Technical Article

Understanding Arduino UNO Hardware Design

July 01, 2016 by Yahya Tawil

This article explains how Arduino works from an electronic design perspective. It also explains the role of every part of the Arduino.

This article explains how Arduino works from an electronic design perspective.

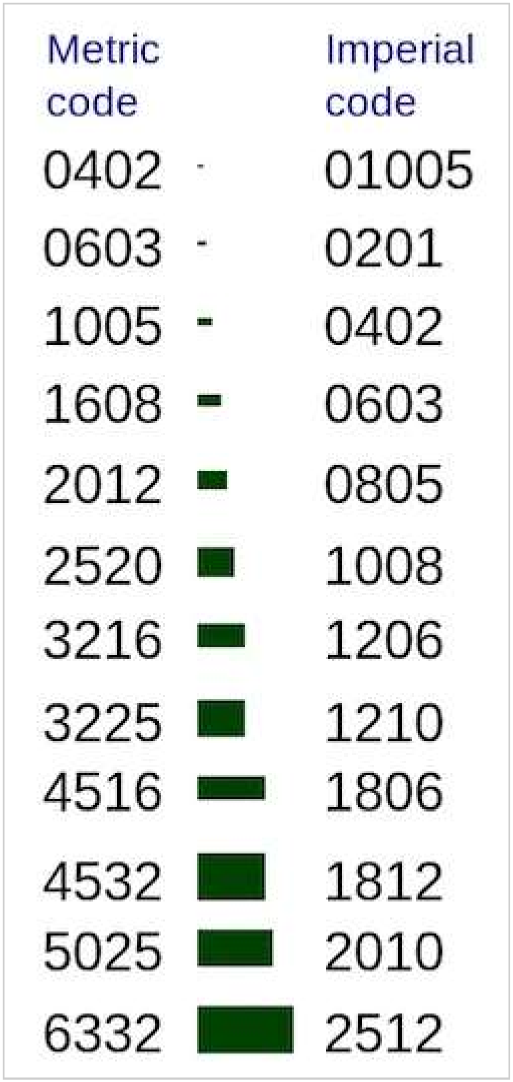
Most articles explain the software of Arduinos. However, understanding hardware design helps you to make the next step in the Arduino journey. A good grasp of the electronic design of your Arduino hardware will help you learn how to embed an Arduino in the design of a final product, including what to keep and what to omit from your original design.

Components Overview

The PCB design of the Arduino UNO uses SMD (Surface Mount Device) components. I entered the SMD world years ago when I dug into Arduino PCB design while I was a part of a team redesigning a DIY clone for Arduino UNO.

Integrated circuits use standardized packages, and there are families for packages.

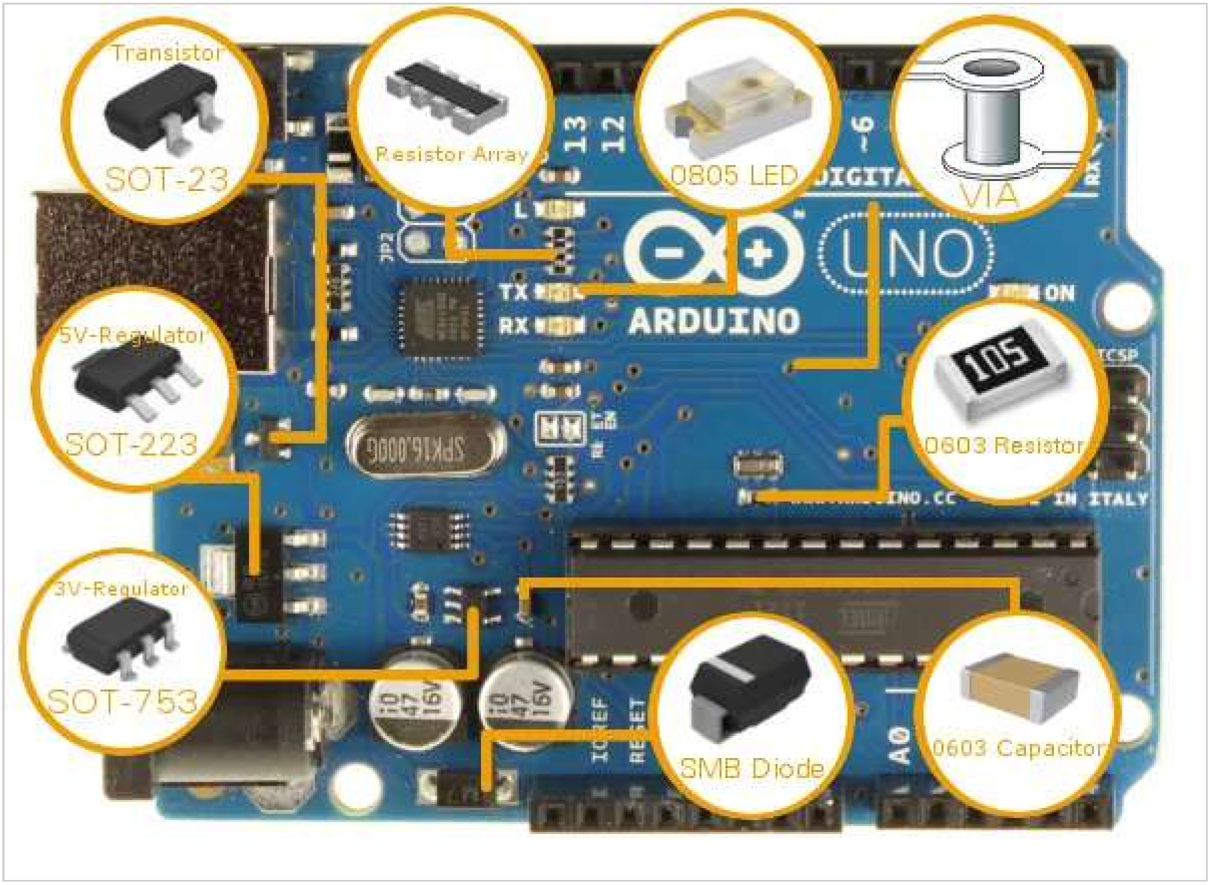
The dimensions of many SMD resistors, capacitors, and LEDs are indicated by package codes such as the following:



Metric code	Imperial code
0402	01005
0603	0201
1005	0402
1608	0603
2012	0805
2520	1008
3216	1206
3225	1210
4516	1806
4532	1812
5025	2010
6332	2512

SMD package code for discrete components such as resistors, capacitors, and inductors. Image courtesy of [Wikimedia](#).

Most packages are generic and can be used for different parts with different functionality. The SOT-223 package, for example, can contain a transistor or a regulator.



In the table below, you can see a list of some components in the Arduino UNO with their respective package:

Part	Package
NCP1117ST50T3G 5V regulator	SOT223
LP2985-33DBVR 3.3V regulator	SOT753/SOT23-5
M7 diode	SMB
LMV358IDGKR dual channel amplifier	MSOP08
FDN340P P-channel MOSFET transistor	SOT23
ATmega16U2-MU	MLF32

Arduino UNO System Overview

Before we can understand the UNO's hardware, we must have a general overview of the system first.

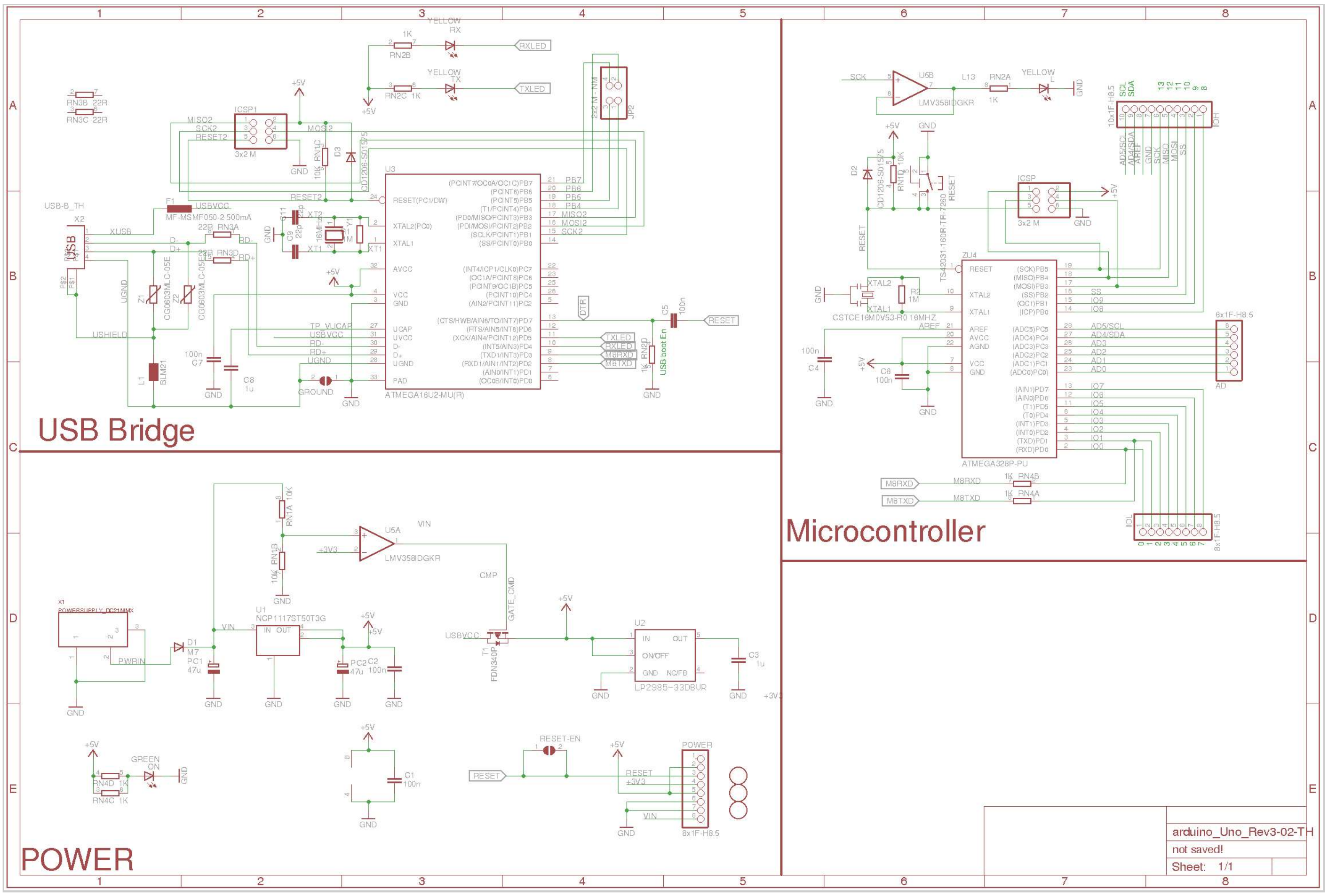
After your code is compiled using Arduino IDE, it should be uploaded to the main microcontroller of the Arduino UNO using a USB connection. Because the main microcontroller doesn’t have a USB transceiver, you need a bridge to convert signals between the serial interface (UART interface) of the microcontroller and the host USB signals.

The bridge in the latest revision is the ATmega16U2, which has a USB transceiver and also a serial interface (UART interface).

To power your Arduino board, you can use the USB as a power source. Another option is to use a DC jack. You may ask, “if I connect both a DC adapter and the USB, which will be the power source?” The answer will be discussed in the “Power Part” section from this article.

To reset your board, you should use a push button in the board. Another source of reset should be every time you open the serial monitor from Arduino IDE.

I redistributed the original Arduino UNO schematic to be more readable below. I advise you to download it and open the PCB and schematic using Eagle CAD while you are reading this article.



Redistributed version of the original Arduino schematic. Click to enlarge.

[Arduino_UNO_R3.zip]

The Microcontroller

It is important to understand that the Arduino board includes a microcontroller, and this microcontroller is what executes the instructions in your program. If you know this, you won't use the common nonsense phrase "Arduino is a microcontroller" ever again.

The ATmega328 microcontroller is the MCU used in Arduino UNO R3 as a main controller. ATmega328 is an MCU from the AVR family; it is an 8-bit device, which means that its data-bus architecture and internal registers are designed to handle 8 parallel data signals.

ATmega328 has three types of memory:

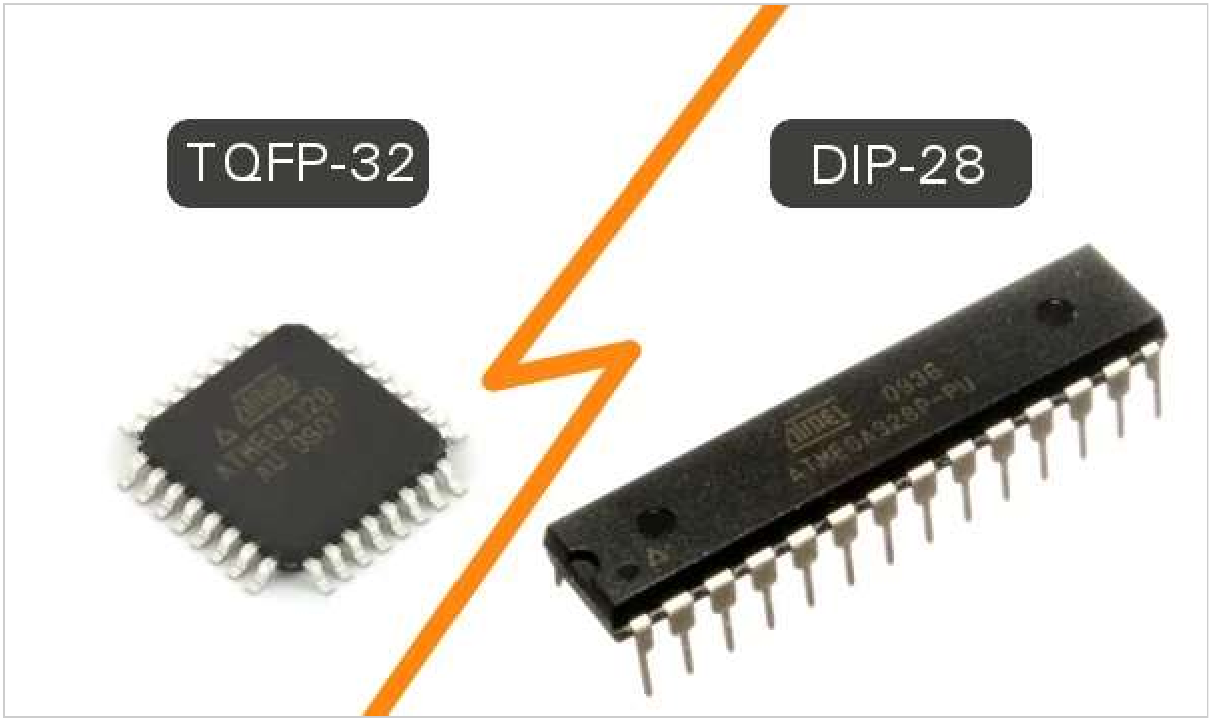
- **Flash memory:** 32KB nonvolatile memory. This is used for storing application, which explains why you don't need to upload your application every time you unplug arduino from its power source.
- **SRAM memory:** 2KB volatile memory. This is used for storing variables used by the application while it's running.
- **EEPROM memory:** 1KB nonvolatile memory. This can be used to store data that must be available even after the board is powered down and then powered up again.

Let us briefly go over some of this MCU's specs:

Packages:

This MCU is a DIP-28 package, which means that it has 28 pins in the dual in-line package. These pins include power and I/O pins. Most of the pins are multifunctional, which means that the same pin can be used in different modes based on how you configure it in the software. This reduces the necessary pin count, because the microcontroller does not require a separate pin for every function. It can also make your design more flexible, because one I/O connection can provide multiple types of functionality.

Other packages of ATmega328 are available like TQFP-32 SMD package (Surface Mount Device).



Two different packages of the ATmega328. Images courtesy of Sparkfun and Wikimedia.

Power:

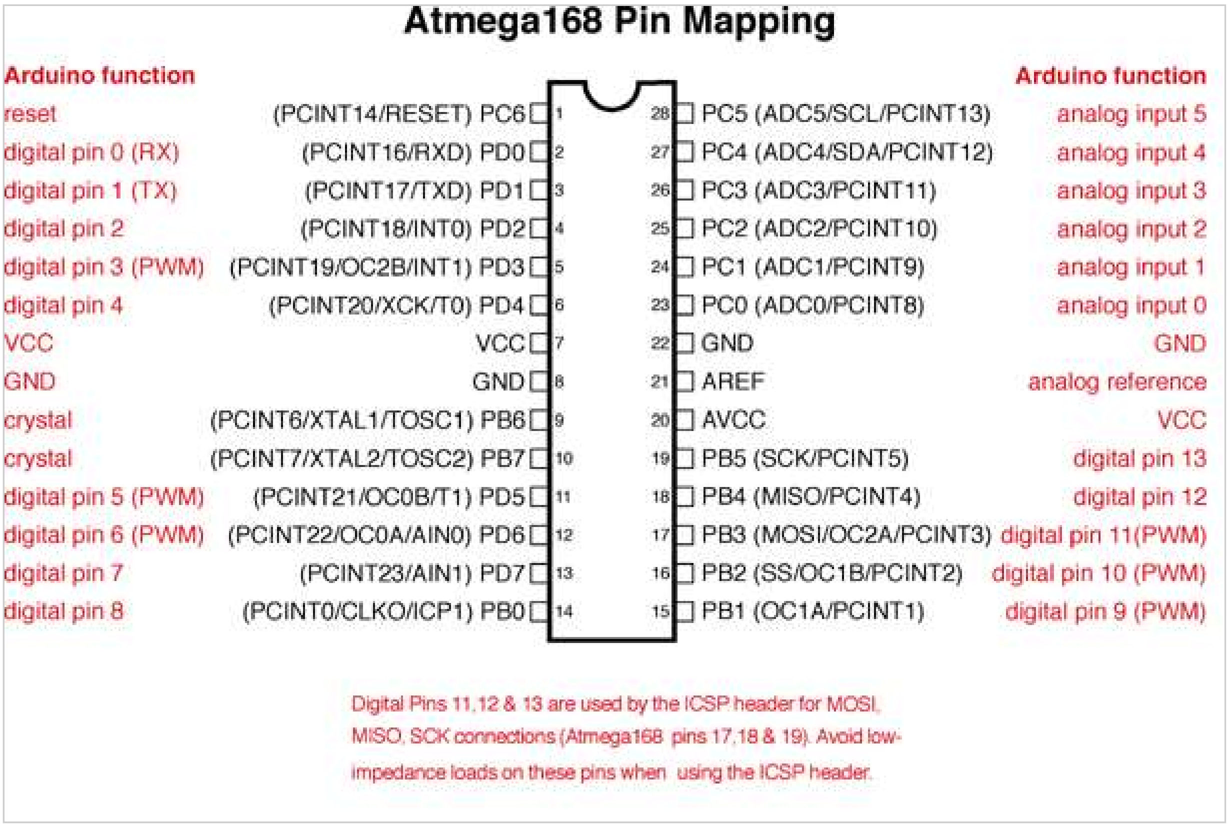
The MCU accepts supply voltages from 1.8 to 5.5 V. However, there are restrictions on the operating frequency; for example, if you want to use the maximum clock frequency (20 MHz), you need a supply voltage of at least 4.5 V.

Digital I/O:

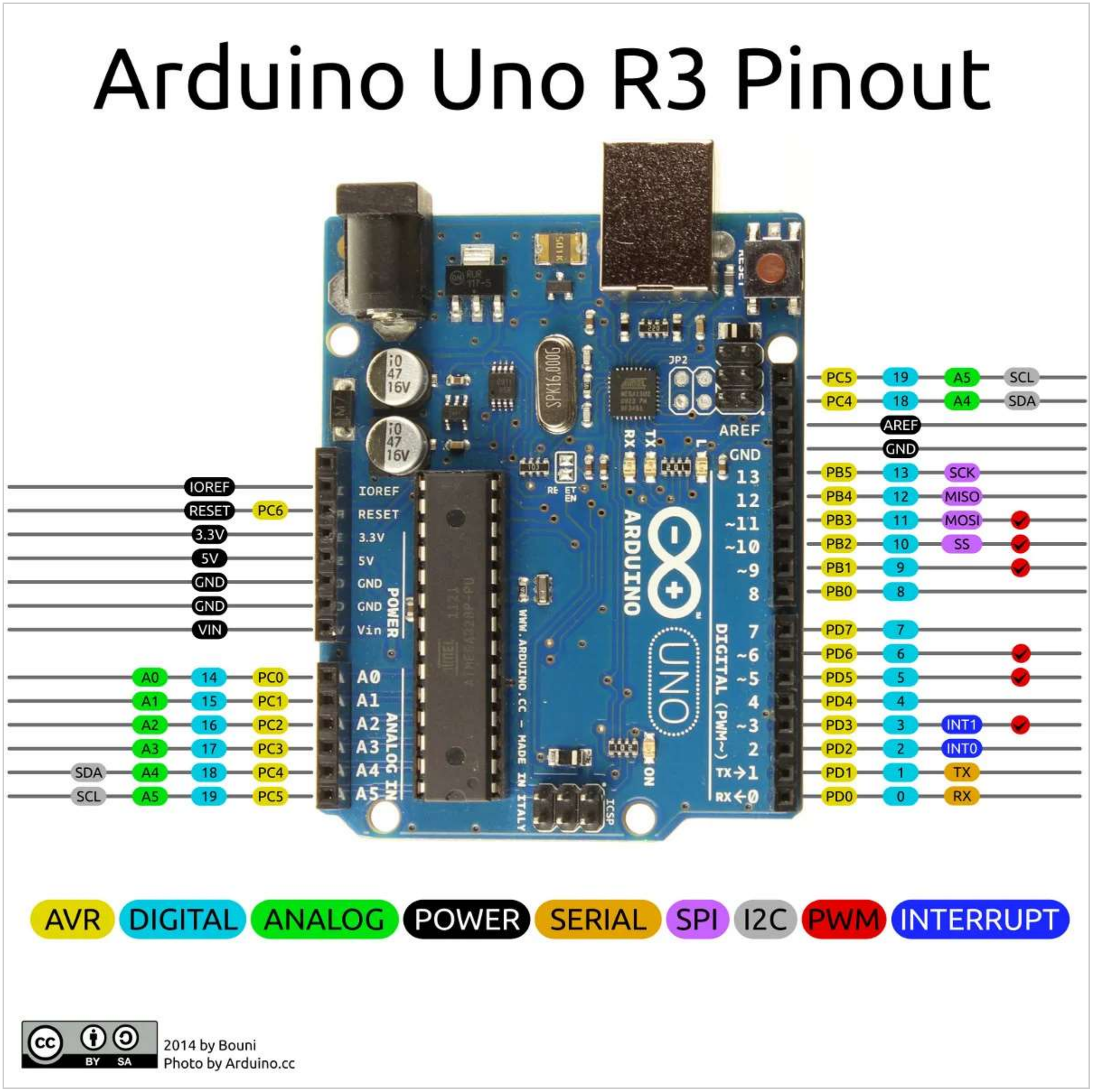
This MCU has three ports: PORTC, PORTB, and PORTD. All pins of these ports can be used for general-purpose digital I/O or for the alternate functions indicated in the pinout below. For example, PORTC pin0 to pin5 can be ADC inputs instead of digital I/O.

There are also some pins that can be configured as PWM output. These pins are marked with “~” on the Arduino board.

Note: The ATmega168 is almost identical to the ATmega328 and they are pin compatible. The difference is that the ATmega328 has more memory—32KB flash, 1KB EEPROM, and 2KB RAM compared to the ATmega168's 16KB flash, 512 bytes EEPROM, and 1KB RAM.


Atmega168 Pin Mapping
Arduino function
Arduino function
reset (PCINT14/RESET) PC6 1
digital pin 0 (RX) (PCINT16/RXD) PD0 2
digital pin 1 (TX) (PCINT17/TXD) PD1 3
digital pin 2 (PCINT18/INT0) PD2 4
digital pin 3 (PWM) (PCINT19/OC2B/INT1) PD3 5
digital pin 4 (PCINT20/XCK/T0) PD4 6
VCC VCC 7
GND GND 8
crystal (PCINT6/XTAL1/TOSC1) PB6 9
crystal (PCINT7/XTAL2/TOSC2) PB7 10
digital pin 5 (PWM) (PCINT21/OC0B/T1) PD5 11
digital pin 6 (PWM) (PCINT22/OC0A/AIN0) PD6 12
digital pin 7 (PCINT23/AIN1) PD7 13
digital pin 8 (PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13) analog input 5
27 PC4 (ADC4/SDA/PCINT12) analog input 4
26 PC3 (ADC3/PCINT11) analog input 3
25 PC2 (ADC2/PCINT10) analog input 2
24 PC1 (ADC1/PCINT9) analog input 1
23 PC0 (ADC0/PCINT8) analog input 0
22 GND GND
21 AREF analog reference
20 AVCC VCC
19 PB5 (SCK/PCINT5) digital pin 13
18 PB4 (MISO/PCINT4) digital pin 12
17 PB3 (MOSI/OC2A/PCINT3) digital pin 11(PWM)
16 PB2 (SS/OC1B/PCINT2) digital pin 10 (PWM)
15 PB1 (OC1A/PCINT1) digital pin 9 (PWM)
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

ATmega168 pinout with Arduino labels; the ATmega168 and ATmega328 are pin compatible. Image courtesy of [Arduino](#).



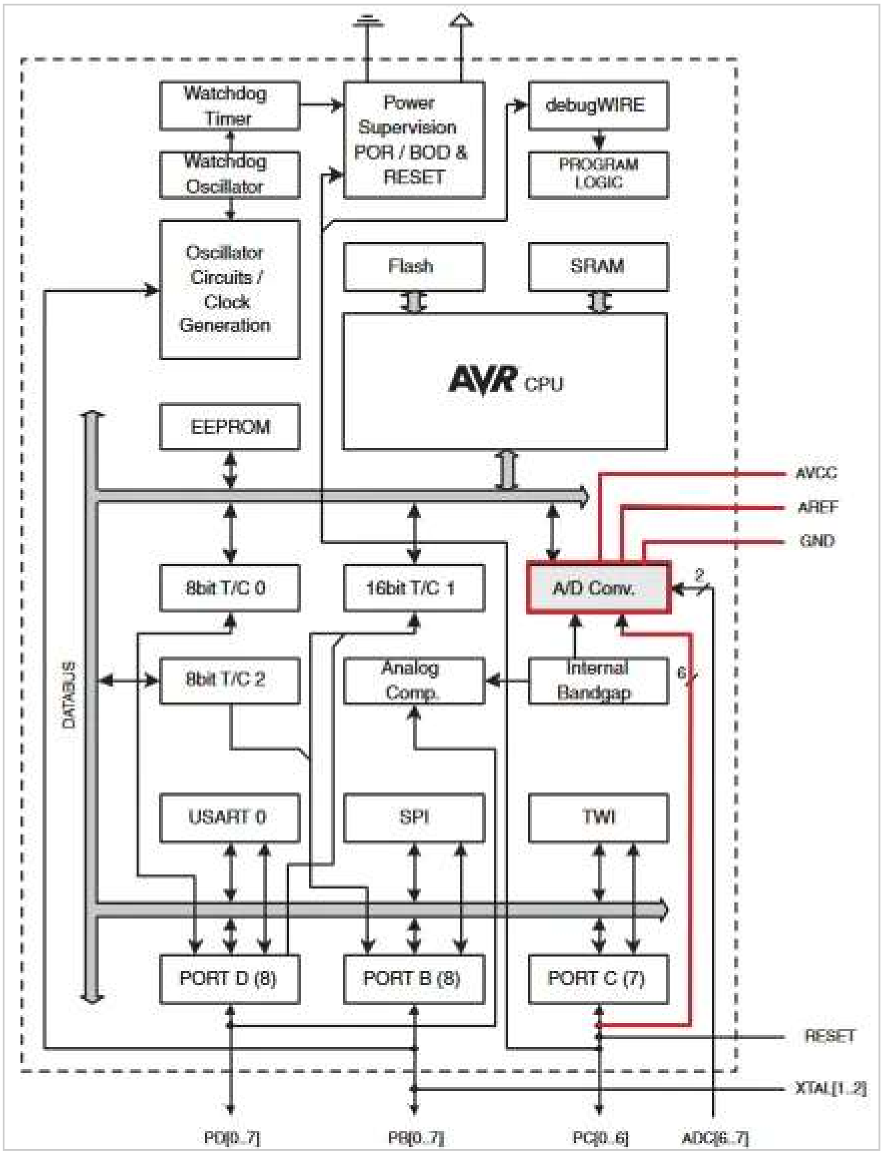
Arduino UNO R3 pinout. Image courtesy of GitHub.

ADC Inputs:

This MCU has six channels—PORTC0 to PORTC5—with 10-bit resolution A/D converter. These pins are connected to the analog header on the Arduino board.

Scroll to continue with content

One common mistake is to think of analog input as dedicated input for A/D function only, as the header in the board states ”Analog”. The reality is that you can use them as digital I/O or A/D.



ATmega328 block diagram.

As shown in the diagram above (via the red traces), the pins related to the A/D unit are:

- AVCC: The power pin for the A/D unit.
- AREF: The input pin used optionally if you want to use an external voltage reference for ADC rather than the internal Vref. You can configure that using an internal register.

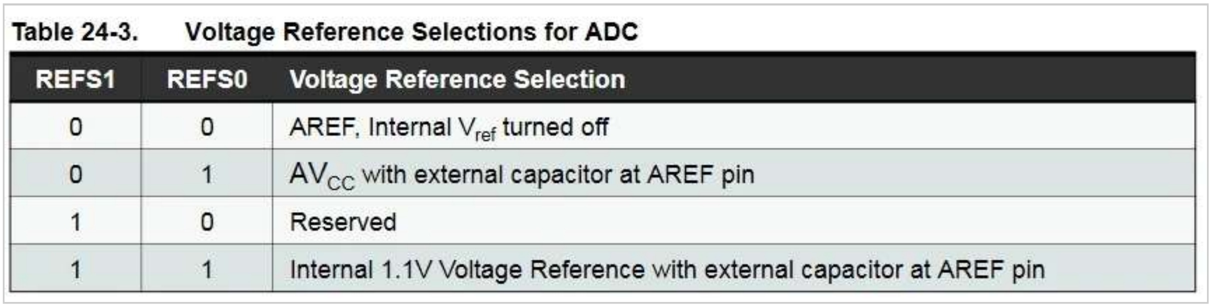


Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Internal register settings for selecting the Vref source.

UART Peripheral:

A UART (Universal Asynchronous Receiver/Transmitter) is a serial interface. The ATmega328 has only one UART module.

The pins (RX, TX) of the UART are connected to a USB-to-UART converter circuit and also connected to pin0 and pin1 in the digital header. You must avoid using the UART if you’re already using it to send/receive data over USB.

SPI Peripheral:

The SPI (Serial Peripheral Interface) is another serial interface. The ATmega328 has only one SPI module.

Besides using it as a serial interface, it can also be used to program the MCU using a standalone programmer. You can reach the SPI's pins from the header next to the MCU in the Arduino UNO board or from the digital header as below:
11<->MOSI
12<->MISO
13<->SCK

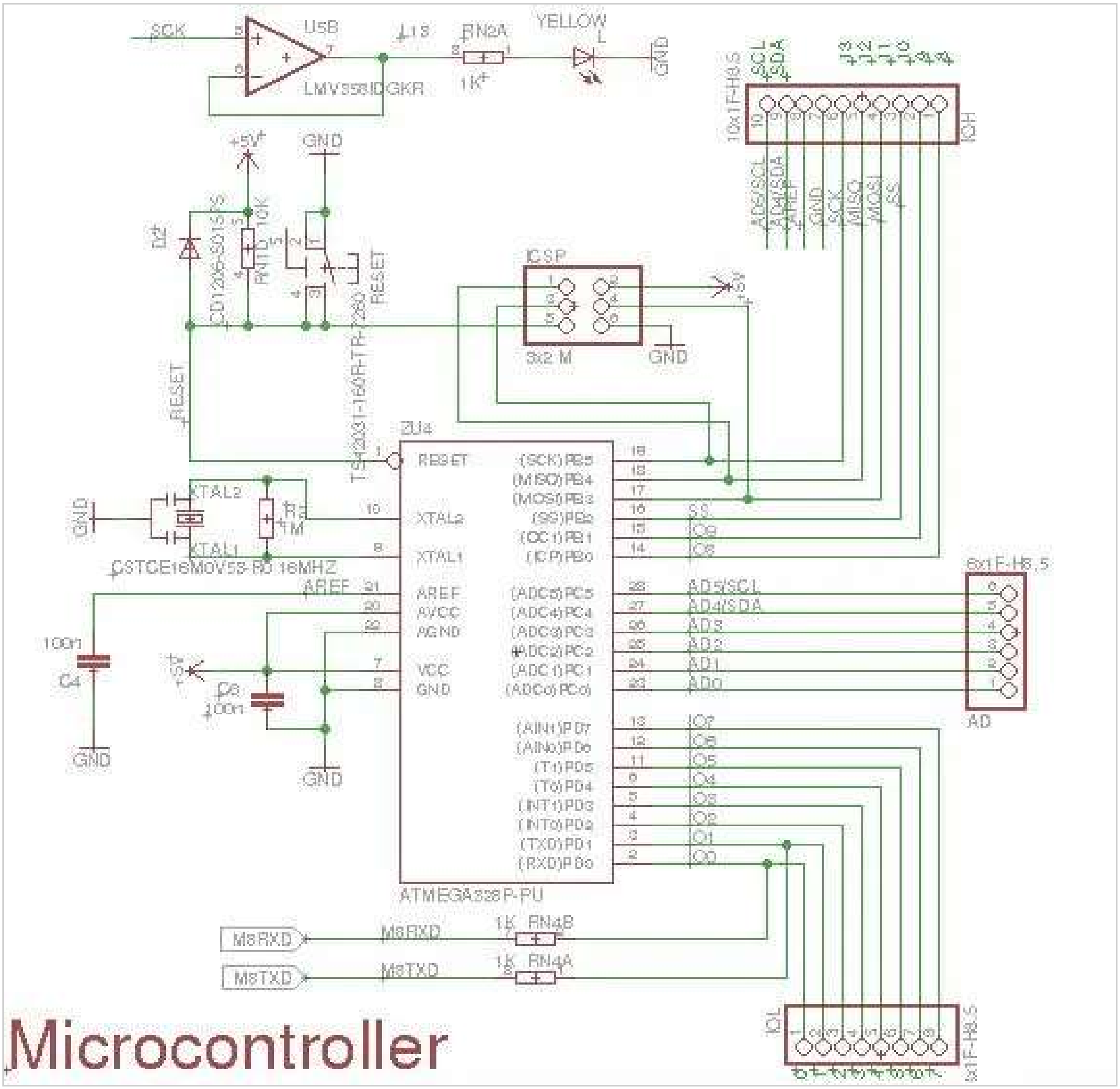
TWI:

The I^{2}C or Two Wire Interface is an interface consisting of only two wires, serial data, and a serial clock: SDA, SCL.

You can reach these pins from the last two pins in the digital header or pin4 and pin5 in the analog header.

Other Functionality:

Other functionality is included in the MCU, such as that offered by the timer/counter modules. You may not be aware of the functions that you don't use in your code. You can refer to the datasheet for more information.



Arduino UNO R3 MCU part.

Returning to the electronic design, the microcontroller section has the following:

- **ATmega328-PU:** The MCU we just talked about.
- **IOL and IOH (Digital) Headers:** These headers are the digital header for pins 0 to 13 in addition to GND, AREF, SDA, and SCL. Note that RX and TX from the USB bridge are connected with pin0 and pin1.
- **AD Header:** The analog pins header.
- **16 MHz Ceramic Resonator (CSTCE16M0V53-R0):** Connected with XTAL2 and XTAL1 from the MCU.
- **Reset Pin:** This is pulled up with a 10K resistor to help prevent spurious resets in noisy environments; the pin has an internal pull-up resistor, but according to the AVR Hardware Design Considerations application note (AVR042), "if the environment is noisy, it can be insufficient and reset may occur sporadically."

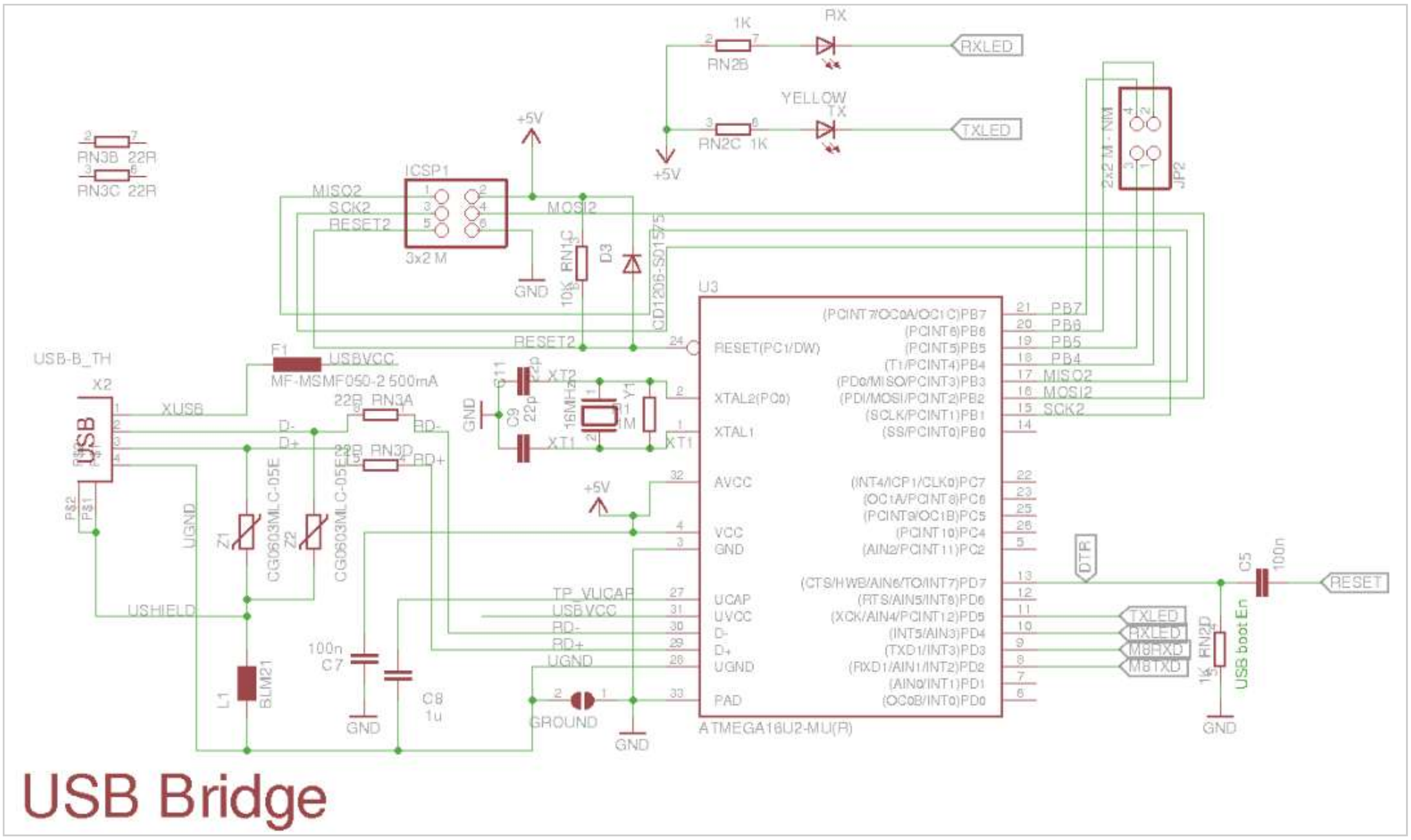
Reset occurs if the user presses the reset button or if a reset is issued from the USB bridge. You can also see the D2 diode. The role of this diode is described in the same app note: “If not using High Voltage Programming it is recommended to add an ESD protection diode from RESET to Vcc, since this is not internally provided due to High Voltage Programming”.

- **C4 and C6 100nF Capacitors:** These are added to filter supply noise. The reactance of a capacitor decreases with frequency:

$$Xc = \frac{1}{2\pi f C}$$

  The capacitors give high-frequency noise signals a low-impedance path to ground. 100nF is the most common value. Read more about capacitors in the AAC textbook.
- **PIN13:** This is connected to the SCK pin from the MCU and is also connected to an LED. The Arduino board uses a buffer (the LMV358) to drive the LED.
- **ICSP (In-Circuit Serial Programming) Header:** This is used to program the ATmega328 using an external programmer. It’s connected to the In-System Programming (ISP) interface (which uses the SPI pins). Usually, you don’t need to use this way of programming because bootloader handles the programming of the MCU from the UART interface which is connected using a bridge to the USB. This header is used when you need to flash the MCU, for example, with a bootloader for the first time in production.

The USB-to-UART Bridge



Arduino USB bridge part. Click to enlarge.

As we discussed in the “Arduino UNO System Overview” section, the role of the USB-to-UART bridge part is to convert the signals of USB interface to the UART interface, which the ATmega328 understands, using an ATmega16U2 with an internal USB transceiver. This is done using special firmware uploaded to the ATmega16U2.

From an electronic design perspective, this section is similar to microcontroller section. This MCU has an ICSP header, an external crystal with load capacitors (CL), and a Vcc filter capacitor.

Notice that there are series resistors in the D+ and D- USB lines. These provide the proper termination impedance for the USB signals. Here is some further reading about these resistors:

1. Why USB data series resistors
2. USB Developers FAQ

Z1 and Z2 are voltage-dependent resistors (VDRs), also called varistors. They are used to protect the USB lines against ESD transients.

The 100nF capacitor connected in series with the reset line allows the Atmega16U2 to send a reset pulse to the Atmega328. You can read more about this capacitor here.

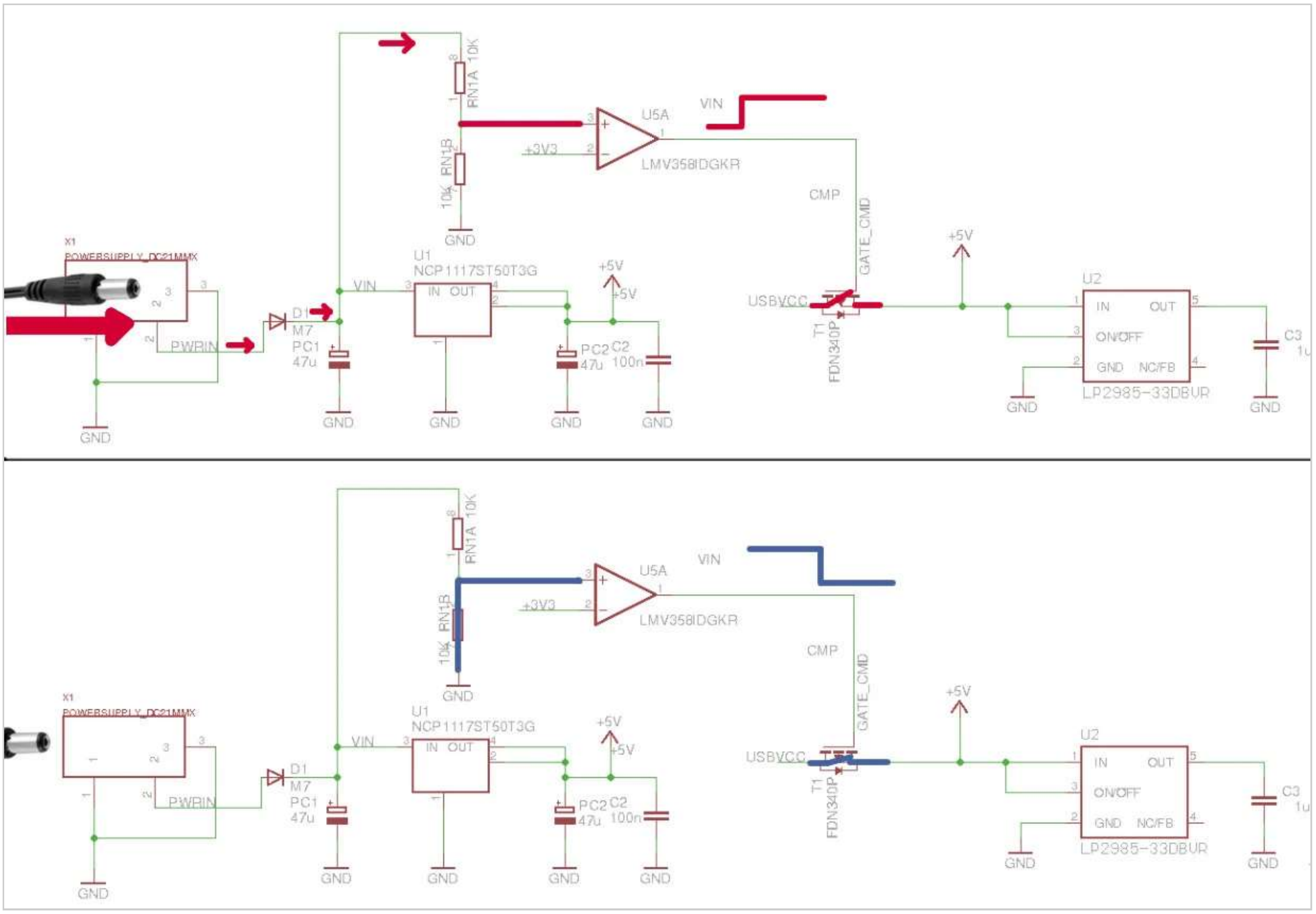
The Power

For a power source, you have the option of using the USB or a DC jack. Now it’s time to answer the following question: “If I connect both a DC adapter and the USB, which will be the power source?”

The 5V regulator is the NCP1117ST50T3G and the Vin of this regulator is connected via DC jack input through the M7 diode, the SMD version of the famous 1N4007 diode (PDF). This diode provides reverse-polarity protection.

The output of the 5V regulator is connected to the rest of 5V net in the circuit and also to the input of the 3.3V regulator, LP2985-33DBVR. You can access 5V directly from the power header 5V pin.

Another source of 5V is USBVCC which is connected to the drain of an FDN340P, a P-channel MOSFET, and the source is connected to the 5V net. The gate of the transistor is connected to the output of an LMV358 op-amp used as a comparator. The comparison is between 3V3 and Vin/2. When Vin/2 is larger, this will produce a high output from the comparator and the P-channel MOSFET is off. If there is no Vin applied, the V+ of the comparator is pulled down to GND and Vout is low, such that the transistor is on and the USBVCC is connected to 5V.

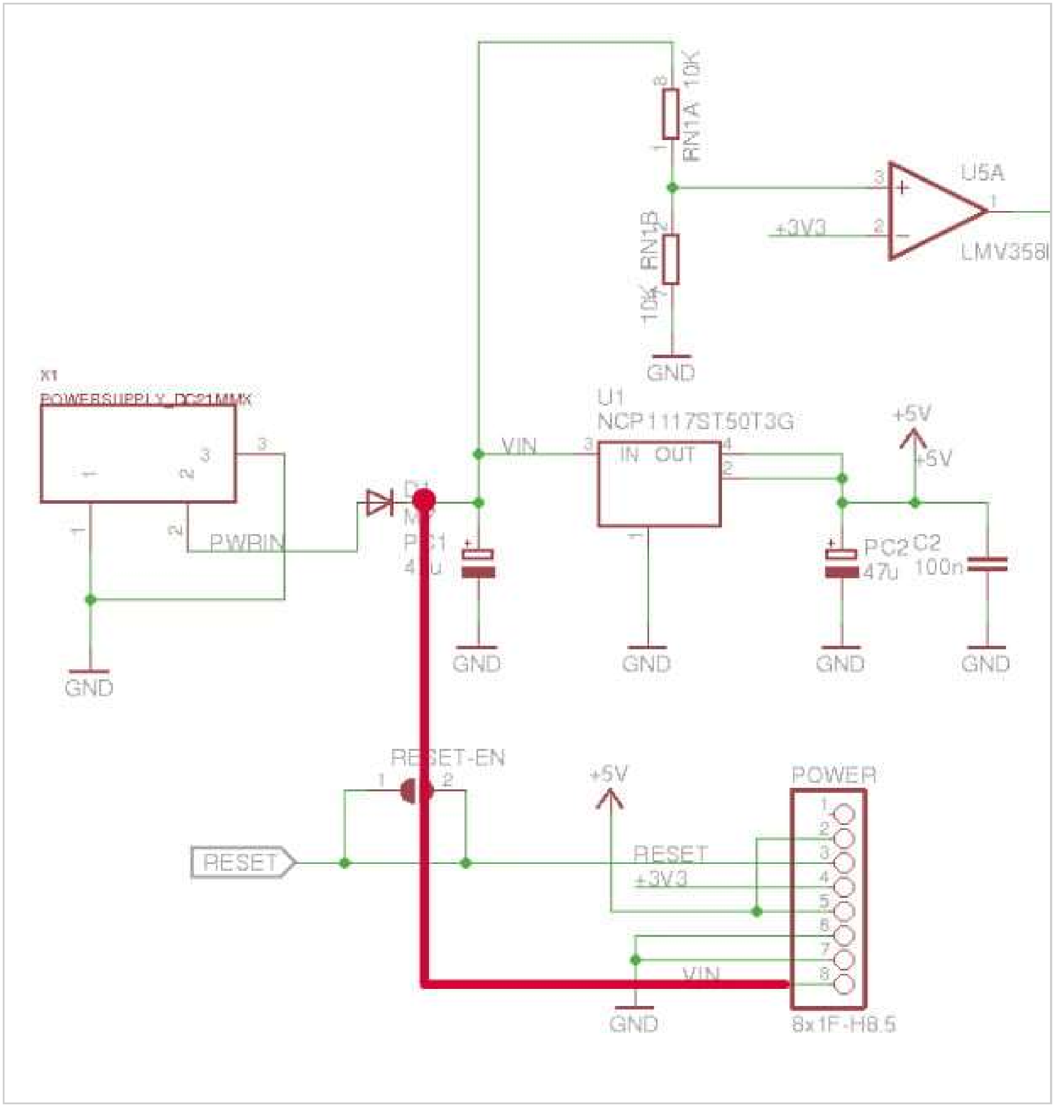


Power source switching mechanism. Click to enlarge.

The LP2985-33DBVR is the 3V3 regulator. Both the 3V3 and 5V regulators are LDO (Low Dropout), which means that they can regulate voltage even if the input voltage is close to the output voltage. This is an improvement over older linear regulators, such as the 7805.

The last thing I'll talk about is the power protection that is provided in Arduino UNO.

As mentioned above, VIN from a DC jack is protected from reverse polarity by using a serial M7 diode in the input. Be aware that the VIN pin in the power header is not protected. This is because it is connected after the M7 diode. Personally, I don’t know why they decided to do that when they could connect it before the diode to provide the same protection.



VIN pin from power header. Click to enlarge.

When you use USB as a power source, and to provide protection for your USB port, there is a PTC (positive temperature coefficient) fuse (MF-MSMF050-2) in series with the USBVCC. This provides protection from overcurrent, 500mA. When an overcurrent limit is reached, the PTC resistance increases a lot. Resistance decreases after the overcurrent is removed.

Reading the Rugged Circuits post about protection in Arduino is very useful.

You should now be more familiar with the Arduino UNO's electronic design and have a better understanding of its hardware. I hope this helps your design projects in the future!

Load more comments