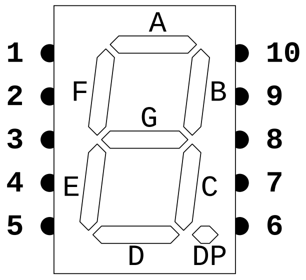
week 10 experimental lab

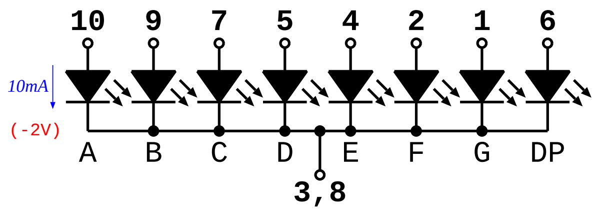
1 LED displays

Seven-segment displays contain seven (or eight, including the decimal point) individual LEDs arranged in a pattern that is convenient for displaying digits (and maybe some letters too).

Unlike bar graph arrays, but similarly to RGB LED arrays, each LED in a seven-segment display shares one of its terminals with all the other LEDs in the package. The shared terminal might be the anode or the cathode, depending on the particular device. (Check the data sheet for your device to find out which.) If all the anodes are connected together the arrangement is called *common anode*; if all the cathodes are connected together, the arrangement is called *common cathode*.

A common cathode seven-segment display therefore requires at least 8 pins (seven for the anodes, and a single pin for the shared cathode connection). Most seven-segment displays also provide a decimal point (the eighth LED), and an additional common connection, making a total of 10 pins.



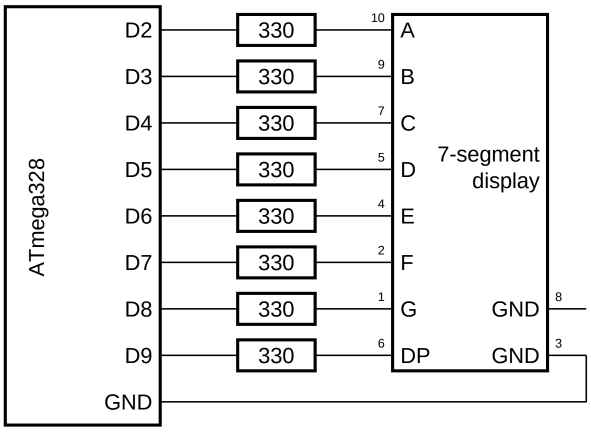


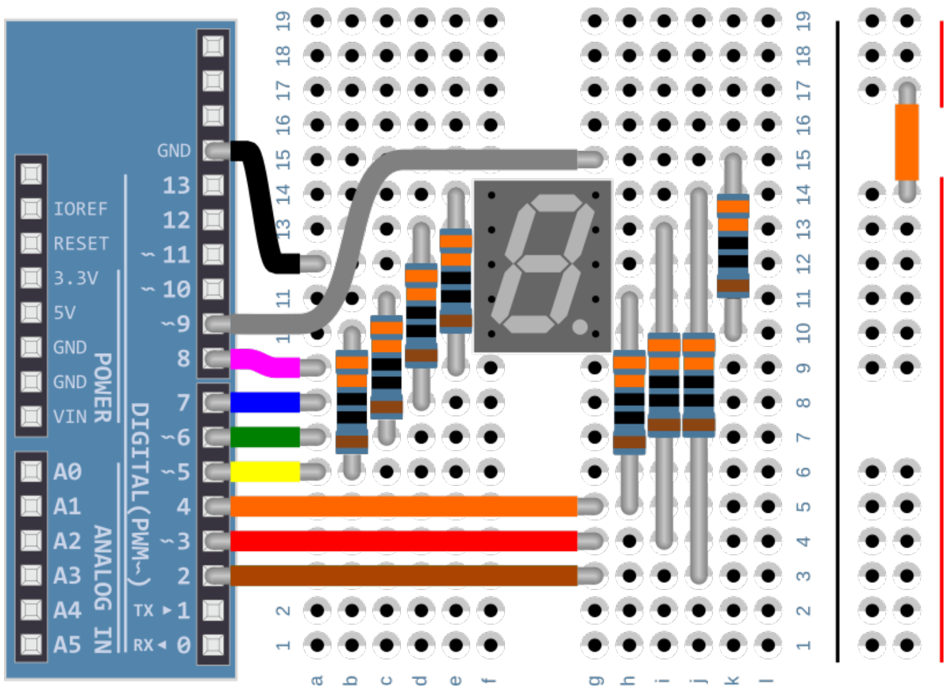
1.1 Seven-segment display circuit

▶ Set up a single seven-segment display on your breadboard. Connect the seven segments as follows:

digital pin 2 → `A` segment
3 → `B`
…
8 → `G`
pin 9 → `DP` segment

Since each segment is a light-emitting diode (LED) you must use the usual 330 Ω current-limiting series resistors.





1.2 Seven-segment display software

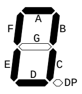
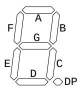
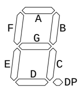
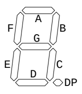
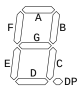
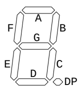
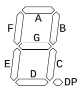
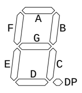
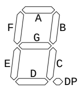
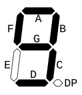
► Create a program to drive the seven-segment display.

If you do not want to create one from nothing, make a copy of an earlier program for an LED array to use as a starting point. Controlling a 7-segment display is not too different.

Test your seven-segment display in some way. (For example, make your `loop()` count in binary on the display's segments, or make it turns on each segment one at a time. Anything with a predictable result, that you can verify visually, is OK.)

2 Using a seven-segment display to show digits

To display a digit we have to turn on the segments in an appropriate pattern. For example, the digit `0` will be displayed if segments `A`, `B`, `C`, `D`, `E`, and `F` are turned on, corresponding to digital pins 2, 3, 4, 5, 6 and 7 being `HIGH`, and pins 8 and 9 being `LOW`. We can set these pins to their required states in parallel using '`setLEDs(0b00111111)`'. (Your `setLEDs()` function from recent labs will work for this with little (or no) modification.)

	segment				*segments*				`setLEDs()`
digit	*pattern*	G	F	E	D	C	B	A	*argument*
0		L	H	H	H	H	H	H	`0b0111111 = 63`
1									
2									
3									
4									
5									
6									
7									
8									
9		H	H	L	H	H	H	H	`0b1101111 = 111`

2.1 Designing the digits

► Complete the table on the right to plan the pattern of segments that need to be lit for each of the ten decimal digits, and derive the corresponding argument to `setLEDs()`:

1. Design a pattern of segments corresponding to the digits `1` to `8`. Using a pencil, fill in the empty segments in the example displays to visualise your requirements.
2. Convert your patterns to the corresponding `H` (high) and `L` (low) values for each segment.
3. Finally, convert the `H` and `L` values for each segment into a series of `1` and `0` bits. The equivalent integer value is the numeric argument for `setLEDs()` that will display the digit.

2.2 Displaying the digits

Displaying a digit requires a call to `setLEDs()` with an argument corresponding to the pattern of segments that should be on to form the digit we want to display.

► Write a function `displayDigit(digit)` that calls `setLEDs()` with the correct parameter for the given `digit`.

A series of `if()` statements can test the value of `digit` and call `setLEDs()` with the corresponding parameter.

```
void displayDigit(int digit) {
  if (0 == digit) setLEDs(0b0111111);
  ...
  if (9 == digit) setLEDs(0b1101111);
}
```

► Test your function with the `loop()` shown on the right.

```
int counter = 0;

void loop() {
  displayDigit2(counter++);
  delay(200);
}
```

2.3 Show useful information on your seven-segment display.

Choose and complete at least one of these three examples of showing useful information on your 7-segment display.

- Use `analogRead()` and `map()` to convert a potentiometer position into a numeric display between `0` and `9`.
- Display the ambient light level as number between `0` and `9`. (Don't forget to make it self-calibrating.)
- Make a light-triggered counter.

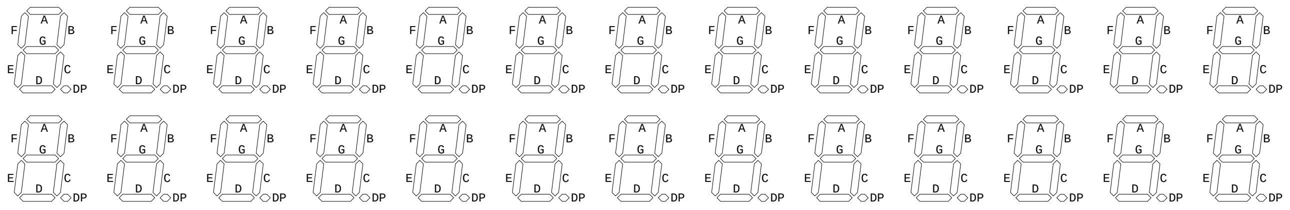
  Hint: for this to work well you will need to use both self-calibration and *hysteresis*. Consider an object absent if the light level is above (e.g.) 60%. Consider an object present if the light level is below (e.g.) 40%. Make your `loop()` code detect a transition from 'absent' to 'present' and at that time increase the value shown on the display by 1.

Important note

Leave you seven-segment display circuit on the breadboard. You will need it to complete the lab experiments next week.

3 Mini project

Design some seven-segment patterns for letters. Supporting the entire alphabet is difficult with only seven segments, but if you design patterns for the letters `A` to `F` then you can display 4-bit numbers as a digit `0`, ..., `9`, `A`, ..., `F`. Use the templates below to design as many additional characters as you want, then add them to your `displayDigit()` function.



4 Challenges

- Improve your `displayDigit()` function.

  There are several ways to write this function. The suggestion above, using `if` statements, is not the most efficient. It would be better to store the `setLEDs()` parameters in an array of `int`egers. Index the array using the `digit` to obtain the parameter to use with `setLEDs()`.

```
int ledValues[10] = {
  0b0111111, // 0
  ...
  0b1101111  // 9
};

void displayDigit(int digit) {
  setLEDs(ledValues[digit % 10]);
}
```

- Modify your `setLEDs()` function to use the `PORTD` and `PORTB` registers directly to set the output pins to `HIGH` or `LOW`.
- Add a second seven-segment display to your circuit.

  Use it to make a counter from `00` to `99`. Use it to display a percentage light level or potentiometer value. Use it to display the temperature in degrees Celsius.
- If you did not already do so, figure out how to add the second seven-segment display to your circuit using only 10 digital outputs instead of the 16 that the obvious approach requires.
- Eight series resistors are required so that eight segments can be lit at the same time without them 'stealing' current from each other. How could you modify the overall system so that only one series resistor is shared between all the segments?