



8BitCADE^{XL} Make Dictionary

With 8BitCADE



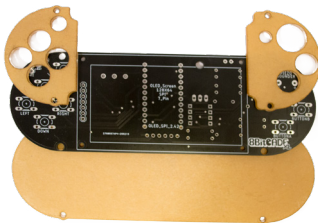
8BitCADE XL: Part Dictionary

Below is an explanation of what each part it is and why we use it on the 8BitCADE. This is a good reference sheet for all of the hardware on your 8BitCADE. If you dont understand something, be sure to reread over the explanation, search the question online or check out the [Arduino Community here.](#)



Right Angle Toggle Switch

The toggle switch has 3 pins, one will be powered by the lipo battery, the other will be open circuit, isolating power so the unit remains off and the last pin connects to the rest of the 8BitCADE circuit to provide power. The switch essentially acts as a break in the circuit. When you toggle the switch, the circuit is closed and allows electricity to flow



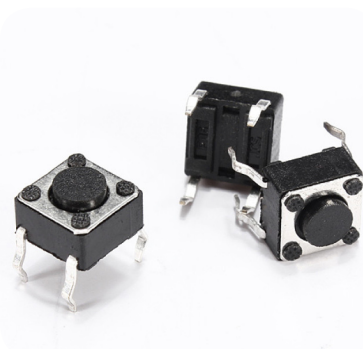
PCB Board

The PCB Board will hold all of the components together and link them up with copper tracks. These tracks are etched into a layer of copper and laminated between sheets of non conductive substrate. Through the use of copper pads and through hole joints, we solder the components in place. On your PCB, you can see white outlines, this is known as silk-screen. This is printed on the PCB last and is simply used to display text/outline on the PCB board.



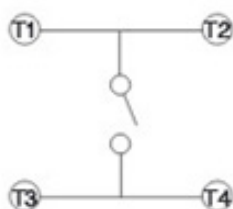
Arduino Programming lead - USB Micro B Cable

Arduino use a special USB lead to connect the microprocessor board to the Arduino IDE Software for powering or programming. The Arduino Pro Micro Board uses a USB Micro B Cable. This cable will provide a steady 5V supply to your Pro Micro from your laptop/PC USB port as well as enabling you to transfer your program.



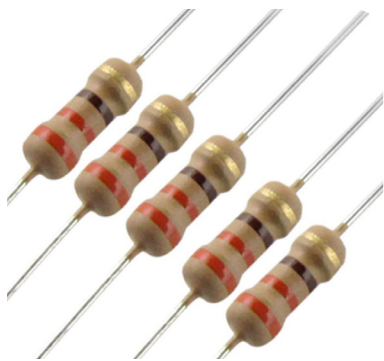
Tactile Switch - Button

A button is a simple on-off switch. There are many kinds of buttons, distinguished by the mechanism used to close or open a circuit, but essentially all buttons belong to one of two families: those that keep the connection in either an open or a closed state, and those that return to their original (default) state once pressure is removed. A momentary button, as shown below, remains closed while pressure is applied to it, then returns to the open position once pressure is removed. This is an ideal button for a hand held game console. 6 buttons are used, Up, Down, Left Right, A and B.



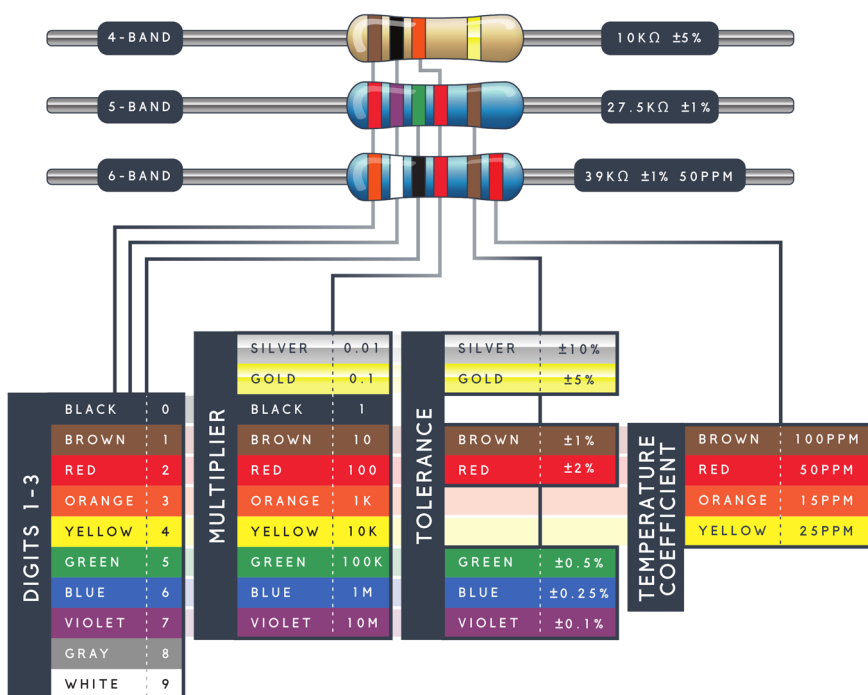
This is an electrical symbol of the switch. As you can see that by pressing the button you are connecting two connections one side of the button to the other. T1 and T2 will be attached to the Pro Micro pins and T3 and T4 will be attached to GND (Ground).

Part Dictionary



330 Ohm Resistors

A resistor is an electrical component that limits or regulates the flow of electrical current in an electronic circuit. This is especially useful when you are limited by one voltage input, like 5V but you have components which require less! Resistors are usually added to circuits where they complement other components like leds! The electrical resistance of a resistor is measured in ohms. The symbol for an ohm is the greek capital-omega: Ω . Resistors come in a variety of shapes, sizes and resistance values. Resistors coloured to show what their resistance is, this is achieved using colour bands.



Resistor Chart

Using a standard four band resistor, the first two bands indicate the two most-significant digits of the resistor's value. The third band is a weight value, which multiplies the two significant digits by a power of ten. The final band indicates the tolerance of the resistor. We use resistor tables to understand resistor codes, just like the one below.

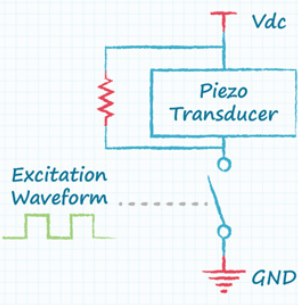
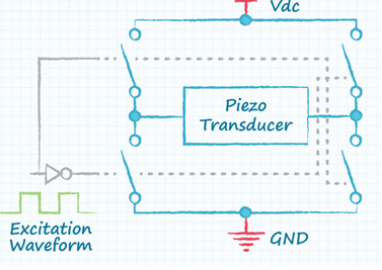
Part Dictionary

Piezo Buzzer

A piezoelectric buzzer is a loudspeaker that uses the piezoelectric effect for generating sound. The initial mechanical motion is created by applying a voltage to a piezoelectric material using a frequency modulated signal (fixed 50% duty cycle but with varying frequency). The motion is typically converted into audible sound by fixing the piezoelectric material to a thin disc and then applying electricity, we can bend the metal back and forth, which in turn creates the noise. The faster you bend the material (by applying a higher frequency signal), the higher the pitch of the noise that's produced. Using Arduino, you can make sounds by selecting a frequency to get a tone. When programming, you have to set which pins the piezoelectric buzzer is on, what frequency (in Hertz) you want, and how long (in milliseconds) you want it to keep making the tone.



The range of frequencies that can be produced depends on the microcontroller's clock frequency (Pro Micro is 16Mhz) and the timer which is being used to apply the frequency to the pin of the buzzer. The frequency to create notes varies from around 31 to 65535 Hz, but human hearing is constrained to sound frequencies below 20kHz.

	
<p>Normally you connect one pin of the buzzer to the ground and the other pin to a square wave, frequency modulated signal output from (with timer function) the microcontroller.</p>	<p>For increased volume on the 8BitCADE, we connect both pins to signal wires on the microcontroller (both require timed functions) and swap which pin is set high or low. This is called a differential drive, producing double the amplitude, gaining a higher volume.</p>

Lithium Polymer Battery

LiPo batteries use a polymer electrolyte (electrically conducting material), instead of a liquid solution found in lithium-ion batteries. This is used to store the electrical power for us to use in our console.



The battery provides 3.7 volts and is rated at 500 mAh. This essentially means that if the battery was fully charged and the current draw of the 8BitCADE was about 50mA, perhaps during game play, it would take about 10 hours before it was fully drained. The reality of course, is that the current draw is much less and the game console would stop working due to low voltage regulation at about 2.7V.

Part Dictionary

Battery Charging Instructions

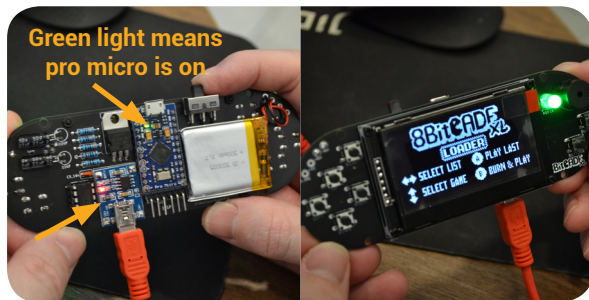
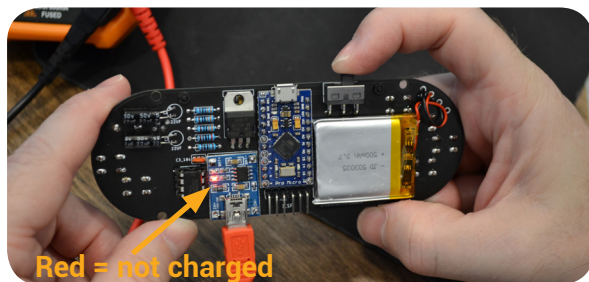
The lipo battery can be safely charged up to 4.2V using the Battery Voltage Arduino Sketch. At 4.3V an alarm will sound indicating that the battery is fully charged and should be disconnected from the USB cable to prevent overcharging.

1. Plugin your 8BitCADE to your PC via the USB programming lead.
2. Push the power switch to the left position, this will use the USB power to charge your battery.
3. Download from the 8BitCADE.com the 'Battery Check Code' Arduino sketch link: <http://8bitcade.com/resources/>
4. Unzip the code and open up the Arduino IDE or Arduboy Hex Loader. Load the sketch (via the IDE) or the Hex file (Via the Arduboy Loader) onto the 8BitCADE.
5. If the power switch is to the left, then the battery is charging and an alarm will go off once the battery reaches 4.3V.

**WE ADVISE TO NOT LEAVE THE BATTERY UNATTENDED WHILE CHARGING.
If you do it is at your own risk.**

The battery also has protection circuitry applied to it as well by the battery manufacturer to protect

- The Battery has overcharge protection voltage rated at 4.20V.
- Over-discharge protection voltage 2.7V
- Over current protection 1.0 amp.



Fast Charge

When plugged into the battery protection board (charging socket) the unit is charging. For fast charging, turn the unit off.

Power Switch to the right (rear view) + Charging
=
Console turned OFF & Fast charging

Slow Charging

You can play the unit while it is charging by pushing the switch to the right. This will still charge, but at a slower rate.

Power Switch to the left (rear view) + Charging
=
Console turned ON & Slow charging

Charging & Programming Caution!

When the console is being programmed, to protect the battery from being overcharged, push the switch into the OFF position (to the left). Please do not charge from the programming port

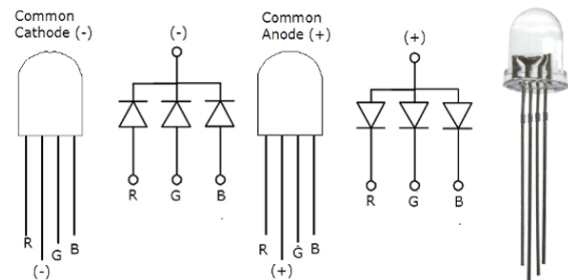
Charging Caution! When the console battery is low, it will turn itself off. From its low state, it would only need about one hour to charge the battery to full capacity. The battery has both over and under charge protection circuitry - but it is not recommended leaving your battery on charge, unattended.

Part Dictionary

330Ω/1k/10k/2k Ohm Resistors

An RGB LED in this kit is common anode (longest leg) and a combination of three LEDs but housed in just one package. The RGB LED normally has 4 pins, one of the pins will be common anode (positive) or cathode (ground) while the other pins will go directly to the 3 LEDs. LEDs are polarity bias, they must be placed on the PCB board or circuit the correct way around to work properly because the electricity must flow in the correct direction.

Each LED within the package has a different colour, red, green and blue. If we just want colours red, green or blue, we can simply switch on each separate LED in turn. To create a wide range of different colours we adjust the brightness level of each LED within the RGB LED package using special code within the Arduino IDE. To adjust the brightness of each LED, we use a special digital signal called PWM or Pulse Width Modulation signal.

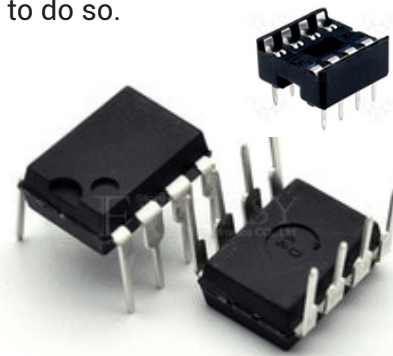


Component Protection

Like all LEDs, RGB LED package is no different and must be protected using a 220 or 330 ohm resistor inline with each LED pin, R, G and B.

8 Pin Serial Flash Chip (16Mb) and Chip Holder

8BitCADE XL has an onboard 8 pin serial flash chip with the capacity of 16mb and operating voltage of around 2.7 to 3.6V. As stated earlier, it is solid state (doesn't use any mechanical parts like a disk/card) is non-volatile (when turned off, the data is still stored and not wiped) memory storage device. It is used to store games and other programs which can be loaded using the 8BitCADE loader (menu screen loader kindly developed by Mr Blinky and the Arduboy Community) to quickly select and load them. Because the files are so small, you can hold about 500 on a single chip. To load the games onto the chip we use a small program to transfer data. The chip has already been programmed with 90+ games, but if you want to update it with additional games or programs then use one of our tutorials to do so.

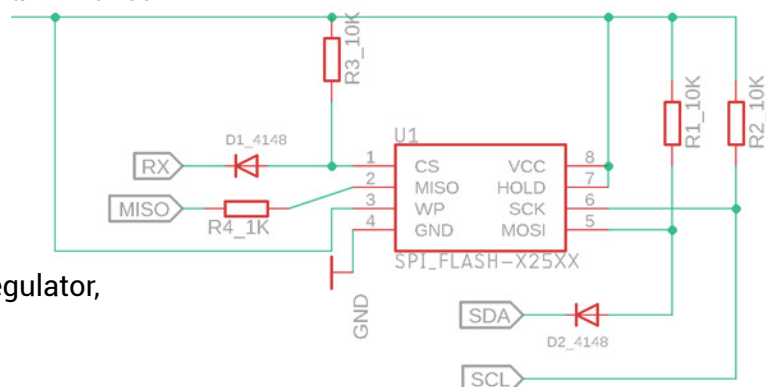


Part Code W25Q128FVIQ

The round circle on the 8 pin DIP (dual inline package) indicates pin 1 of the chip. Diodes and resistors in this circuit are used for level shifting, in other words, to obtain the correct voltage at the pins RX/SCK/MOSI/MISO, as we have a 2.7 to 3.7V component working with a 5V Arduino ProMicro, so we need to ensure that we have the correct voltage levels between communication pins. Part of the schematic for the 8BitCADE XL is below showing the memory chip, resistors and diodes.

Pins:

- CS – Chip Select,
- MISO - Master In Slave Out,
- WP - ,
- GND – ground connection,
- VCC 3.3V supply via voltage regulator,
- HOLD - ,
- SCK – Serial Clock,
- MOSI – Master Out Slave In



Part Dictionary

Pins and their functions:

CS > Chip Select

The chip select pin enables and disables the memory chip operation. When the CS pin is held high the chip cannot send data via any of the data output pins (MOSI) and the devices power consumption will be at standby levels unless an internal erase, program or write status register cycle is in progress. When CS pin is held low the device will be enabled, power consumption will increase to active levels and instructions can be written to and data read from the device. A pull-up resistor on the CS pin is used to pull high when not in use.

WP > Write Protect Pin

A hardware protection pin which can be used when held low to protect about 4kb worth of memory of the chip. This pin is permanently held high on the 8BitCADE XL as it is powered by VCC as seen in the schematic.

GND > Ground Connection.

VCC > 2.7 - 3.6V supply voltage (we supply 3.3V via voltage regulator)

HOLD - Hold pin or reset

The HOLD pin on the 8BitCADE XL is held permanently high by the VCC supply as seen in the schematic. The HOLD pin allows the device to be paused while it is actively selected. When HOLD is brought low, while CS is low, the MISO pin will be off and signals on the DI and CLK pins will be ignored. When HOLD is brought high, device operation can resume. The HOLD function can be useful when multiple devices are sharing the same SPI signals.

SCK – Serial Clock (with pullup resistor)

Serial Clock pin receives a timed input signal from the Pro Micro (Master device) to provide timing for data read/write functions.

MOSI – Master Out Slave In (Data Input with Pullup Resistor) & MISO > Master In Slave Out (Data Output)

The W25Q128FV supports standard SPI (as used on the 8BitCADE XL), Dual SPI and Quad SPI operation. The 8BitCADE XL uses standard SPI instructions, using unidirectional DI (input via MOSI) pin to serially write instructions, addresses or data to the device on the rising edge of the Serial Clock (SCK) input pin. Standard SPI also uses the unidirectional DO (output via MISO) to read data or status from the device on the falling edge of (SCK).

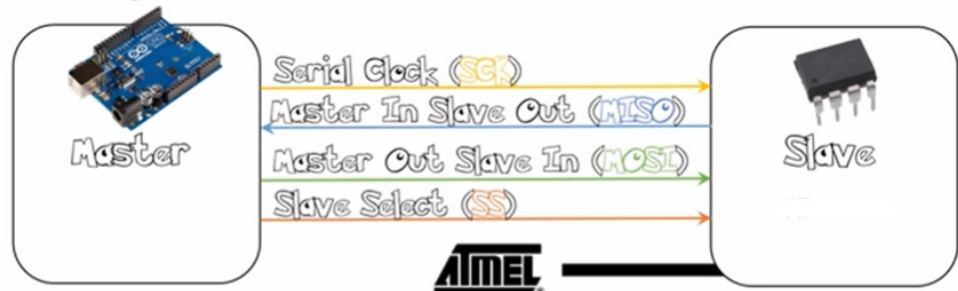
Operation

The W25Q128FV is accessed through an SPI compatible bus consisting of four signals: Serial Clock (CLK), Chip Select (/CS), Serial Data Input (MOSI) and Serial Data Output (MISO). Once the chip select pin is held low, standard SPI instructions use the DI input pin to serially write instructions, addresses or data to the device on the rising edge of CLK signal. The DO output pin is used to read data or status from the device on the falling edge of CLK signal.

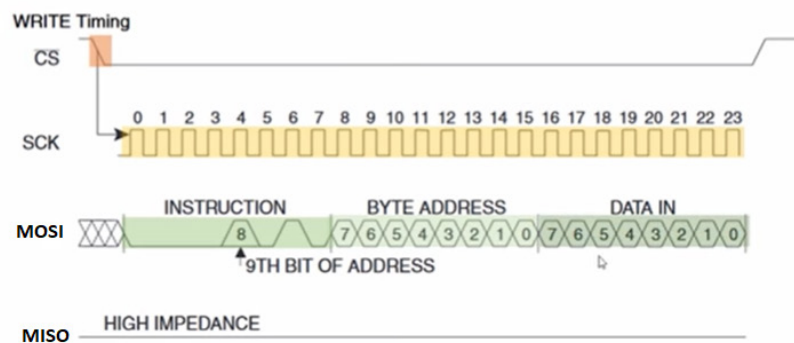
Part Dictionary

Read and Write Protocol

Write Protocol



CS goes low
SCK rising edge
MOSI Data in
MISO Held high by
pull up resistor



Read Protocol



CS goes low
SCK falling edge
MOSI Instruction in
MISO Data out

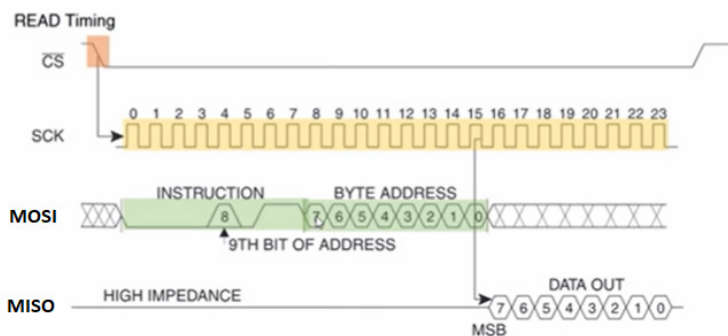


Figure 1 and 2 are good examples of SPI communication but are NOT specific to W25Q128FV chip

Further resources on SPI: <https://www.youtube.com/watch?v=AuhFr88mjt0>

Video on general Serial Communication: <https://www.youtube.com/watch?v=lyGwvGzrqp8>

Part Dictionary

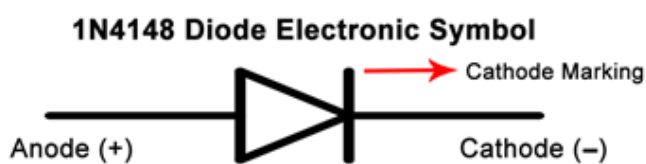
Diodes (1N4148)

Standard signal or commonly called fast switching diodes like 1N4148 have a medium-high forward voltage drop of about 1.0 volt and a low maximum current rating of about 200mA. On the 8BitCADE XL the diodes are used to provide a voltage drop (level shifting), essentially reducing the voltage going to other parts of the circuit. They are fast switching as they are attached to communication pins on the circuit which have high data transmission rates so they turn on and off very quickly, hence they need diodes which can do the same.

What is a diode?

The theoretical function of a diode is to control the direction of current-flow. Current passing through a diode should only go in one direction which is referred to as the forward direction or forward bias. Current trying to flow in the reverse direction is theoretically blocked. Essentially this makes them like a one way valve, only conducting electricity in one direction.

Diodes have 2 terminal and are polarity bias (polarised), must be connected in a circuit the correct way around to operate correctly. The positive end of a diode is called the anode, and the negative end is called the cathode. Current can flow from the anode end to the cathode, but theoretically, not the other direction.



In the real world diodes do actually consume some power when conducting current in the forward bias direction this give us a small voltage drop. And they won't block out all reverse current completely. This gives diodes special characteristics relating to both voltage and current and how they respond in the forward and reverse direction.

1. **Forward bias:** When the voltage across the diode is positive the diode is "on" and current can run through it. The voltage should be greater than the forward voltage (0.7V) in order for the current to be anything significant.
2. **Reverse bias:** This is the "off" mode of the diode, where the voltage is less than (0.7V) but greater than -25V. In this mode current flow is (mostly) blocked, and the diode is off. A very small amount of current called the reverse saturation current is able to flow in reverse through the diode.
3. **Breakdown:** When the voltage applied across the diode is very large and negative, lots of current will be able to flow in the reverse direction, from cathode to anode.

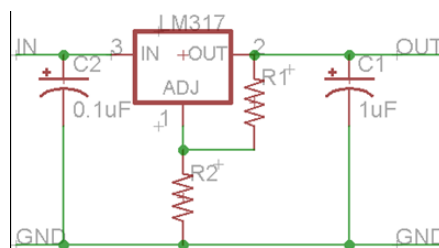
Finally, in order to turn a diode on in the forward bias direction and enable current to flow you need about 0.7V, which is called the forward voltage or V_F . This forward voltage is dependent on the material the diode is made of.

Part Dictionary

Electrolytic Capacitors 0.22UF (220NF) for Decoupling

The capacitors for the 8BitCADE XL are electrolytic and used for decoupling. Decoupling capacitors suppress high-frequency noise in power supply signals. They take tiny voltage ripples, which could otherwise be harmful to delicate integrated circuit, out of the voltage supply. They also act as temporary power supplies, charging when there is an excess of voltage and discharging when the voltage drops, maintaining a stable and continuous power supply. The 8BitCADE has two 0.22UF capacitors fitted in the circuit next to the LM317 voltage regulator.

LM317 is a voltage regulator and requires two electrolytic capacitors, one on the 5V supply (IN) and the other on the 3.3V output (out). Without these capacitors or with the incorrect value and type of capacitor the electrical interference created would stop the microprocessor or Serial Flash.



Decoupling capacitors are often connected between the power source (5V, 3.3V, etc.) and ground. It's not unusual to use two or more of different values, as shown above, because some capacitor values will be better than others at filtering out certain **frequencies of noise**. When physically placing decoupling capacitors, they should always be located as close as possible to an integrated circuit as possible, the further away they are, they less effective they'll be.

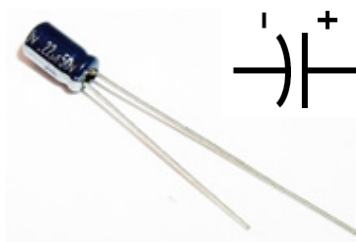
What is a capacitor?

A capacitor is a two-terminal, electrical component which essentially **stores electrical energy** and releases it at a given time. Apart from energy storage, capacitors are also used for **voltage spike suppression (decoupling)**, and **complex signal filtering** (very good at filtering signals at certain frequencies).

Not all capacitors are the same material or have the same use. Each capacitor is built to have a specific amount of capacitance. The capacitance of a capacitor tells you **how much electrical energy it can store**, more capacitance means more capacity to store this electrical energy. The standard unit of capacitance is called the **farad**, which is abbreviated F.

Electrolytic Capacitors (Excellent Signal Filtering)

The capacitors for the 8BitCADE XL are electrolytic. Electrolytic caps are usually polarized. They have a positive pin (anode) and a negative pin (cathode). When voltage is applied to an electrolytic capacitor, the anode must be at a higher voltage than the cathode. The cathode of an electrolytic capacitor is usually identified with a '-' marking, and a coloured strip on the case. The leg of the anode is often longer to help identify it.



WARNING - If voltage is applied in reverse on an electrolytic cap, they'll break by popping open. After popping an electrolytic will behave like a short circuit and would damage the 8BitCADE XL circuit.

These capacitors are poor for current leakage, essentially allowing small amounts of current to run through the dielectric from one terminal to the other. This makes electrolytic capacitors less-than-ideal for energy storage, but ideal for signal filtering.

Ceramic Capacitors (Small Capacitance & Fast charge/discharge Frequencies)

A popular capacitor is the ceramic capacitor which is often rather small in size and value, maximum is about 10µF. Ceramics capacitors are very stable (much lower leakage currents), but their small capacitance value can be very limiting. They are well-suited for high-frequency (fast) coupling and decoupling applications and are often seen on Arduino boards fitted to the resonator



Continue on next page...

Part Dictionary

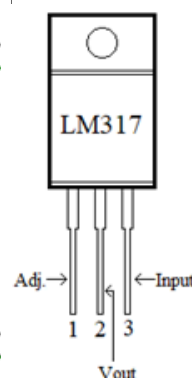
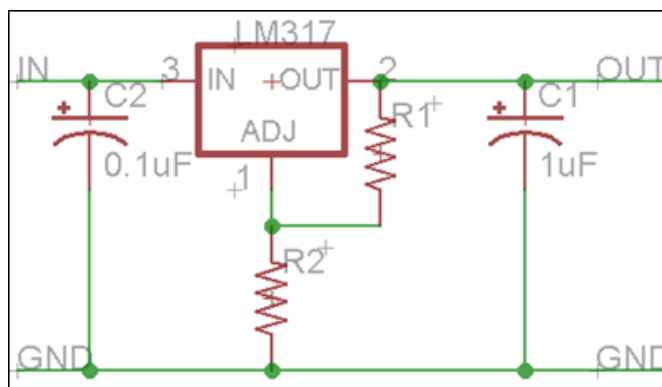
Selecting Capacitors

Decoupling capacitors are added to the circuit in order to smooth out the power supply voltage. A good rule of thumb for digital circuits is to use a single 100nF ceramic capacitor for each integrated circuits, as well as a single larger (up to a few hundred μF) electrolytic capacitor on a circuit segment. The larger electrolytic capacitor stores most of the energy in the circuit and decouples lower frequencies. However, electrolytic capacitors have poor high-frequency characteristics, and parts of the circuit like microprocessors have operating frequencies around 16MHz range. At these higher frequencies, ceramic capacitors provide better decoupling.

Voltage Regulator LM317T

The LM317T is a 3 pin, fully adjustable voltage regulator capable of supplying 1.5 amps with an output voltage ranging from around 1.25 volts to just over 30 volts. By using the ratio of two resistances, we can set the output voltage to the desired level with a corresponding input voltage being anywhere between 3 and 40 volts. The LM317T variable voltage regulator also has built in current limiting and thermal shut down capabilities which makes it short-circuit proof.

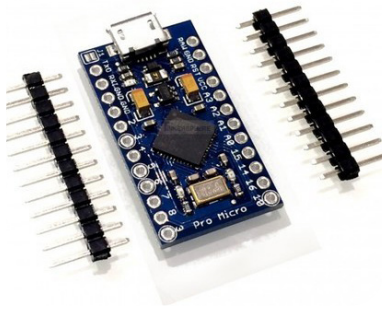
As stated, the output voltage of the LM317T is determined by ratio of the two feedback resistors R1 and R2 which form a potential divider circuit across the output terminal as shown below. The 8BitCADE XL uses two fixed resistors to set the voltage output at around 3.3V to power the Serial Flash Chip. Other products, like a variable power supply might use a variable resistor for R2 so that manual adjustment of the output voltage can be achieved by the user.



- Pin 1 = Voltage output from the potential divider circuit R1 and R2
- Pin 2 = Voltage output, 3.3V to the Serial Flash Chip
- Pin 3 = 3.7-5.0V input from either the lipo battery or USB supply.

Additional capacitors in the circuit support the stability of the regulator and can be anywhere between 100nF and 330nF. Sometimes an additional 100 μF output capacitor helps smooth out the inherent ripple leaving the regulator. This large value capacitor placed across the output of a power supply circuit is commonly called a decoupling or smoothing capacitor.

Part Dictionary



The Brains of our 8BitCADE- Arduino Pro Micro with ATmega32U4 5V 16MHz

The Arduino Pro Micro is a development board which contains a microchip called an ATmega 32U4, seen below as a large black square, with small pins and the name Atmel, the original manufacturers of the chip before the company was bought by Microchip Technology.

The development board holds the chip and supports it by giving it power, breaking out the pins of the chip into numbered connections, programming the chip via usb connection and adding LEDs so you know when power is connected or when the chip is being programmed by flashing leds labeled TX (transmit) & RX (receive). The Arduino Pro Micro Development Board has 18 input/output, often called Input/output pins. Every pin can be used as a digital input or output. Nine of these pins have analog to digital converters (ADCs) and can be used as analog inputs (A0, A1, A2, A3, A4, A5, A6, A7, A10). These pins are useful for reading analog devices.

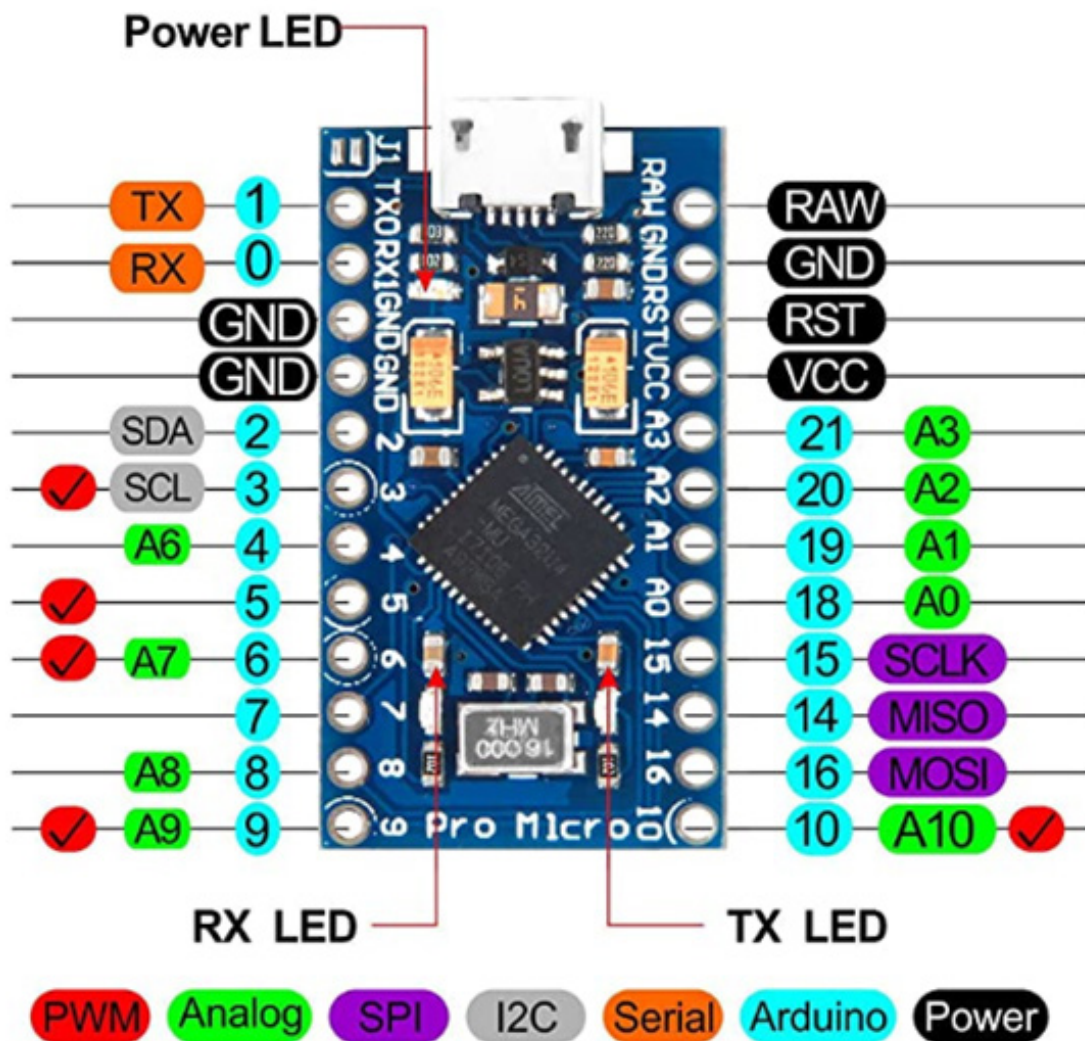
ATmega32u4 Microchip main features

- The ATmega32u4 has built-in USB communication, eliminating the need for a secondary microchip to be added to the board to assist in programming it (converting USB communication into serial).
- The 32U4 chip is low-power.
- The 32U4 chip is an 8-bit microcontroller featuring 32KB of flash program memory, 2.5KB SRAM, 1KB EEPROM.
- Provides 18 input/output pins which 9 can be used for connecting with analogue devices and 5 are pulse width modulated.
- Provides I2C communication

The development board main features:

- Provides voltage regulation to power and protect the board
- Has a power led
- Has programming leds which flash when the 32u4 chip is being programmed
- Can be powered using a 3.7v lipo battery.
- Can communicate with hardware using serial

Part Dictionary



What do these pins do on the 8BitCADE

RAW Pin:

This is the pin which can be normally used to supply power from 5V to 12V input voltage. This voltage will then be regulated by the on board voltage regulator and applied to the ATmega32U4 microchip.

USB Connection:

If the board is powered via USB connection, the voltage at this pin will be about 4.8V (USB's 5V minus a small voltage drop). VCC is the voltage supplied to the on-board ATmega32U4 microchip.

VCC:

If used as an input voltage then no more than 5V must be supplied. This pin can be used as an output pin if voltage is supplied to the RAW pin and can be used to power other components as long as the maximum of 200ma is not exceeded. 8BitCADE uses this pin to power the development board and ATmega32U4 microchip using power from the lipo battery.

GND

GND is the common, ground voltage (0V reference) for the system.

Part Dictionary

RST

RST (Reset) can be used to restart the Pro Micro (restart existing program). This pin is pulled high by a 10kΩ resistor which is on the board and must be connected to ground to initiate a reset. The Pro Micro will remain “off” until the reset line is pulled back to high.

Input/Output Pins:

The Pro Micro has 18 input/output or I/O pins. Every pin can be used as a digital input or output. Nine of these pins have analog to digital converters (ADCs) and can be used as analog inputs (A0, A1, A2, A3, A4, A5, A6, A7, A10). These pins are useful for reading analog devices.

On-Board LEDs:

There are three LEDs on the Pro Micro. One red LED indicates whether power is present. The other two LEDs help indicate when data is transferring over USB. A yellow LED represents USB data coming into (RX) the Pro Micro, and a green LED indicates USB data going out (TX).

Pro Micro Serial Communication

How does the Pro Micro controller communicate internally and externally with hardware (other microchips, sensors) and your computer?

We need to understand the following:

1. Analog communication (read information from analogue sensors to measure something)
2. Digital communication
 - PWM Communication (Pulse Width Modulation Pins 3,5,6,9,10)
 - USB communication (Universal Serial Bus connected via USB Connector)
 - Serial Communication (Asynchronous Serial Pins TX & RX)
 - SPI Communication (4 wire Synchronous Serial Pins SCLK,MISO,MOSI,CS)
 - I2C Communication (2 wire Synchronous Serial Pins SDA, SCL)

Part Dictionary

Screen OLED 2.42 inch 128x64



OLED stands for Organic Light emitting Diode, as it uses a thin carbon based material sandwiched between a positive and negative layer that passes electricity through it. This 'emits light in dots or pixels, in fact the OLED screen is 128 pixels wide by 64 pixels high'. These pixels are turned on and off, individually to make images on the screen. The OLED displays have high resolution (lots of dot per inch). These displays have no back light and makes their own light, that's why these are very low power devices and ideal for portable game consoles.

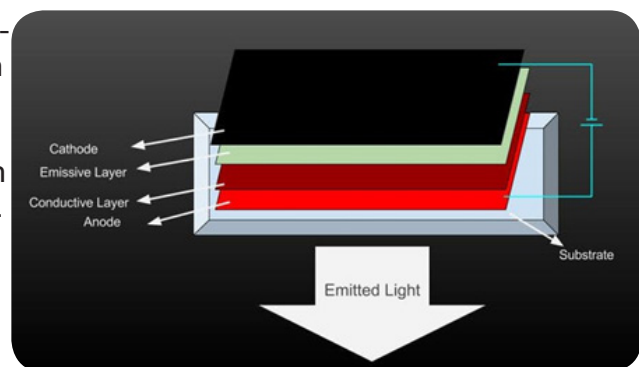
At the heart of the OLED module is a powerful single-chip called an OLED driver controller and named SSD1309. It can communicate with the main 8BitCADE XL microcontroller in multiple ways including I2C and SPI (communicate over a single wire for convenience or multiple wires for speed). SPI is generally faster than I2C but requires more input and output pins, so more wiring. While I2C requires only two pins (wires) and can be shared with other I2C components on a sort of network. It's a trade-off between pins and speed. The OLED SSD1309 controller of the display has complex drivers. Vast knowledge on programming (memory addressing) is required in order to use the OLED SSD1309 controller. Fortunately, Arduino uses library files to help us send basic instructions to operate such OLEDs. They were written to hide away the complexities of the SSD1309 controller so that we can issue simple commands like draw graphics and display images. You will use these later on in the tutorial!

Operation of an OLED

When the voltage is applied to the OLED. The current flows from the cathode to anode through the organic layers of the OLED. The cathode gives the electrons to the emissive layer of organic molecules and the anode removes electrons from the conductive layer of organic molecules.

At the boundary between the conductive and emissive layer, electron holes are created. These holes are filled by the electrons and the OLED emits light. The colour of the OLED depends upon the organic molecules used.

The OLED on the 8BitCADE XL has individual 128X64 white OLED pixels. It is 2.42 inch in size. The OLED's of other sizes are also available. The OLED used in this tutorial is monochrome (Only one colour) but you can also get the OLED's having several colours. This OLED uses the SPI communication to communicate with Arduino. The SPI communication is very fast (much faster than I2C communication) so this will make our display faster, great for gaming where animations like characters move quickly.

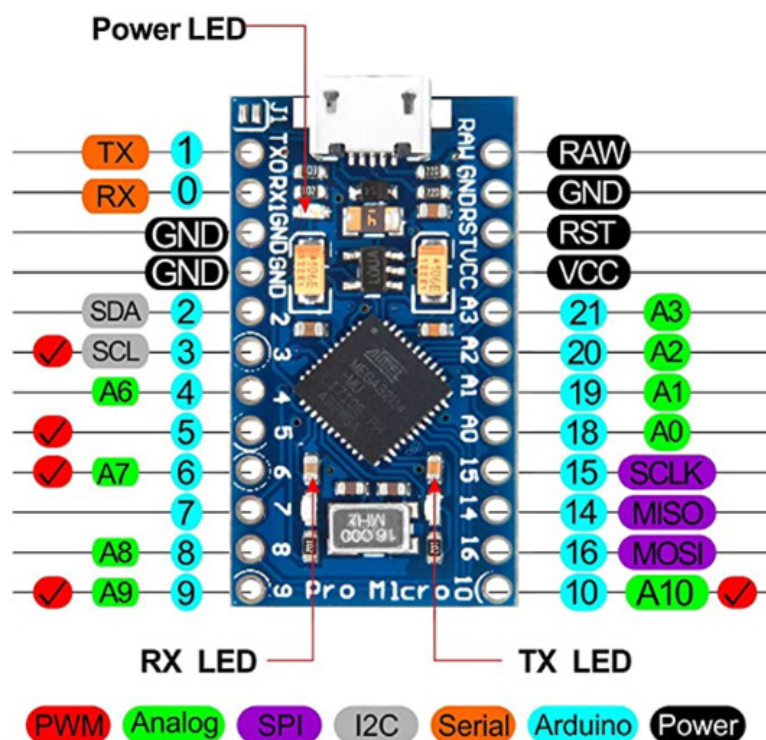


Overview

The maximum size of picture to be drawn on the OLED should be 128X64 pixels and the picture should be black and white because our OLED is monochrome which means that it has only one colour.

Part Dictionary

Analog (green) and Digital Pins (all the rest)



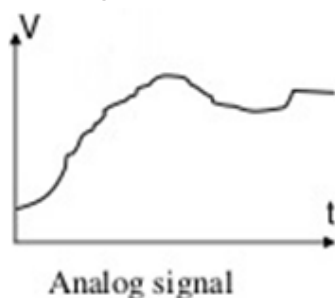
The Pro Micro has many individual pins (coloured blue 1-21) but these pins can be divided into 2 categories, digital and analog. Analog is the green pins and digital are all the other pins on the board. The Pro Micro can receive input information in both digital and analog signals but can only output digital signals.

When a microcontroller like the Pro Micro is powered and receives information from a digital pin (0,1,2,3,5,7,18,15,16) the information will be generally in one or two forms, either 5V or 0V, it will understand zero volts (0V) as a binary 0 and a five volts (5V) as a binary 1.

An analog pin (A0, - A9) might receive a signal value of 2.50V. Is that a zero or a one? We often need to measure signals that vary in voltage; these signals are called analog signals and come from analog sensors. A 5V analog sensor may output a voltage between 0.01V to 4.99V.

Analog Communication Signals

An analog input signal being received by the microcontroller can take the form of any number of values from just below the supply voltage, which could be 4.9 (from a 5V supply) to 0.1 volts (just above 0 volts level) and is generally measured over time. They are generally used to measure some sort of quantity like temperature, pressure or level of light and return a voltage level which represents the value being measured.

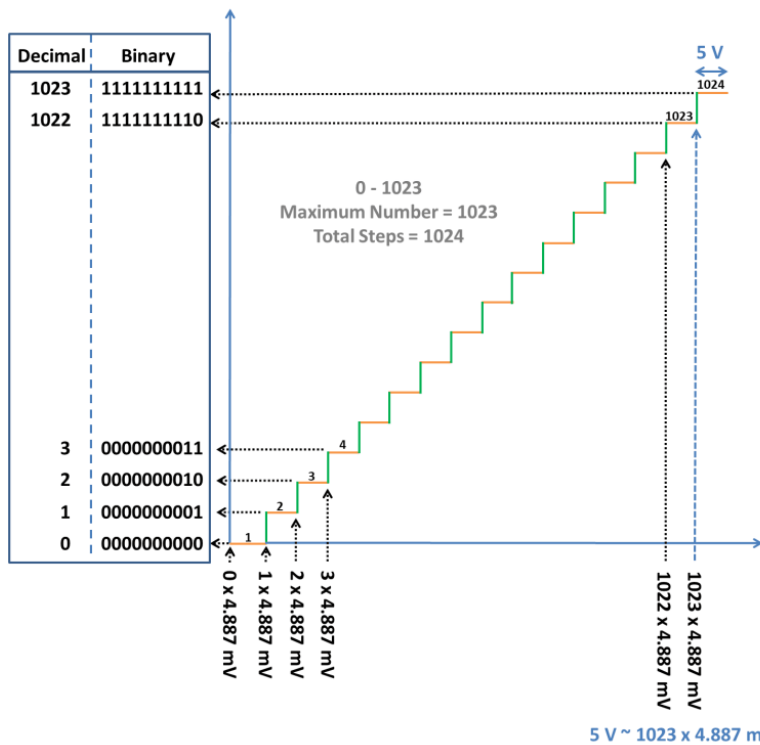


An analog waveform (V= voltage t= Time)

To enable your Pro Micro to understand this information it needs to be in a digital format. Arduino's have many built-in analog-to-digital converters which are called ADC's. These measure the value of analog signal and convert it into a binary number. The analog-to-digital converter changes the analog voltage into a digital value to enable the microprocessor to read it, as microcontrollers communicate using digital forms like binary and hexadecimal and not voltages.

This function converts the value of an analog input pins voltage and returns a digital value from 0 to 1023 which is later changed to a binary number. So when an analogue value is read by your microcontroller it returns a number, not a voltage value but essentially this value represents the voltage which has been converted.

Part Dictionary



The voltage in the graph varies between 0V and 5V. The built in ADC will convert this into a value from 0 - 1023. This gives us $5000 \text{ (mV)} / 1023 = 4.887 \text{ mV/per step}$ which is equal to one increment of 1023 and is represented as 0000000001 in binary.

So if each step is worth 4.887 (mV) then what would the voltage be if 511 steps were returned as a digital value?

$4.887 \text{ mV/step} \times 511 \text{ steps} = 2,497 \text{ (mV)}$ or 2.5V.

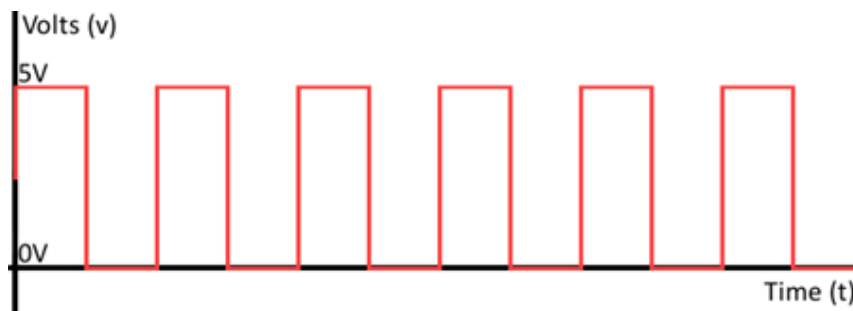
Binary 0111111111

<https://www.binary-code.org/binary/10bit/0000011111/>

The graph and table show the relationship between voltage, the converted ADC (10bit) value of 0-1023 and the binary equivalent.

Simple Digital Signal

A digital signal, on the other hand, has only two values: HIGH which is maximum voltage (5V) and LOW (0V). This will return a value of either binary 0 (low or 0V) or binary 1 (high or 5V).



These signals are typically used for input sensor like a switch. High being 'switched on' and low being 'switched off' or an LED (light-emitting diode), high being on and low being off.

Review:

From the signal we have seen so far, we know that the microcontroller can:

- receive analog signals of varying voltages (0.1 - 4.9V) and convert them into a digital format using an Analog-to-Digital Converter (ADC) which converts the supply voltage into 1023 steps (because it has 10-bit resolution, 1111111111 = 1023).
- receive digital signals which are either full voltage or zero voltage (on or off).
- send digital signals which are either on or off.

Key question: How do you send a varying voltage? Perhaps to dim a light or change the speed of a motor?

The answer is by using a **pulse width modulated** signal (PWM) via pins 3,5,6,9,10. This method of digital communication will provide an average voltage which can be varied from a low voltage to the maximum supply voltage (0-5V)

Part Dictionary

PWM Communication

(Pulse Width Modulation Pins 3,5,6,9,10) We know that we can turn a signal on a microcontroller's output pin high sending the maximum supply voltage out of the pin, for example, 5V. We can also turn it low which is off. But if we switch the signal on and off very quickly, about 500 times per second, and we vary the amount of time the signal is high, compared to how long the signal is low we can vary the voltage output.

Key point: Essentially we can change the proportion of time the signal is high compared to when it is low over a consistent time interval. This is referred to as a Duty Cycle Ratio.

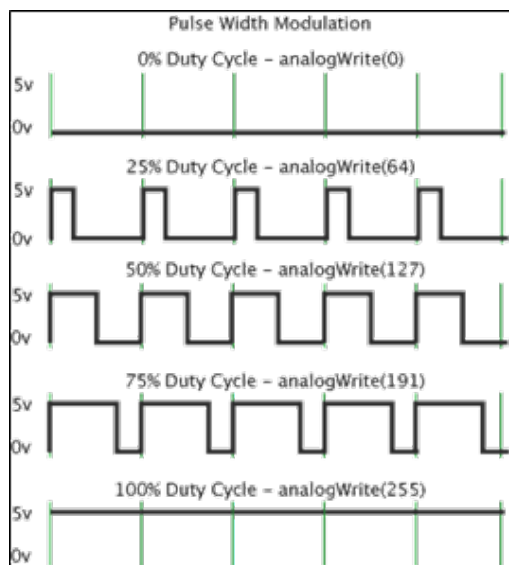
If a digital signal spends half of the time on and the other half off, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave. If the percentage is higher than 50%, the digital signal spends more time in the high state than the low state.

PWM has several uses:

- Providing an analog output between 0-5V.
- Dimming an LED
- Moving servo motors
- Generating audio signals.
- Providing variable speed control for motors.

The frequency of the PWM signal is approximately 490 Hz on pins 5,6,9,10 and 980 HZ on pins 3.

100% duty cycle would be the same as setting the voltage to 5 Volts (high). 0% duty cycle would be the same as grounding the signal.

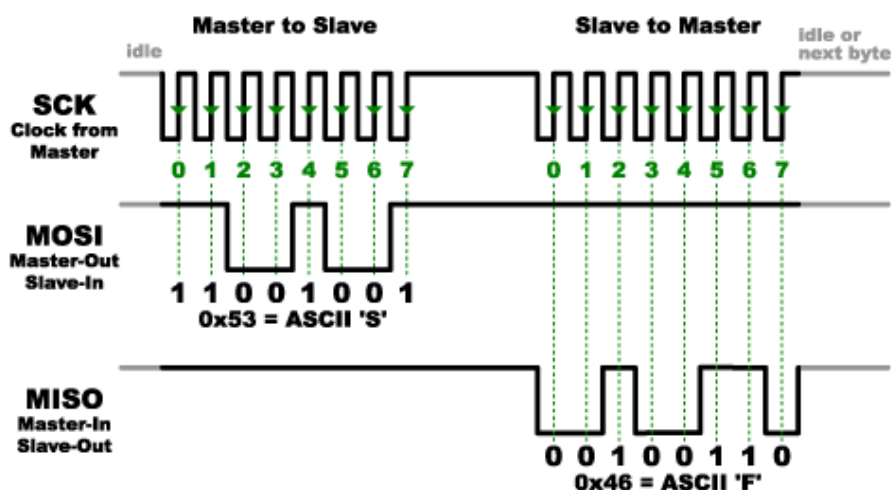


Complex Digital Communication Signals

Most communication between integrated circuits is digital. The main interfaces used within an Arduino are:

1. Parallel (lots of wires sending data)
2. Serial (Few wires and are categorised as Synchronous & Asynchronous)

All of the types transmit data via a **coded sequence of square waves** to transfer information, lots



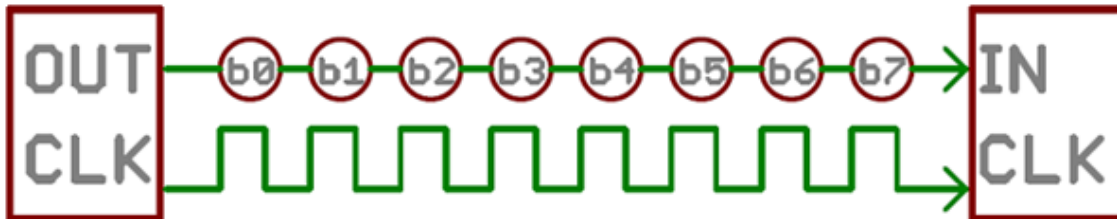
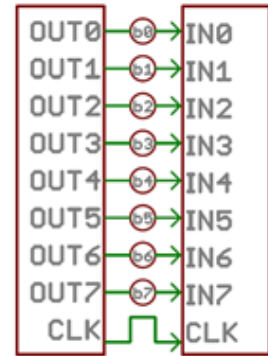
Some systems have many wires and transmit information very quickly but if speed is not as important, some use only 2 wires.

Part Dictionary

Parallel Vs Serial Communication

Parallel interfaces transfer multiple bits of data, through multiple wires at the same time. They usually require 'buses' of data – information transmitting across grouped wires of eight, sixteen, or more. As stated, data is transferred in 1's and 0's.

The example shows an 8-bit data bus, controlled by a clock, transmitting a byte every clock pulse. 9 wires are used. Very fast data can be transferred but more expensive and time-consuming to wire-up.



Example of a serial interface, transmitting one bit every clock pulse. Just 2 wires required!

Serial interfaces send their data, one single bit at a time. These interfaces can operate on as little as one wire, usually never more than four. Parallel communication certainly has its benefits. It is fast and easy to implement but requires much more input/output (I/O) lines.

Synchronous & Asynchronous Serial

The synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock signal (for signal timing). This helps for faster serial data transfer but requires at least one extra wire between communicating devices, due to the addition of the clock signal.

Where does the clock signal come from for Synchronous data signals?

The clock signal comes from the on board 16.000 MHz crystal oscillator or ceramic resonator. Both have the same function but work slightly differently.



The Pro Micro has an on board external 16Mhz ceramic resonator which is used to provide a time input to the main Pro Micro microprocessor, it essentially acts like a metronome, ticking at a constant interval and is used as a reference to keep communication signals in time.

Asynchronous means that data is transferred **without support from an external clock signal**. This means that both hardware devices which want to communicate must be set to the same data communication speed. This is called **baud rate**. The baud rate specifies **how fast** data is sent over a serial line. This value determines how long the hardware which is transmitting and receiving data for. The only requirement is that both devices to operate at the same speed. One of the more common baud rates is **9600 bps**. When using Arduino IDE software on your PC and the 8BitCADE is plugged in via the USB connector, you can send data directly to it (Pro Micro) via the Serial Monitor. Because the Serial monitor is communicating via the USB port, you must set the baud rate to 9600 on your computer (via the Arduino IDE) and set the baud rate in your sketch (program) which you load on to your 8BitCADE game console. This will synchronise data exchange without the need of a clock signal.

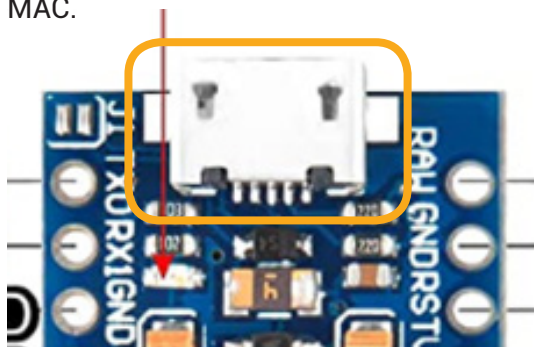
Part Dictionary

Types of Serial Protocols used on the Pro Micro

1. USB communication (Universal Serial Bus connected via USB Connector)
2. PWM Communication (Pulse Width Modulation Pins 3,5,6,9,10)
3. Serial Communication (Asynchronous Serial Pins TX & RX)
4. SPI Communication (4 wire Synchronous Serial Pins SCLK,MISO,MOSI,CS)
5. I2C Communication (2 wire Synchronous Serial Pins SDA, SCL)

USB communication (Universal Serial Bus connected via USB Connector)

The USB port your Pro Micro can be connected to the USB port of your computer, provided that the USB driver is installed (included in Arduino IDE installation). The driver causes the USB device to appear as an additional COM port on your computer. This allows the Arduino IDE software to access the Pro Micro's main microprocessor to load sketches and run programs in the same way as it would access a standard COM port. The USB port is, therefore, more suitable for user applications running on Microsoft Windows operating systems and MAC.



Essentially this is a bridge between two different types of serial communication. In the old days' computers would have a serial connector and would have been able to connect directly to you Pro Micro via a serial port. Because technology has moved on quickly and we now have USB connectivity, so we now use a USB serial interface, essentially allowing one type of serial (on your computer) to communicate with another type (on the 8BitCADE).

Bridging the gap - USB Serial > Arduino Serial

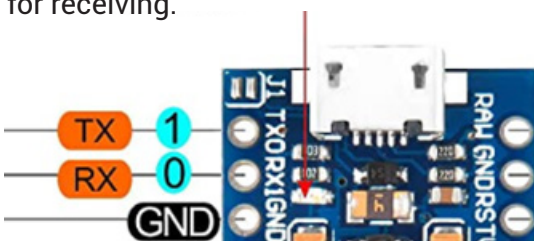
When microcontrollers and other low-level ICs communicate serially, they usually do so at a TTL (transistor-transistor logic) level. TTL serial signals must operate between a microcontroller's voltage supply range - usually 0V to 3.3V or 5V. A computer's USB connection has different voltage levels as well as other connection differences which create 'a gap in serial communication'. To bridge this gap Arduino generally use one of two options, add microchip to interface with the USB serial (FTDI Chip) and TTL serial for the main microcontroller or to use a UART processor or circuitry (can be inside the main microprocessor) to interface with the USB connection.

IMPORTANT NOTE: When connecting two serial devices, it's important to make sure their signal voltages match up.

The Pro Micro main microprocessor (ATmega32u4) includes a USB-to-UART bridge, eliminating the need for a secondary microprocessor. This allows the Pro Micro to appear to a connected computer as a mouse and keyboard (UART1), in addition to a virtual (CDC) serial / COM port (UART2).

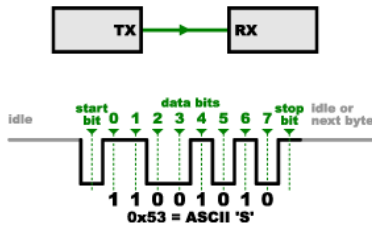
Serial Communication (Asynchronous Serial Pins TX & RX)

An asynchronous serial bus consists of only two wires - one for sending data and another for receiving.

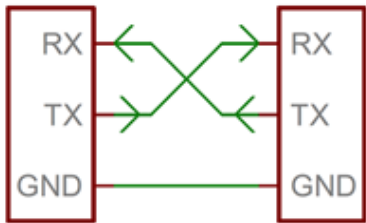


These wires are used to exchange data but because it does not make use of the clock signal from the on board ceramic resonator, there is no control over when data is sent or any guarantee that both microcontrollers are running at precisely the same speed (baud rate).

Part Dictionary



To work around this problem, asynchronous serial connections add extra information in their data to help the receiver synchronise the data as it arrives. Both microcontrollers must also agree on the transmission speed (such as 9600 bits per second) in advance.



Both serial devices should have two serial pins, one is the receiver and named, RX, and the other is the transmitter, TX. On the Pro Micro, both TX & RX pins are also connected to LEDs so that when serial communication is occurring, for example, when loading a program (sketch) onto the main microprocessor via a computer with the USB connection, the LED's will flash to show that data is being received or transmitted.

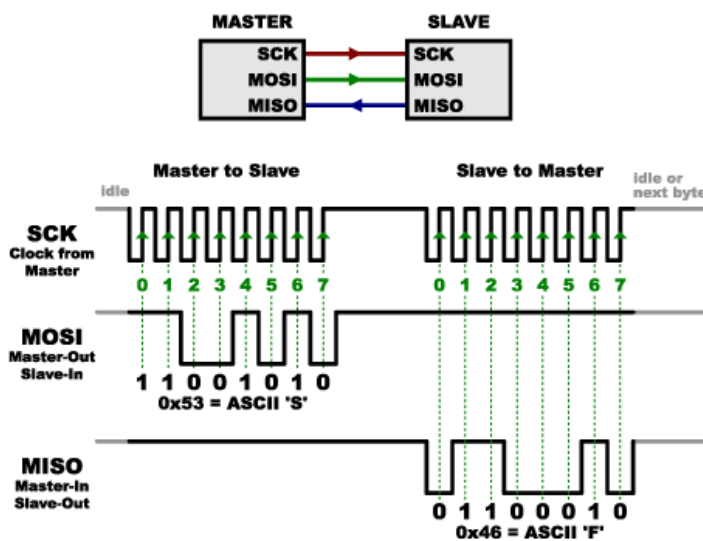
It's important to note that those RX and TX labels are concerning the device itself. So the RX from one device should go to the TX of the other,

Overview:

1. Simple layout, easy to connect to.
2. Lines must have the same transmission speed set-up (baud rate) otherwise it will not work.
3. Data bits are lost as it has additional information to help synchronise it between the transmitter and receiver (10 bits of transmission time are required for every 8 bits of data sent).

SPI Communication (4 wire Synchronous Serial Pins SCLK,MISO,MOSI,CS)

The synchronous serial bus consists of at least 4 wires - one for sending data, one for receiving it, a clock signal to synchronise the data transmission and a select pin to indicate when the data should be read. Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and smaller devices such as memory integrated chips and OLED's (8BitCADE screen). It is very reliable as it is highly synchronised using the clock for signal timing, separate data lines and the select line to control when signals are read by each device. Because the timing signal comes from one device, this device is referred to as the master and the other devices are slaves. SPI is fast, it can send and receive data at the same time so is very good for controlling hardware used in gaming with fast-moving graphics.



SCLK- Serial Clock, comes from the ceramic resonator and is used for data transmission timing.

MISO- Master In Slave Out, receives data on this line

MOSI- Master Out Slave In, sends data on this line

CS- Control Select or Slave Select, this controls when data is read which is normally activated by sending a low signal to the line (as it is normally held high).

Part Dictionary

Overview:

1. Fast transmission rate
2. Highly synchronised data exchange.
3. Lots of wiring or connections, especially if you have multiple slaves!
4. Easy to program using commands in libraries in Arduino.
5. Slaves can't communicate directly with each other, all information must go through the Master.

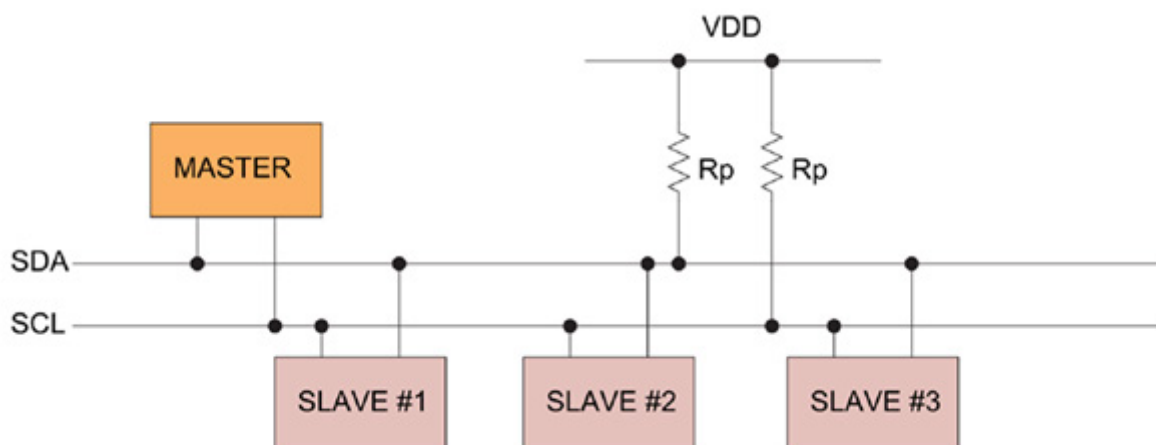
I2C Communication (2 wire Synchronous Serial Pins SDA, SCL)

The Inter-Integrated Circuit (I2C) protocol was created to enable multiple "slave" digital integrated circuits to communicate with one or more other chips irrespective of whether they are master or slave. The main problem with SPI is that you have multiple slave IC's having to communicate with a master (the hardware with the clock signal) instead of communicating with each other. You also have increased electrical connections with the addition of CS/SS pins to select the hardware to read the data. But of course, the data is processed very quickly.

Important: When the data transmission rate is not important and needs to be easily shared within a network of many integrated circuits, I2C become highly effective.

I2C bus consists of two signals: SCL which is the clock signal, this is generated by the master (Pro Micro in this case) and SDA which is the data line. All information is shared on the 2-wire network. Resistors are used to hold the signal lines high (5V). To start communication on the line the master will pull the SDA line low, advising all devices that transmission of data is now possible. Data is sent using an address which corresponds to either one of the slaves or the master device. The device with the address will read the data while all other devices will ignore it. Devices on the 2-wire network send, receive and ignore data, making it very easy to add hardware and remove it with little addition of wiring or programming.

Important: Voltages on the network should ideally be the same – either 5V or 3.3V, if they are not, the signal voltage level will need shifting (an IC to help match the voltage levels) must be used to bring them in line, this is called **level shifting**.



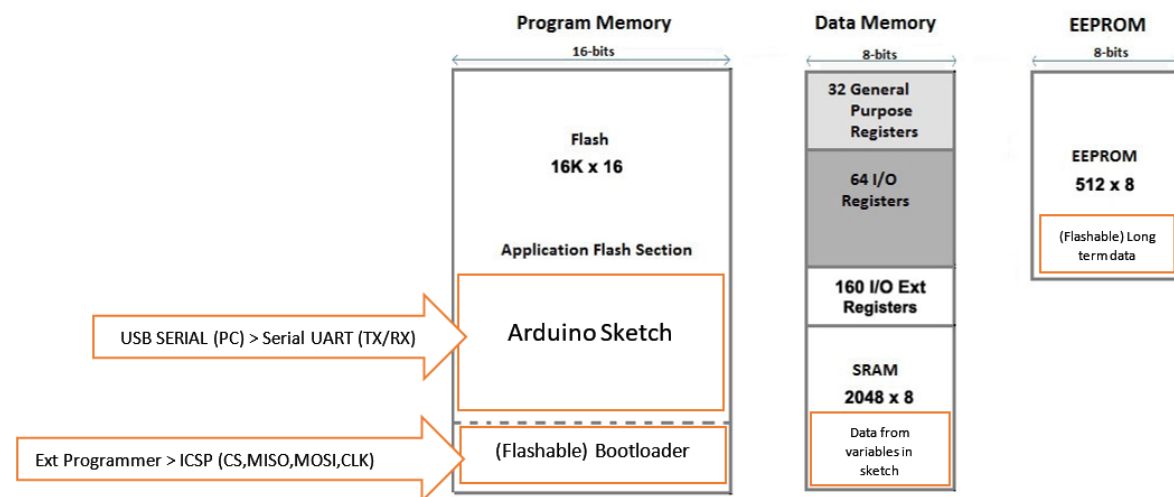
Part Dictionary

ATmega 32U4 has three types of memory

The chip contains 32 kilobytes of internal flash memory, 1 kilobyte of EEPROM and 2.5 kilobytes of SRAM. The flash and EEPROM memory are rewritable electronically, using the Arduino IDE. The information is written to the memory and remains (called volatile) after the chip is powered down (switched off) but the SRAM is a memory which only saves information while power is supplied (called non-volatile) and when the power removed all the information is erased. The program memory contains your sketch you loaded via the Arduino IDE and the bootloader, which is a short, protected program that runs when you turn the chip on, or press the reset button. Its main function is to wait for the Arduino software on your computer to send it a new program which it then writes to memory. The bootloader is an important sketch and is positioned at the end of Program Memory to protect it and can only be rewritten using ICSP (In Circuit Serial Programming).

When we refer to “boot loading” the ATmega 32U4 chip, we are talking about using a special device (called an In-Circuit-Serial-Programmer or ICSP) to replace the bootloader software.

Overview of the Different Types of Memory of an AVR Chip

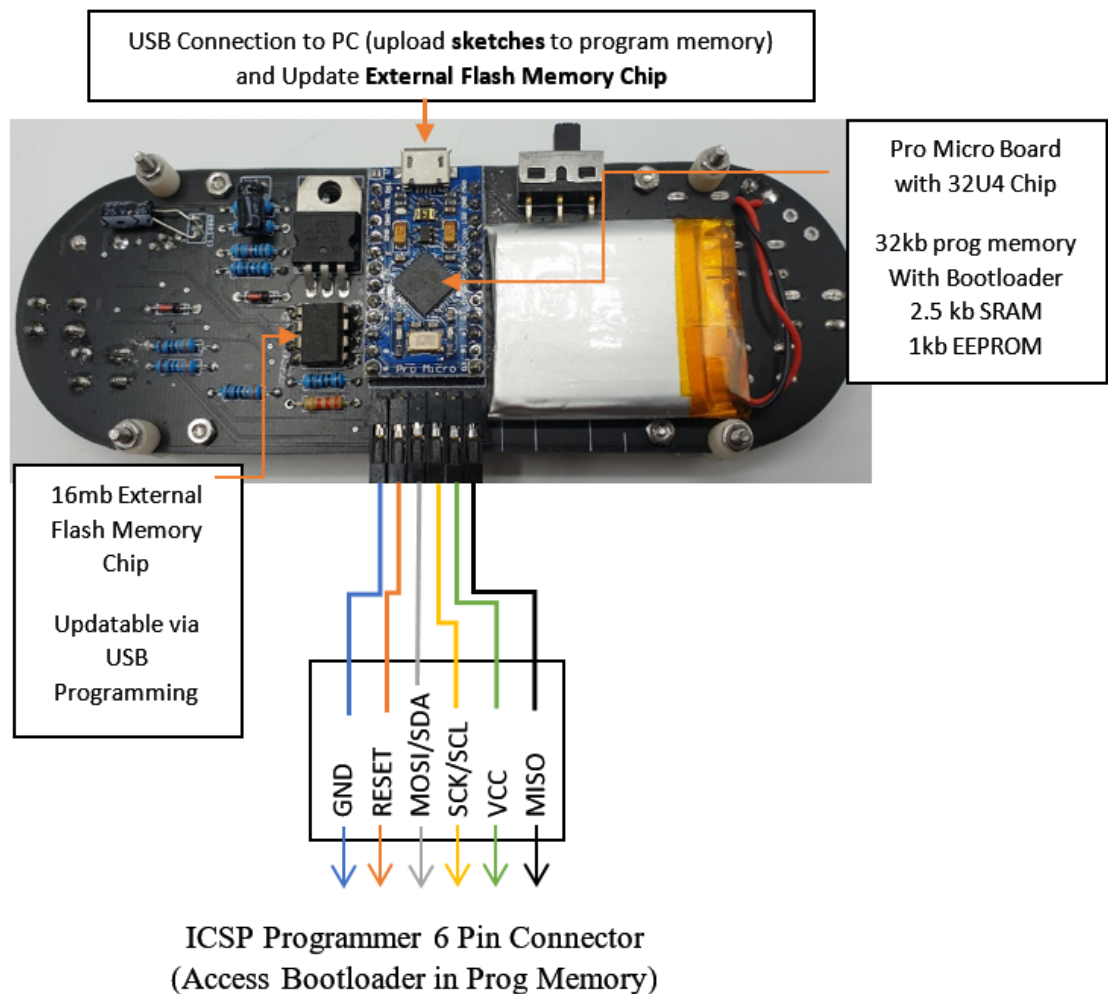


Program Memory - 32KB nonvolatile memory. This is used for storing your sketch (program) and the bootloader. This code remains in the memory of the chip even if the power is removed from the circuit. The program or actual sketch code is loaded via an FTDI chip (a UART Bridge) converting your PC USB serial to TTL serial the AVR chip can read. Boot Flash Section (Bootloader) enables you to program the Arduino board easily using a PC attached via a USB connector. Its protected and is only accessible via special pins (SPI) if you want to reprogram it using ICSP.

SRAM Memory -2.5 KB volatile memory. This is used for storing data from your sketch variables while the program is running. Once the power is turned off this data is lost. A good example of the type of data might be the position of a player on the screen which needs to be tracked so the position can be updated, like the ball in the game 'Pong'.

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. A good example of data stored here would be high scores of a game, small amount of data which would need to be saved to be displayed for later use.

Part Dictionary



Re-Programming the Bootloader - Introduction

There are two ways to program the main microcontroller (ATmega 32u4 chip) on the 8BitCADE XL. One is to reprogram the entire chip using a specialist hardware called an AVR ICSP Programmer. The other is to use a bootloader that is pre-programmed onto the main microcontroller that allows the chip to re-program itself.

The bootloader is basically a .hex file that runs when you turn on the microcontroller. It is very similar to the BIOS that runs on your PC. It does two things. First, it looks to see if the computer is trying to program it. If it is, it takes the program from the computer and uploads it into the ICs memory (in a specific location so as not to overwrite the bootloader). This is why when you try to upload code, the Arduino IDE resets the chip. This basically turns the chip off and back on again so the bootloader can start running again. If the computer isn't trying to upload code, it tells the chip to run the code that's already stored in memory. Once it locates and runs your program, the Arduino continuously loops through the program and does so as long as the board has power.

Programming the chip with an updated bootloader will add access to a menu system on the external serial flash memory chip which can store up to 500 games. Arduino chips (AVR chips) like the ATmega 32u4 chip on the Pro Micro, generally have the bootloader added at the factory. Its protected so that new users or beginners don't overwrite it by mistake while programming the chips using software like the Arduino IDE. To reprogram it with a new bootloader we need to use the SPI protocol, SPI connections and an AVR programmer like a USB ASP (USB Tiny) or ICSP Programmer (like the DIYMORE Shield).

Part Dictionary

Procedure

MR.Blinky created the Arduboy-homemade-package for homemade Arduboy (we use this to program our 8BitCADE XL). His package includes the board drivers and library of Arduboy that works with different versions of the original Arduboy like the 8BitCADE XL.

To achieve this you need 2 things:

1. Correct set-up in the Arduino IDE for the Bootloader
2. Correct connections between an ICSP Programmer and the 8BitCADE giving access to the bootloader memory inside the ATmega 32u4 chip on the main microcontroller.

(See video in learn section for a visual guide)

Correct Set-up for Arduino IDE

1. Browse to MR.Blinky's GitHub folder for the homemade Arduboy.

<https://github.com/MrBlinky/Arduboy-homemade-package>

2. Follow the instructions on the GitHub to configure your Arduino IDE with the homemade package or follow the instructions below.

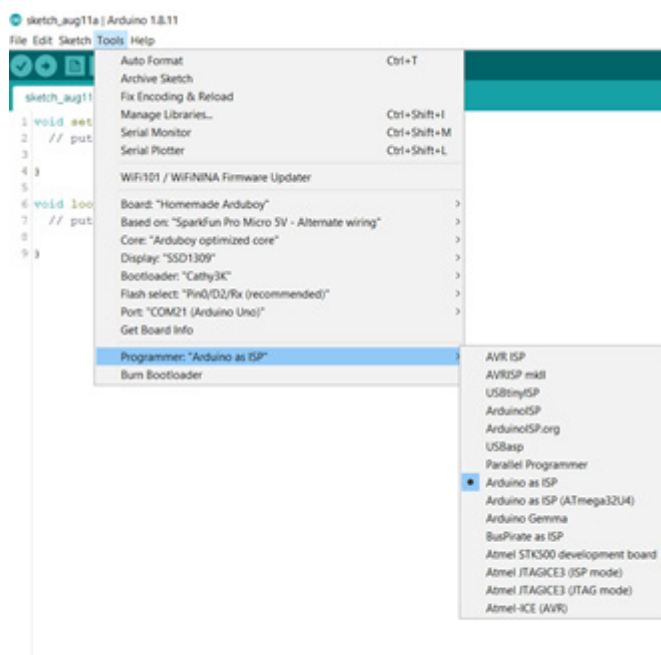
3. First copy the url of the "Additional board manager" for Arduboy homemade package.

https://raw.githubusercontent.com/MrBlinky/Arduboy-homemade-package/master/package_arduboy_homemade_index.json

4. Start Arduino IDE. Click Preferences from the Arduino top menu. Paste this text into the "Additional Boards Manager URLs" Note: If you already have other text on this field, insert this additional text at the beginning, then add a ", " and keep the other text intact.

5. Exit Arduino IDE and start the IDE again to take effect of the change above.

6. Click Tools > Board: Board Manager. Enter homemade to search. Select to install the Arduboy homemade package by Mr.Blinky. Then click update to get the latest version. The package will be added to Arduino.



7. Now select Tools>Board: "**Home-made Arduboy**" and select the following parameters for Homemade Arduboy based on:

Board:

"SparkFun Pro Micro 5V" - "alternative wiring"

Core:

"Arduboy Optimized core"

Bootloader:

"Cathy3K"

Programmer:

USBasp or Arduino as ISP (if using DIYMORE Shield)

Part Dictionary

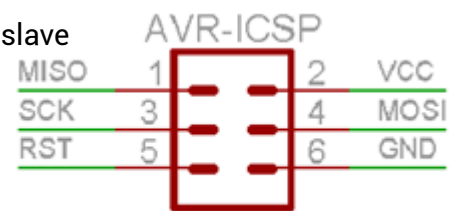
8. WARNING: Turn off the 8BitCADE – the unit must only be powered by your AVR ICSP Programmer. Attached the cables from the AVR Programmer (we use DIYMORE Shield) IC-SP_6Pin connector or other AVR ICSP Programmer and connect them as follows:

Correct Connections Between AVR ICSP Programmer & 8BitCADE XL with explanation

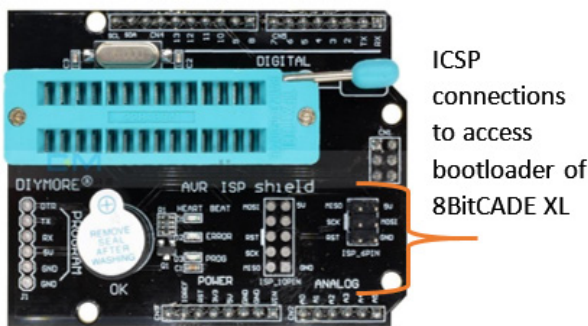
As previously mentioned, to access the bootloader section of your program memory of your 8BitCADE XL, we need to use an AVR 'In Circuit Serial Programmer'. This will help you get access to all the program memory of your 8BitCADE Microcontroller (Pro Micro with the 32u4 chip) including the bootloader. This programming can actually be done in a number of different ways, although the principles are the same, we need to use another microcontroller to use SPI to program the bootloader on the microcontroller inside the 8BitCADE XL. At 8BitCADE, we use an AVR In-Circuit-Serial-Programmer by DIYMORE as we can use this for other operations like programming ATmega chips!

The ICSP (In-Circuit Serial Programming) header pins connects the SPI (Serial Peripheral Interface) bus and we use the SPI protocol to transfer the data from the Arduino IDE on your PC to the 8BitCADE via the ICSP Programmer. See the Make Guide, part dictionary for information on SPI. The pins for the ICSP 6PIN header are as follows:

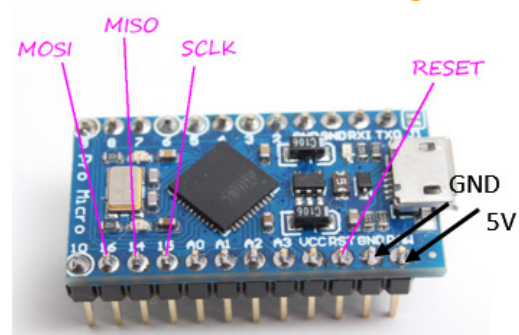
- 1 - MISO - Master Input, Slave Output - output from slave to master
- 2 - VCC - 5V
- 3 - SCK - Serial Clock - keeps the communicated data in sync
- 4 - MOSI - Master Output, Slave Input - output from master to slave
- 5 - RST - Reset
- 6 - GND - Ground



Connections on Microcontroller



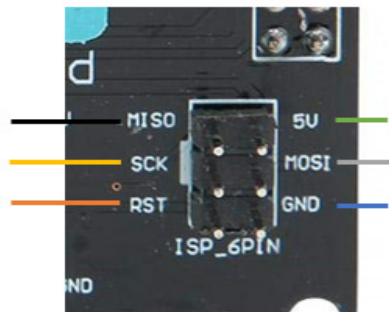
In-Circuit-Serial-Programming (ICSP)



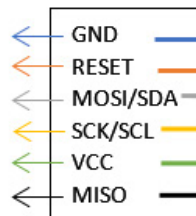
The connections on the microcontroller have been broken out to the main ICSP connector on the 8BitCADE XL Board for easy connection. See below.

Part Dictionary

How to connect to the 8BitCADE XL using the ICSP Connector



ICSP Programmer 6 Pin Connector



09. Connect AVR ICSP Programmer (DIYMORE Shield) to your PC via the USB connection. This will connect the ICSP Programmer to the Arduino IDE.

10. Your 8BitCADE should be powered on now through DIYMORE Shield ISP_6Pin connector or other ICSP Programmer.

11. Click the Tools> Boards>Burn Bootloader button on the Arduino IDE.

12. Check the message to see if the bootloader burn is successful.

13. If not, check the cable and make sure your connections are correct.

14. If successful, the 8BitCADE XL will show the following screen: USB Boot Logo (see picture) all you need to do now is disconnect the AVR Programmer, turn the unit on and it should reboot to the 8BitCADE Splash Screen.



USB Boot Logo in the middle of the screen shows you have successfully programmed the boot loader.

Compile and upload single games to 8BitCADE using Arduino IDE

Games for Arduboy can be downloaded from the following sources:

- <https://community.arduboy.com/c/games>
- <https://github.com/topics/arduboy-game>
- Erwin: Arduboy collections
- <http://arduboy.ried.cl/>

You can download the source code of the game that you can load to Arduino and upload to the 8BitCADE. See the video on how this is done: 8bitcade.com/game/8bitcadexl

Compile and upload single games to 8BitCADE in hex file format

Hex file is a text file containing binary codes resulting from the compilation of your Arduino program (sketch), but represented in a text file format using two digit hexadecimal numbers 0-9, A-F.

You can get these hex file in different ways.

1. We can download hex files from the different sources we explained above:
2. Alternatively, you can make your own hex file.

Open a game in the Arduino IDE, go to >Sketch >Export Compiled Binary. Your sketch will be compiled, then a copy of the compiled .hex file will be output to the directory of your sketch. Browse on your PC to the sketch folder or in the IDE select Sketch>Show Sketch to view the code.

Folder to see the hex file.

If you installed MR.Blinky homemade package, two versions of the .hex file will be created.

For example, if you compile the picovaders.ino sketch, the following two .hex files will be created.

1. picovaders.ino-arduboy-promicro-ssd1306.hex
2. picovaders.ino with_bootloader-arduboy-promicro-ssd1306.hex (this one is not needed and can be deleted)

Arduboy Hex File Uploader to program 8BitCADE with Hex Files

We will only use the first file: picovaders.ino-arduboy-promicro-ssd1306.hex to upload hex file to 8BitCADE. To do this you need to use an uploader. There are many on the internet. For Example, MR.Blinkys uploader is very simple to use. Browse to <https://github.com/MrBlinky/Arduboy-Python-Utilities> and follow the instructions there to install MR.Blinkys Arduboy Python utilities. If you do not have a python installed, you need to follow the instruction to install python and the required python modules first.

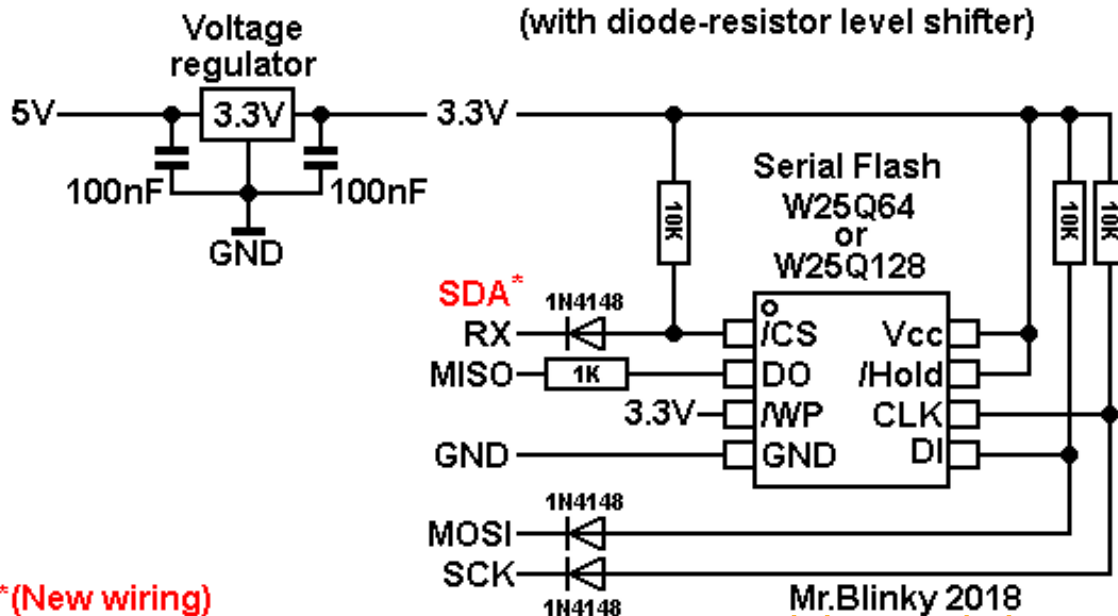
1. Connect the Arduboy to the USB port of your computer. Switch on the 8BitCADE.
2. Start the shell program in your operating system e.g. the terminal app in Mac OSX or the command prompt in windows to type the following commands to upload the hex file to the 8BitCADE.
3. Taking our previous sketch picovaders.ino as an example. python uploader.py picovaders.ino-arduboy-promicro-ssd1306.hex
4. Once the game is uploaded, the Arduboy will reset and start the game.

Part Dictionary

Write games to serial flash Chip - Hardware requirements

Arduboy flash cartridge

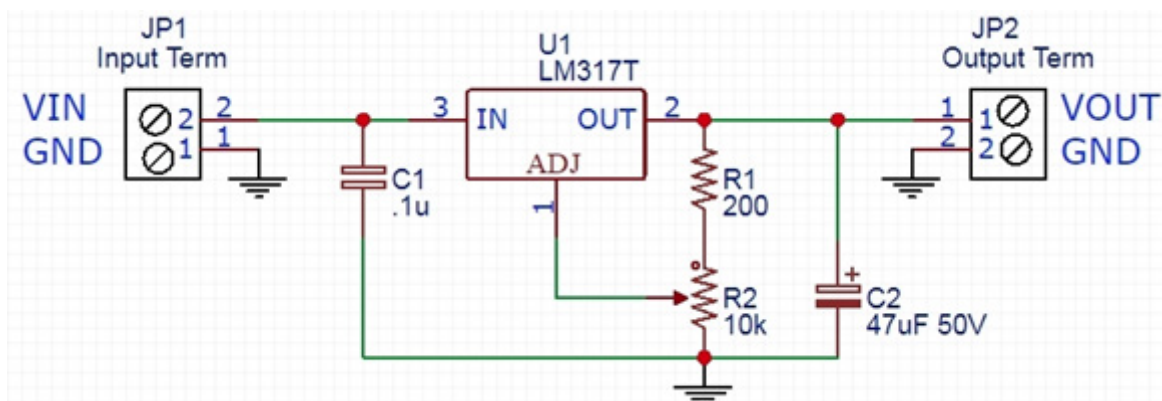
(with diode-resistor level shifter)



*(New wiring)

Mr.Blinky 2018
1N4148 Diode not required on SCK line

LM1117T Adjustable Voltage Regulator 3.3V output



Writing Games to Serial Flash Chip

(Also see video for support in game Section of Website)

To write the consolidated game file to serial flash, you need to use MR.Blinky Arduboy Python Utilities again. You should have this already installed if you follow the previous step. Otherwise, browse to <https://github.com/MrBlinky/Arduboy-Python-Utilities> and follow the instructions there to install MR.Blinky Arduboy Python Utilities.

1. Follow the instructions to install python and the required python modules first.
2. Create the index file for the consolidated game image file to hold as many as 500 games.

Part Dictionary

We will use the 'Use the flashcard-builder.py' script to build a consolidated game image files for all the games you want to store into the serial flash of the Arduboy. A 16MB Serial flash can hold as many as 500 games. This script builds a binary flash image from an index file (.csv) and the following 2 files for each game:

1. **hex files** that is the text file containing the hexadecimal codes of the binary images of the compiled Arduboy games. Refer to the 'example-flashcart\flashcard-index.csv file' for example syntax. This file is included in the package if you click Clone or Download.

2. **png graphical image files** to be displayed on the bootloader menu on Arduboy so you know which game you will be selecting. Some of the game repositories will have this graphical file together with their game source file or hex file. If you cannot find it, you can create your own by either running the game on the Arduboy emulator (<https://felipemanga.github.io/ProjectABE/>) and capture the screen. Or just type the name of the game on a power point, then screen capture that. Then the graphics you captured/created down to 128x64 pixel using paint brush in windows, or preview in Mac OSX. The YouTube video also explains how to put things the right place of this.csv index file.

One thing to note, the examples.csv file from MR.Blinkys GitHub is used in Windows PC, backslash\ are used in the pathnames. If you are using a Linux system or MAC OSX you need to change it to /.

To get a quick start, you can download my package of 63 games from:
<https://github.com/cheungbx/ArduBaby63games.zip>

This package contains the hex files and .png files of the 63 games I have chosen, plus the **games.csv index** file and the games-image.bin file that is built using the flashcard-builder.py script. You can add more games into games.csv and build your own consolidated game binary image file to be written to the serial flash.

You can put max 500 games on the 16M serial flash. I will explain how to make the.csv file using the games.csv that you can download from my GitHub. Even though the.csv file can be opened using excel. DO NOT use excel to open the file. It will corrupt the file. Please use a plain text editor only. You can use notepad in windows.

The first line of the.csv file is the header you can ignore. List;Discription;Title screen;Hex file

The second line point to the graphical image file (must be 128x64 pixel in png file format) for the boot loader menu screen. Bootloader is named in example-flashcarts> arduboy_loader.png.

The games are configured **starting from the third line**. Games are organised into groups in the bootloader menu called categories. This line is the group title of the list of games for that group e.g. Action Game. It also points to the graphical image file for the group of games.

The beginning denotes group number 1. All games that follows this group will start with this number.

1;Action Games;category-screens\Action.png. **Then you add one line for each game within that group.**

Starting with group number 1, name of the game, and the path of the graphic file for a snapshot of the screen, and the path of the hex file. All separated by " " "Add one more " " to skip the parameter for the save file.

Part Dictionary

Example flash cart

```
List;Discription;Title screen;Hex file;Data file;Save file
0;Bootloader;arduboy_loader.png;;;
1;Action Games;category-screens\Action.png;;;
1;SanSan;Action\Sansan.png;Action\Sansan.hex;;
2;Arcade Games;category-screens\Arcade.png;;;
2;1943;Arcade\Nineteen43.png;Arcade\Nineteen43.hex;;
2;Ardu-Whack;Arcade\Ardu-Whack.png;Arcade\Ardu-Whack.hex;;
3;Platformer Games;category-screens\Platformer.png;;;
3;CastleBoy;Platformer\CastleBoy.png;Platformer\CastleBoy.hex;;
4;Puzzle Games;category-screens\Puzzle.png;;;
4;Hangman;Puzzle\Hangman.png;Puzzle\Hangman.hex;;
4;LATE;Puzzle\LATE.png;Puzzle\LATE.hex;;
4;Minesweeper;Puzzle\Minesweeper.png;Puzzle\Minesweeper.hex;;
5;Racing Games;category-screens\Racing.png;;;
5;Ard-Drivin;Racing\Ard-Drivin.png;Racing\Ard-Drivin.hex;;
6;RPG Games;category-screens\RPG.png;;;
6;Rick & Morty;RPG\Rick-and-Morty.png;RPG\Rick-and-Morty.hex;;
7;Shooter Games;category-screens\Shooter.png;;;
7;Night Raid;Shooter\Night-Raid.png;Shooter\Night-Raid.hex;;
```

The last line has a save file in the parameter which is a cartoon movie.

To build the consolidated game image file, type the command, where games.csv is your game index file. `python flashcart-builder.py games.csv`. This will create a file named `games-image.bin`. Write the consolidated game image file to 8BitCADE. We use [MR.Blinkys flashcart-writer.py script](#) to write the consolidated game image file to the serial flash memory of the 8BitCADE.

If you are using my sample `games-image.bin` file you can type this command. `python flashcart-writer.py games-image.bin`

If you are using an [SSD1309 OLED](#) screen instead of the SSD1306 OLED on the standard build, you can patch the screen driver on the fly. To automatically apply the SSD1309 patch to the uploaded image, make a copy of `flashcart-writer.py` and rename it to `flashcart-writer-1309.py`. Then type `python flashcart-writer-1309.py games-image.bin`

Play games from serial flash

To play games from serial flash, switch on the 8BitCADE. If you already have a game loaded, the game will start automatically. Press the reset button on the top of the 8BitCADE once to go to the bootloader menu.

The bootloader menu will be displayed. The RGB LED will light up in sequence. If you see an icon that looks like a USB port displayed instead, that means your serial flash memory chip is not working. Pls check the wiring.

If you do not press any keys within 12 seconds, the game already stored in the ATmega32U4's internal flash memory will be run. To go back from a game to the bootloader menu, just press the Reset button once. You can press the left or right button to scroll through the different category (group) of games. Press the down or up button to scroll through the games within a category (group). Press B button to copy the game from the serial flash memory onto the ATmega32U4s internal Flash memory. The game will start within a second.

Part Dictionary

Now you have a tiny game console that you can play on the road. I challenge you to collect and load up your 16M Serial flash with 500 games. I haven't seen anyone who's done that yet to fill up the serial flash. If you can do that, do share that consolidated game file with us. MR.Blinkys GitHub link for python utilities for game upload and serial flash memory operations.

<https://github.com/MrBlinky/Arduboy-Python-Utilities>

Extract from Blinky Python:

Installing dependencies

- Download and install python 2.7.x from <https://www.python.org/downloads/> if it is not already installed
- Make sure the option 'Add python.exe to path' is checked on install options (Windows)
- After install run 'python -m pip install pyserial' from command line. For OSX run 'easy_install pyserial' from terminal.

Note: Not all utilities work with Python 3.7.x yet.

Uploader

- Works with both Python 2.7.x AND 3.7.x
- Requires pySerial: python -m pip install pyserial
- .Hex file and .Arduboy uploader for Arduboy
- Double click the uploader-create-send-to-shortcut.vbs for right click Send to upload option(Windows only)

Features

- Supports uploading to Arduboy, DevKit, and homemade Arduboys
- Uploads .hex files, .hex files in .zip and .arduboy files
- Protects unprotected bootloaders from being overwritten by large hex files
- Supports on the fly patching for SSD1309 displays
- Supports on the fly RX and TX LED polarity patching for Arduino /Genuino Micro

Usage:

- Right click a .hex file, .arduboy file or .zip file containing a hex file and choose Send To Arduboy uploader
- Drag and drop .hex, .zip or .arduboy files on the uploader.py file
- Command line: uploader.py [filetoupload]

SSD1309 display support

To patch Arduboy hex files for use on Homemade Arduboy's with SSD1309 displays, make a copy of uploader.py and rename it to uploader-1309.py Also make sure you run the uploader-create-send-to-shortcut.vbs again to create a Send To shortcut for it. Files will be patched on fly, original files will not be altered.

reverse RX and TX LED polarity support

To patch Arduboy hex files for use with Homemade Arduboy's based on Arduino / Genuino Micro, make a copy of uploader.py and rename it to uploader-micro.py and run the uploader-create-send-to-shortcut.vbs again to create a Send To shortcut for it.

EEPROM backup

You can backup your Arduboy's EEPROM by double clicking the eeprom-backup.py python script. The backup is saved to a time stamped file in the format eeprom-backup.py-YYYYMMDD-HHMMSS.bin

EEPROM restore

You can restore a previously made EEPROM backup simply by dragging the eeprom-backup.py-YYYYMMDD-HHMMSS.bin file onto the eeprom-restore.py python script

Part Dictionary

EEPROM erase

Erases the EEPROM content (An erased EEPROM contains all 0xFF's).

Erase sketch

Erases the application/sketch start-up page to keep the bootloader mode active indefinitely. This solves problematic (time sensitive) uploads using Arduino IDE.

Flash cart builder

- Works with both Python 2.7.x AND 3.7.x
- Requires PILlow: `python -m pip install pillow`

Builds a binary flash image from an index file and supporting resource files (.png images and .hex files). Use the `flashcart-writer.py` script to write the output to a flash cart. See the `example-flashcart\flashcart-index.csv` file for example syntax.

example: `python flashcart-builder.py example-flashcart\flashcart-index.csv`

Flash cart writer

- Works with both Python 2.7.x AND 3.7.x
- Requires pySerial: `python -m pip install pyserial`

Writes a binary flash image to external flash memory of Arduboy FX and Arduboy (clones) with added serial flash memory (Cathy3K v1.3+ bootloader required). Use the `flashcart-builder.py` script to build the image. To automatically apply the SSD1309 patch to the uploaded image, make a copy of `flashcart-writer.py` and rename it to `flashcart-writer-1309.py`.

example: `python flashcart-writer.py example-flashcart\flashcart-image.bin`

For development purposes external program data and save data can be stored at the end of external flash memory using `-d` and `-s` switches.

example: `python flashcart-writer.py -d datafile.bin`

Flash cart backup

- Works with both Python 2.7.x AND 3.7.x
- Requires pySerial: `python -m pip install pyserial`

Backup your existing flash cart to a binary image that can later be re-written to the Arduboy using the `flashcart-writer.py` script. The backup is saved to a time stamped file in the format `flashcart-backup-image-YYYYMMDD-HHMMSS.bin`

example: `python flashcart-backup.py`

Image Converter

- Works with both Python 2.7.x AND 3.7.x
- Requires PILlow: `python -m pip install pillow`

Converts .bmp or .png image files to C++ include file. Image width and height can be any size. Tile sheets and sprite sheets with optional spacing can be converted by specifying the width and height and optional spacing in the filename. When an image contains transparency information the converted data will include a sprite mask. Script can convert multiple files in one go by supplying multiple filenames.

example: `python image-converter.py tilesheet_16x16.png`