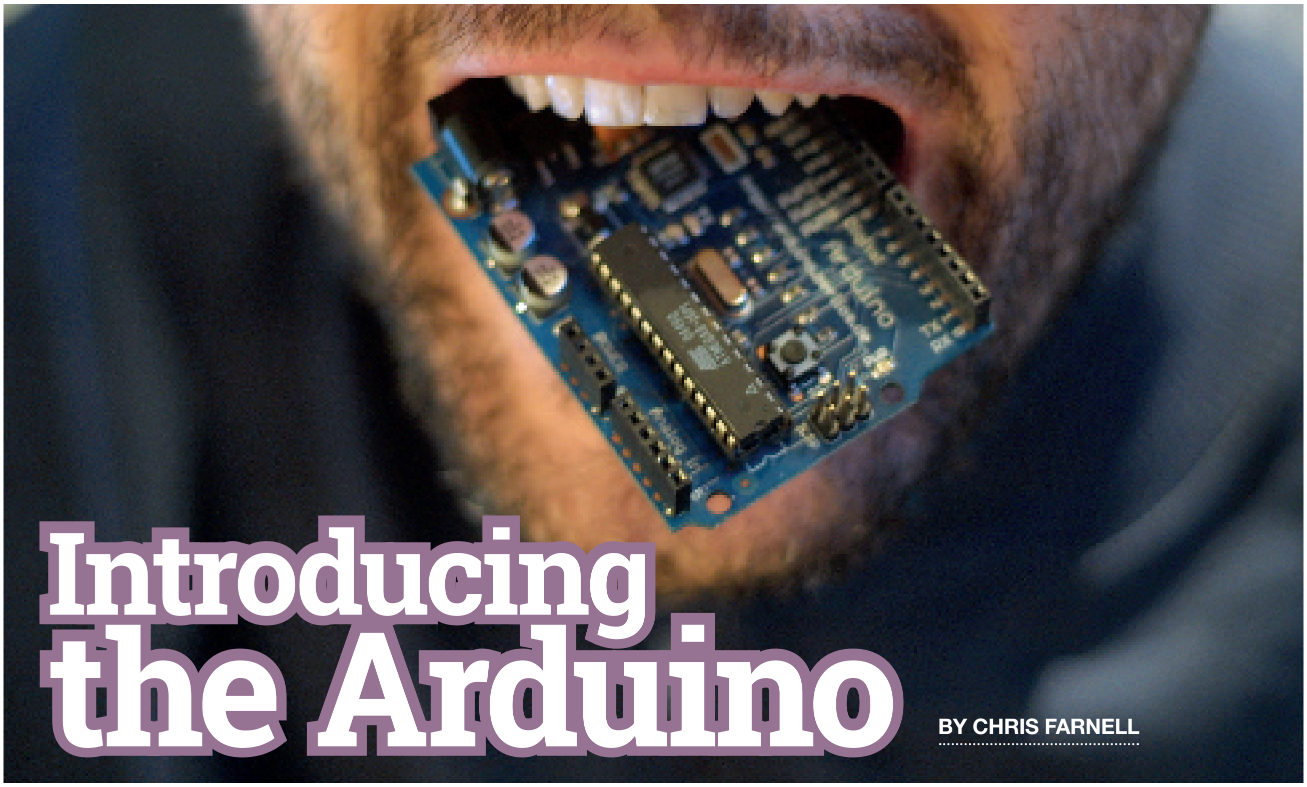


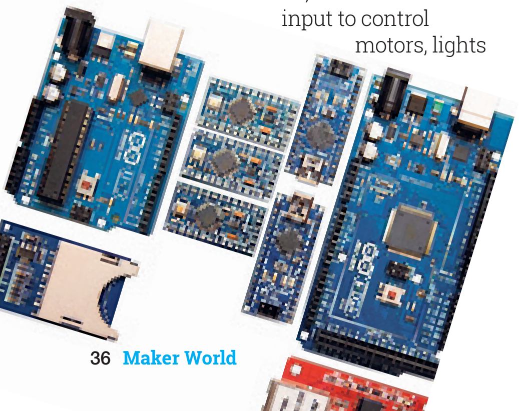
Introducing the Arduino

BY CHRIS FARNELL

If you're just starting out experimenting with electronics, there's a good chance that somebody has recommended you try out the Arduino platform. It's a simple, programmable circuit board that is easy for beginners and non-experts to use because you can load new programs onto the board with a simple USB cable. Arduino also uses a simplified C++ programming language, making it ideal for beginner programmers.

The advantage of the Arduino in its simplest form is that it allows people used to working in other disciplines a chance to try their hand at programming and electronics. Arduinos can be become very complicated but a simple PCB (printed circuit board) is a very user-friendly prospect. It might not seem like the most powerful bit of computing kit you've ever seen, but it can receive input from sensors, and can act on that input to control motors, lights and more. With an Arduino you can build a new, stand-alone piece of hardware, or you can build something you can control from your computer.

The hardware will usually consist of a board build around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM, loaded with a standard programming language compiler and boot loader (the programme that boots up a computer's operating system).



Why Arduino?

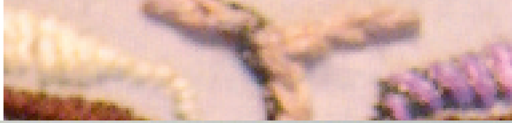
Arduino has become popular with everyone who wants to incorporate electronic components in their work, from programmers and engineers to artists.

It's not difficult to see the appeal of working with Arduinos, they're relatively inexpensive compared to other similar platforms, with pre-assembled modules costing little more than £30, and the raw components are available even more cheaply if you don't mind assembling the module yourself. The Arduino can be run on Windows, Linux or Macintosh OSX, where most other microcontrollers can only run with Windows.

The Arduino also offers an easy-to-understand Integrated Developing Environment (IDE), geared towards beginners but flexible enough for advanced users to get the most out of, making it a popular tool for teachers as well as the self-taught.

But perhaps the biggest advantage of the Arduino is that it is almost completely open source. All the software you need to run an Arduino is available free online, with a programming language that can be easily expanded using C++ libraries and AVR C code that you can put directly into Arduino programs.

More crucially, Arduino also presents the rise of open source hardware. The designs for their modules are



published online under a Creative Commons licence, meaning that you're free to design and build your own version of the Arduino module, customising and improving it as you like.

Arduino in the Wild

While researching our other articles for this magazine, we found that nearly everyone we talked to had something to say about the Arduino. Sarah Angliss, an artist who works with robots to make on-stage performances (see p106), found that Arduino-compatible tools were a great way to create a pared-down version of the much more advanced tools she'd been using in a motion-capture studio.

"I'm using things like the Razor IMU, and it's brilliant. It's around £80 and it's an accelerometer, a magnetometer, and a gyro all built into this one Arduino-compatible object", she tells us. "With something I can buy off the internet, I can capture a lot of data from subjects without them having to put a motion capture suit on."

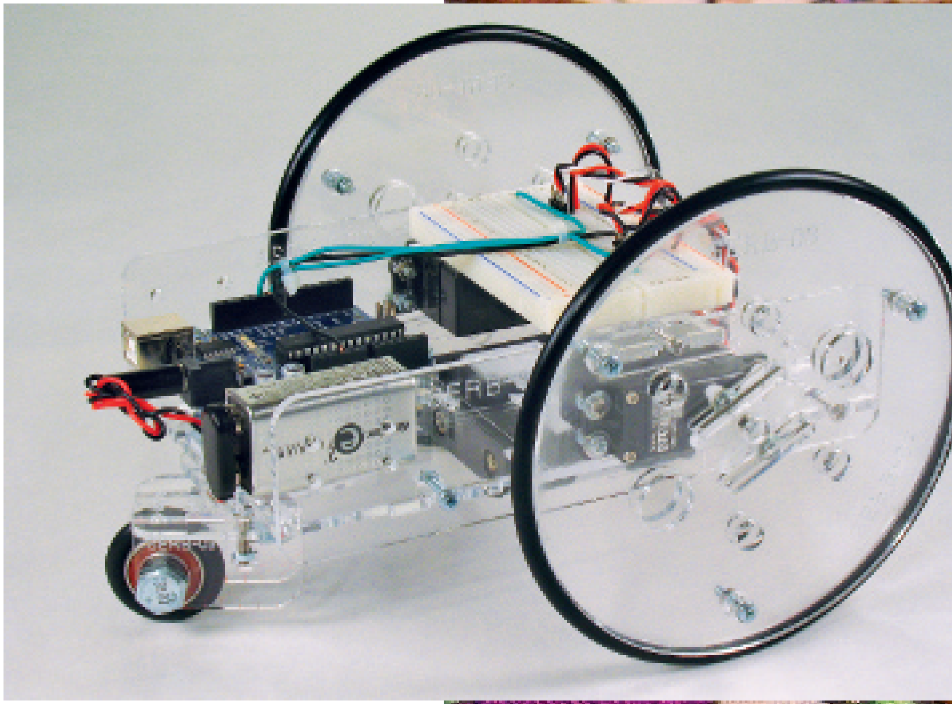
For other Makers, the Arduino was actually an impetus to build something. Sandy Noble (see p36) began working on his linear clock as an excuse to try out the capabilities of his Arduino. "The linear clock was at first something I put together, partly as a trial exercise to get used to Arduino and stepper motors", Sandy admits, suggesting that while the Arduino is probably not an ideal component for a finished, packaged product, it is perfect for prototyping and even for selling kits.

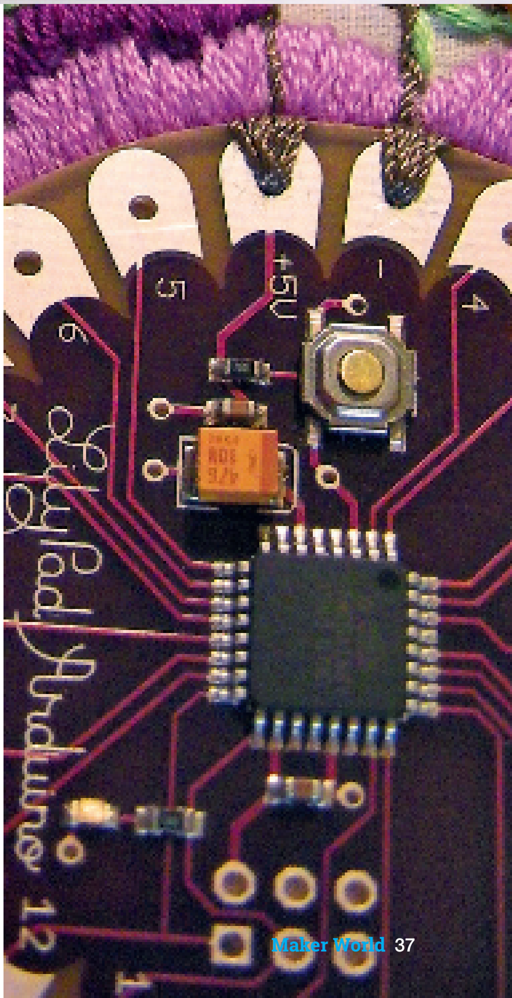
"The Arduino is an expensive way of getting a microcontroller running", he explains. "It can all be squashed onto a much smaller, much simpler board that would cost less."

A quick glance at Instructables shows that people are doing all sorts with the Arduino, from jack1986's Arduino Fingerprint Lock and the spikec's *duino Keg Temp Monitor, to ST-Geotronics showing how an Arduino can help you build a replica of the Enigma machine and dschurman demonstrating how to build a DIY Robotic Hand controlled by a glove. The options are seemingly limitless.

Sizing up the Pi

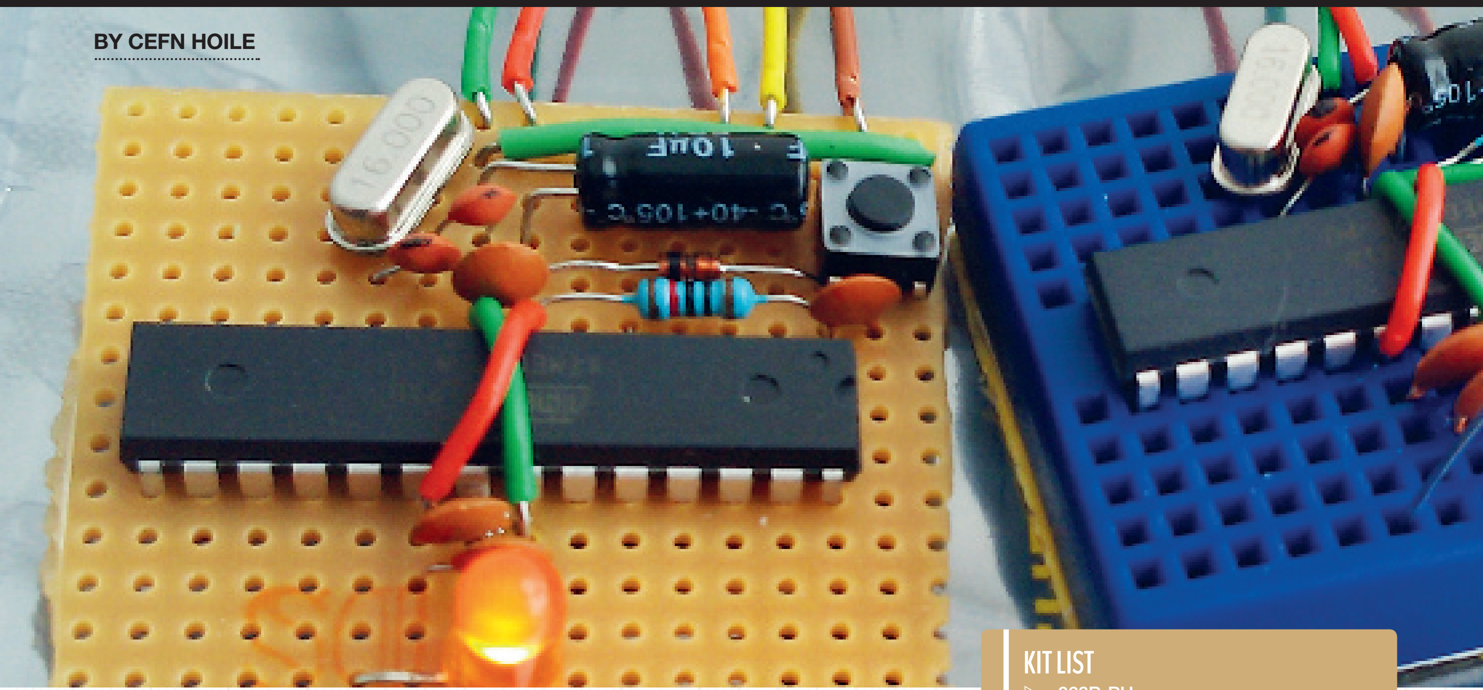
You might assume because of their joint popularity and visual similarities that the Raspberry Pi is the Arduino's only serious rival when it comes to developing and prototyping projects. However, this is something of a misconception as the two devices have very different functions. The truth of the matter is that the Arduino is a micro-processor, whereas the Raspberry Pi is a mini computer. A microprocessor like the Arduino forms just a small part of what makes a computer run and is best for performing a series of (relatively!) simple tasks quickly and efficiently, which is why it's great for prototyping and tests. The Pi is a much more complex machine that can execute actions that demand greater processing power - any gadget that relies on interaction, like the Postbot (p52) for instance, would be better off utilising the Pi. Playing with Arduinos is, however, an excellent place to begin a self-guided education in electronics - our tutorial on how to build your own replica Arduino Uno in the form of a Shrimp board (overleaf) is an ideal place to start, though we say it ourselves! If you do want to purchase rather than make your first Arduino, however, be mindful of where you source your materials and make sure you get an official Arduino board rather than purchasing a shoddy counterfeit version. It's not a bargain if it doesn't function! There's a guide to identifying fake Arduinos here:
http://bit.ly/fakearduinos
In the meantime, happy tinkering!. MW





Make a DIY Arduino-

BY CEFN HOILE



Arduinos are very useful for physical computing projects, but official boards are expensive and a printed circuit board is hard to decipher. Instead, why not build your own Arduino-Uno-compatible 'Shrimp' circuit on breadboard or stripboard for a fraction of the cost, learn about the components as you go, and remix freely to suit your needs? Cefn Hoile, maintainer of **http://shrimping.it** and 'sculptor of open source hardware', shows us how...

Build a Shrimp, Blink an LED

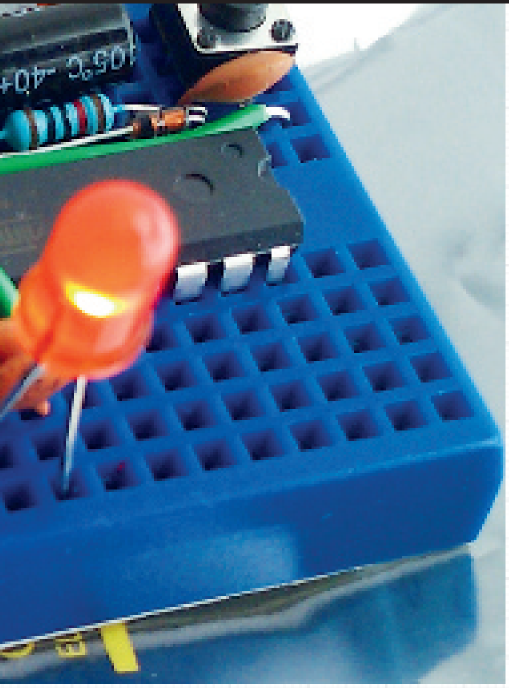
In this step-by-step illustrated guide, we show you how to build an Arduino-compatible Shrimp from scratch, and upload your first code to control it.
A Shrimp is great for deploying your first interactive inventions, doing physical computing using sensors and actuators. It is programmed using the Arduino IDE, and thinks it's an Arduino Uno but is much cheaper, and teaches more electronics fundamentals. You can use it for any project that uses an Arduino Uno.
You make a Shrimp by pushing components into holes in a solderless breadboard. You can source them yourself following the materials list below, or buy pre-bagged bundles from us here: **http://bit.ly/buyshrimp**
We use the 'Blink' build at the beginning of all our workshops, and when prototyping for ourselves, to prove that everything works before adding more challenging modules or behaviours. The minimal circuit can take 20 minutes or so when attempted the first time, plus troubleshooting.

KIT LIST

▷ 328P-PU
▷ 2 x 22pF capacitor
▷ 4 x 100nF capacitor
▷ 1 x 10μF capacitor
▷ 1 x 10KΩ resistor
▷ 16MHz quartz crystal
▷ 6mm tactile switch
▷ Various coloured wires or jumpers
▷ One LED and matching resistor for testing
▷ 1 x 170 point breadboard
▷ CP2102 (with DTR pin) and a pre-bootloaded ATMEGA chip (for a no hassle/soldering/bootloading make)
▷ 40-pin male header strip

No matter what kind of Arduino you have, don't forget to download the Arduino software from http://bit.ly/arduinosoftware so you can program your board. If you need extra help sourcing your components, go to http://shrimping.it/blog/bill-of-materials/

-compatible board



So, get hold of your Shrimp components and a breadboard, sit down with your laptop and favourite beverage and let's get started!

Meet your breadboard

1 A solderless breadboard allows you to make circuits by pushing wires into holes, instead of soldering.

A gap from top to bottom separates the breadboard into two halves. A metal rail behind each 5-hole row grips the legs of inserted components, connecting them. Components in rows above, below or across the gap remain separate.

A 400 tie-point solderless breadboard is shown, which has 'power rails', four full-height metal rails visible as columns down the sides. Although these holes are also grouped into five, everything in each single column is connected to everything else inserted in that column. They are called power rails as they are usually wired to 5V (plus, or power) and 0V (minus, or ground), two connections which are used a lot throughout a typical circuit.

The Shrimp layout works on a mini-breadboard or soldered onto stripboard without changes. Our breadboards have readable column letters and row numbers, but even if you bought your own, this guide can be used positioning the components by eye relative to each other.

1. Place the breadboard with its central spine vertically, and with row numbers starting from 1 at the top (if your rows and columns are labelled).

The ATMEGA chip

2 The ATMEGA microcontroller is a black oblong with numbers printed on it, and 28 silver legs, and looks a bit like an insect. It is the computer at the heart of a Shrimp, and has inputs and outputs – sensing or triggering things out there in the world.

Check the legs don't splay out too much. Forcing the chip in can break the legs. If the legs are not at right angles to the chip, ease them into position by gently pressing one side of the chip against the table top (14 pins at a time). Check the pins are lined up well on the correct holes before pushing down softly to slide them in, then finally push down hard to ensure a good connection.

2. Carefully align the chip, checking the half-moon shape is at the top, with two empty breadboard rows above the chip. Check the legs are cleanly aligned with the breadboard below. Once the legs are all aligned, press down until the chip slides fully into the board (about 3mm movement).

100 nanoFarad 'decoupling' capacitor

3 Look for the 100 nanoFarad ceramic capacitor, a small disc with two thin wires coming out of it. These have no orientation; each leg is the same as the other.

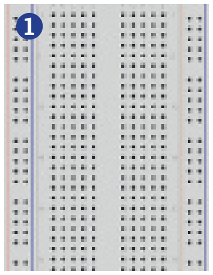
This 'decoupling' capacitor smoothes electrical spikes, so that the reboot signal sent through to pin 1 is stable and reliably detected. The chip should reboot only when you request it (e.g. when you're reprogramming the microcontroller).

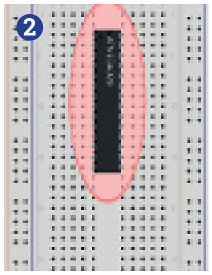
The digits 104 indicate capacitance in picoFarads using scientific notation. The last figure '4' tells us how many zeroes to add, so the capacitance starts '10' and then continues with a further four zeroes – 100,000 picoFarads. Since 1 nanoFarad is 1000 picoFarads, there're 100 nanoFarads in a capacitor marked '104'.

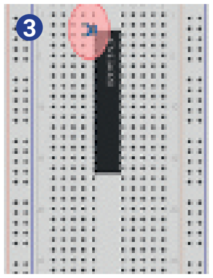
3. Insert the capacitor marked '104' between the top left leg of the chip (pin 1, immediately anticlockwise from the half-moon shape) and the empty row immediately above that.

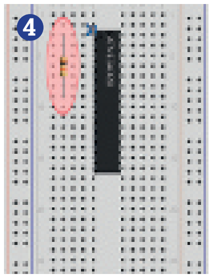
10 kiloOhm 'pull-up' resistor

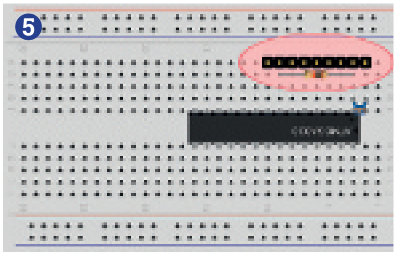
4 Look for a brown or blue cylinder with a wire at each end, with stripes of brown, black and orange. Resistors have no orientation, so you can't wire them backwards.

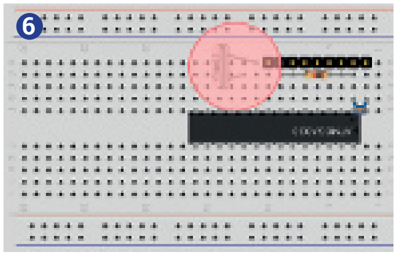


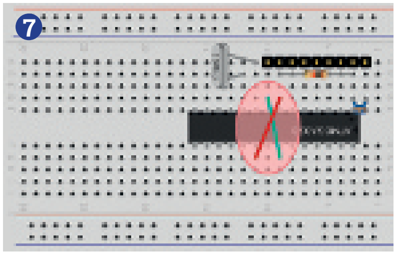


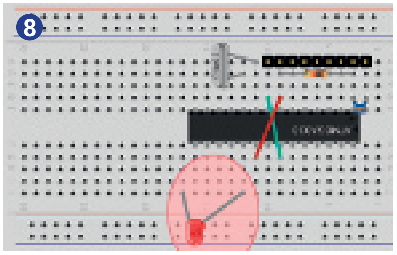


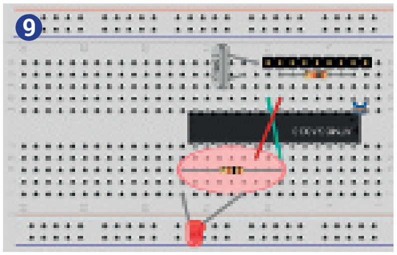


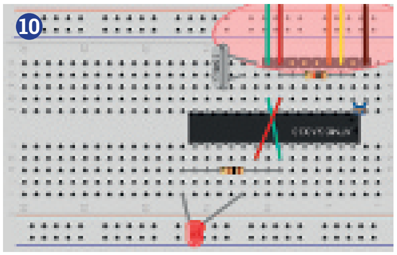
5


6


7


8


9


10


This will be used as a pull-up resistor. A positive voltage, usually 5V, 'pulls up' the reboot pin (pin 1) through this resistor. The ATMEGA will keep running so long as it gets a high voltage on this pin. When reprogramming, the brown wire is briefly connected directly to 0V (a stronger signal than the flow from 5V limited by the resistor). Pin 1 therefore gets pulled to 0V, which causes the chip to reboot. After rebooting the ATMEGA listens for a new program sent over the orange wire. If nothing is sent, it runs the last program.

The coloured stripes are decoded just like the '104' capacitor earlier, but colours are used instead of digits. Decoding the colors as brown=1, black=0, orange=3 gives us the number 103. That means the resistance in Ohms starts '10' and continues with a further three zeroes – 10,000 Ohms or 10 kiloOhms.

4. Attach the resistor with brown, black and orange stripes between row 9 on the breadboard (this will be the positive power line, which is approximately 5 volts) and the top left leg of the chip (pin 1, immediately anticlockwise from the half-moon shape).

9-pin power and programming header

5 A series of copper pins will be used to program and provide power to the Shrimp. Break off a strip of 9 pins from the 40 pin male header strip, keeping the strip intact as you slide the 9 pins in. Not all of the 9 pins will be used in this circuit, but having all 9 in the right place helps you position everything else.

5. Push the 9-pin header strip into the top left corner of the board, leaving just one empty row at the top of the board.

16 MHz crystal

6 A silver box with rounded ends, and two wires, marked '16.000'.

A computer is a bit like clockwork. The first ever digital computer was built using clock-making techniques, with cogs representing numbers! This quartz crystal acts like the clock's pendulum, causing the mechanism to tick along.

The '16.000' indicates the number of back-and-forth movements this crystal generates per second, in megahertz. One hertz means once per second, and one megahertz means one million times per second.

6. Insert the crystal with one leg in the row immediately below the 9-pin header, and the other leg in the row below that.

Power and ground wires

7 Strip the ends of your red and green power wires, if needed.

The ATMEGA chip is broken up internally into separate parts, each of which needs a stable power supply. Power and ground wires will soon be attached to the 9-pin header. Two wires are needed to connect power and ground across to the correct legs on the right-hand half of the microcontroller.

7. Connect a green wire to the last pin of the 9-pin header, across the chip and up one row.

Connect a red wire to the second-to-last pin of the 9-pin header, across the chip and down two rows.

Light emitting diode (LED)

8 A diode only allows electrical current to flow in one direction. Electricity should flow into the long leg and out of the short leg. Round LEDs also have one slightly flatter side, which corresponds with the short, negative leg of the LED.

8. Insert an LED, inserting the long leg into the row below the red line on the right-hand side of the chip (power, or 5 volts) and the short leg in the first empty row below the microcontroller.

100 Ohm 'current-limiting' series resistor

9 Our circuit is running at approximately 5 volts, and LEDs are rated around 1–2 volts. This resistor is connected in series with the LED, limiting the amount of electricity flowing to prevent the LED from overheating and destroying itself. Some of the voltage will be used up by the resistor, and only a suitable share of it is applied across the LED.

For an explanation of the coloured stripes, see the earlier section describing the 10 kiloOhm pull-up resistor.

9. Insert the resistor between the short leg of the LED and the row containing the green wire on the right hand side of the chip (ground or 0 volts)

USB to UART (CP2102 module).

10 The CP2102 is a green- or blue-coloured circuit board that has a USB connector at one end and 6 pins at the

other end.

This device enables your desktop or laptop machine to communicate with the Shrimp, for example to send new programs to it with the free Arduino IDE, or to exchange other information with a program when it's running on the Shrimp.

10. Attach the rainbow wires from the 9-pin header to the CP2102's labelled pins.

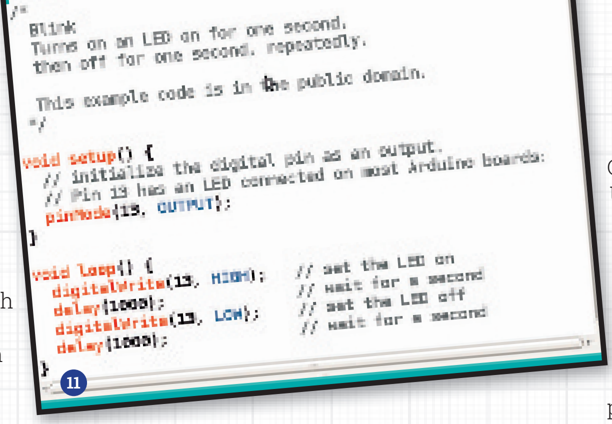
Red to 5V
Orange to TXD
Yellow to RXD
Green to GND
Brown to DTR

Upload the 'Blink' program

11 To upload a new program, you need to have the correct UART drivers installed in your machine. You can download CP2102 installers for Windows and Mac at **http://shrimping.it/drivers/**. Linux machines can automatically connect to the CP2102 device, as the drivers are built in.

You will also need to install the Arduino IDE to write and upload new programs to your Shrimp, available to download for free for Mac, Windows and Linux at **http://bit.ly/arduinosoftware**.

Install UART drivers.
Install the Arduino IDE.
Load 'Blink' from the Arduino IDE menu **(File -> Examples -> Basics -> Blink)**.
Check there is a tick next to the correct entry in **Tools -> Serial Ports**


11

```
/*
  Blink
  Turns on an LED on for one second,
  then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Check that there is a dot next to Arduino Uno in **Tools -> Board**
Click on the horizontal arrow in the toolbar to upload the program.

Success: start coding!

You should now have a working Shrimp! The LED should be flashing on and off.

You can change bits of your code to prove that your machine is able to send new behaviours to your microcontroller. Why not change the number of milliseconds in the delay request so that the light blinks much more quickly, much more slowly, spends longer on for a long blink, or spends longer off for a short blink? **MW**

i ▷ You can explore other Shrimp projects and see what others have made at: shrimping.it. You can also find the Shrimp Project on Twitter: @ShrimpingIt With thanks to Fritzing.org for electronics vector graphics.