

How to develop your own custom Arduino

Discover how to turn your Arduino project into a custom PCB



John Teel

@JohnTeelEE

John Teel is president of Predictable Designs, which specialises in helping entrepreneurs, startups, makers, inventors, and small companies develop and launch new electronic products. John was previously a microchip design engineer for Texas Instruments.

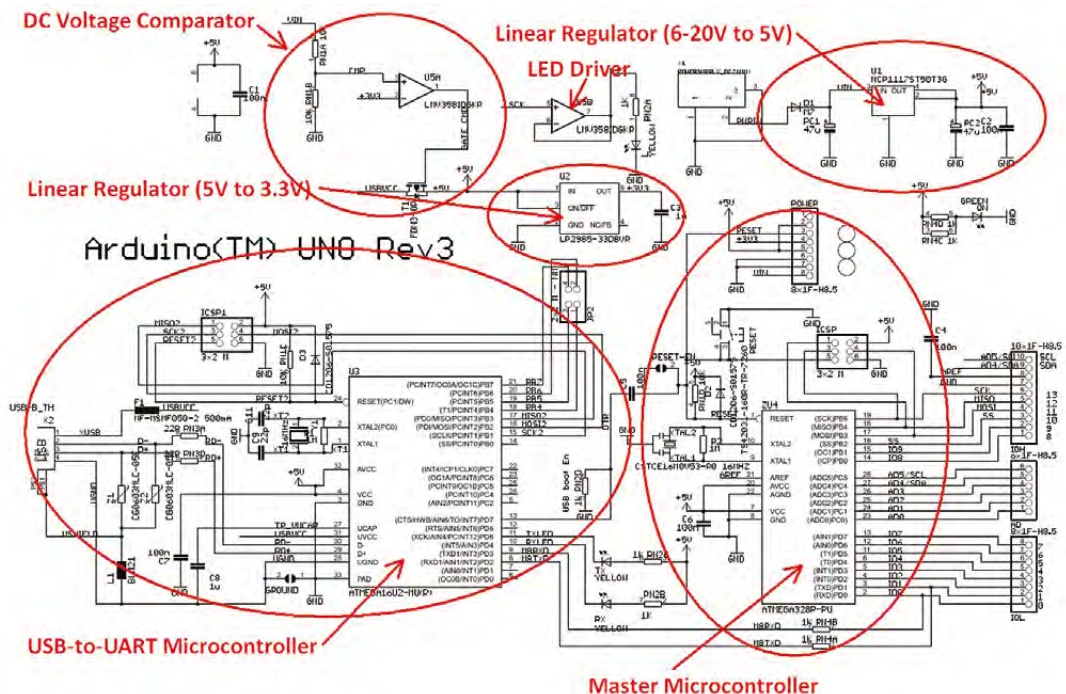


Figure 1 Open-source schematic circuit diagram for the Arduino Uno

In this article you'll discover, step-by-step, how to transition from an Arduino-based project to a custom PCB design.

The Arduino microcontroller development platform is an easy way to create your own electronic DIY project. But, if your ultimate goal is to bring your product idea to market, then you're going to eventually need a custom PCB design.

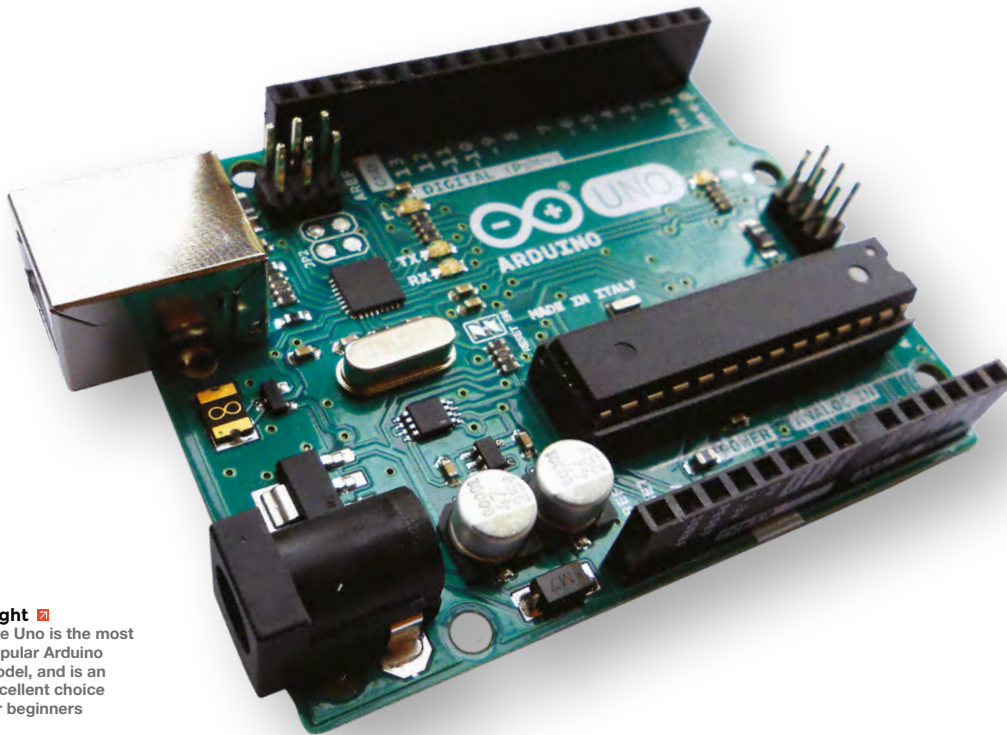
In this article you'll discover exactly how you can develop your very own custom microcontroller PCB. The considerable time and cost required to develop a custom PCB only makes sense if you hope to eventually sell your product. It can also be a fun learning experience to develop a custom PCB for your personal projects.

CHOOSING THE MICROCONTROLLER

Most models of the Arduino use an 8-bit Atmel AVR microcontroller. There are a few that are based on a more powerful 32-bit microcontroller, including the Arduino Zero, Due, MKRZero, and MKR1000. We'll be primarily focusing on the Arduino Uno in this article, since it is the most popular model available.

Note that the Uno microcontroller is packaged in a Dual In-line Package (DIP), which is then mounted in a through-hole socket. A socket facilitates the easy replacement of the microcontroller if it were to become damaged. Use of a socket may be handy in a dev kit, but a socket is almost never a good idea for a custom design you plan to sell.

Arduinos are open-source, which of course means that you can simply download the schematic circuit



Right The Uno is the most popular Arduino model, and is an excellent choice for beginners

YOU'LL NEED

- ◆ Arduino Uno
- ◆ Atmel ATmega328P microcontroller
- ◆ Nokia 5110 LCD module
- ◆ Texas Instruments TLV70233 linear regulator
- ◆ ST Microelectronics STBC08PMR USB battery charger
- ◆ ST Microelectronics STC3100IQT battery level monitor
- ◆ AVR programmer

diagram and the PCB layout design files for reference, or to use as a really handy starting point for your own custom board.

In the Arduino Uno schematic diagram shown in **Figure 1**, you may notice there are actually two separate microcontrollers (labelled U3 and U4).

U4 is an Atmel ATmega328P microcontroller, whereas U3 is an Atmel ATmega16U2 microcontroller. Both of these microcontrollers are part of Atmel's 8-bit AVR line.

The ATmega328P is the core microcontroller that is actually running your sketch code. The ATmega16U2 microcontroller is programmed to only act as a USB to UART converter. This is necessary because the ATmega328P does not provide any embedded USB functionality.

As you may already be aware, the main purpose of USB on an Arduino is for programming purposes. It's the simplest way to connect your Arduino to a computer without the need to lug out any extra fiddly hardware.

As this article will discuss in further detail, this is not the case when designing your own custom board. Instead, you will need a special piece of external hardware to handle this USB-to-UART translation. Unless you require USB for some other reason, the ATmega16U2 portion of the circuit can be removed.

Shown in **Figure 2** is the schematic diagram for the custom version of the Arduino Uno discussed in this article. The main differences are that the

// The considerable time and cost required to develop a custom PCB only makes sense if you hope to eventually sell your product **//**

QUICK TIP

You will minimise the cost to assemble your PCB if you strictly use surface-mount technology (SMT) components, and no through-hole packages.

ATmega16U2 has been removed, the power circuitry has been redesigned to use a battery, and an LCD display has been added.

DISPLAY

The Nokia 5110 LCD display can interface to any microcontroller using only three to five digital output pins. The display is powered by 3.3V, so you'll either need to run it at 3.3V, or use level shifters between the display and the microcontroller. →

NEED SOMETHING MORE POWERFUL? UPGRADE TO 32 BITS

There are countless microcontrollers available with much higher performance than the relatively simple 8-bit Atmel microcontrollers used in most Arduinos.

For example, much more powerful ARM Cortex-Mx microcontrollers are available. Cortex-Mx is a very popular 32-bit processor architecture implemented by many microcontroller manufacturers. Microcontrollers are available that are not only much faster than the ATmega328, but which also include significantly more memory and peripherals, to boot!

TUTORIAL

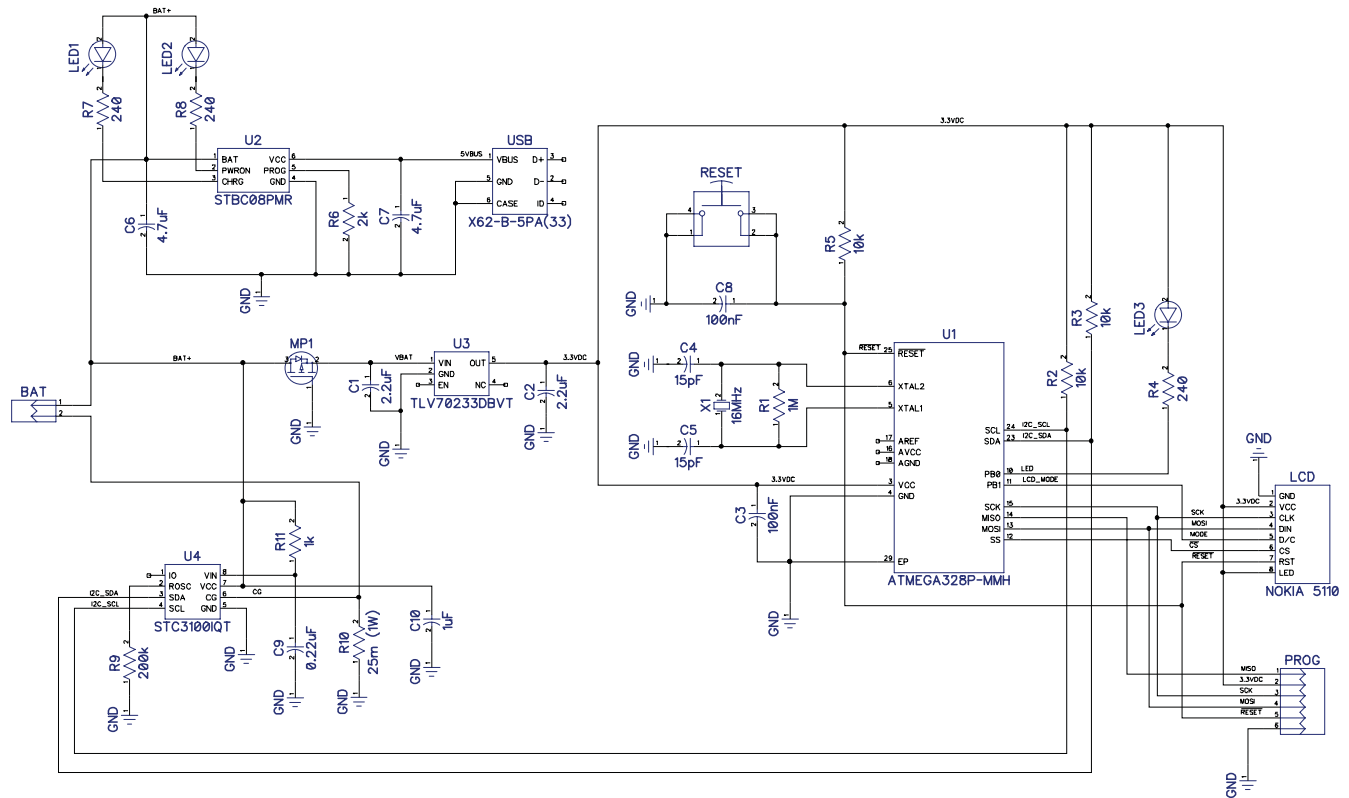


Figure 2 The schematic diagram for the custom battery-powered Arduino design discussed in this article

The Arduino Uno runs the microcontroller at 5V, so level shifters would be required. However, we've chosen to power the microcontroller from 3.3V, which eliminates the need for any level shifters.

Referring to **Figure 2**, the display should be connected as follows:

First, connect the VCC and LED pins to a 3.3V supply, and the GND pin to ground. Now, verify that the backlight LEDs illuminate.

Next, you need to connect the digital data lines. Connect the RST (reset), CS (chip select), D/C (data/command), DIN (data in), and CLK (clock) lines to GPIO pins on the microcontroller.

The RST pin could also be connected to the reset signal, so the display resets automatically any time the microcontroller is reset. The chip-select line can also simply be tied to ground so the display is always selected. Wiring the reset and chip-select pins in this fashion reduces the number of GPIO pins required from five to only three.

POWER CIRCUIT

Arduinos are designed to be powered by an external DC power supply. A 6-20V supply can be connected to the power adapter plug and from that a 5V supply is generated (U1 in **Figure 1**). Alternatively, 5V can be supplied directly via the USB connector.

For the custom Arduino however, we're going to make it function with a lithium-ion battery which can

be recharged via the USB port. A lithium-ion battery provides a voltage supply of about 3.7V. In our design, we'll use a linear regulator to step down this voltage to 3.3V.

The Arduino Uno uses a Texas Instruments LP2985 linear regulator rated for load currents up to 150mA. That is sufficient for our application, but in order to give more room to grow we are instead using a TLV70233 rated at 300mA.

The big downside to linear regulators, compared to switching regulators, is that they can often be extremely inefficient and waste power by dissipating it as heat. This can be especially critical for battery-operated designs, of course. The amount of power wasted by linear regulators is equal to the load current times the voltage difference between

REGULATIONS SWITCHING EFFICIENCY

Switching regulators rapidly turn a series switch on/off at a controllable duty cycle. The duty cycle determines how much charge is sent to the output. Inductors and capacitors are energy storage components that supply energy during the off period. Since the series switch is either on or off, and never operates in the linear region, it wastes very little power. Switching regulators may have an efficiency of 90–95% whereas linear regulators are less than 50% efficient in some applications.

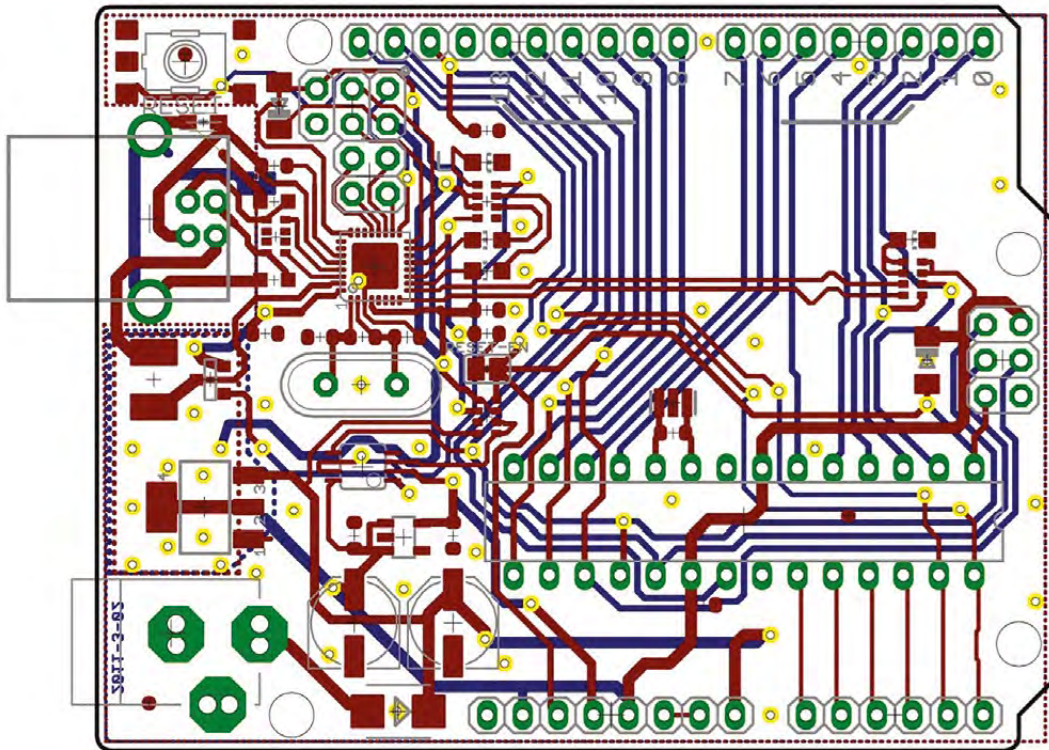



Figure 3  The PCB layout for an Arduino Uno is available to use as a reference for your own PCB design

the input and the output pins: $P = VI = (V_{in} - V_{out}) * LoadCurrent$.

If an application requires a large input-output voltage differential, and/or a high load current capacity, then linear regulators are not likely the best solution.

A linear regulator works well in this application because the difference between the input and output voltage is low ($3.7V - 3.3V = 0.4V$), and the current draw isn't that high.

For recharging the lithium-ion battery we're using the STBC08 battery charger from ST Microelectronics. Always use a battery charger IC specifically designed for the type of battery being used, no cheap substitutes.

DISPLAY RETRO COOL

The LCD display used in this project was originally used in the Nokia 5110 mobile phone that was introduced way back in 1998. By today's smartphone standards, these are quite primitive monochrome displays, but they still have lots of useful applications. They measure about 1.5" diagonally with a resolution of 84×48 pixels, and can be used to display graphics or text. They also feature a white backlight for improved readability in low-light conditions.

For the custom Arduino, we are going to make it function with a lithium-ion battery which can be recharged via the USB port

Charging a lithium-ion battery is comprised of three stages: pre-charge, constant current (CC) fast-charge, and constant voltage (CV) termination charge.

During the pre-charge stage, the battery will be charged using a small trickle current that is about 10% of the fast-charge current.

Once the battery voltage reaches about 3.0V, the charger will enter the fast-charge constant-current (CC) phase.

The battery continues to be charged in constant current mode (fast charge) until the battery reaches about 4.2V. At that point, the charger switches to constant-voltage (CV) mode.

On the STBC08, the fast-charge current is set with a resistor up to a maximum of 800mA. When charging via a USB port you must be careful to not exceed the maximum current capacity of the USB port. The easy solution is to limit the charge current to a maximum of 500mA since all USB ports can handle at least 500mA.

Normally, the fast-charge current should be limited to a maximum 1 C charge rate to prevent →

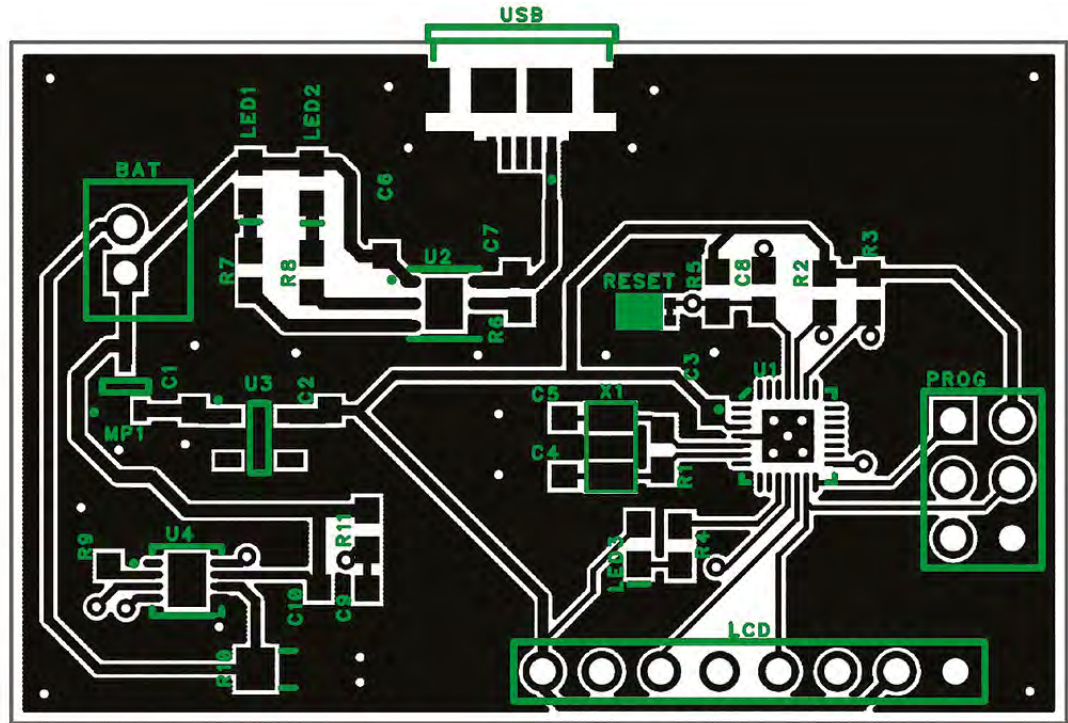


Figure 4 Shown here is the PCB layout for the custom Arduino board

any overheating and early battery degradation. But, every battery is different, so be sure to refer to the datasheet for the specific battery being used.

The other function that is typically required for a battery charger design is a way to indicate the battery charge level. A lithium-ion battery has a very non-linear discharge curve, so it's impossible to measure the state of charge by simply measuring its voltage.

To accurately monitor the charge level for a lithium-ion battery, the best solution is to use a chip designed for this purpose, such as the STC3100 from ST Microelectronics.

The STC3100 interfaces to the microcontroller via a simple two-wire serial interface called I2C. The microcontroller can now precisely monitor and report the battery charge level.

PRINTED CIRCUIT BOARD (PCB)

The schematic circuit is an abstract engineering diagram. Design of the PCB layout is necessary to turn the abstract schematic diagram into a real-world circuit board.

All Arduinos are open source, so you can easily access the PCB layout design files (see **Figure 3**). However, your PCB design will most likely need to be developed from scratch to meet the size and shape requirements for your specific product.

A microcontroller running at only a few tens of megahertz (MHz), without any wireless functionality, is a moderately easy PCB to design. Designing a PCB requires significant technical skills, but is well within the capability of most makers.

Always be aware that PCB layout design becomes significantly more complex as clock speeds reach into hundreds of MHz, and inevitably, even more so for GHz speeds. You'll find that the PCB layout for a high-speed microprocessor board, such as a

Raspberry Pi, is going to be considerably more complicated.

There are two general precautions for laying out a microcontroller PCB.

First, the crystal (which provides a highly precise clock for timing purposes),

and its associated load capacitors, must be carefully laid out. This includes placing these capacitors as near as you possibly can to their corresponding pins on the microcontroller chip.

Secondly, be sure to place decoupling capacitors as near as possible to the pins being decoupled.

Once you have the PCB layout design completed (**Figure 4**), and have run the necessary verification to ensure it matches the schematic, it's now time to order the boards.

You will need to send your PCB shop the PCB layout files (in a format called Gerber), along with your bill of materials (BOM).

// All Arduinos are open source, which means you can easily access the PCB layout design files

//

QUICK TIP

Older linear regulators required the input voltage to be a couple of volts higher than the voltage on the output. Now, most linear regulators are classified as Low-DropOut (LDO) regulators, meaning they can operate with an input voltage only a few hundred millivolts above the output voltage.

FIRMWARE DEVELOPMENT

As previously stated, an Arduino is programmed by connecting it to your computer via USB. Using USB allows the Arduino to be programmed from any computer without the requirement for any special hardware.

On the other hand, a custom microcontroller circuit is usually programmed using a simpler serial protocol such as UART, SPI, SWD, or JTAG.

Since these protocols aren't supported by most computers, a specialised hardware device called an in-system programmer (ISP) is required.

They are called in-system programmers because they allow programming while the microcontroller is embedded in the full system. Programming the microcontroller in this fashion allows for much easier testing and debugging.

An ISP is generally used with a specific line of microcontrollers. For example, the Atmel ISP is called the AVRISP mkII (**Figure 5**) and is used to program their 8-bit AVR line of microcontrollers such as the ATmega328P, discussed in this article.

The next step is to begin porting your Arduino sketch over to the native code required for the specific microcontroller chosen.

An Arduino, and custom microcontrollers, are both programmed using C++. But Arduino has greatly simplified the programming process by including an extensive library of numerous functions.

For example, on an Arduino, in order to define a GPIO pin as an output you would call the `pinMode()` function as follows:

```
pinMode(PinNumber, OUTPUT);
```

Then, to set this output low you would call the `digitalWrite()` function as follows:

```
digitalWrite(PinNumber, LOW);
```

Whenever you call either of these functions, the critical code is executed within these two already defined functions.

These functions won't be automatically available when transitioning to a custom microcontroller. Instead, you will just have to develop this fundamental code yourself.

FINAL THOUGHTS

The Arduino is celebrated as a great place to get started with a proof-of-concept prototype, or for just learning all you can about electronics. However, if you hope to sell your product then you'll eventually need a custom PCB in order to

reduce your product's cost, and to fit your desired form factor.

Although by no means trivial, it is within the capabilities of most makers to design their own custom PCB. This is especially true for relatively low-frequency circuits, such as for the microcontrollers used in Arduino.

For higher frequency circuits, such as high-performance microprocessors or wireless functions, it would be advisable to use modules in most cases. The use of a module will lower the design complexity and reduce your FCC certification costs.

Finally, it is recommended that you always be sure to get your design reviewed by an electronics engineer before you proceed with ordering prototype boards. □

Figure 5 ♦
A custom microcontroller without a USB communication port requires a special hardware device called an in-system programmer (ISP). Shown is the Atmel AVRISP mkII used for programming the firm's AVR series of microcontrollers



QUICK TIP

The 1C charge rate means, for example, if you have a battery with a capacity of 500mAh, then the maximum fast-charge current should be 500mA. A 2C charge rate would indicate you can charge this battery with up to 1000mA of current.