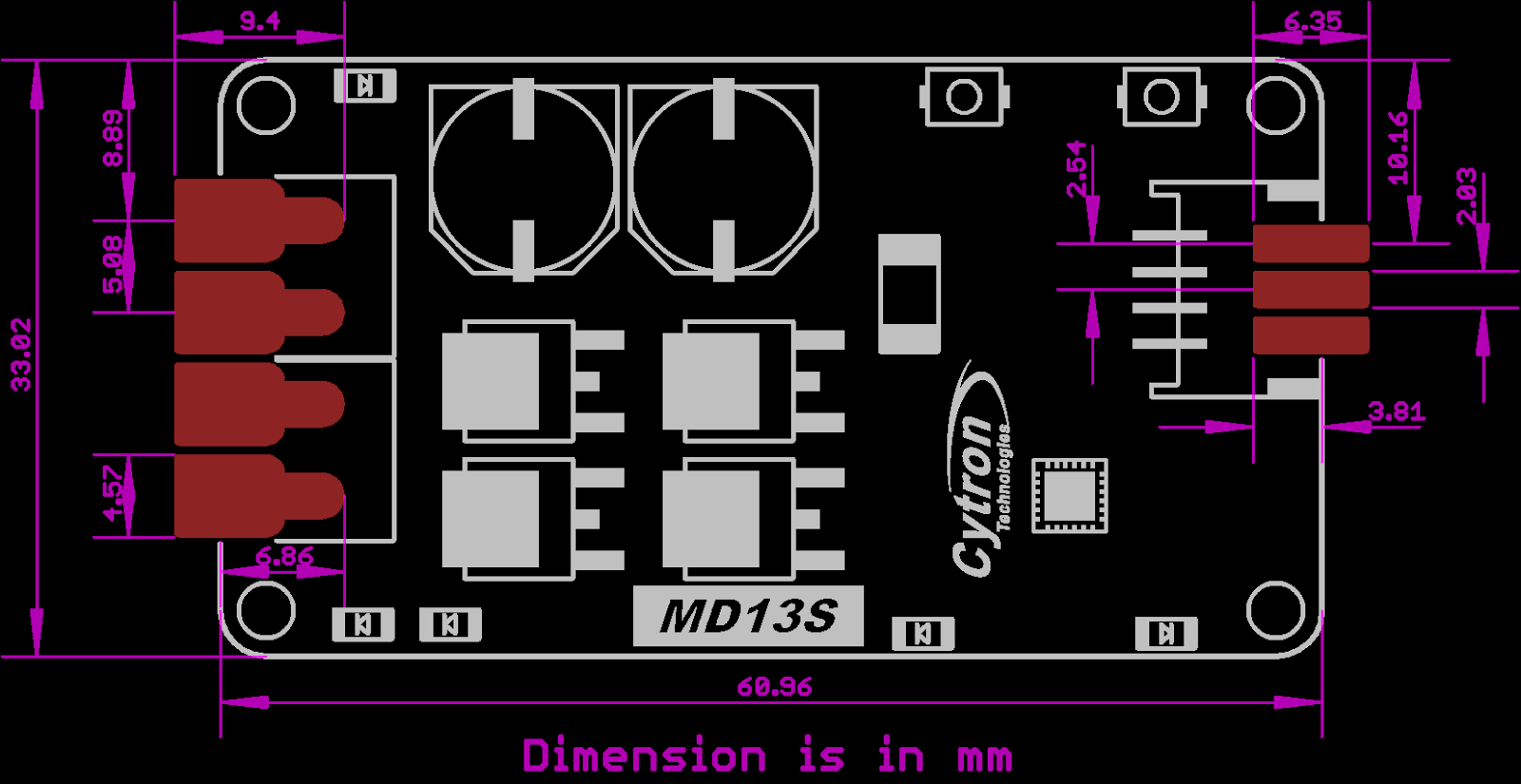
3/31/21 pin 27 and 28 can be used for I2C in future

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Pin** | **Description** | **328Pin** | **UNO** | **Description** | **Function REVC** |
|  | +5V | 1 | RESET | PC6 | Reset |
| 4 | USB TXD (or)  Program Mode button | 2 | 0 | PD0 – IN | Digital Pin (RX) |
| 5 | USB RXD | 3 | 1 | PD1 – OUT | Digital Pin (TX) |
| 37 | RC1\_input | 4 | 2 | PB0 – IN | Digital Pin |
| 38 | RC2\_input | 5 | 3 | PB1 – IN | Digital Pin (PWM) |
| 23 | BUZZER | 6 | 4 | PD2 – OUT | Digital Pin |
| 11 | VDD | 7 |  | Vcc | Positive Voltage (Power) |
| 12 | GND | 8 |  | GND | Ground |
| 13 | XTAL 1 | 9 |  | XTAL 1 | Crystal Oscillator |
| 14 | XTAL 2 | 10 |  | XTAL 2 | Crystal Oscillator |
| 21 | Windlass\_LMP | 11 | 5 | PD5 – OUT | Digital Pin (PWM) |
| 18 | Winch\_OUT | 12 | 6 | PD6 – OUT | Digital Pin (PWM) |
| 17 | Winch\_IN | 13 | 7 | PD7 – OUT | Digital Pin |
| 16 | PWM1\_direction | 14 | 8 | PD3 – OUT | Digital Pin |
| 15 | PWM1 helm | 15 | 9 | PD4 – OUT | Digital Pin (PWM) |
| 22 | Bat\_LOW\_LED | 16 | 10 | PB2 – OUT | Digital Pin (PWM) |
| 27 | Winch\_LED | 17 | 11 | PB3 – OUT | Digital Pin (PWM) |
| 28 | Helm\_LED | 18 | 12 | PB4 – OUT | Digital Pin |
| 29 | Heartbeat\_LED | 19 | 13 | PB5 – OUT | Digital Pin |
| ?? | AVCC | 20 |  | AVCC | Positive voltage for ADC (power) |
| 5 | Vref joystick | 21 |  | AREF – IN | ADC Ref Voltage = Vcc |
| 31 | GND | 22 |  | GND | Ground |
| 2 | Joystick helm | 23 | A0 | PC0 – IN | Analog Input |
| 3 | Joystick wind | 24 | A1 | PC1 – IN | Analog Input |
|  | S/P helm | 25 | A2 | PC2 – IN | Analog Input |
|  | S/P wind | 26 | A3 | PC3 – IN | Analog Input |
|  | Bat sensor | 27 | A4 | PC4 – IN | Analog Input SDA |
| 4 | MODE | 28 | A5 | PC5 – IN | Analog Input SCL |



**Remote RC: RadioLink T8S transmitter and R8EF Receiver**

T8S transmitter must be configured via the RadioLink IPhone App.

Channels: For T8S, channels ONE (RC1 HELM) and THREE (RC2 WINCH) are connected to Mk VI PCB ports. Note that polarity of both T8S channels must be REVERSED.

RX Failsafe Mode: As long as it is powered by the mk VI, the R8EF cannot be configured to stop transmitting pulses to the Mk VI, whether or not a TX is present. Failsafe Mode (F/S) is set at 50% PWM. When TX is turned OFF (or out of range), PWM will be NEUTRAL and not interfere with joystick/sip and puff commands.

NOTE: while using RadioLink App, settings can only be changed when TX is disconnected (go figure!). 1) READ configuration from TX, 2) then DISCONNECT, 3) make changes, 4) then RECONNECT and WRITE changes to TX.

**A screenshot of a computer

Description automatically generated with medium confidence**

**ATMega328 pinout**

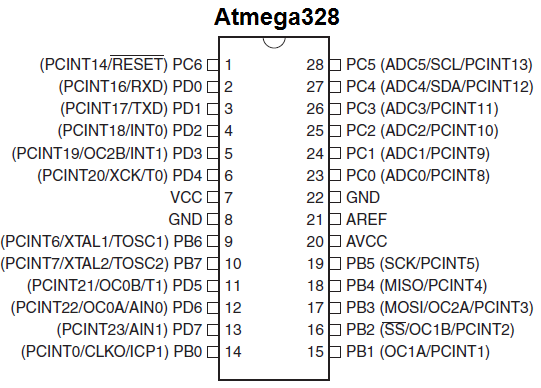
In this article, we will go over the pinout of the Atmega328 chip.

The Atmega328 is a very popular microcontroller chip produced by Atmel. It is an 8-bit microcontroller that has 32K of flash memory, 1K of EEPROM, and 2K of internal SRAM.

The Atmega328 is one of the microcontroller chips that are used with the popular Arduino Duemilanove boards. The Arduino Duemilanove board comes with either 1 of 2 microcontroller chips, the Atmega168 or the Atmega328. Of these 2, the Atmega328 is the upgraded, more advanced chip. Unlike the Atmega168 which has 16K of flash program memory and 512 bytes of internal SRAM, the Atmega328 has 32K of flash program memory and 2K of Internal SRAM.

The Atmega328 has 28 pins.

It has 14 digital I/O pins, of which 6 can be used as PWM outputs and 6 analog input pins. These I/O pins account for 20 of the pins.

The pinout for the Atmega328 is shown below.  
  


The table below gives a description for each of the pins, along with their function.

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Description** | **Function** |
| 1 | PC6 | Reset |
| 2 | PD0 | Digital Pin (RX) |
| 3 | PD1 | Digital Pin (TX) |
| 4 | PD2 | Digital Pin – EXT INT |
| 5 | PD3 | Digital Pin (PWM) – EXT INT |
| 6 | PD4 | Digital Pin |
| 7 | Vcc | Positive Voltage (Power) |
| 8 | GND | Ground |
| 9 | XTAL 1 | Crystal Oscillator |
| 10 | XTAL 2 | Crystal Oscillator |
| 11 | PD5 | Digital Pin (PWM) |
| 12 | PD6 | Digital Pin (PWM) |
| 13 | PD7 | Digital Pin |
| 14 | PB0 | Digital Pin |
| 15 | PB1 | Digital Pin (PWM) |
| 16 | PB2 | Digital Pin (PWM) |
| 17 | PB3 | Digital Pin (PWM) |
| 18 | PB4 | Digital Pin |
| 19 | PB5 | Digital Pin |
| 20 | AVCC | Positive voltage for ADC (power) |
| 21 | AREF | Reference Voltage |
| 22 | GND | Ground |
| 23 | PC0 | Analog Input |
| 24 | PC1 | Analog Input |
| 25 | PC2 | Analog Input |
| 26 | PC3 | Analog Input |
| 27 | PC4 | Analog Input - I2C SDA |
| 28 | PC5 | Analog Input – I2C SCL |

As stated before, 20 of the pins function as I/O ports. This means they can function as an input to the circuit or as output. Whether they are input or output is set in the software. 14 of the pins are digital pins, of which 6 can function to give PWM output. 6 of the pins are for analog input/output.

2 of the pins are for the crystal oscillator. This is to provide a clock pulse for the Atmega chip. A clock pulse is needed for synchronization so that communication can occur in synchrony between the Atmega chip and a device that it is connected to.

The chip needs power so 2 of the pins, Vcc and GND, provide it power so that it can operate. The Atmega328 is a low-power chip, so it only needs between 1.8-5.5V of power to operate.

The Atmega328 chip has an analog-to-digital converter (ADC) inside of it. This must be or else the Atmega328 wouldn't be capable of interpreting analog signals. Because there is an ADC, the chip can interpret analog input, which is why the chip has 6 pins for analog input. The ADC has 3 pins set aside for it to function- AVCC, AREF, and GND. AVCC is the power supply, positive voltage, that for the ADC. The ADC needs its own power supply in order to work. GND is the power supply ground. AREF is the reference voltage that the ADC uses to convert an analog signal to its corresponding digital value. Analog voltages higher than the reference voltage will be assigned to a digital value of 1, while analog voltages below the reference voltage will be assigned the digital value of 0. Since the ADC for the Atmega328 is a 10-bit ADC, meaning it produces a 10-bit digital value, it converts an analog signal to its digital value, with the AREF value being a reference for which digital values are high or low. Thus, a portrait of an analog signal is shown by this digital value; thus, it is its digital correspondent value.

The last pin is the RESET pin. This allows a program to be rerun and start over.

And this sums up the pinout of an Atmega328 chip.

**Related Resources**

[How to Use the LM741 Op Amp as a Comparator](http://www.learningaboutelectronics.com/Articles/LM741-op-amp-voltage-comparator.php)  
  
[How to Build an LM339 Quad Voltage Comparator Circuit](http://www.learningaboutelectronics.com/Articles/LM339-quad-voltage-comparator-circuit.php)  
  
[How to Build a Dark-activated Switch](http://www.learningaboutelectronics.com/Articles/Dark-activated-switch.php)  
  
[How to Build a Hall Effect Sensor Circuit](http://www.learningaboutelectronics.com/Articles/Hall-effect-sensor-circuit.php)  
  
[How to Build a Touch Sensor Circuit](http://www.learningaboutelectronics.com/Articles/How-to-build-a-touch-sensor-circuit)  
  
[How to Build an Accelerometer Circuit](http://www.learningaboutelectronics.com/Articles/Accelerometer-circuit.php)  
  
[How to Build a Motion Detector Circuit](http://www.learningaboutelectronics.com/Articles/Motion-detector-circuit.php)  
  
[How to Build a Motion Detector Alarm Circuit](http://www.learningaboutelectronics.com/Articles/Motion-detector-alarm-circuit.php)

**23. Analog-to-Digital Converter**

23.1 Features

● 10-bit resolution

● 0.5 LSB integral non-linearity

● ±2 LSB absolute accuracy

● 65 to 260µs conversion time

● Up to 15kSPS

● 6 multiplexed single ended input channels

● 2 additional multiplexed single ended input channels

● Temperature sensor input channel

● Optional left adjustment for ADC result readout

● 0 to VCC ADC input voltage range

● Selectable 1.1V ADC reference voltage

● Free running or single conversion mode

● Interrupt on ADC conversion complete

● Sleep mode noise canceler

23.2 Overview

The Atmel® ATmega328P features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel analog multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The ADC contains a sample and hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 23-1 on page 206.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3V from VCC. See Section 23.6 “ADC Noise Canceler” on page 211 on how to connect this pin.

Internal reference voltages of nominally 1.1V or AVCC are provided on-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

The power reduction ADC bit, PRADC, in Section 9.10 “Minimizing Power Consumption” on page 36 must be disabled by writing a logical zero to enable the ADC.

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB.

Optionally, AVCC or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

# Digital Pins

The pins on the Arduino can be configured as either inputs or outputs. This document explains the functioning of the pins in those modes. While the title of this document refers to digital pins, it is important to note that vast majority of Arduino (Atmega) analog pins, may be configured, and used, in exactly the same manner as digital pins.

### Properties of Pins Configured as INPUT

Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode() when you're using them as inputs. Pins configured this way are said to be in a **high-impedance state**. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megohm in front of the pin. This means that it takes very little current to move the input pin from one state to another, and can make the pins useful for such tasks as implementing [a capacitive touch sensor](https://www.arduino.cc/playground/Code/CapacitiveSensor), reading an LED as a [photodiode](https://www.arduino.cc/playground/Learning/LEDSensor), or reading an analog sensor with a scheme such as [RCTime.](https://www.arduino.cc/en/Tutorial/RCtime)

This also means however, that pins configured as pinMode(pin, INPUT) with nothing connected to them, or with wires connected to them that are not connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

### Pullup Resistors with pins configured as INPUT

Often it is useful to steer an input pin to a known state if no input is present. This can be done by adding a pullup resistor (to +5V), or a pulldown resistor (resistor to ground) on the input. A 10K resistor is a good value for a pullup or pulldown resistor.

### Properties of Pins Configured as INPUT\_PULLUP

There are 20K pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed by setting the pinMode() as INPUT\_PULLUP. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on.

The value of this pullup depends on the microcontroller used. On most AVR-based boards, the value is guaranteed to be between 20kΩ and 50kΩ. On the Arduino Due, it is between 50kΩ and 150kΩ. For the exact value, consult the datasheet of the microcontroller on your board.

When connecting a sensor to a pin configured with INPUT\_PULLUP, the other end should be connected to ground. In the case of a simple switch, this causes the pin to read HIGH when the switch is open, and LOW when the switch is pressed.

The pullup resistors provide enough current to dimly light an LED connected to a pin that has been configured as an input. If LEDs in a project seem to be working, but very dimly, this is likely what is going on.

The pullup resistors are controlled by the same registers (internal chip memory locations) that control whether a pin is HIGH or LOW. Consequently, a pin that is configured to have pullup resistors turned on when the pin is an INPUT, will have the pin configured as HIGH if the pin is then switched to an OUTPUT with pinMode(). This works in the other direction as well, and an output pin that is left in a HIGH state will have the pullup resistors set if switched to an input with pinMode().

Prior to Arduino 1.0.1, it was possible to configure the internal pull-ups in the following manner:

pinMode(pin, INPUT); // set pin to input

digitalWrite(pin, HIGH); // turn on pullup resistors

**NOTE:** Digital pin 13 is harder to use as a digital input than the other digital pins because it has an LED and resistor attached to it that's soldered to the board on most boards. If you enable its internal 20k pull-up resistor, it will hang at around 1.7V instead of the expected 5V because the onboard LED and series resistor pull the voltage level down, meaning it always returns LOW. If you must use pin 13 as a digital input, set its pinMode() to INPUT and use an external pull down resistor.

### Properties of Pins Configured as OUTPUT

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), or run many sensors, for example, but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins, or attempting to run high current devices from them, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often this will result in a "dead" pin in the microcontroller but the remaining chip will still function adequately. For this reason it is a good idea to connect OUTPUT pins to other devices with 470Ω or 1k resistors, unless maximum current draw from the pins is required for a particular application.

# analogRead()

[Analog I/O]

### Description

Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit. See the table below for the usable pins, operating voltage and maximum resolution for some Arduino boards.

The input range can be changed using [analogReference()](https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference), while the resolution can be changed (only for Zero, Due and MKR boards) using [analogReadResolution()](https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution).

On ATmega based boards (UNO, Nano, Mini, Mega), it takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

| BOARD | OPERATING VOLTAGE | USABLE PINS | MAX RESOLUTION |
| --- | --- | --- | --- |
| Uno | 5 Volts | A0 to A5 | 10 bits |
| Mini, Nano | 5 Volts | A0 to A7 | 10 bits |
| Mega, Mega2560, MegaADK | 5 Volts | A0 to A14 | 10 bits |
| Micro | 5 Volts | A0 to A11\* | 10 bits |
| Leonardo | 5 Volts | A0 to A11\* | 10 bits |
| Zero | 3.3 Volts | A0 to A5 | 12 bits\*\* |
| Due | 3.3 Volts | A0 to A11 | 12 bits\*\* |
| MKR Family boards | 3.3 Volts | A0 to A6 | 12 bits\*\* |

\*A0 through A5 are labelled on the board, A6 through A11 are respectively available on pins 4, 6, 8, 9, 10, and 12  
\*\*The default analogRead() resolution for these boards is 10 bits, for compatibility. You need to use [analogReadResolution()](https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution) to change it to 12 bits.

### Syntax

analogRead(pin)

### Parameters

pin: the name of the analog input pin to read from (A0 to A5 on most boards, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

### Returns

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: int.

### Example Code

The code reads the voltage on analogPin and displays it.

int analogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3

// outside leads to ground and +5V

int val = 0; // variable to store the value read

void setup() {

Serial.begin(9600); // setup serial

}

void loop() {

val = analogRead(analogPin); // read the input pin

Serial.println(val); // debug value

}

### Notes and Warnings

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).