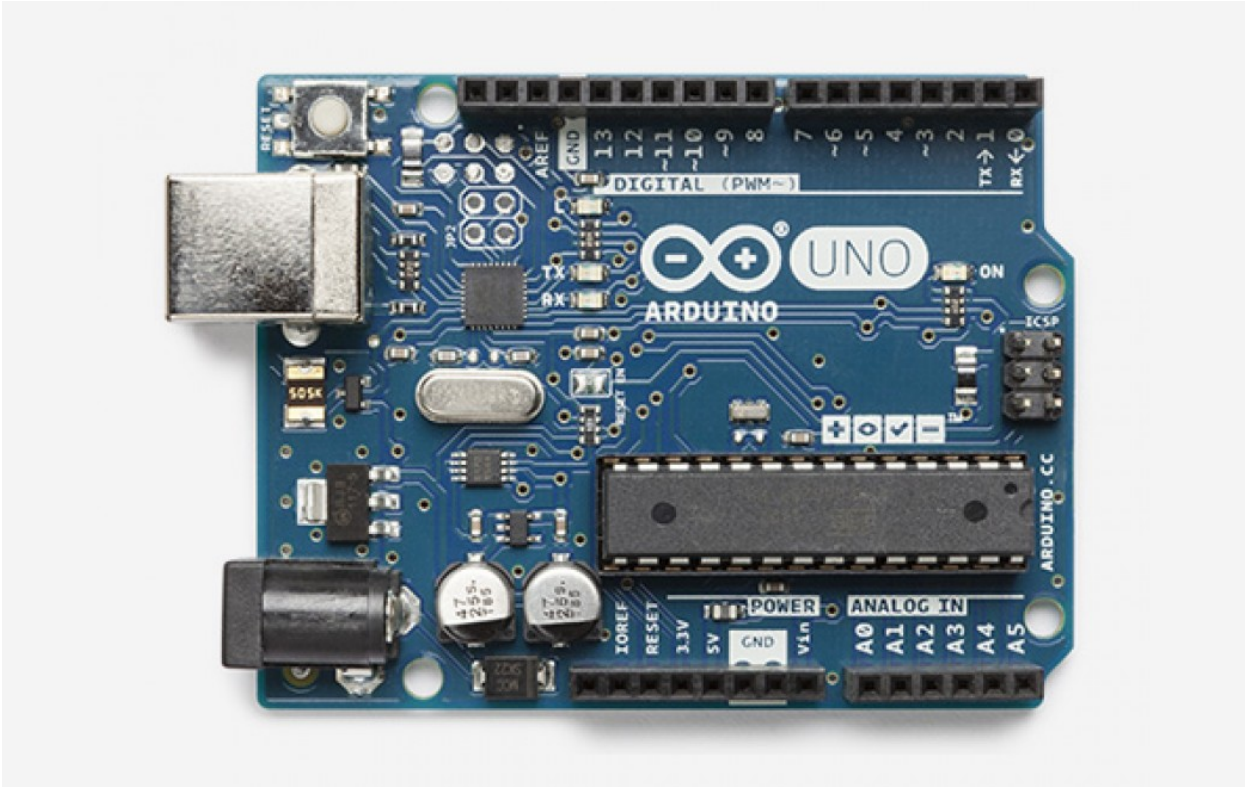
From Arduino Prototype to Manufacturable Product

predictabledesigns.com/from-arduino-prototype-to-manufacturable-product/



Article Technical Rating: 7 out of 10

Creating a prototype based on an Arduino (or Genuino outside the US) is an excellent start to bringing a new electronic hardware product to market.

The Arduino is an ideal platform for proving your product concept. However, there is still a lot of engineering work required to turn it into a product that can be manufactured and sold.

NOTE: Want to learn even more about turning your Arduino project into a product you can sell? If so, then download **The Complete Guide from Arduino to Sellable Product**.

The most straightforward strategy to transition from an Arduino prototype to a truly sellable product is to use the same microcontroller as used in the Arduino. Although there may be higher performance and lower cost microcontrollers available, the simplest option is to just use thesame microcontroller.

There are two key reasons why using the same microcontroller is the easiest option.

First, the firmware you've already developed is more easily ported over to the manufacturable version of your product.

Secondly, Arduino is open-source hardware so for the most part the circuit schematics can be simply copied.

Design the Schematic Circuit Diagram

The first step in migrating to a manufacturable product is to select the microcontroller. The simplest option is to use the same microcontroller as your Arduino model.

Selecting the microcontroller

The large majority of Arduino models are based on an Atmel AVR 8-bit microcontroller (see Table 1 below). The exceptions are the Arduino Due, Zero, MKR1000, and MKRZero all of which are based on 32-bit ARM Cortex-M architecture microcontrollers from Atmel (ARM Cortex-M is a very popular architecture implemented by many microcontroller manufacturers).

For this article I will be focusing solely on Atmel AVR 8-bit microcontrollers, and I will cover how to transition to an ARM Cortex-M microcontroller in a future article.

Considering that I was just recently selected as an Arm Innovator, it's no secret that I like Arm Cortex-M based 32-bit microcontrollers. However, an 8-bit AVR microcontroller is sufficient for many applications, and is by far the most common microcontroller used in Arduino kits.

For this article we'll be specifically dissecting the Arduino Uno which uses the Atmel ATmega328 microcontroller.

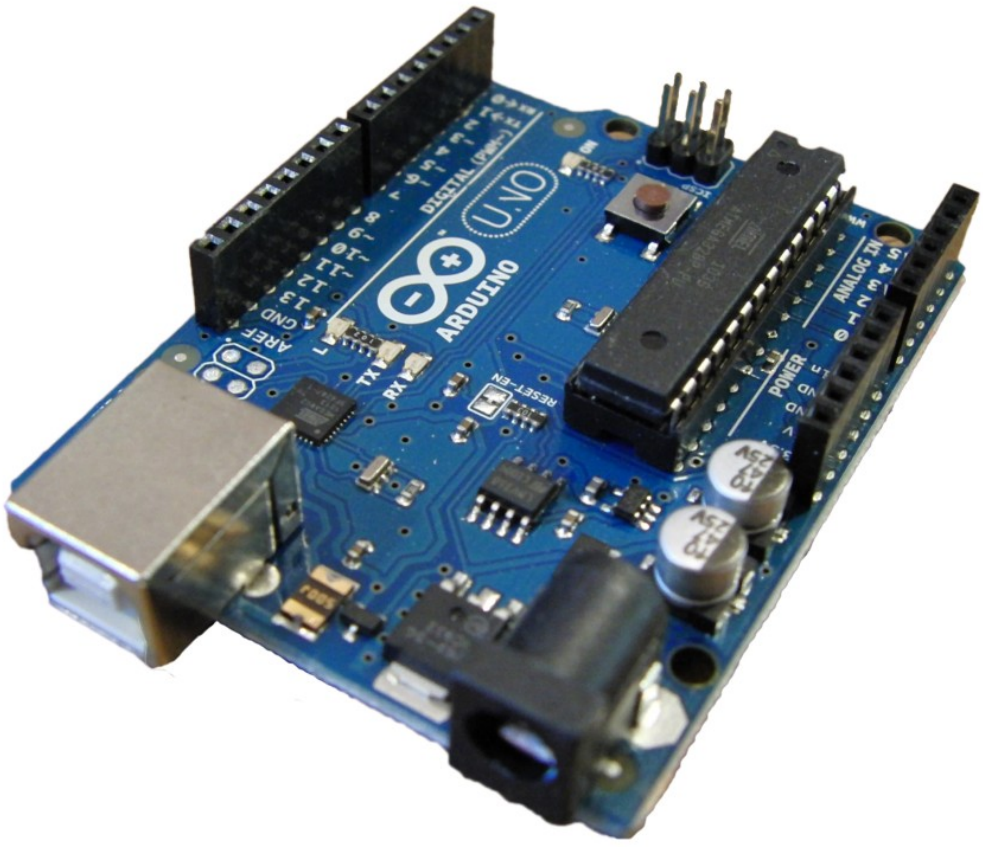


Figure 1 – Arduino Uno uses an ATmega328 microcontroller with 32kB of Flash memory
The ATmega328 used in the Uno is a through-hole DIP (Dual-Inline Package) version in a socket. Use of a socket allows the microcontroller to be easily swapped out if it becomes damaged. A socketed microcontroller may be a good idea for a development kit, but not for a production product.

First of all, a DIP package is going to be significantly larger than a SMT (Surface-Mount-Technology) package, especially with the added size of a socket. Secondly, your PCB assembly costs will be lower if you avoid through-hole packages entirely.

Finally, DIP packages tend to be more expensive since they are not generally used in high volume production. For example, the ATmega328 costs $1.66 @ 100 pieces in a DIP package, but only $1.18 for a SMT package.

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
101	Intel® Curie	3.3 V/ 7-12V	32MHz	6/0	14/4	-	24	196	Regular	-
Gemma	ATtiny85	3.3 V / 4-16 V	8 MHz	1/0	3/2	0.5	0.5	8	Micro	0
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7-5.5 V	8MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
Mega 2560	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
MKR1000	SAMD21 Cortex-M0+	3.3 V/ 5V	48MHz	7/1	8/4	-	32	256	Micro	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512 1	1 2	16 32	-	1
Pro Mini	ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	1	2	32	-	1
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Zero	ATSAMD21G18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	512	2 Micro	4

Table 1 – The microcontrollers used in the various models of Arduino.
There are numerous microcontrollers on the market that are more powerful and cheaper than the ATmega.

For example, there are 32-bit ARM Cortex-M microcontrollers available for about half the price of the ATmega328. Not only are they half the price but they include twice the memory and several times the processing power. In fact, there are 32-bit microcontrollers available for under $1!

However, using with the same microcontroller as your development kit will make the transition significantly less complicated. Both the hardware design and the software development will be simplified.

Schematic Review of Arduino Uno

One of the great things about the Arduino is that it's an open-source platform. This means that you can easily view both the schematic and PCB layout for any of the Arduinos. Let's start by looking at the schematic circuit for the Uno.

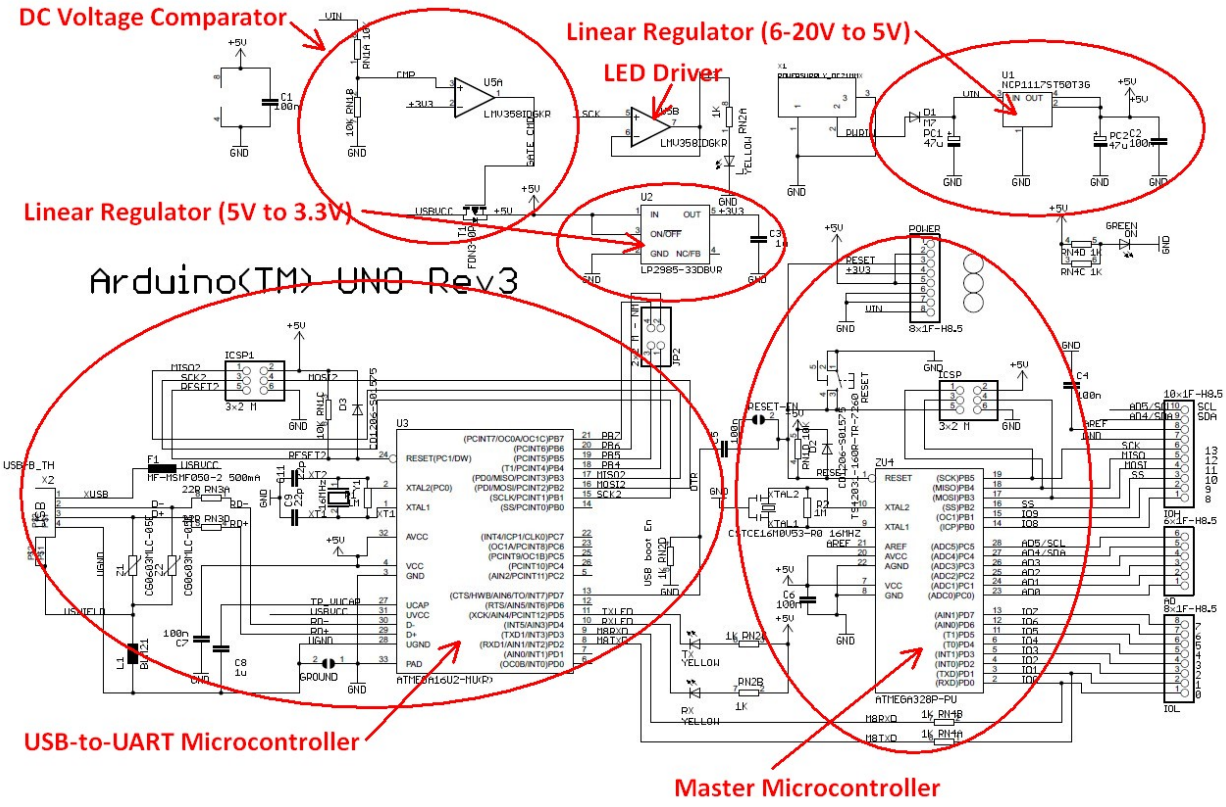


Figure 2 – Marked up schematic diagram for an Arduino Uno.

Looking at the schematic for the Arduino Uno you'll see there are two primary integrated chips: U3 and U4. U4 is an ATmega328P microcontroller, and U3 is an ATmega16U2 microcontroller. Wait a second, why are there two microcontrollers?

The ATmega328P (U4) is the primary microcontroller. The second microcontroller (U3 – ATmega16U2) is solely there to provide a USB to UART conversion function since the ATmega328P doesn't include a built-in USB port for programming. Older Arduinos instead used a specialized USB-to-UART chip from FTDI called the FT232RL.

By changing the FTDI chip to the ATmega16U2 it not only lowers the cost of the Arduino, but it also allows advanced users to use the USB port for other types of devices such as a keyboard or mouse. In general, a microcontroller based solution will provide more flexibility than a specialized solution like the FTDI chip.

As I will discuss later in regards to programming, with a custom microcontroller circuit you will no longer need a USB port in your design for programming purposes. So, if your product doesn't require a USB communication for other purposes (USB charging is different), then you don't need U3.

If you do require a USB port for your product then I would instead suggest you use a microcontroller that includes an embedded USB port, such as the ATmega32U4 microcontroller used on the Arduino Leonardo.

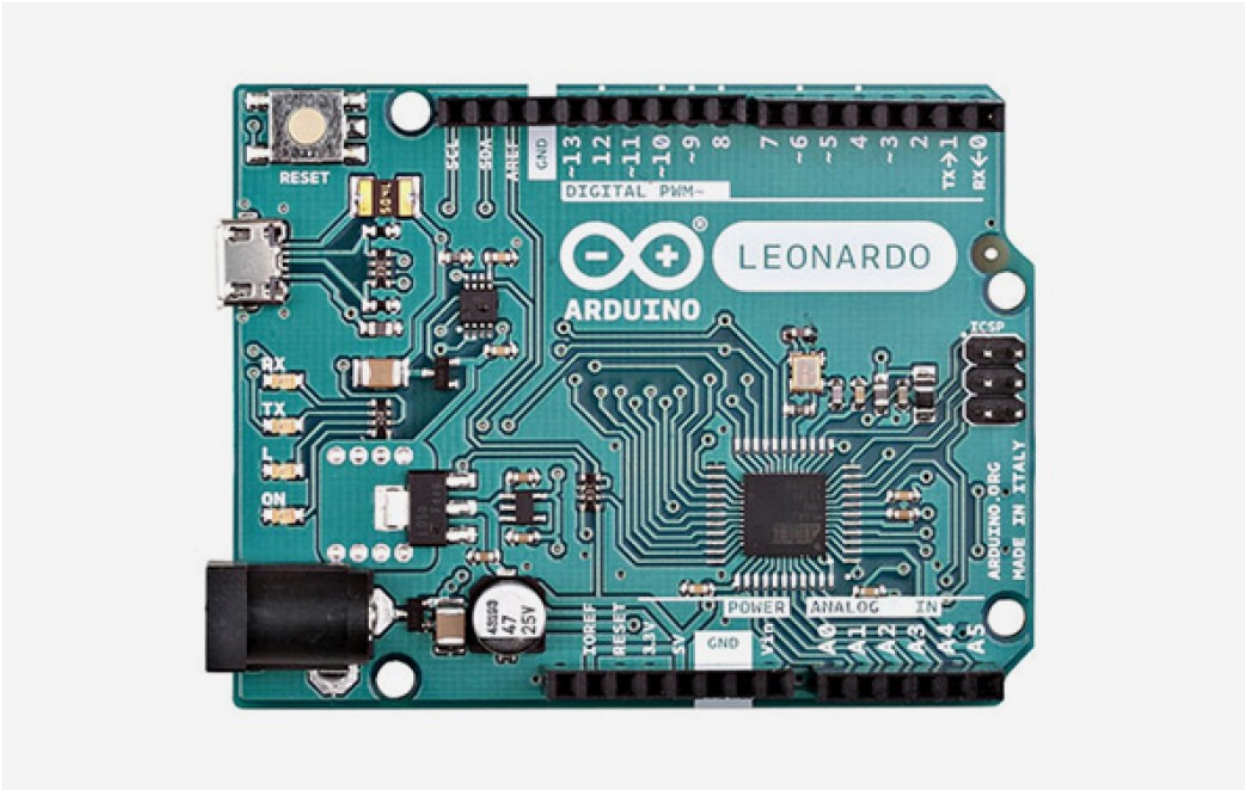


Figure 3 – Arduino Leonardo uses an ATmega32U4 microcontroller with a built-in USB port
That being said, the ATmega32U4 is quite pricey at around $3.47 @ 100 pieces. ST Microelectronics offers an ARM Cortex-M 32-bit microcontroller with USB functionality for only $1.92 @ 100 pieces.

Microcontroller Support Circuitry

Each of the two microcontroller circuits in the Uno consists of a crystal oscillator running at 16 MHz, various GPIO (General Purpose Input/Output) signals, multiple serial interfaces including one for programming, a power supply, and lots of decoupling capacitors.

U5 is a dual op-amp (operational amplifier) called the LMV358IDGKR from Texas Instruments. One of the two op-amps (U5A) is operated as a comparator since it has no feedback. This comparator is used to determine if the Arduino is being powered by the DC input or via the USB port.

If the 6-20V DC input voltage is present then the 5V supply is generated by an on-board linear regulator (as I discuss in detail shortly). On the other hand, if the 6-20V DC input is not present then the 5V supply voltage comes from the USB port.

So, if there is a 6-20V DC input voltage supplied then the positive input of the U5A comparator is higher than the negative input (3.3VDC). In this case the output of the comparator will be high, and PMOS transistor T1 will be turned off. This disconnects the internal 5V signal from the USB supply voltage.

If the 6-20V DC input is not present then the output of U5A will be low which turns on T1, thus the internal 5V supply comes from the USB port.

The other op-amp in U5 (U5B) is connected in a configuration known as a unity-gain feedback amplifier. This is a fancy way of saying that it has a gain of 1 which means it acts as a simple buffer.

Whatever voltage you put on the input of U5B is what you get on the output. The purpose is that now the output is able to drive a much larger load. In this case, this buffer is there simply to flash an LED whenever the serial programming clock (SCK) signal is present.

Power Circuit

The power circuit for the Uno is based on an NCP1117 linear regulator from ON Semiconductor. This regulator generates a 5V DC voltage from the 6-20V DC input voltage and can source up to 1A of current.

The use of a linear regulator in this situation is fine for some products, but not if your product is powered from a battery, or consumes large amounts of current. A linear regulator such as the NCP1117 is extremely inefficient when the input voltage is significantly higher than the output voltage. Being inefficient means it wastes a lot of the power by dissipating heat.

For example, on the Uno the input voltage can be as high as 20V, and the output voltage is only 5V. This means the input-output differential voltage is 15V. If you pull the maximum current of 1A from this regulator the power dissipated by the linear regulator would be (Vin – Vout) * Iout = (20V – 5V) * 1A = 15W!

If the NCP1117 didn't have an internal thermal shutdown feature, it would literally cook while trying to dissipate this much power. Regardless, you will waste all of this power as heat.

If you need to step-down a high voltage to a significantly lower voltage then a switching regulator is a much better choice. I won't get into the details of switching regulators in this article, but they are many times more efficient than linear regulators. However, switching regulators are also considerably more complex than linear regulators.

On the Uno a LP2985 linear regulator from Texas Instruments is used to create a 3.3V voltage. The LP2985 is rated for 150mA of load current. This type of linear regulator is also called a Low-Drop-Out (LDO) regulator because it requires very little differential voltage from the input to the output.

Older, non-LDO linear regulators required the input voltage to be a few volts above the output voltage. However, from a power dissipation standpoint it's best to operate a linear regulator with an input voltage close to the output voltage.

The 3.3V voltage is fed into a comparator (U5A) that is used to switch to USB power, if available, when no power supply is plugged in.

The LP2985 doesn't dissipate that much power so a linear regulator is a good choice for this regulator. This is because the input-output voltage differential is only 5V – 3.3V = 1.7V, and the maximum current is only 150mA.

A very common strategy is to use a switching step-down regulator (also called a buck regulator) followed by a linear regulator. In addition to the increased complexity, the other downside of a switching regulator is it provides a "noisy" output voltage. This is fine for many applications. However, if you require a cleaner supply voltage it's best to add a linear regulator to clean up the output voltage from the switching regulator.

What About Any Shields?

You are probably also using some shields that will also need to be converted to a custom schematic. If the shield is providing wireless functionality then in most cases you are better off using a surface-mounted module which solders directly on to your PCB.

There are two reasons for using a module for wireless functions. First, the PCB for a wireless radio can be quite complex to lay out correctly. Secondly, the use of pre-certified modules will simplify the process of getting your product certified.

Other functions, such as sensors and motor controllers, are best accomplished with a custom circuit design.

In future articles I will discuss in detail how to migrate various Arduino shields to a production circuit.

Design the Printed Circuit Board (PCB)

Now that the schematic design is completed, it's time to turn it into Printed Circuit Board (PCB). A schematic is simply an abstract technical diagram, but a PCB is how you turn your design into a real-world product.

Since the Arduino is open-source hardware the PCB layout design is available for reference. However, you will almost surely need to redesign the PCB for your specific product size requirements.

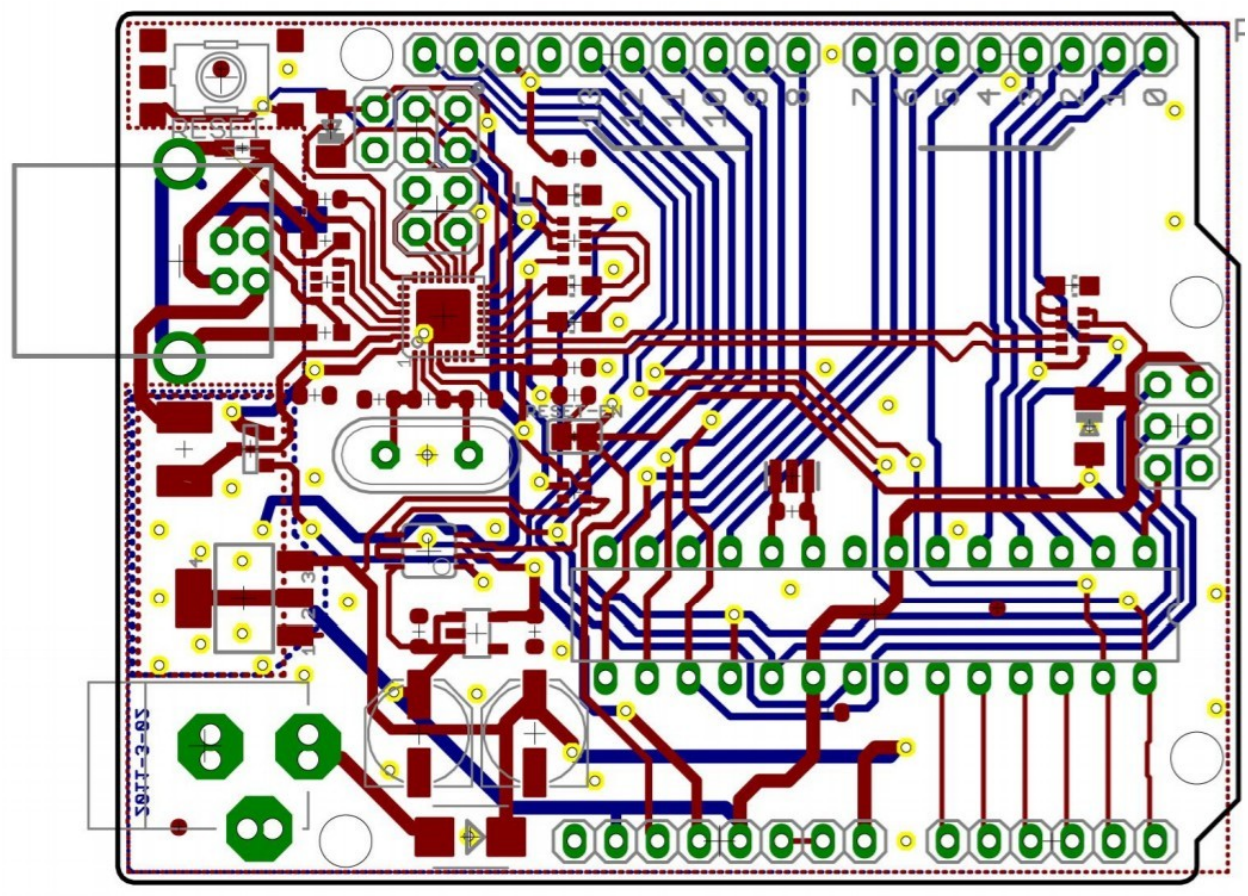


Figure 4 – PCB layout for the Arduino Uno

A microcontroller circuit with a clock speed of only 16 MHz, without wireless functionality, is a fairly simple PCB layout to design (assuming you know how to do PCB layout). Things become much more complicated once speeds approach hundreds of MHz, or especially GHz.

Be cautious about two things when laying out an Arduino Uno equivalent microcontroller circuit. First, the crystal and it's two load capacitors need to be laid out correctly and placed as close as possible to the microcontroller pins.

Secondly, carefully lay out any decoupling capacitors so they are as close as possible to the pin that is being decoupled. Be sure to always review the microcontroller datasheet for PCB layout guidelines.

Order PCB Prototypes

Once the PCB layout is completed it's now time to order the boards. However, before ordering any PCB prototypes you should really get an independent design review of the schematic and PCB layout.

Regardless of the designer's experience level, an independent design review reduces the likelihood that mistakes will make their way into your prototype. I always get other engineers to review my own designs and so should you. I offer both schematic and PCB layout design reviews.

In some cases you may have two different vendors make your boards. One vendor will produce the blank PCB's, and then another supplier will solder the components onto the board.

In other cases, a single vendor will perform both steps. For example, Seeed Studio's Fusion service can supply you with completely assembled boards at an incredibly affordable cost.

For your first prototype version I suggest ordering only 3-10 boards. This is because the first version will likely have various bugs that will need to be fixed. In most cases it's a waste of money to order a large quantity on the first version.

Once you've tested and debugged the first version, then increase the quantity for the second order.

Develop the Firmware/Software

One aspect of an Arduino that is different than your own, custom microcontroller circuit is how the programming is done. An Arduino is programmed via a USB port. This allows it to be programmed from any computer without the need for special hardware.

On the other hand, a custom microcontroller is usually programmed via a serial port protocol such as SPI, SWD, UART, or JTAG. In order to program a microcontroller using one of these serial programming protocols you'll need a special piece of hardware called an In-Circuit Serial Programmer (ICSP) or In-System Programmer (ISP).

You'll also sometimes see "programmer" substituted with "debugger" since this hardware device also allows you to see the inner workings of the microcontroller for debugging purposes.

These devices are called In-Circuit or In-System programmers/debuggers because the microcontroller can be programmed directly in the system without any need to remove the microcontroller.

The old method of programming required the microcontroller be removed from the circuit for programming, then re-inserted back into the circuit. This is a very inefficient method of programming a microcontroller during development.

The AVRISP from Atmel is an example of an in-system programmer for the Atmel AVR line of microcontrollers.

Figure 5 – The Atmel AVRISP mkII In-System Programmer (ISP)
This special programming hardware isn't required for an Arduino since it is essentially already embedded in the Arduino. As already discussed, the Arduino Uno incorporates a USB-to-UART converter to allow programming via a standard USB port.

Once you have the necessary programming hardware it's time to port over your Arduino sketch to native firmware code.

Just as with a custom microcontroller, an Arduino is programmed using the C language. However, programming is greatly simplified on the Arduino since it already contains a huge library of various functions.

For example, to setup a GPIO pin as an output on an Arduino, and then output a low logic level, you would use the following two functions:

pinMode(PinNumber, OUTPUT);

digitalWrite(PinNumber, LOW);

When you execute these two functions, the real work is performed by the library code behind these functions. For a custom microcontroller circuit the library code for these two functions must also be ported over to your microcontroller code (this will covered in more detail in a future article).

Finally, remember that you don't have to use the exact same microcontroller as your Arduino to simplify programming. Selecting a microcontroller from the same line of microcontrollers will still significantly simplify the transition from Arduino to production.

For example, porting your Arduino code over to any 8-bit AVR microcontroller will be considerably less complex than porting it over to a 32-bit microcontroller.

Test, Debug, and Repeat

It doesn't really matter how good you are at designing circuits. Unless your product is exceptionally simple, you are almost guaranteed to make at least one or two mistakes in your design.

Be sure to account for this fact in your planning. That being said, accurately planning for debugging is extremely challenging since you are inherently dealing with unknown and unexpected problems.

Conclusion

In this article we've looked at the simplest example of migrating from an Arduino prototype to a manufacturable product. However, the simplest method is rarely the best method.

The Atmel AVR microcontrollers used in most Arduino kits are a great choice for learning about microcontrollers, but they are not necessarily the best choice in a production product.

There are many other microcontrollers available that will give you significantly more *bang for your buck*. In future articles I will discuss how to migrate from a 8-bit Arduino to a more powerful, and typically lower cost, 32-bit microcontroller.

You may also be interested in these other articles:

– Tutorial: How to Design Your Own Custom STM32 Microcontroller Board (with Video)
– Microcontroller or Microprocessor: Which is Right for Your New Product?
– Introduction to Microcontrollers

15 comments

←Previous post Next post→