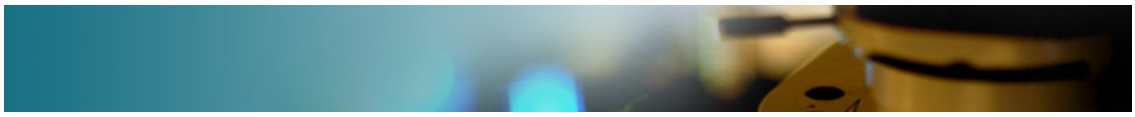
devices

Device Support

Building Micro-Manager device adapters

Free third-party tools for testing communications with hardware



Arduino

Summary:	Adapter for the **Arduino electronics prototyping platform**
Author:	Nico Stuurman
License:	LGPL
Platforms:	Linux, Mac, and Windows
Since version:	1.3.18
Automated Serial Port Setup:	Yes
Peripheral device discovery:	No
Wiki page:	Nico Stuurman Pariksheet Nanda Roy Wollman

The **Arduino** is a low cost programmable digital IO board. It has some digital inputs and digital outputs and can communicate with the computer through a serial interface (that is hidden in a USB connection). A very nice and simple programming language makes the Arduino very simple to program. The Micro-Manager interface to the Arduino consists of an Arduino program (the 'firmware') that you need to upload to the Arduino first. The current version of the Micro-Manager has facilities to use the Arduino as a shutter, and as a shutter that only opens when a TTL is set high (for instance, when the camera is exposing). Also, timed sequences of TTL output can be setup. There are also facilities for programmable analogue out, however, those need connection of a DA chip and requires constructions of a 'daughterboard'/'shield'.

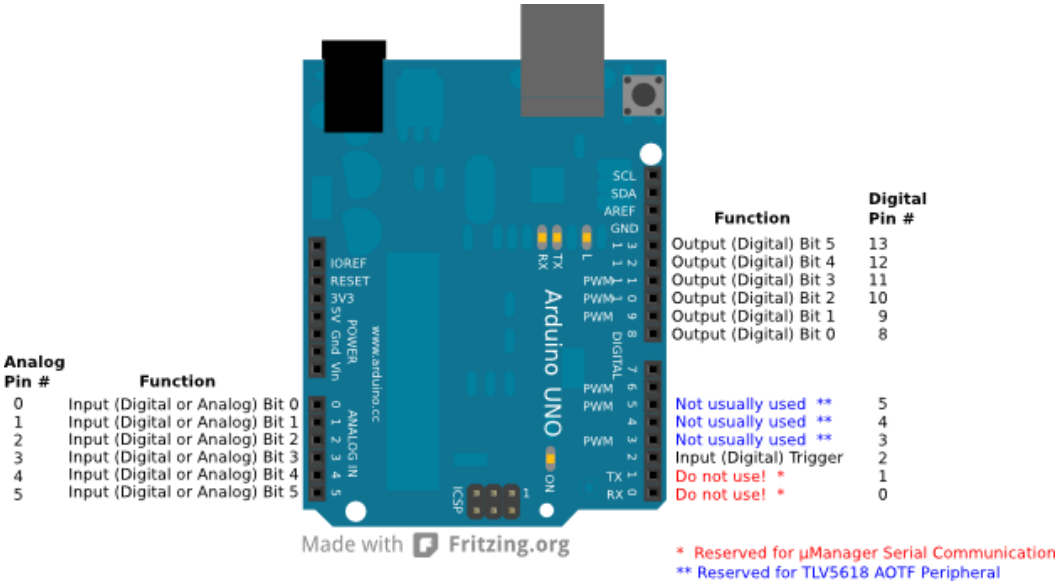


Figure: Functional pinout set by firmware version 2
(**Image source .svg file**)

Contents

Installation

Arduino Software (IDE)

The Arduino IDE is required to upload the firmware to the Arduino. Once this step is done, it should no longer be required. Download and install the **Arduino Software** for your platform. Establish a connection with your Arduino using the instructions that come with the Arduino software (this includes installing a driver for the USB to serial adapter built into the Arduino.

Firmware

For the Arduino Uno and similar boards, download the **firmware source code**. Bonno Meddens wrote compatible firmware for the ESP32, ItsyBitsy M4, and Teensy 3.x boards that can be found **here**. Copy the firmware into a blank Arduino Sketch window. Send to the Arduino, and your Arduino is programmed to work with Micro-Manager.

Hardware Configuration Wizard

Hub

Add the Arduino-Hub. You may want to set the "Verbose" property of the port to 0 to avoid extensive logging of the communcation with the device, especially if you are using the input functionality (which queries the Arduino every second).

Pre-Initialization Properties	Description
Logic	**Inverted** causes digital inputs and outputs to be considered HIGH when 0 Volts and LOW when +5 Volts.

Serial Setting	Value
AnswerTimeout	500
BaudRate	57600
DelayBetweenCharsMs	0
Handshaking	Off
Parity	None
StopBits	1
Verbose	0

> **Note** If you get the following error message when trying to *Add* the Arduino device in the Wizard: `There are no unused ports available!`, first check that the Arduino is connected to the computer. Also, under GNU/Linux systems,make sure that your user is in the `dialout` system group (**see details**).

Peripheral Devices Setup

Name	Description
Arduino-Switch	Digital output pattern set across pins 8 to 13. See usage in **#Digital IO**.
Arduino-Shutter	Toggles the digital outputs pattern across pins 8 to 13. Set all pins off when the shutter is closed, and restores the value set in `Switch-State` when the shutter is opened.
Arduino-Input	Reports, both, the digital and analog (0-1023) state of the analog input pins 0 to 5.
Arduino-DAC{1,2}	(Not usually used) Reserved for TLV5618 AOTF Peripheral. TLV5618 is a chip acting as a DAC (digital-to-analog converter) that can be added as a shield to the Arduino. This way, the Arduino can be used to control analog devices such as AOTF or stages. See **#DAC**.

Initialization Properties	Description
Pin	Reads all 6 input pins when set to **All**, otherwise reads the individual 0-5 Pin number.
Pull-Up-Resistor	Choose whether the Arduino should use its internal 20 kOhm internal pull-up resistor. If not sure, choose **On**.

Test Communication

Arduino boards Duemilanove and newer feature an LED attached to pin 13. One can verify the Arduino is communicating by changing the state of pin 13 and verifying the LED changes. Setting pin 13 HIGH requires setting the `Arduino-Switch-State` to 32 (i.e. 2^5) in the `Device/Property Browser`. Toggle the `Arduino-Shutter` in the main MM window (the output pins will only go high when the Shutter Device is open, so you will need both the Shutter and State devices).

Usage Notes

Digital IO

All possible combinations of digital outputs that can be set across pins 8 to 13 can be set by a single number from from 0 to 63, with 0 turning off all pins and 63 turning all pins on.

To choose the set of pins to be on for your `Arduino-Switch-State` pattern, simply add all the decimal values for the pin from the table below; so for pins 8 and 10 to be on, needs a decimal value of 1 + 4 = **5** to be set in `Arduino-Switch-State`.

Pin	Bit	Decimal Value when On
8	0	1
9	1	2
10	2	4
11	3	8
12	4	16
13	5	32

The reason one has to do this is MM encodes the pins as bits in binary. Pin 13 corresponds to the most significant bit 5, whose decimal value is 2^5 = 32. Similarly pin 10 will be bit 2 which has a decimal value of 2^2 = 4.

Blanking

The "blanking" feature of the device adapter allows to ensure that the sample is illuminated only when the camera is exposing. By connecting a TTL output of the camera (active when the camera is exposing) to the arduino input and enabling blanking, the digital output of the arduino will be kept at zero as long as the TTL is inactive.

Device Properties

Properties	Description
Switch-Blanking Mode	Allows **On** or **Off**: enables "blanking" when the TTL input is inactive.
Switch-Blank On	Allows **Low** or **High**: defines the active state of the TTL input.
Switch-HubID	
Switch-Label	Set the digital output pattern across pins 8 to 13. Allows values 0-63. See usage in **#Digital IO**
Switch-Sequence	**Off** operates the Arduino in normal "switching" mode. **On** enables triggered "sequencing" mode. One defines and upload multiple pattern states to the Arduino **using a Beanshell script**. Then the digital input signal to pin 2 switches between the sets of patterns. Typically, one would connect the camera exposure signal to pin 2.
Switch-State	Set the digital output pattern across pins 8 to 13. Allows values 0-63. See usage in **#Digital IO**
Input-AnalogInput{0:5}	10-bit value of ADC input.
Input-DigitalInput	Value depends on how the `Pin` initialization property was set. Shows binary state of the input pin, if a single in was selected. Shows the 5-bit value of all input pins if "All" was selected.
Input-HubID	
DAC{1,2}-HubID	(Not usually used) Reserved for TLV5618 AOTF Peripheral.
DAC(1,2}-Volts	(Not usually used) Reserved for TLV5618 AOTF Peripheral.

DAC

A *DAC* (digital-to-analog) converter is required for the Arduino to output analog signals. Indeed, the Arduino can only *simulate* analog outputs using a **PWM**. To output *true* analog output, a chip that perform a digital-to-analog conversion is required.

PWM

When a pin is configured in PWM mode (using the `analogWrite` Arduino command), the pin constantly switches between the binary *HIGH* and *LOW* values such that the temporal average of the signal is the required analog output: although the average of the voltage on a given pin is different from the *HIGH* and *LOW* values, the pin actually outputs a high frequency square wave, which can be detrimental to some devices that require a true analog signal. A way to smooth the PWM output to its average value is to use a **RC circuit**.

Furthermore, the default PWM frequency of the Arduino is relatively low: about 1 kHz. It is possible to **go up to 30-60kHz**.

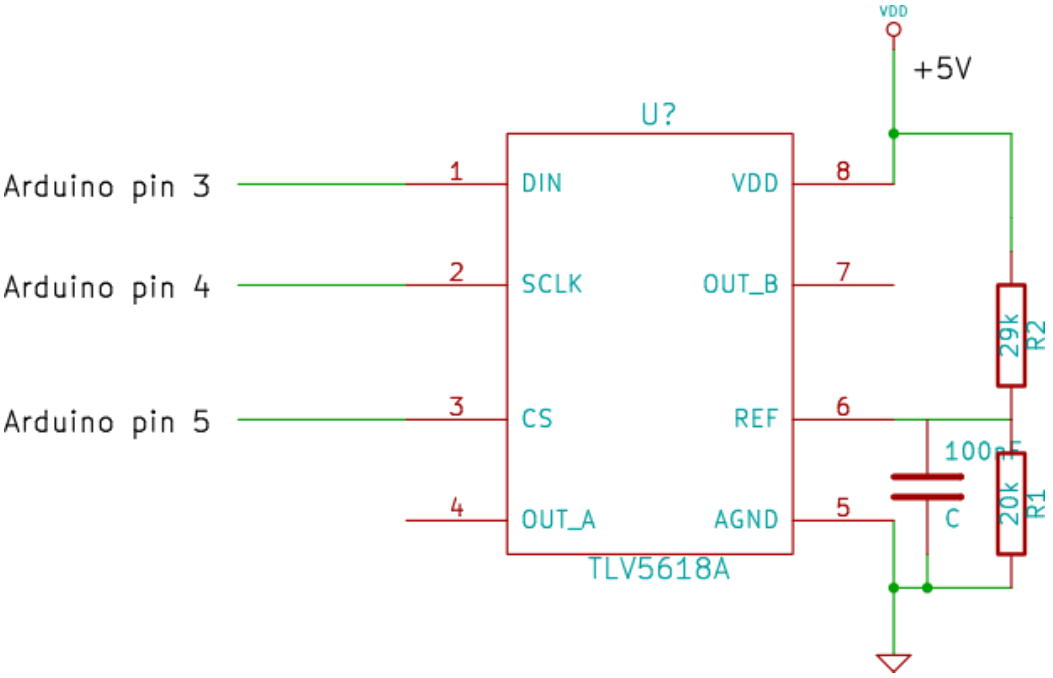
DAC - Description

One DAC chip is the **Texas Instrument TLV5618**. It is a 12 bit DAC. It is easy to handle and the Arduino firmware can talk natively to it through a builtin **SPI protocol**.

It can be wired as follows on pins 3, 4 and 5 of the Arduino and will require a few additional components:

- A decoupling capacitor (we used a 100 nF ceramic capacitor)
- Resistors/potentiometers to build a voltage divisor and power the *REF* pin of the DAC. The reference has to be set at 2.04 V. This can be achieved using a 20 kOhm resistor on one side and a 29 kOhm resistor on the other side (29 kohm can be obtained with 20+6.8+2.2 kOhm serial resistors).

Those should be wired as follow:



Notes:

- *OUT_A* and *OUT_B* are the analog outputs.
- The pair of resistors *R1* and *R2*can be tweaked as long as the voltage on the *REF* pin of the TLV5618 remains 2.04 V
- The *+5V* and *GND* should be wired to the 5V and GND pins of the Arduino

Alternative wiring

The current (as of Oct. 2016) implementation of the SPI interface in the Arduino firmware limits the speed of the DAC to approximately 1 kHz. However, it is possible to do better by using the **builtin SPI interface** of the Arduino. This can shrink the response down to 25 µs (40 kHz).

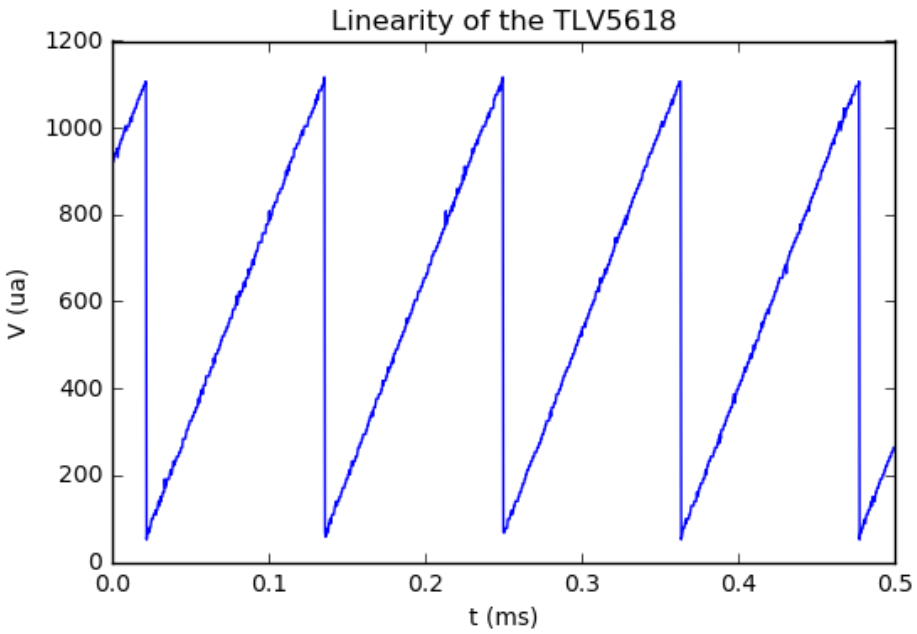
However, the SPI interface is only available on pins 10, 11, 13, which requires a little bit of reorganization of the Arduino code to work out-of-the box, and the `analogueOutput` function has to be re-written to use the SPI interface (**see an example**)

Additional information is available on those websites:

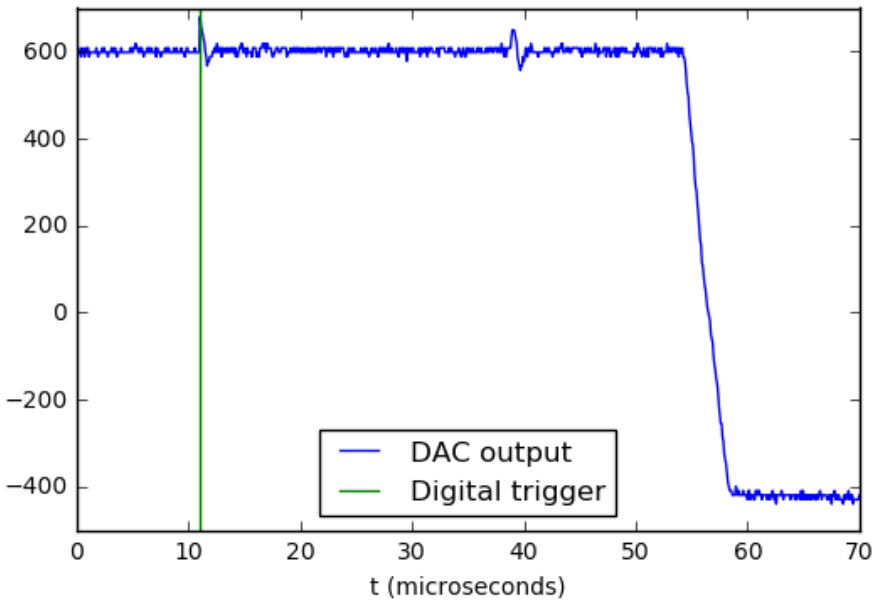
- **Digital-to-analog converter**
- **Laser harp construction**

DAC - performance

Linearity: If the voltage on the reference pin *REF* of the TLV5618 is properly set, the response to a ramp command is fairly linear (see image below). Note that the y scale is arbitrary and the x scale is ms.



Response time:



Project Tutorials

- **Step-by-step to controlling multiple light sources with an Arduino for sequenceable MDA** by Guillaume Witz & Thomas Julou.
- **TTL control of multiple LEDs** by David Knecht.
- **Detect tube lens position** of Nikon Ti Eclipse
- **Control laser shutter with Arduino** by Rocco D'Antuono.
- **3D printed enclosure with 8 BNC connectors for an Arduino** by Kurt Thorn
- **Open LED Illuminator** by Jens B. Bosse et al.