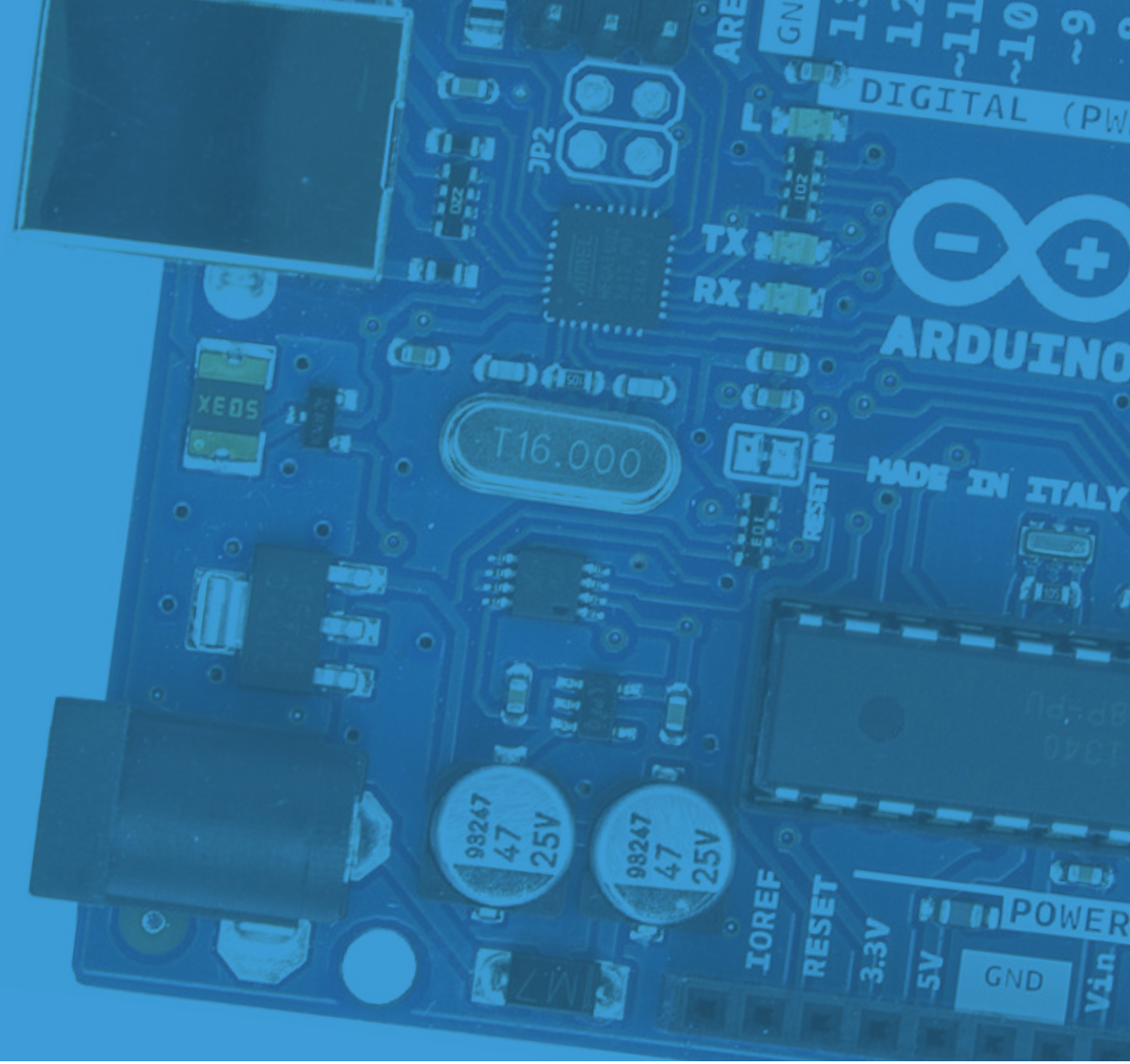
How to Turn Your Arduino Project into a Sellable Product



This guide is written by John Teel and brought to you in cooperation by:

PREDICTABLE DESIGNS

hackster.io

Table of Contents

Introduction 3
Step #1 - Preliminary Design 4
Critical Components Selection 5
Common Microcontroller Peripherals 7
Notable Microcontroller Cores 8
Estimate the Manufacturing Cost 9
Step #2 - Design the Schematic Circuit 10
Review of Arduino Uno Schematic Diagram 13
What About Any Shields? 16
Step #3 - Design Printed Circuit Board (PCB) 17
Step #4 - Order PCB Prototypes 19
Step #5 - Develop the Firmware/Software 21
Step #6 - Test, Debug, and Repeat 24
Step #7 - Electrical Certifications 25
Conclusion 27

Introduction

Creating a prototype based on an Arduino 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 to the masses.

But why can't you simply manufacture your product with an Arduino inside? The good news is you can, but most likely you shouldn't.

There are two main reasons this is not usually a good idea. First of all, the Arduino is large so for many products embedding an Arduino will make the product much too big to sell as a finished product.

Secondly, and most importantly, you won't make any money unless your product is really expensive with high profit margins. Very few products have a retail price high enough to justify using an Arduino for production. But, if you are lucky enough to have such a product then using an Arduino for production can be a low cost way to get your product to market very quickly.

Arduinos may seem cheap when buying just one for a DIY project, but they are very expensive once you move on to high-volume production. You can design your own microcontroller circuit to replace an Arduino for only a few dollars, and that is the focus of this guide.

Step #1
Preliminary Design

Before you jump head first into spending big money on full development I highly recommend you first look at the big picture. This basically means you select all the critical components, identify any development or manufacturing risks, and estimate the cost to develop, prototype, scale, and manufacture the product.

Most entrepreneurs make the grave mistake of jumping right into spending big money (and/or tons of time) on full development without any idea of the costs, steps, and complexities that lie ahead. No big tech company would ever do this. They always analyze the big picture before spending the big money, and so should you.

Critical Components Selection

The first step to design the electronics is to select all of the critical components. This includes the various microchips (i.e. integrated circuits), sensors, displays, connectors, and other electronic devices needed based upon the desired functions and target retail price of your product.

Most products require a master microcontroller with various components (displays, sensors, memory, etc.) interfacing to the microcontroller via multiple serial port protocols (I2C, SPI, UART, I2S, USB, etc.).

Should you select the microcontroller first or last? I recommend that you begin by creating a detailed system block diagram.

A system block diagram is invaluable for this early planning. It can tell you how many input and output (I/O) pins and serial communication ports are needed for the project. Once you have that information then you can select the best microcontroller.

I recommend searching for microcontrollers via an electronics component distributor such as Newark.

This will allow you to narrow down your search to only microcontrollers that are actively available. It also allows you to quickly compare prices without limiting your choices to any particular manufacturer.

One word of caution when selecting components. Take the time to do your due diligence to ensure that you won't run into supply issues with any of the components in the future. First of all, make sure you only select components that are currently in active production. Secondly, confirm that all critical components are available from multiple distributors. Finally, contact the component manufacturers to find out the forecasted end-of-life, or use a paid service such as SiliconExpert.

The most easiest strategy to transition from an Arduino prototype to a sellable product is to use the same microcontroller as the Arduino you used for your project. Most Arduinos are based on simple 8-bit AVR microcontrollers from Microchip.

Although there may be higher performance and lower cost microcontrollers available, the simplest option is

to just use the same microcontroller as your Arduino. 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 as-is, or at least serve as a reference starting point.

That being said, the easiest solution is rarely ever the best solution. I almost never use 8-bit microcontrollers. Instead my standard go-to microcontrollers are 32-bit microcontrollers based on Arm Cortex-M processor cores.

For example, there are 32-bit Arm 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!

My specific preference is the STM32 line from ST Microelectronics. The STM32 line is huge. It includes everything from relatively simple microcontrollers all the way up to powerful microcontrollers running at hundreds of MHz. The upper echelons of the STM32 line approach the performance of some microprocessors.

Common Microcontroller Peripherals

In order to select the best microcontroller you need to first understand the peripherals and features commonly included with microcontrollers.

Memory: Most microcontrollers available today include built-in FLASH and RAM memory. FLASH is non-volatile memory used for program storage, and RAM is volatile memory used for temporary storage. Some microcontrollers also include EEPROM memory for permanently storing data, although usually a separate EEPROM chip is required.

Digital General Purpose Input and Output (GPIO): These are logic level pins used for input and output. Generally they can sink or source up to a few tens of milliamps and can be configured as open drain or push pull.

Analog input: Most microcontrollers have the ability to precisely read an analog voltage. Analog signals are sampled by the microcontroller via an Analog to Digital Converter (ADC).

Analog output: Analog signals can be generated by the microcontroller via a Digital to Analog Converter (DAC) or a Pulse Width Modulation (PWM) generator. Not all microcontrollers include a DAC but they do offer PWM capabilities.

In Circuit Programming (ISP): ISP allows you to program a microcontroller while it is installed in the application circuit, instead of having to remove it for programming. The two most common ISP protocols are JTAG and SWD.

Wireless: If your product needs wireless capabilities then there are specialized microcontrollers available that offer Bluetooth, WiFi, ZigBee, and other wireless standards.

Universal Asynchronous Receiver Transmitter (UART) is a serial port that transmits digital words, typically of length 7 to 8 bits, between a start bit and an optional parity bit and one or two stop bits. A UART is commonly used along with other standards such as RS-232 or RS-485.

UART is the oldest type of serial communication. UART is an asynchronous protocol which means there is no clock signal. Many microcontrollers also include a synchronous version of a UART called a USART.

Serial Peripheral Interface (SPI): SPI is used for short distance serial communication between microcontroller and peripherals. SPI is a synchronous protocol which means it includes a clock signal for timing. SPI is a 4 wire standard that includes data in, data out, clock, and chip select signals.

Inter Integrated circuit (I2C): I2C also written as I2C is a 2-wire serial bus used for communications between the microcontroller and other chips on the board. Like SPI, I2C is also a synchronous protocol.

However, unlike SPI, I2C uses a single line for both data in and data out. Also instead of a chip select signal, I2C uses a unique address for each peripheral. I2C has the advantage of only using 2 wires, but it's slower than SPI.

Universal Serial Bus (USB) is a standard that is familiar to most people. USB is one of the fastest serial communication protocols. It is generally used for connecting up peripherals that require large amounts of data transfer.

Controller Area Network (CAN) is a serial communication standard developed specifically for use in automotive applications.

Notable Microcontroller Cores

There are several microcontroller cores that have some notoriety and are worth describing. Below are four of the most common ones:

Arm Cortex-M
The 32-bit Arm Cortex-M series is one of the most commonly used microcontroller cores used today. Arm doesn't actually make and sell microcontrollers, instead they license their architecture to other chip makers.

Many companies offer Cortex-M microcontrollers including ST Microelectronics, Freescale Semiconductor, Silicon Labs, Texas Instruments, and Microchip.

Cortex-M series microcontrollers are my favorite choice for products that will be brought to market. They are low cost, powerful, and widely used. Cortex-M microcontrollers are the most popular microcontroller in use for commercial products.

8051
The 8-bit 8051 microcontroller was developed by Intel way back in 1980. It's the oldest microcontroller core commonly still used today. The 8051 is currently available in enhanced modern versions sold by at least 8 different semiconductor manufactures. For example, the popular Bluetooth Low-Energy chip from CSR (CSR101x) uses an 8051 core.

AVR
The microcontroller line known as AVR from Microchip (originally from Atmel) is best known for being the brains in most versions of the Arduino. So for many makers it's an easy transition from an Arduino to an AVR microcontroller. However, I've found that you can usually get one of the other cores with similar, or better, performance for several dollars cheaper.

The large majority of Arduino models are based on an 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.

PIC
PIC is another family of microcontrollers from Microchip. They are very popular and come in a wide array of options. The PIC microcontroller line includes 8-bit, 16-bit and 32-bit versions. The number of pins, package styles, and selection of on chip peripherals are offered in an almost endless array of combinations.

Estimate the Manufacturing Cost

I highly recommend that you estimate the production cost for your product before you design the full schematic. Most entrepreneurs and developers skip this step and proceed right to the schematic design. That's a mistake!

It's critical to know as soon as possible how much it will cost to manufacture your product. You need this number in order to determine the best sales price, the cost of inventory, and most importantly how much profit you can make.

Once you've selected all of the major components then you should have enough information to accurately estimate the production cost for your product (Cost of Goods Sold – COGS).

Don't make profit an afterthought. Does Apple start developing a new product before knowing how much profit they can make? Of course not, and neither should you.

The total COGS consists of much more than just the electronic component costs. For more information see my article that describes in great detail the manufacturing cost for a new electronic hardware product.

Step #2

Design the Schematic Circuit

For the rest of this introductory guide we'll be focusing primarily on 8-bit AVR microcontrollers since that is the easiest transition from an Arduino.

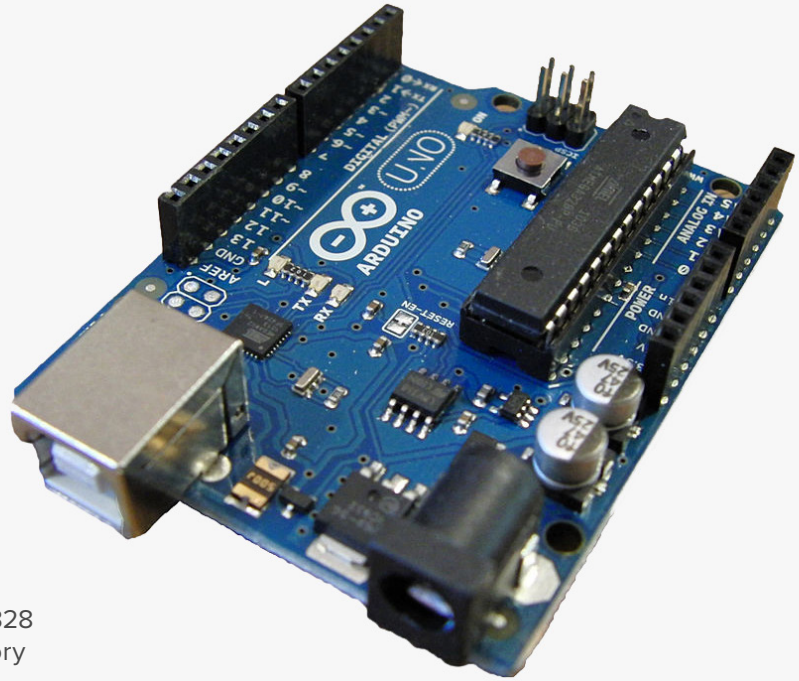


Figure 1 - Arduino Uno uses an ATmega328 microcontroller with 32kB of Flash memory

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 much smaller 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 / 7-12 V	32 MHz	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	8 MHz	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 / 5 V	48 MHz	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.

Review of Arduino Uno Schematic Diagram

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 take a look at the schematic diagram for the Uno.

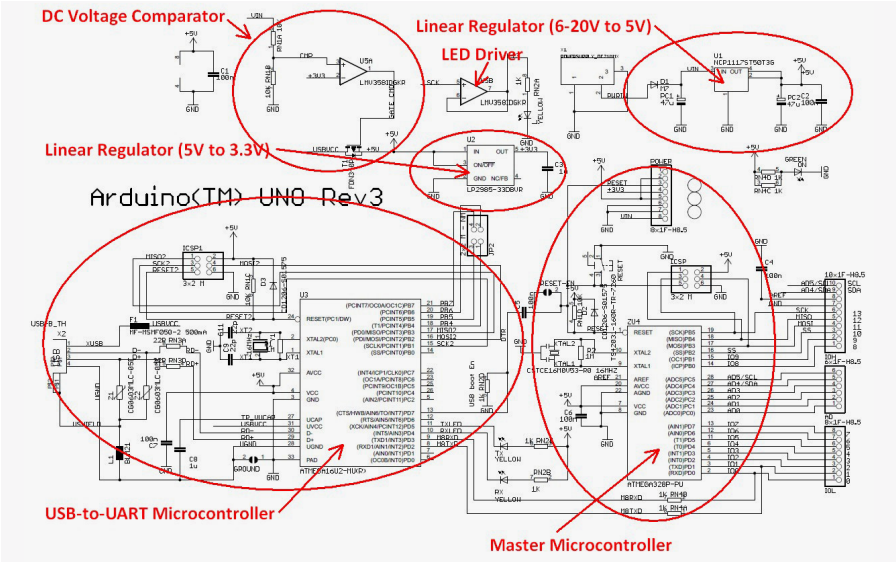


Figure 2 - Marked up schematic diagram for an Arduino Uno.

First of all, 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.

Unlike an Arduino, with a custom microcontroller circuit you no longer need a USB port for programming purposes (I discuss this in more detail later). 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.

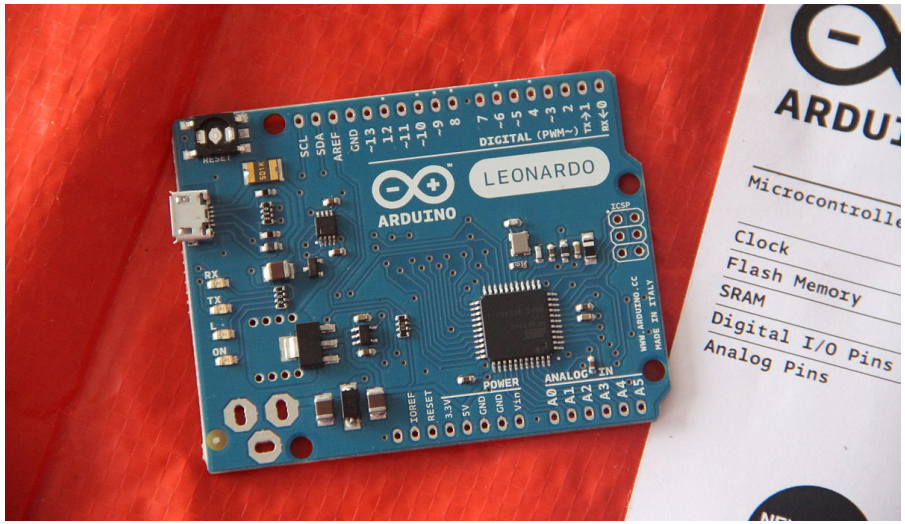


Figure 3 - Arduino Leonardo uses an ATmega32U4 microcontroller with a built-in USB port

That being said, the ATmega32U4 is quite pricey at around $3.47 @ 100 pieces. Whereas, ST Microelectronics offers an Arm Cortex-M 32-bit microcontroller with USB functionality for only $1.92 @ 100 pieces.

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. Depending on the complexity of the function provided by the shield you'll either want to replace it with a custom circuit or a module solution.
If the shield is providing wireless functionality then in most cases you are better off using a surface-mounted module which will solder directly on the main 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.

Note that many wireless solutions (whether a module or a chip) include an embedded microcontroller that can also serve as the master microcontroller for your product. In such as case there is no need for a separate microcontroller unless your product requires a more advanced microcontroller than embedded in the wireless module/chip.

For less complex functions like sensors, relays, and motor controllers it's usually best to implement them as a custom circuit on your own PCB.

Step #3

Design Printed Circuit Board (PCB)

Now that the schematic design is complete, 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.

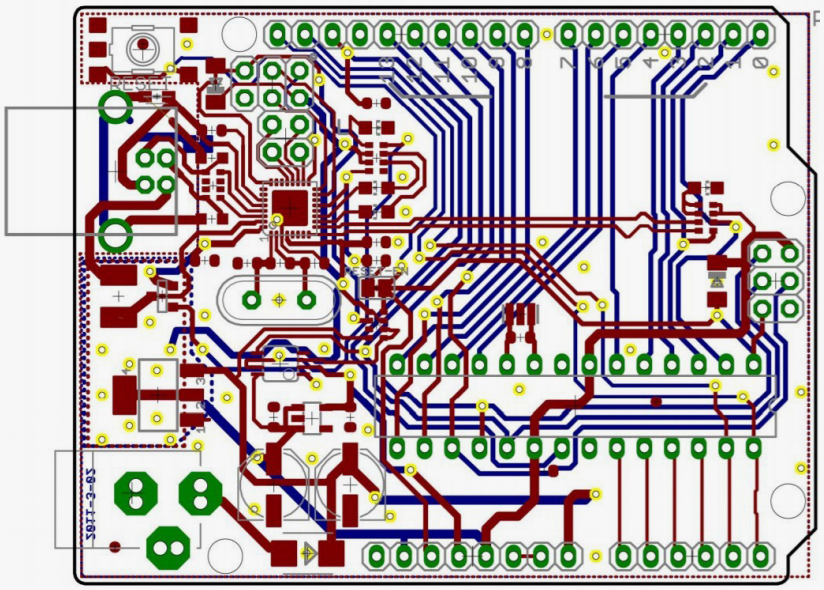


Figure 4 - PCB layout for the Arduino Uno

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.

Some general PCB layout tips include avoiding 90 degree bends in signal traces and making sure any traces carrying significant current are sized properly. If leadless packages are used be sure to also include test points for debug purposes.

Step #4

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.

Once you are finally ready to order boards you will need to generate Gerber files for the PCB layout. There are countless PCB design software packages and each has its own proprietary file format. Gerber files, on the other hand, are an industry standard supported by all PCB design tools. Gerber files will be used to prototype your boards as well as for production.

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 depending on your confidence level that all major issues have been fixed.

Step #5

Develop the Firmware/Software

As I eluded to earlier, one aspect of an Arduino that is different than a 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 is an example of an in-system programmer for the AVR line of microcontrollers. Unfortunately, most microcontrollers require their own custom programmer so be sure the one you purchase works with your microcontroller.

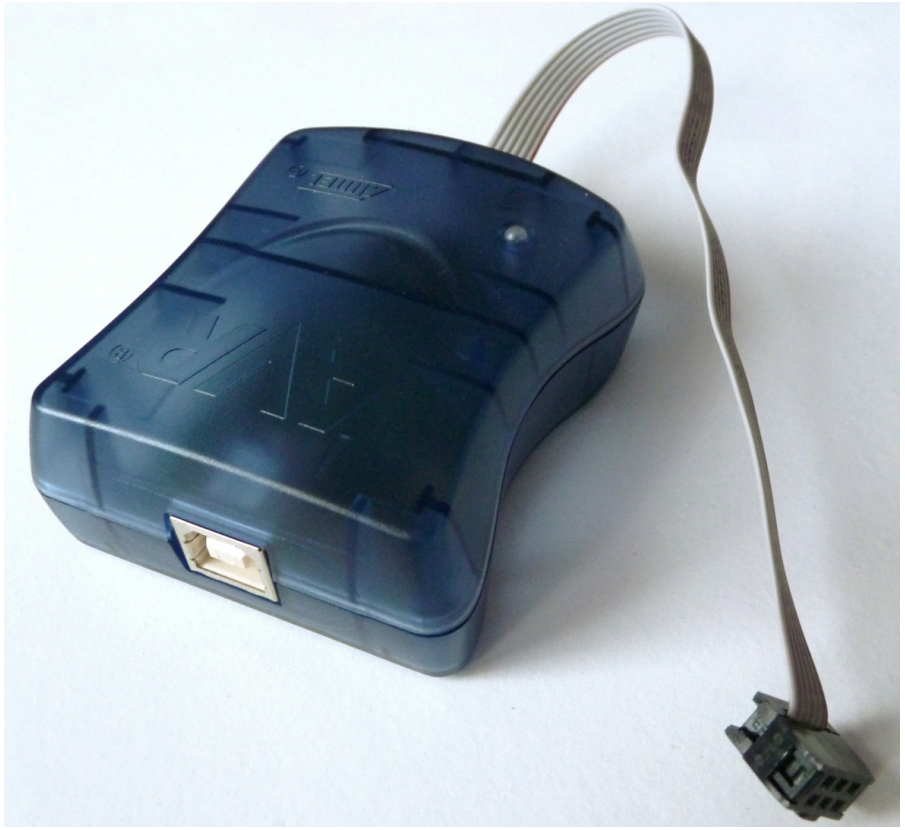


Figure 5 - The AVRISP mkII In-System Programmer (ISP)

.

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.

Step #6

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, and likely many more.

So be sure to account for this fact in your forecast planning. That being said, accurately forecasting debug time is extremely challenging since you are inherently dealing with unknown and unexpected problems. This step almost always consists of both hardware and firmware debug.

Step #7

Electrical Certifications

In order to sell a new electronic product in most countries there are several types of certification required. The exact certifications needed depend on the country/region where the product will be sold.

I'll warn you that obtaining certifications isn't cheap and most products will cost at least several thousand dollars to certify. Below is a quick overview of the certifications required in the USA, Canada, and Europe.

FCC certification is required for all electrical products sold in the USA. Products that don't purposefully radiate electromagnetic energy (i.e. no wireless functions) are classified as non-radiators.

On the other hand, wireless products purposefully transmit electromagnetic energy and are classified as intentional radiators. It is much more expensive to obtain FCC certification for an intentional radiator. Fortunately, there are ways to reduce this cost such as by using pre-certified wireless modules.

UL (Underwriters Laboratories) or CSA (Canadian Standards Association) certification is required for any electrical sold in the USA and/or Canada that plugs into an AC electrical outlet.

Products running on only batteries with no recharging capabilities do not require UL/CSA certification. However, many retail chains and/or product liability insurance companies will require UL/CSA certification for any electronic product.

CE (Conformité Européene) certification is required for products sold in the European Union (EU). It is similar to the FCC and UL certifications required in the USA.

RoHS (Restriction of Hazardous Substances) certification is required for electrical products sold in the European Union (EU). It certifies the electronics are free of lead.

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 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. The 32-bit Arm Cortex-M line of microcontrollers are the most common choice for production. They are powerful, low-cost and widely available from multiple chip makers.

Getting your product to the point of having a production quality prototype is a huge accomplishment. However, there is still considerable work required to scale a product from a prototype to mass producing thousands or millions of units.

Before you jump into developing your product you should first look at the big picture. Doing so will give you insight into all of the steps and costs that lie ahead in your path to market domination.