



# Arduino Bare Metal

## Arduino Baremetal / Barebone???

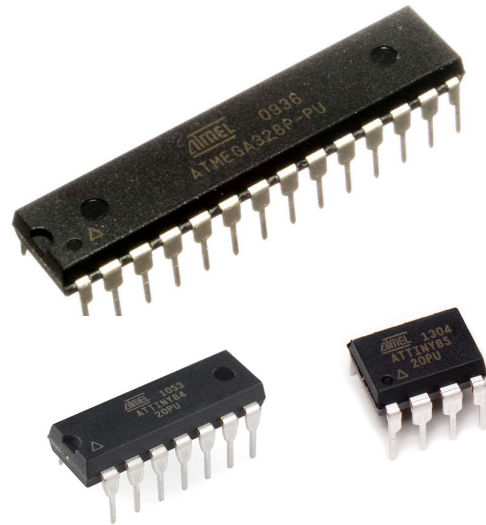
### bare bones

Examples Word Origin

See more synonyms on [Thesaurus.com](https://www.thesaurus.com)

plural noun

1. the irreducible minimum; the most essential components:  
*Reduce this report to its bare bones. There is nothing left of the town but the bare bones—a couple of stores, a church, and a few houses.*



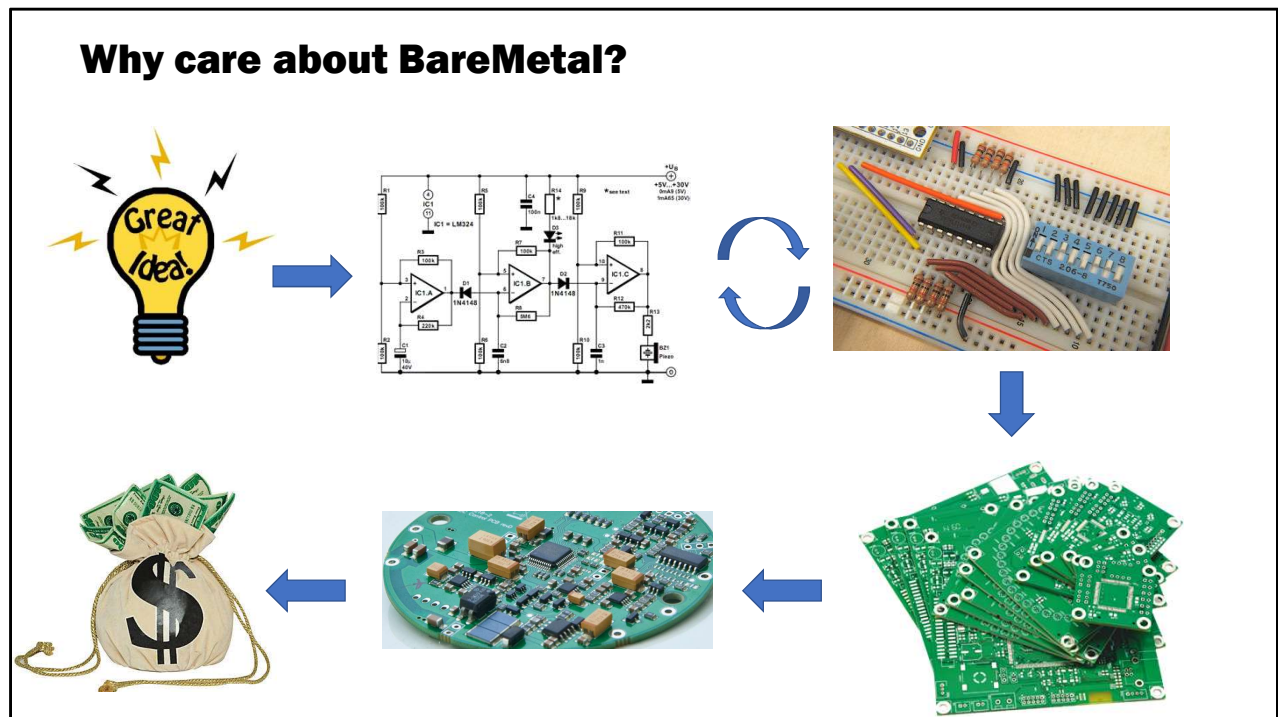
### Arduino Barebone

Barebone stands for the bare minimum / essential components to something. In this case it means we're going to create an Arduino compatible circuit using the bare minimum amount of components.

A normal Arduino Uno board has the same micro controller: the ATmega 328, but it doesn't stop there. Because UNO is a development board created for students and hobbyists it has components and parts that are not essential to its normal operation, like status LEDs, USB to Serial converter, etc.

When you move from UNO board to barebone, you might miss the 9 volts DC jack, the USB connector, 5 volts regulator, ICSP (in circuit serial programming) connector and power connections. But you still have all the 14 digital I/O pins and the 6 analog inputs.

By loading the code to this micro controller you can expect the SAME behavior you would get from a UNO board.



### Why Arduino Barebone?

Going barebone has its advantages when you don't really need all functionalities from an Uno board. However, this is not the main advantage.

The value of going barebone is to learn how the micro controller behaves and the components you need to add around it to allow its functionality. This is the stepping stone of electronics product development.

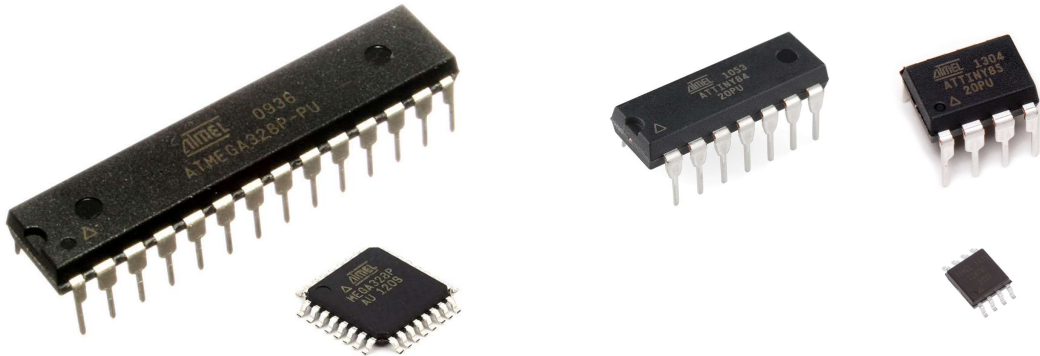
Following the picture above we have:

1. **Idea:** this is when you envision your new product
2. **Circuit:** your idea gain life through the careful design of your product. This is where learning how to use the components will help.
3. **Breadboard:** you can prototype your circuit using a breadboard much like the one we're using today at the workshop, learn, adjust and iterate. Expect to go back and forth within steps 2 and 3 several times.
4. **PCB (printed circuit board):** once you have the circuit design complete, you can

create a board to match the circuit and have it fabricated.

5. **Assembly:** add components to your board
6. **Monetize:** the final step in this loop is when you monetize your project.

Several micro controller options (all ATMEL AVR)



| Microcontroller | Package | Memory | SRAM | EEPROM | I/O Pins | A/D | SPI | I <sup>2</sup> C | PWM | USART |
|-----------------|---------|--------|------|--------|----------|-----|-----|------------------|-----|-------|
| ATmega 328      | PDIP28  | 32k    | 2048 | 1024   | 23       | 6   | Yes | Yes              | 6   | Yes   |
| ATtiny 85       | PDIP8   | 8K     | 512  | 512    | 6        | 4   | Yes |                  | 2   |       |
| ATtiny 84       | PDIP14  | 8K     | 512  | 512    | 12       | 8   | Yes |                  | 2   |       |

Micro controller options – several ones

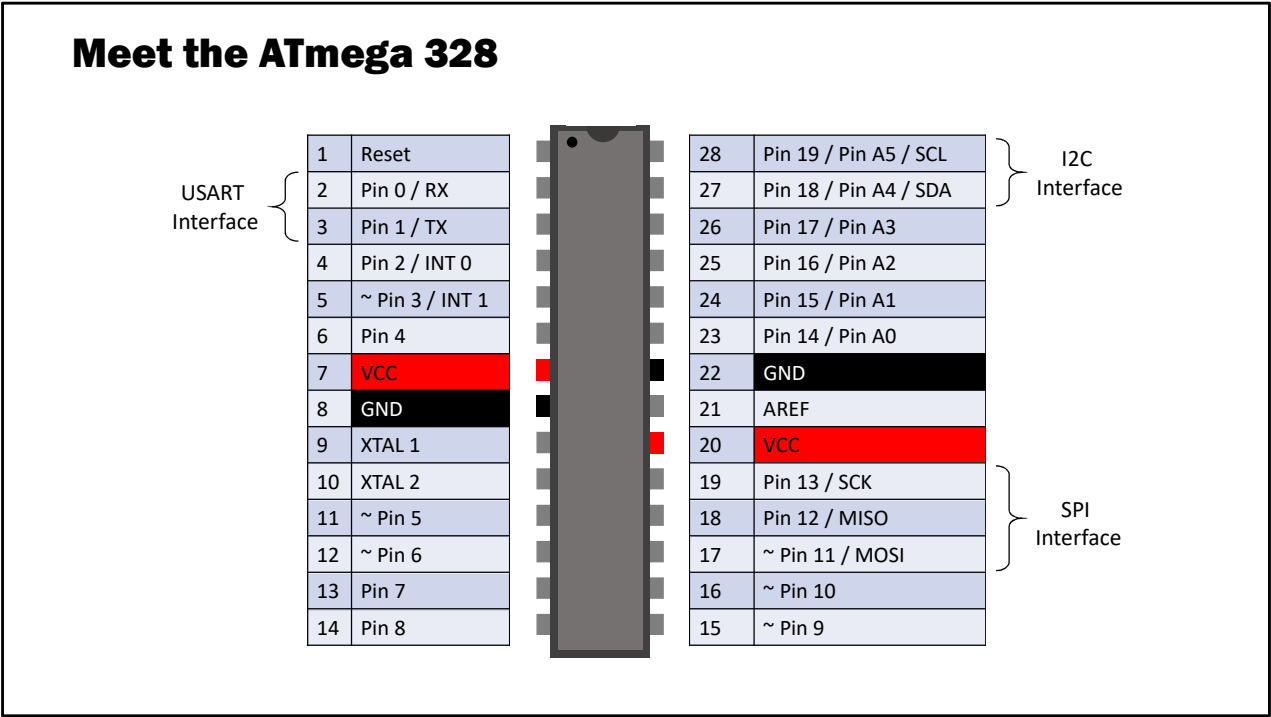
A very important decision for your project is to select the best micro controller for your needs. There’s no such thing as which one is the best one, but rather what’s the best for your project.

Micro controllers varies in size, shape (package), power requirements, digital pins, analog pins, wi-fi enabled, memory, speed, etc.

The examples above are all from the same family: Atmel AVR. Beyond AVR, Microchip also manufactures the famous PIC micro controllers. Between AVR and PICs there are hundreds of chips to choose.

As the picture shows, there are dual in line chips which are breadboard friendly and there are the SMD sized ones, which are great for small footprint projects.

Reference: [www.futurlec.com/ICAtmel.shtml](http://www.futurlec.com/ICAtmel.shtml)



**ATmega 328 Pinout**

The ATmega 328 micro controller comes in different packages, but the simplest one is the PDIP (Plastic dual in line package) with 28 pins as shown in the picture above.

Note that several pins have multiple functions, even though you can only use one of the functions at a time, just like I/O (Input/Output) pins which can be EITHER input OR outputs, but never both at same time.

Pinout description (pins are shown in the mechanical order, starting with 1 at the top left, down to pin 14 and then going from 15 to 28 in the right side of the chip – make sure the chip mark is up to count the pins).

Note that Pin # refer to the pin number you would use in the Arduino IDE interface and not the physical pin on the chip.

List below shows all the physical pins followed by its functionality:

- 1. Reset – pull this pin down to reset the Micro controller
- 2. Pin 0 – RX - connected internally to the USART circuit which allows inbound Serial

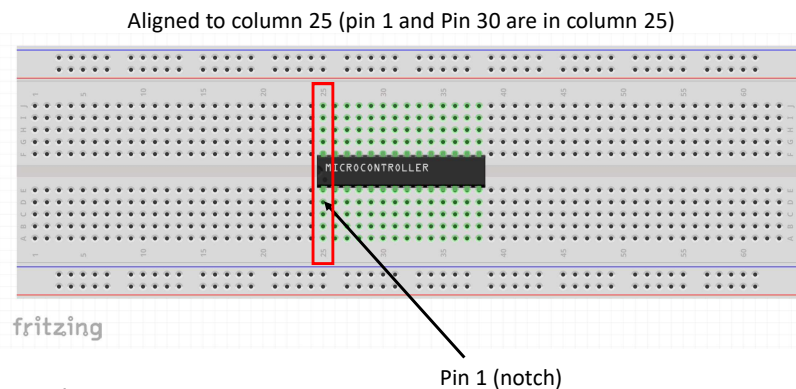
communication to the chip.

3. Pin 1 – TX - connected internally to the USART circuit which allows outbound Serial communication from the chip.
4. Pin 2 – Can be used as a digital I/O and is also connected to the Interrupt 0
5. Pin 3 – Can be used as a digital I/O, is PWM enabled, and is also connected to the Interrupt 1
6. Pin 4 - Can be used as a digital I/O
7. VCC – Supply voltage
8. Ground
9. XTAL 1 – pin to be connected to the external Crystal Oscillator.
10. XTAL 2 – pin to be connected to the external Crystal Oscillator.
11. Pin 5 – Can be used as a digital I/O and is PWM enabled
12. Pin 6 – Can be used as a digital I/O and is PWM enabled
13. Pin 7 – Can be used as a digital I/O
14. Pin 8 – Can be used as a digital I/O
15. Pin 9 – Can be used as a digital I/O and is PWM enabled
16. Pin 10 – Can be used as a digital I/O and is PWM enabled, also connected internally to the SPI (Serial Peripheral Interface) circuit working as SS (Slave Select)
17. Pin 11 – Can be used as a digital I/O, is PWM enabled, also connected internally to the SPI (Serial Peripheral Interface) circuit working as MOSI (Master Out Slave In)
18. Pin 12 – Can be used as a digital I/O, also connected internally to the SPI (Serial Peripheral Interface) circuit working as MISO (Master In Slave Out)
19. Pin 11 – Can be used as a digital I/O, also connected internally to the SPI (Serial Peripheral Interface) circuit working as SCK (Serial Clock)
20. VCC – Supply voltage
21. AREF is the analog reference pin for the A/D converter
22. Ground
23. Pin 14 – Can be used as a digital I/O, or as an Analog input A0
24. Pin 15 – Can be used as a digital I/O, or as an Analog input A1
25. Pin 16 – Can be used as a digital I/O, or as an Analog input A2
26. Pin 17 – Can be used as a digital I/O, or as an Analog input A3
27. Pin 18 – Can be used as a digital I/O, or as an Analog input A4, also connected internally to the TWI (two wire serial interface) also known as I2C (Inter Integrated Circuit) working as SDA (Serial Data Line)
28. Pin 19 – Can be used as a digital I/O, or as an Analog input A5, also connected internally to the TWI (two wire serial interface) also known as I2C (Inter Integrated Circuit) working as SCL (Serial Clock Line)

## Assembling the bareMetal – ATmega 328

### Bill of Materials:

1. Micro Controller ATmega 328
2. Push button (Reset)
3. 16Mhz Crystal
4. 22pF capacitor (x2)
5. 100nF capacitor (x2)
6. 10K resistor
7. 330R resistor
8. Led
9. CP2102 Serial x USB adaptor / board
10. Full size (830 pins) breadboard
11. Jumper wires



### Assembling a barebone Arduino

Whenever you're in the process of creating your prototype, the first step is to define what functionality and what components will be required, because that will determine if a half size breadboard will be enough or even if multiple full sized boards would be the case.

For this workshop we'll use a full-sized board to make it simpler to work and to add components for learning.

**Attention:** Most ICs like the micro controller we're using here are intolerant to static electricity, so avoid touching the pins on the chip directly.

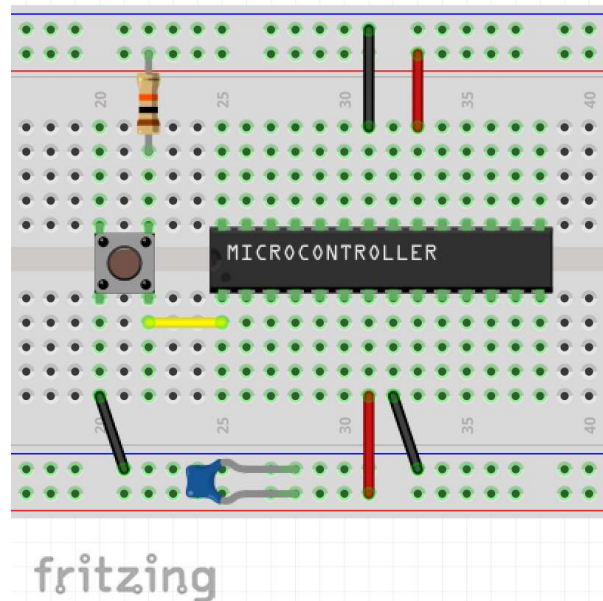
Never used a breadboard before? <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>



## Assembling the bareMetal – ATmega 328

Assembly steps:

1. VCC to pins 7 and 20 (red wire)
2. GND to pins 8 and 22 (black wire)
3. Push button aligned to column 20
4. 10K (brown, black, orange) resistor from VCC to column 22
5. GND to column 20 (push button)
6. 100nF (it reads 104) capacitor between VCC and GND
7. Pin 1 (reset) to push button (yellow wire)



### Assembling details

Note that the chip requires power (VCC and GND) on both sides.

The 100nF capacitor (it reads 104) between VCC and GND acts as a decoupling capacitor.

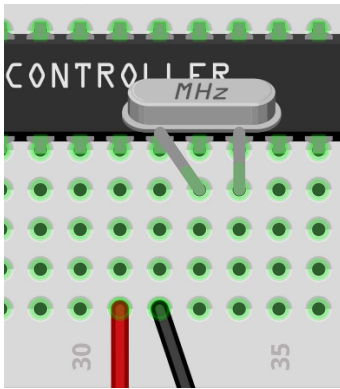
A decoupling capacitor's job is to suppress high-frequency noise in power supply signals.

When physically placing decoupling capacitors, they should always be located as close as possible to an IC. The further away they are, they less effective they'll be.

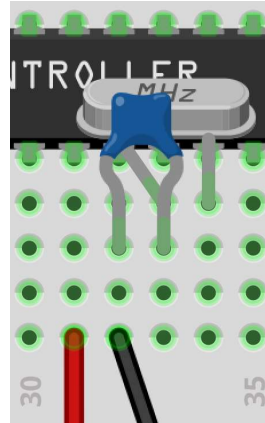
Reference: <https://learn.sparkfun.com/tutorials/capacitors/application-examples>

## Assembling the bareMetal – ATmega 328

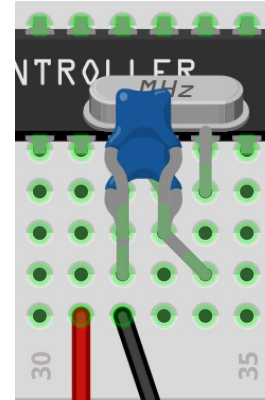
Step 1:  
Add the 16MHz Crystal  
between pins 9 and 10



Step 2:  
Add a 22pF capacitor  
between GND and pin 9



Step 3:  
Add a 22pF capacitor  
between GND and pin 10



### Assembling details

Separating this step in 3 to make it easier to follow.

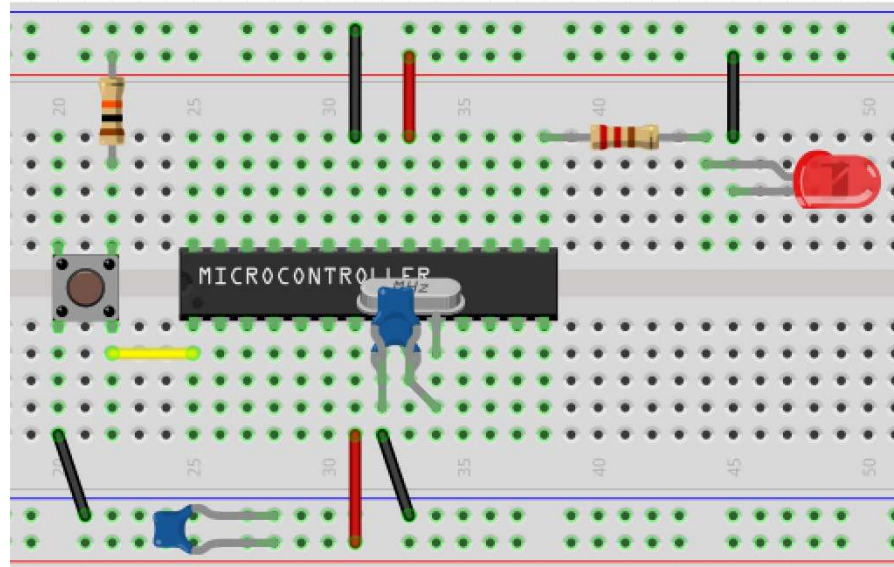
Several similar circuits available in the Internet shows a different approach positioning the components far away from the chip. While this makes the assembling easier, it will add more noise to the line and increase the risk of random issues in the execution.

This is the reasons we're opting for shorter connections, even if they are a little more complex to understand.

## Assembling the bareMetal – ATmega 328

Led:

1. Add an Led between columns 44 and 45 (longer leg goes to 44)
2. Add a black wire between GND and column 45
3. Add a 330R (orange, orange, brown) resistor between column 44 and Pin 15



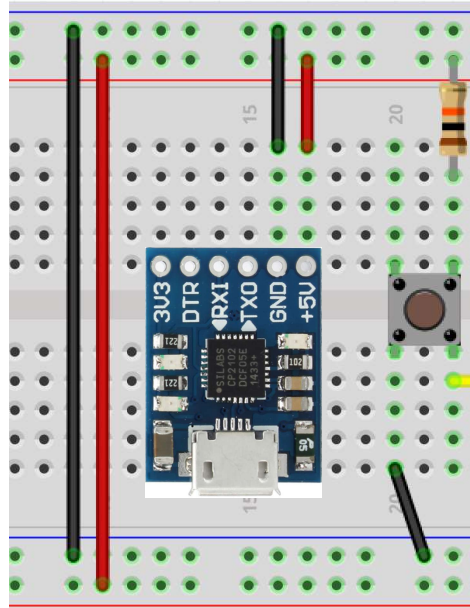
### Assembling details

In this step we're just adding an LED to the pin 15 so we can blink it in a while.

## Assembling the bareMetal – ATmega 328

Last assembly steps:

1. Add a red wire between the top VCC rail to the bottom VCC rail (red to red)
2. Add a black wire between the top GND rail to the bottom GND rail (blue to blue)
3. Add a black wire between GND and column 16
4. Add a red wire between VCC and column 17
5. Plug the CP2102 board aligning +5V to VCC as shown in the picture



### Assembling details

In this step we're adding the CP2102 board, which is an USART interface (Serial to USB). At this moment this board will be only providing the 5 Volts we need to power the circuit (note that VCC and GND are the only pins connected to the breadboard).

In future slides we'll be connecting the other pins to allow the communication between the PC and the ATmega 328 chip.



### Powering your breadboard

Connect your CP2102 board via USB to a 5V power supply.

Could that be my laptop? **NO!!!** There is a significant risk of short circuiting your laptop power supply circuit, damaging it. We absolutely don't recommend doing that. Use a power supply until you're sure your circuit works.

At this moment we're only using this board to supply 5V to the breadboard.

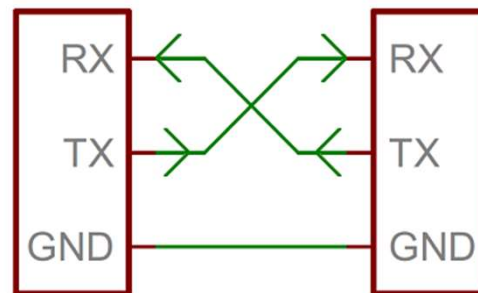
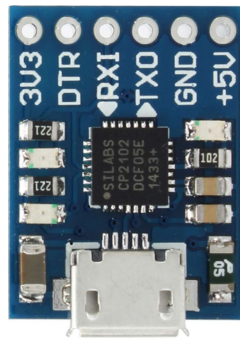
If everything is correct until this moment, the Led should be blinking (this microcontroller was pre-loaded before the class with a blinking sketch to blink Led on pin 15)

Pressing the push-button will reset the code, with the Led on for 5 seconds, going back to blinking every 500ms.

At this moment you have a fully functional circuit including the micro controller and

the reset circuit, but you don't have a way to load new sketches into the micro controller.

## USART (Universal Synchronous and Asynchronous receiver-transmitter)



### USART

A universal synchronous and asynchronous receiver-transmitter is a device for asynchronous serial communication in which the data format and transmission speeds are configurable.

A serial bus consists of just two wires - one for sending data and another for receiving. As such, serial devices should have two serial pins: the receiver, **RX**, and the transmitter, **TX**.

It's important to note that those *RX* and *TX* labels are with respect to the device itself. So the RX from one device should go to the TX of the other, and vice-versa. It's weird if you're used to hooking up VCC to VCC, GND to GND, MOSI to MOSI, etc., but it makes sense if you think about it. The transmitter should be talking to the receiver, not to another transmitter.

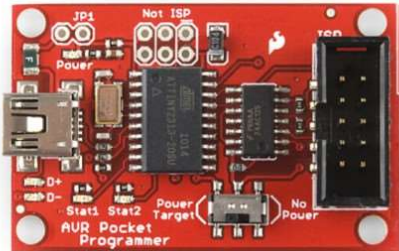
**Baud rates** are like the languages of serial communication. If two devices aren't speaking at the same speed, data can be either misinterpreted, or completely

missed. If all the receiving device sees on its receive line is garbage, check to make sure the baud rates match up.

Reference: <https://learn.sparkfun.com/tutorials/serial-communication>

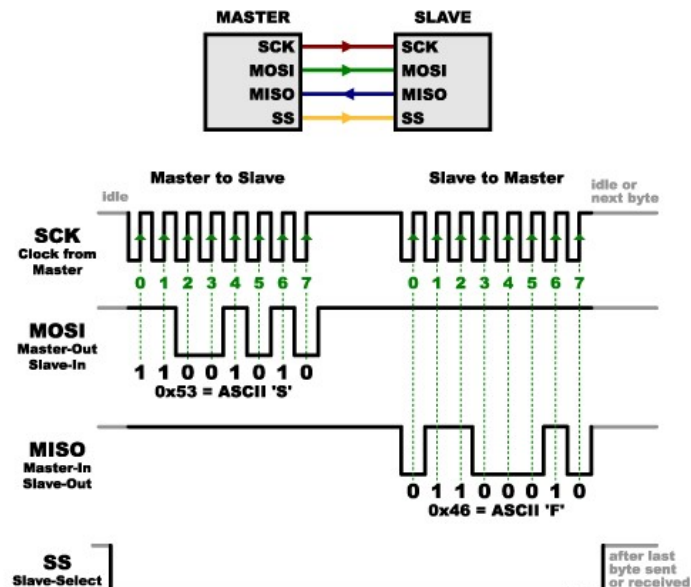


## SPI (Serial peripheral interface)



Serial peripheral interface

Uses 4 wires to send and receive data  
(MISO, MOSI, CLK and RST)



## SPI

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

### Advantages of SPI

- It's faster than asynchronous serial
- The receive hardware can be a simple shift register
- It supports multiple slaves

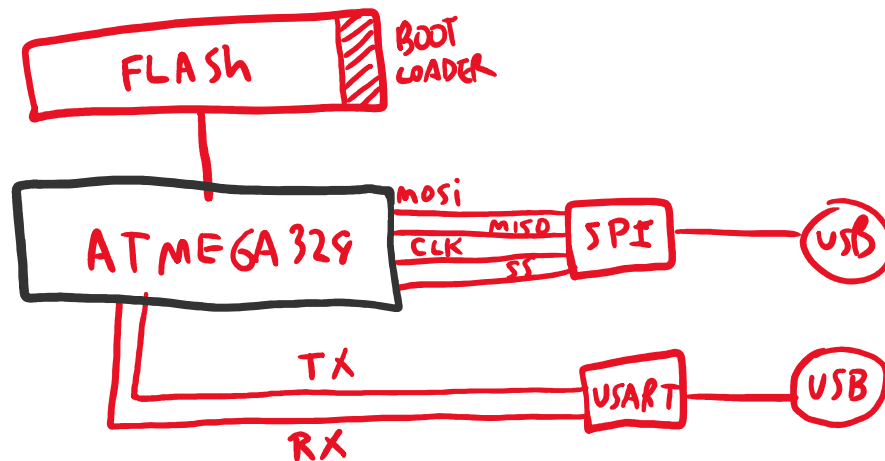
### Disadvantages of SPI

- It requires more signal lines (wires) than other communications methods
- The communications must be well-defined in advance (you can't send random amounts of data whenever you want)
- The master must control all communications (slaves can't talk directly to each other)
- It usually requires separate SS lines to each slave, which can be problematic if

numerous slaves are needed.

Reference: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>

## Talking to the ATmega 328



## Talking to the ATmega 328

The ATmega 328 chip has multiple communication interfaces (SPI, USART, I2C), but when it comes to load new sketches and burn the Bootloader, the main alternatives are SPI and USART.

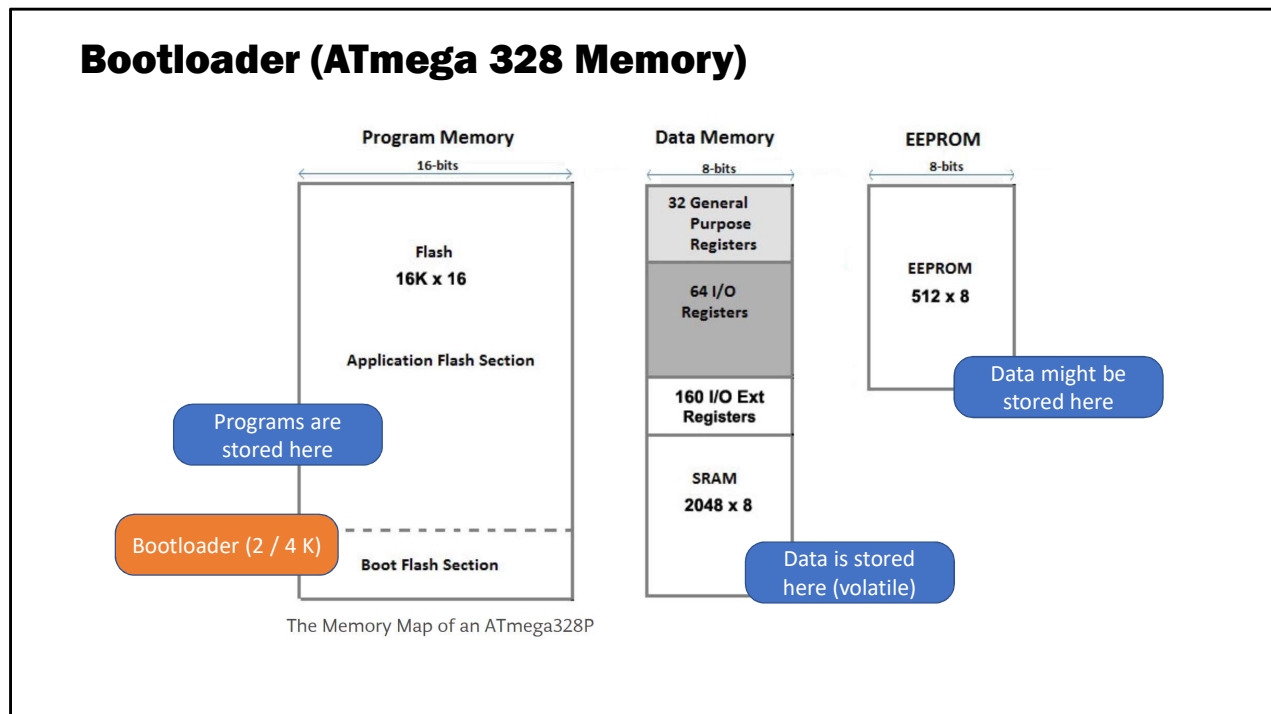
Both of them allow USB to SPI/USART converters to be in place and translate the conversation between the chip and the PC.

Note that USART interface requires only two wires, while SPI requires 4.

While some micro controllers in the AVR family have multiple USART circuits, others have none. In this case (ATTiny Family) SPI is the only alternative to load sketches to the chip. Our ATmega 328 has one USART so we have the choice of using either one or the other.

In the Arduino Uno board, all communication goes via USART interface, and the SPI is exposed via the ICSP (in circuit serial programming) header.

Note that USART interface only works when the bootloader is in place, which is the code running as soon as we power the chip on and which listens to the RX port. Therefore, no bootloader no USART.



## Bootloader

What makes an Arduino what it is? Many things, but one of the most important ones is the way every Arduino board is easily programmed with the Arduino Software (IDE). It is enough to connect it to the computer USB port and press the “Upload” icon to start a process that transfers your sketch into the Flash memory of the microcontroller.

The behavior described above happens thanks to a special piece of code that is executed at every reset of the microcontroller and that looks for a sketch to be uploaded from the serial/USB port using a specific protocol and speed. If no connection is detected, the execution is passed to the code of your sketch.

This little (usually 512 bytes) piece of code is called the “Bootloader” and it is in an area of the memory of the microcontroller – at the end of the address space - that can’t be reprogrammed as a regular sketch and had been designed for such purpose.

### **Is the bootloader ALWAYS needed?**

Not really. The bootloader is needed to allow you to load your sketches into the micro controller, but in order to provide you with this functionality the price to pay is the memory it takes.

If you're shipping your product with an embedded ATmega 328, most of the time, having a bootloader is an unnecessary burden.

Note that brand-new micro controllers don't come with the bootloader, so if you need it you're going to need to burn it first.

### **Saving memory space**

- Unused Libraries - Are all the #include libraries actually used?
- Unused Functions - Are all the functions actually being called?
- Unused Variables - Are all the variables actually being used?
- Unreachable Code - Are there conditional expressions which will never be true

Reference: <https://cdn-learn.adafruit.com/downloads/pdf/memories-of-an-arduino.pdf>

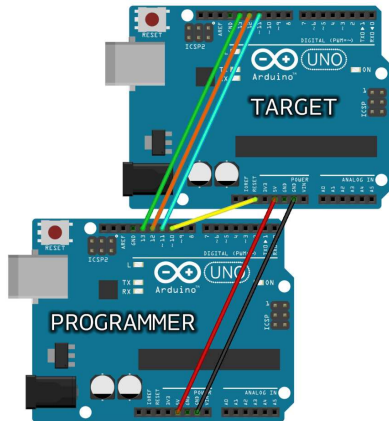
<https://www.arduino.cc/en/Tutorial/ArduinoISP>

EEPROM usage: <https://randomnerdtutorials.com/arduino-EEPROM-explained-remember-last-led-state/>

## Burning the Bootloader - DEMO

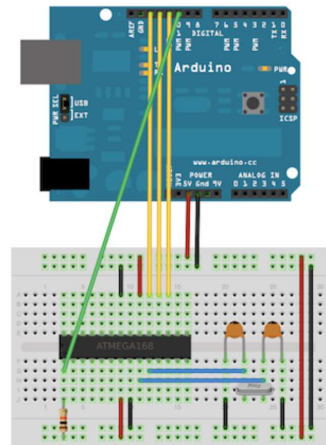
### Option 1

Using an Arduino Uno as the programmer to burn the bootloader into an ATmega 328 P sitting on another Uno board.



### Option 2

Using an Arduino Uno as the programmer to burn the bootloader into an ATmega 328 P sitting on a breadboard (like ours)



## Burning the bootloader

As stated in previous pages, without the Bootloader, no communication via USART is possible. One of the simplest ways to burn the bootloader into the ATmega 328 is to use an Arduino Uno board as the programmer.

The targets could be another Arduino Uno board where we inserted a blank ATmega 328 chip or it could also be a bare bone breadboard circuit.

In both cases SPI is the method used to talk to the chip.

Reference: <https://www.arduino.cc/en/Tutorial/ArduinoISP>

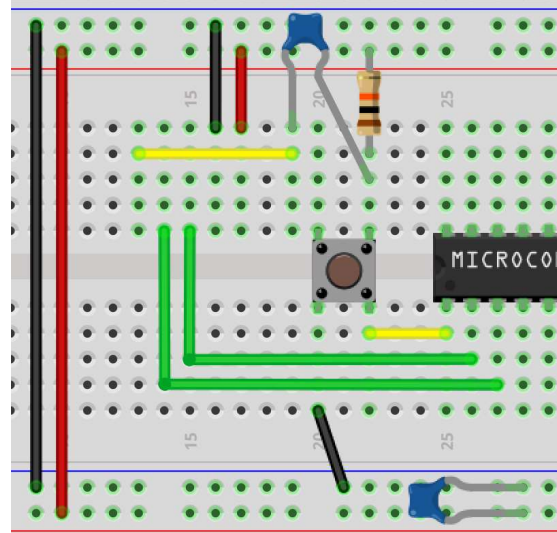
Once you have your boards connected, open Arduino ISP sketch (File – Examples – 11.Arduino ISP – Arduino ISP), then Tools, burn bootloader.

Now, the “burn bootloader” function will load whatever sketch you have opened into the Bootloader section of the micro controller. That’s why is crucial you have the Arduino ISP sketch opened BEFORE burning it to the chip. Don’t worry if you got that wrong, you can always overwrite the bootloader with a newer one.

## Assembling the bareMetal – ATmega 328

Adding USART wires:

1. Remove the CP2102 board from your breadboard
2. Add an yellow wire between column 13 and column 19
3. Add a 100nF capacitor between column 19 and column 22
4. Add a green wire between column 14 and Pin 3
5. Add another green wire between column 15 and Pin 2
6. Plug the CP2102 board again, still aligning +5V to VCC



### Talking to the chip via USART

Assuming your chip already has the bootloader it means this chip is ready to listen to the USART serial ports.

Being the simplest and most common way to interface with the chip, we'll be using this approach for the rest of this workshop.

Green wires shown above are connected to the RX and TX pins of the ATmega 328.

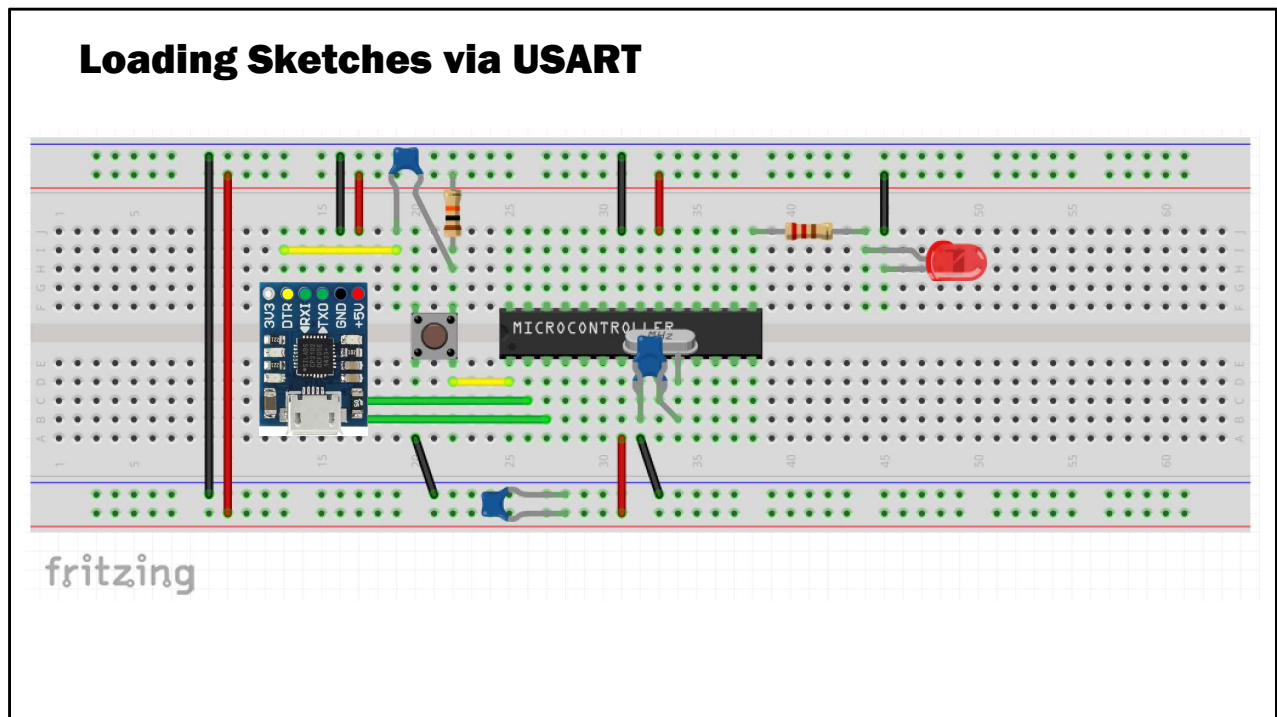
The CP 2102 board is the USB x USART converter we'll be using.

It has 6 output pins:

1. +5V: can provide 5V coming from the PC to the board
2. GND: ground connection
3. TXO: this is the transmitter pin, which should be connected to the RX pin on the Chip
4. RXI: this is the receiver pin, which should be connected to the TX pin on the Chip
5. DTR: this is the reset command, which should be connected to the RST pin on the chip via a 100nF capacitor



6. 3V3: this pin can provide 3.3V coming from the PC to the board



### Breadboard ready!

The picture above shows how your breadboard should look like once you're finished assembling it.

Before connecting the board to the computer via an USB cable, take some time to review all connections.

Once you complete the review, do that again to all VCC and GND connections!!

The chips provided with this workshop have the bootloader and a sketch already loaded which will make the LED blink.

Actually, it will turn the LED on, wait for 5 seconds and then blink.

Try pressing the button, which is connected to the reset pin on the chip to see if it actually resets the ATmega 328 correctly.

If your board is not behaving like explained, please go back to reviewing all steps.

## Loading Sketches via USART

Remember: code is case sensitive!!

```

1  void setup() {
2  pinMode(13, OUTPUT);
3  }
4  void loop() {
5  digitalWrite(13, HIGH);
6  delay(1000);
7  digitalWrite(13, LOW);
8  delay(1000);
9  }

```

- Use Arduino IDE to write this code
- Using the device manager, find the COM port in use by the CP2102 board
- Select Arduino UNO as your board (Arduino IDE can't tell the difference between the full board and just the chip!!)
- Compile and load your sketch into the ATmega 328
- Did it work?



### USART loading

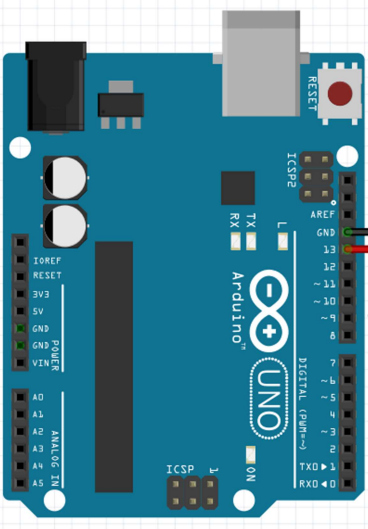
Once your board is finished, it will behave just like an Arduino Uno. It has on the chip all I/O and Analog inputs. All sketches you might have previously will work on this board as well.

In the Arduino IDE you can just select Arduino Uno as the board. It will work just fine!

The differences between the Arduino Uno board and this breadboard are:

1. Breadboard version consumes less power to work
2. Once you load your sketch, you don't need to leave the CP2102 board attached to the breadboard. It is only needed when you need to load another sketch.
3. It will take some time for you to memorize the pins on the chip, the headers on the Arduino Uno help a lot!

### Arduino IDE Pins x ATmega 328 Pins



|    |                 |    |                       |
|----|-----------------|----|-----------------------|
| 1  | Reset           | 28 | Pin 19 / Pin A5 / SCL |
| 2  | Pin 0 / RX      | 27 | Pin 18 / Pin A4 / SDA |
| 3  | Pin 1 / TX      | 26 | Pin 17 / Pin A3       |
| 4  | Pin 2 / INT 0   | 25 | Pin 16 / Pin A2       |
| 5  | ~ Pin 3 / INT 1 | 24 | Pin 15 / Pin A1       |
| 6  | Pin 4           | 23 | Pin 14 / Pin A0       |
| 7  | VCC             | 22 | GND                   |
| 8  | GND             | 21 | AREF                  |
| 9  | XTAL 1          | 20 | VCC                   |
| 10 | XTAL 2          | 19 | Pin 13 / SCK          |
| 11 | ~ Pin 5         | 18 | Pin 12 / MISO         |
| 12 | ~ Pin 6         | 17 | ~ Pin 11 / MOSI       |
| 13 | Pin 7           | 16 | ~ Pin 10              |
| 14 | Pin 8           | 15 | ~ Pin 9               |

Pin usage on the barebone

As mentioned in the last page, having the pin numbers in the Arduino Uno board helps a lot in finding the right pin to connect.

A cheat sheet can be used in this case to help you find the right pin to use and to refer to in your sketch.

Take physical pin 15 for instance. It is at the bottom right in the picture, and at the top right in your breadboard. This pin is the equivalent to the Pin 9 when you’re writing your sketch, so you’ll have to use syntax like “PinMode(9,INPUT);” every time you need to connect something to this pin.

It is simple to realize, as explained before, that several pins have multiple functions, but they can only operate in a single function at a time.

Previous sketch didn’t work because the code declares pin 13, but the LED is physically connected to pin 15, which is pin 9 for the Arduino IDE.

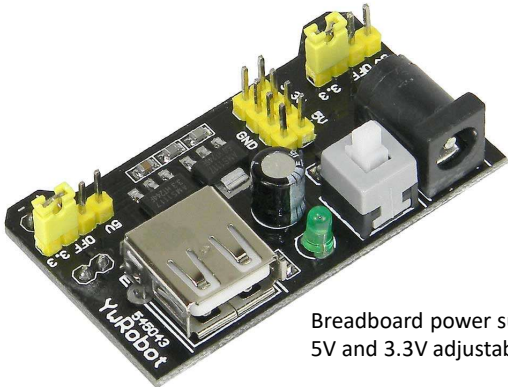
### Power consumption - alternatives

Power limits for ATmega 328 **1.88V to 5.5V**

Better safe than sorry: Use a nice 5V regulated power supply

Power consumption (1 x 3mm LED)

| Led | Barebone | Uno   |
|-----|----------|-------|
| ON  | 25 mA    | 73 mA |
| OFF | 19 mA    | 70 mA |



Breadboard power supply  
5V and 3.3V adjustable



5V and 3.3 V regulators

### Powering your Barebone

There are several alternatives to power your barebone circuit:

1. Use a 5V power adaptor
2. Use a breadboard power supply (as shown in the picture above) which allow you to select 3.3V or 5V, with one selection for each rail.
3. If you're designing your own circuit for later fabrication, you might consider 5V regulators like the 7805 which can take up to 1.5A and SMD regulators such as the AMS1117 in versions for 3.3V and 5V, both can take up to 1A.
4. Use a 3.7V battery with a booster to 5V

Once your code is finished and loaded, if you remove the CP2102 board from the breadboard, the power consumption should be about a third of the equivalent Arduino Uno board, as shown in the table above.

Reference: <https://www.sparkfun.com/products/107> – 5V 1.5A regulator

## Troubleshooting - testing



### Troubleshooting

- Try removing the **Crystal** – code stops. No crystal = no running program. If your code is not running, this would be my first guess.
- Erratic behavior – Add a 100nF **decoupling capacitor** to the power line, as close as possible to the AT Mega 328. Bad power supplies can generate a lot of noise in the power line, which is minimized by using one (or more) decoupling capacitors. The closer to the chip pins, the better.
- **Review your connections** – ATmega is able to survive even to inverted power connections for a while, but test before plugging... (don't ask how we know this!)
- It's easy to **mix up 22pF capacitor** by the 100nF ones. If you do this, your code won't run. (again, don't ask!)
- Got a bad breadboard? Replace it... Hard to fit components and bad quality will lead to several issues from intermittent connections to short circuits.
- Remember the Chip has **2 VCC and 2 GND pins** that need to be connected to the power rails. Even though the datasheet shows the upper VCC connected only to the ADC converter, if not connected the chip won't work.

- Test your power supply when in doubt. Chip won't tolerate excessive voltage. **Multimeter** is your friend!
- Nothing works? Get an Arduino UNO and plug the chip there for a test. Be extra careful and gentle when removing/inserting the chip. A bent pin might break when you're trying to fix it, damaging irreversibly you chip.
- Can't load the code? Are you sure you got the **Bootloader** into your chip? If you can't load you sketch even after moving your chip to an Arduino Uno board, and you keep getting "out of sync" error messages, it is very likely your chip is missing the bootloader. If you have an SPI adapter at hand, try loading the code using SPI. If it works via SPI, then – definitely – the bootloader is missing.

## What's next?

- What do you want to build? @Snoco Makers
- Different boards, different needs
  - Size, Speed, Memory, #Ports
  - Wifi - IoT projects, CNC, Robots
- Advanced topics:
  - Memory management
  - Manual multi threading
  - Interrupts / Semaphore controls
  - Bit shifting operations
- Electronics
- NeoPixels
- Radio control
- ATTiny microcontrollers (barebone)
- Fritzing (prototype software)
- IoT 101
- Digital analyzer
- Laser cutter
- Robotics club

Internet references:

YouTube channels (EEV blog, Educ8TV, etc)

Instructables, Hackster, sparkfun, Adafruit, Arduino.cc

## Next Steps

In this workshop you've learned how to build an Arduino-like device, how to interface with the chip, how to identify and setup the initial design for a simple circuit.

Now what's next?

There is not a simple answer to this question. It depends on what types of project you want to build.

For each project there will be a different board:

- **Memory management:** we saw in this workshop the ATmega 328 has 3 different types of memories. Advanced uses of memory will be necessary when moving to more advanced projects. Different chips will have different memory sizes.
- **#Ports:** As we saw in this workshop, ATmega 328 has 14 I/Os. AtTiny will have only 6 while the AT MEGA 2560 (chip on the Arduino Mega) has



54. How many do you need?

- **Manual multi threading:** our sketches are executed directly in the chip, there's no OS (operating system) to help us here. Understanding the concepts of multi-threading will extend your ability to create more advanced code.
- **Interrupts:** interrupts are amazing resources when you need some real time responses from the chip. ATmega 328 has 2 interrupt that can be used in your sketches.
- **Bit shifting operations:** these are a more complex way to control your chip, but the tradeoffs are worth it: smaller and faster code is the result of working with bit shifting operations.
- **WIFI:** the ability to connect your board to the Internet will allow you to think about connected devices and IoT scenarios.

At **Snoco Makerspace** space you'll have options to dig deeper in the Arduino world. Some of the Workshops / Meetings we have:

- **NeoPixels:** how to connect them, what libraries to use, how to get started with Neopixels
- **Radio control:** how to use radio expansion boards to allow P2P communication to your projects
- **ATTiny microcontrollers** (barebone): Shrink your project going barebone! How to use the AtTiny 85 to create a project.
- **Fritzing** (prototype software): how to draw your electronics diagram, the PCB board and send that for fabrication using Fritzing, a free software used in this material to create all wiring pictures.
- **Digital analyzer:** Sometimes the Serial monitor can't help you in your debugging adventures. Instruments like a multimeter are limited when it comes to high speeds. The digital analyzer is a tool that will allow you see what is happening on several Arduino pins at once. A must for more advanced projects.
- **Laser cutter:** Mandatory training if you're intending to use the laser cutter in your projects.
- **Robotics club:** an ongoing series of meeting for robot lovers! Create your own robot and have them compete against other robots in the club.

Some useful references where you can learn more:

1. Adafruit: [www.adafruit.com](http://www.adafruit.com)
2. Instructables: [www.instructables.com](http://www.instructables.com)
3. Hackster: [www.huckster.io](http://www.huckster.io)
4. Sparkfun: [www.sparkfun.com](http://www.sparkfun.com)

5. Arduino: [www.Arduino.cc](http://www.Arduino.cc)

You tube channels worth watching:

1. EEV Blog
2. Explaining Computers
3. Andreas Spiess
4. Ben Eater
5. Educ8s.tv