Input/output pins on the Arduino

ENGR 40M
Chuan-Zheng Lee

April 28, 2017

An *input/output pin*, or *I/O pin*, is the interface between a microcontroller and another circuit. It can be configured in the microcontroller's software to be either an input or an output. On the Arduino, this configuration is accomplished using the `pinMode()` function.

This handout concerns three functions: `pinMode()`, `digitalWrite()` and `digitalRead()`. The details of all of these can be found in the Arduino reference at `https://www.arduino.cc/en/Reference/HomePage`. (If you're reading a printed copy of this document, the links all go there.)

1 Output pins

When configured as an *output pin*, a pin provides V_{DD} or 0 V to whatever is connected to it (if anything). When the pin provides V_{DD}, we say that it is in the `HIGH` state. When the pin provides 0 V, we say that it is in the `LOW` state. We set the state of an output pin using the `digitalWrite()` function.

An output pin achieves this by making a connection to V_{DD} or a connection to ground internally, inside the microcontroller. This, in turn, is accomplished with transistors whose drains are connected to the output pin, as shown in Figure 1.

When the output is set to `HIGH`, the PMOS transistor turns on and provides a connection to V_{DD}. When the output is set to `LOW`, the NMOS transistor turns on and provides a connection to ground.

1.1 Output resistance

If these transistors were ideal, then they would provide a direct connection to V_{DD} or ground, depending on which output state (`HIGH` or `LOW`) was set by software. Of course, these transistors aren't ideal; as we understood from our discussion of transistors, they have some (hopefully small) *on resistance*. Furthermore, the outputs of microcontrollers are typically designed to power low loads, so this resistance isn't negligible.

The resistance thus "seen" by a device connected to a pin is called the *output resistance* of the pin. Of course, there is not just one output resistance—it depends on the state of the pin. When the output is `HIGH`, it is the resistance to V_{DD} that is relevant ($R_{\texttt{HIGH}}$ in Figure 2); in our model, this is the on resistance of the PMOS transistor. When the output is `LOW`, it is the resistance to ground; in our model, the on resistance of the NMOS transistor.

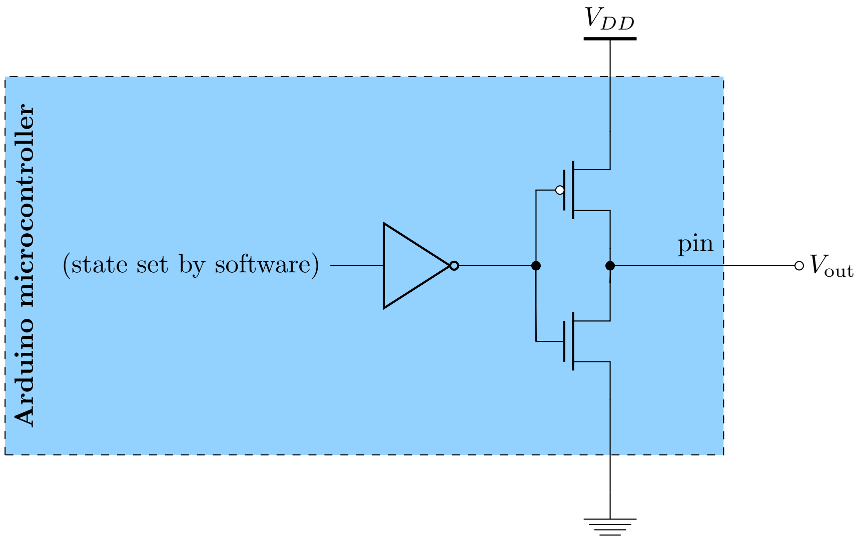

Figure 1: Schematic of an I/O pin when configured as an output

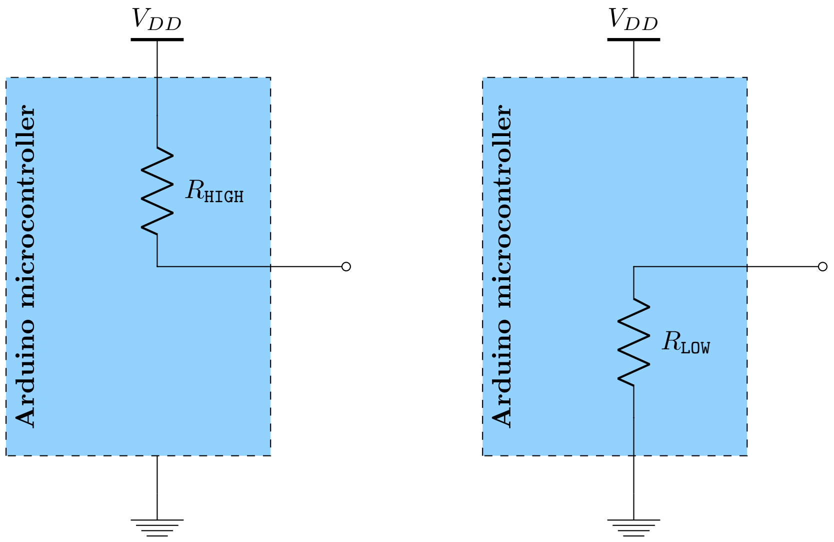

Figure 2: Equivalent circuits showing the output resistance when the output is high (left) and low (right)

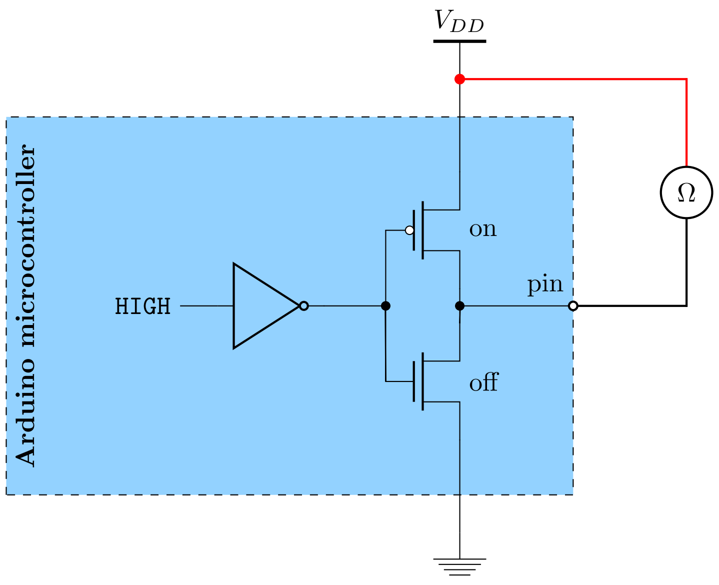

Figure 3: Measuring the output resistance of a pin when it is set to `HIGH`

1.2 Measuring the output resistance

Say we wished to measure the resistance of an output pin when it is set to `HIGH`. First, we would need to set the output pin to high, using the `pinMode()` and `digitalWrite` functions. Then, we can measure the resistance with an ohmmeter, as shown in Figure 3.

Recall, however, that what we're measuring isn't a resistor, but a transistor that is on. Therefore, it is polarized, and unlike with a resistor, the polarity of the leads matters. The ohmmeter will try to push a test current from the red lead to the black lead, so we should place our leads so that this test current goes in the expected direction. When measuring the resistance of a PMOS transistor, this means putting the red lead at the source (VDD), and the black lead at the drain (the pin). Similarly, when measuring the resistance of the NMOS transistor, the red lead should be at the drain (pin), and the black lead at the source (ground).

For devices like LEDs, this output resistance isn't too much of an issue: we don't wish to draw that much current from the pin anyway. For motors, however, this output resistance is prohibitively high, and we need to place a driver circuit in between the output pin and the motor, so that the motor can draw enough current. Such a driver circuit often uses *power transistors*, transistors that are designed to provide higher current (lower drain-source resistance) than those in the microcontroller.

2 Input pins

An *input pin* reads the voltage on the pin as if it were a voltmeter, and returns either `HIGH` (`1`) in software if the voltage is close to V_{DD}, or `LOW` (`0`) if it is close to 0 V. An input pin can be read using the `digitalRead()` function.

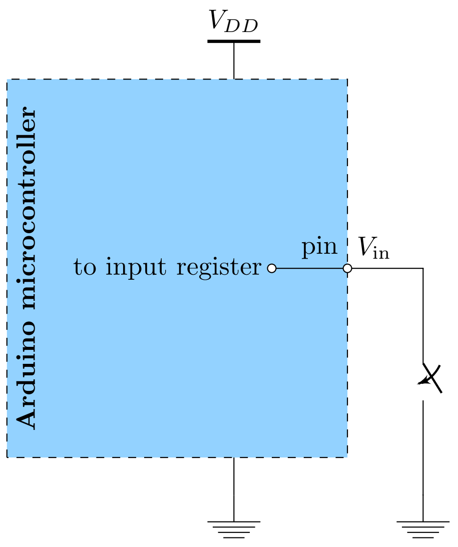

Figure 4: A switch circuit that does not work

Note that the value returned by `digitalRead()` is only well-defined when the input pin voltage is close to V_{DD} or $0\,\text{V}$. The precise meaning of "close" varies between microcontrollers, but for the Adafruit Metro Mini[1] as it's used in our circuit, the input pin voltage needs to be at least $0.6V_{DD}$ to qualify as `HIGH`, and at most $0.3V_{DD}$ to qualify as `LOW`. In the middle (say, at $0.45V_{DD}$), the behavior of the pin is undefined.

An (ideal) input pin takes (approximately) no current, like the gate of a transistor or a voltmeter. In the projects we do in this class, modeling the input pin as ideal is a good approximation.[2]

2.1 Connecting a switch to an input pin (pull-up resistors)

How do we connect a switch to an Arduino? This is not as obvious as it sounds: remember that switches govern connections, not voltages. So simply connecting the switch between the pin and some other point in the circuit, say, ground, would *not* work (Figure 4). When the switch is closed, the input pin would be shorted to ground, and V_{in} would be $0\,\text{V}$, which is fine. However, when the switch is open, the input pin is floating, and we have no idea what its voltage is. (In practice, it will "remember" the last voltage it was driven to, which in this case is $0\,\text{V}$.)

Consider, instead, the circuit shown in Figure 5. When the switch is closed, the pin is still shorted to ground, so $V_{\text{in}} = 0\,\text{V}$. However, this time, when the switch is open, no current flows through the resistor R_{pu} (there's nowhere for it to go, since the input pin takes no current), so the voltage drop across the resistor is zero, so $V_{\text{in}} = V_{DD}$. Thus, we can reliably distinguish between the closed and open states of the switch: when the switch is closed, the input pin will read `LOW`,

[1] More precisely, it's the ATmega328, which is the microcontroller used in the Adafruit Metro Mini. The full datasheet can be found at `http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf`; this information is on page 313.

[2] The input leakage current is specified in the datasheet to be at most $1\,\mu\text{A}$.

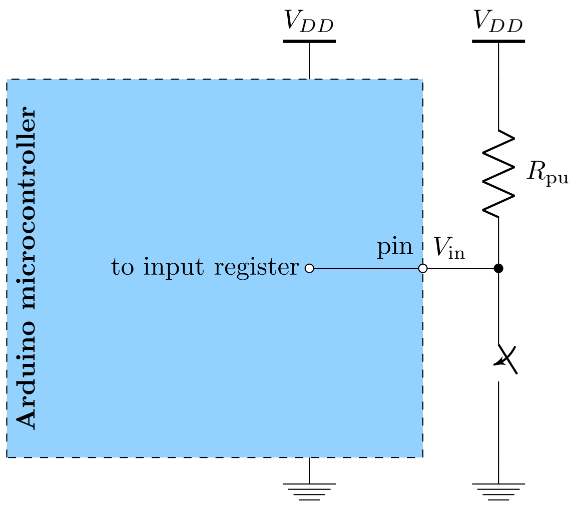

Figure 5: Using an external pull-up resistor

and when the switch is open, it will read `HIGH`.

You can think of the role of the resistor as being to enforce a “default” state on the pin. When we directly connect the pin to ground, V_{in} is set to 0 V by that connection—the resistor won’t get in the way (current will flow through it, but that doesn’t affect the input pin voltage). In the *absence* of such a connection, though, the resistor comes into play, *pulling up* the voltage V_{in} to V_{DD}. For this reason, the resistor R_{pu} is often referred to as a **pull-up resistor**.

We could also do this the other way round, connecting the switch between V_{DD} and the pin, and the resistor between the pin and ground, so that would be a “pull-down resistor”. However, connecting the switch to ground and the resistor to V_{DD} tends to be the more common practice. It’s so common, in fact, that many microcontrollers (including the Adafruit Metro Mini) implement a pull-up resistor internally, inside the microcontroller, as shown in Figure 6. The pull-up resistor can be enabled or disabled in software. To enable it, we pass the `INPUT_PULLUP` constant as the second argument to `pinMode()`: `pinMode(pin, INPUT_PULLUP)`. To disable it, we pass the constant `INPUT` instead: `pinMode(pin, INPUT)`.

The enabling/disabling of the internal pull-up resistor is implemented using—you guessed it—a PMOS transistor, connected to VDD. When the transistor is off, it looks like an open circuit, effectively removing the pull-up resistor from the circuit.

If you use the internal pull-up resistor, you naturally don’t need the external one. This makes it acceptable to connect your switch as in Figure 4, so long as you enable the pull-up resistor, making the equivalent circuit in Figure 7.

Finally, a couple of cautionary notes. First, we haven’t yet talked about the value of the resistance R_{pu}. This is partly because it doesn’t really matter—for any reasonable resistor value, this circuit will function as we described; the precise resistance only affects how much current flows when the switch is closed. But there is another aspect to this: the internal pull-up resistor isn’t specified to have a precise value. Typically, it’s on the order of tens of kiloohms; for the Adafruit Metro Mini, it’s specified to be between 20 kΩ and 50 kΩ. So you shouldn’t use the internal pull-up resistor if you need a precisely-known resistance.

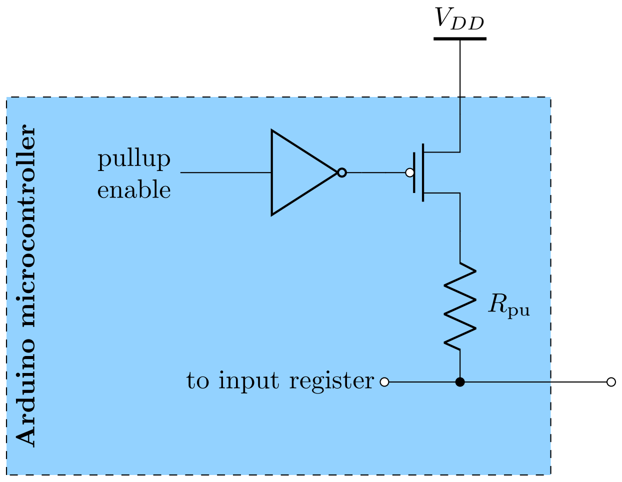

Figure 6: Internal pull-up resistor schematic

Secondly, the pull-up resistor obviously creates a path to V_{DD}, which can sometimes make a pin you intended to be an output pin seem like it's working even when you forgot to include `pinMode(pin, OUTPUT)` in your `setup()` function. This is because, in a slight quirk of the Arduino, when a pin is configured as an *input* pin, the `digitalWrite()` function actually enables (`HIGH`) or disables (`LOW`) the pull-up resistor. (That is, `pinMode(pin, INPUT_PULLUP)` is actually just shorthand for `pinMode(pin, INPUT); digitalWrite(pin, HIGH)`.)

So if you forget to configure a pin to output, you're actually just enabling and disabling the pull-up resistor. For a multitude of reasons this is a *bad idea*: First, as discussed above, the pull-up resistor does not have a precisely-defined resistance. Also, when the pull-up resistor is disabled, the pin is floating—which is acceptable for an LED, but if you're using this pin to drive the gate of a power transistor, it will mean you have a floating gate. For this reason, it's important always to remember to configure the `pinMode()` of all pins that you're using in your code.[3]

[3]If you don't configure a pin, it defaults to being an input. However, to make your code easier for others to read, you should explicitly configure your input pins, too.

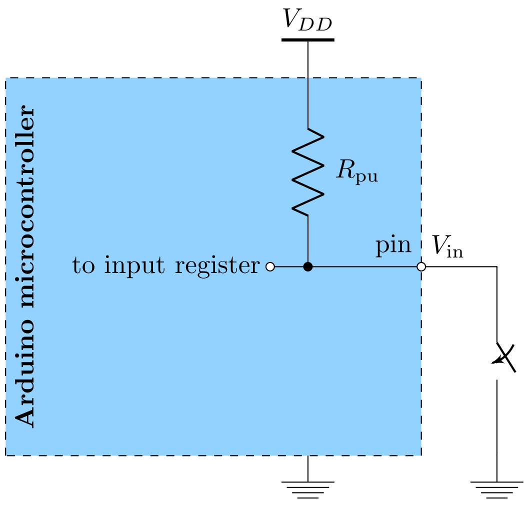

Figure 7: Using an internal pull-up resistor

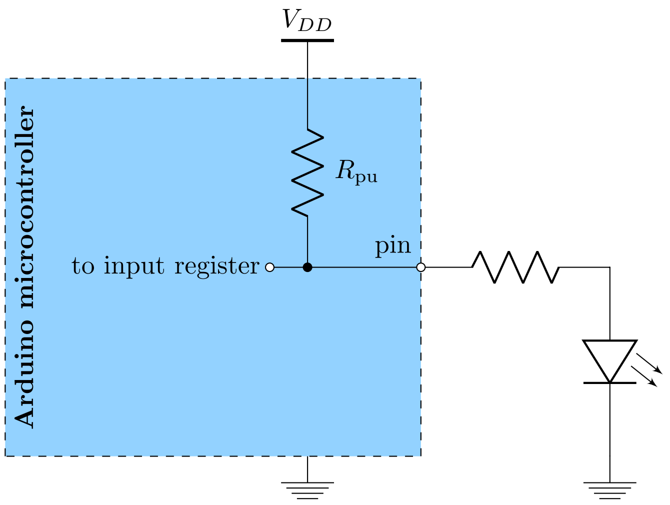

Figure 8: When you forget to `pinMode(pin, OUTPUT)` (bad, don't do this)