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 in a whole world of projects.

You make a Shrimp by pushing components into holes in a solderless breadboard. You can source them yourself following our guide, or buy pre-bagged bundles from us.

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 if you didn't follow the instructions quite right.

Get hold of your Shrimp components and a breadboard, sit down with your laptop and favourite beverage, then click 'Next Step'

Meet your breadboard

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

A gap from top to bottom separates it in two halves. A metal rail behind each 5-hole row grips the legs of inserted components, connecting them. Components in other 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 5, 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.

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

The ATMEGA microcontroller is a black oblong with numbers printed on it, and 28 silver legs, looking 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. It can break the legs if you force the chip in. 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.

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

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 smooths 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 4 zeroes - 100,000 picoFarads. Since 1 nanoFarad is 1000 picoFarads, there's 100 nanoFarads in a capacitor marked '104'.

Insert the capacitor marked '104' between

```
<li>the top left leg of the chip (pin 1, immediately anticlockwise from the half-moon shape)
<li>the empty row immediately above that</li>
```

10 kiloOhm 'pull-up' resistor

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.

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 colored stripes are decoded just like the '104' capacitor earlier, but colors are used instead of digits. Decoding the colors Brown=1, Black=0, Orange=3 gives us the number 103. That means the resistance in Ohms starts '10' and continues with a further 3 zeroes - 10,000 Ohms or 10 kiloOhms.

Attach the resistor with Brown, Black and Orange stripes between...

row 9 on the breadboard (will be the positive power line, approx. 5 Volts)
the top left leg of the chip (pin 1, immediately anticlockwise from the half-moon shape)

9-pin power and programming header

A series of copper pins will be used to program and provide power to the Shrimp. Find the line of 9 pins encased in black plastic, keeping the strip intact as you slide the 9 pins in. Not all of the 9 pins be used in this circuit, but having all 9 in the right place helps you position everything else.

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

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.

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

Look for one Red and one Green wire, stripped to show silver at each end. 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.

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

Look for a red or clear dome having two wire legs. 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.

Insert an LED, inserting the long leg to the row below the Red line on the right hand side of the chip (power, or 5Volts) and the short leg in the first empty row below the microcontroller.

100 Ohm ‘current-limiting’ series resistor

Look for a brown or blue cylinder with a wire at each end, with strips of Brown, Black, Brown. This has no orientation, and can be attached either way round.

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 10kiloOhm pull-up resistor.

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 0Volts)

USB to UART, (CP2102 module)

Look for a green or blue-coloured circuit board with 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.

For the latest Baite CP2102 modules, (with DTR as sixth pin on the front) attach the rainbow wires from the 9pin header to the CP2102’s labelled pins like...

```
<li>Red -&gt; 5V</li>
<li>Orange -&gt; RXD</li>
<li>Yellow -&gt; TXD</li>
<li>Green -&gt; GND</li>
<li>Brown -&gt; DTR</li>
```

For older Baite CP2102 modules (with DTR pin labelled on the back) exchange TX and RX to be like...

```
<li>Orange -&gt; TXD</li>
<li>Yellow -&gt; RXD</li>
```

Upload the ‘Blink’ program

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 here.

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

Next, why not build a POV circuit, a Memory Game, solder it on to Stripboard Alternatively take a look at what other people are building with Arduino-compatibles, now you have one of your own!