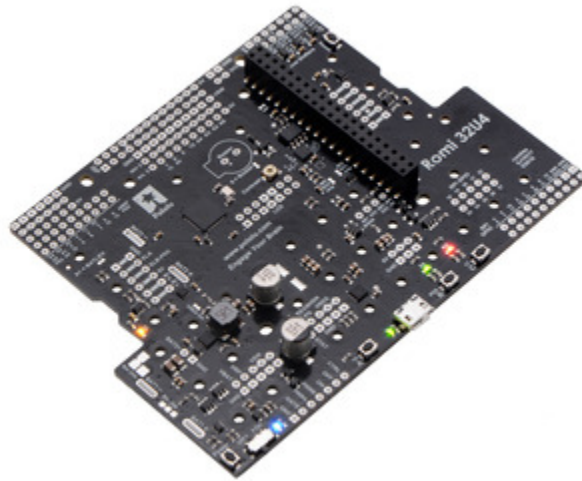


Pololu Romi 32U4 Control Board User's Guide



1. Overview	3
1.1. Included components	3
1.2. What you will need	4
1.3. Supported operating systems	5
2. Contacting Pololu	6
3. Romi 32U4 Control Board	7
3.1. Microcontroller	7
3.2. User interface	7
3.3. Motor drivers and encoders	9
3.4. Inertial sensors	11
3.5. Power	12
3.6. Expansion areas	16
3.7. Raspberry Pi interface and level shifters	19
3.8. Pin assignments	22
3.9. Adding electronics	25
3.9.1. Controlling a servo	28
3.10. AVR timers	28
3.11. Schematics and dimensions	29
4. Assembling the Romi 32U4 Control Board	31
5. Programming the Romi 32U4 Control Board	33
5.1. Installing Windows drivers	33
5.2. Programming using the Arduino IDE	35
5.3. Programming using avr-gcc and AVRDUDE	40
6. Romi 32U4 Arduino library	43
7. The Romi 32U4 USB interface	44
8. The A-Star 32U4 Bootloader	46
9. Reviving an unresponsive Romi 32U4	49
9.1. Reviving using the Arduino IDE	49
9.2. Reviving using AVRDUDE	52
10. Related resources	53

1. Overview

The Romi 32U4 Control Board is designed to be assembled with a **Romi chassis** [<https://www.pololu.com/category/202/romi-chassis-and-accessories>] to create a capable integrated robot platform that can easily be programmed and customized.

Like our **A-Star 32U4 programmable controllers** [<https://www.pololu.com/category/149/a-star-programmable-controllers>], the Romi 32U4 Control Board is built around a USB-enabled Atmel ATmega32U4 AVR microcontroller, and it ships preloaded with an Arduino-compatible bootloader. The control board features two H-bridge motor drivers and is designed to connect to a **Romi Encoder Pair Kit** [<https://www.pololu.com/product/3542>] (available separately) to allow closed-loop motor control. It also includes a powerful 5 V switching step-down regulator that can supply up to 2 A continuously, along with a versatile power switching and distribution circuit. A 3-axis accelerometer and gyro enable a Romi 32U4 robot to make inertial measurements, estimate its orientation, and detect external forces. Three on-board pushbuttons offer a convenient interface for user input, while indicator LEDs, a buzzer, and a connector for an optional LCD allow the robot to provide feedback.

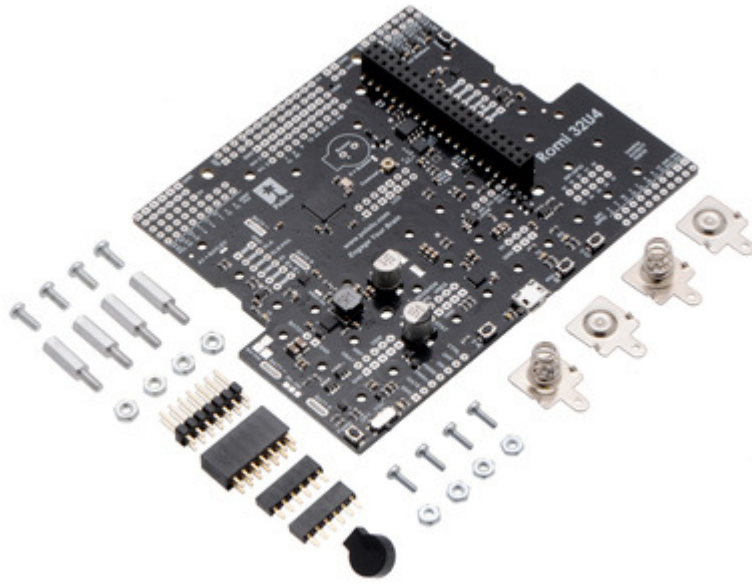
The Romi 32U4 Control Board can be used either as a standalone control solution or as a base for a more powerful Raspberry Pi controller. Its on-board connector and mounting holes allow a compatible Raspberry Pi (Model B+ or newer, including **Pi 3 Model B** [<https://www.pololu.com/product/2759>] and **Model A+** [<https://www.pololu.com/product/2760>]) to plug directly into the control board. Integrated level shifters make it easy to set up I²C communication and interface other signals between the two controllers, and the control board automatically supplies 5 V power to an attached Raspberry Pi. In this setup, the Raspberry Pi can handle the high-level robot control while relying on the Romi 32U4 Control Board for low-level tasks, like running motors, reading encoders, and interfacing with other analog or timing-sensitive devices.

The I/O lines of both the ATmega32U4 and the Raspberry Pi are broken out to 0.1"-spaced through-holes along the front and rear of the control board, and the board's power rails are similarly accessible, enabling sensors and other peripherals to easily be connected.

A software add-on is available that makes it easy to program a Romi 32U4 robot from the Arduino environment, and we have Arduino libraries and example sketches to help get you started. A **USB A to Micro-B cable** [<https://www.pololu.com/product/2072>] (not included) is required for programming.

1.1. Included components

The following components are included with the Romi 32U4 Control Board:



- two low profile female headers for motors and encoders
- **buzzer** [<https://www.pololu.com/product/1484>]
- **2×7 female header** [<https://www.pololu.com/product/1027>] and **male header** [<https://www.pololu.com/product/966>] for LCD
- battery terminals
- four 3/16" #2-56 screws and nuts
- four **M2.5 standoffs** [<https://www.pololu.com/product/1952>] (11 mm length), **screws** [<https://www.pololu.com/product/1968>], and **nuts** [<https://www.pololu.com/product/1967>] for mounting Raspberry Pi



An LCD and Raspberry Pi are **not** included with the Romi 32U4 Control Board.

1.2. What you will need

These additional items are also needed for using and assembling the Romi 32U4 Control Board:

Required accessories

- a **Romi Chassis Kit** [<https://www.pololu.com/category/203/romi-chassis-kits>] (this includes motors, wheels, one ball caster, and battery contacts)
- a **Romi Encoder Pair Kit** [<https://www.pololu.com/product/3542>]
- six **AA batteries** [<https://www.pololu.com/product/1003>]. The Romi chassis and control board

work with both alkaline and NiMH batteries, though we recommend rechargeable NiMH cells.

Assembly tools

- soldering iron and solder (we recommend one with adjustable temperature control like the **Hakko FX-888D Digital Soldering Station** [<https://www.pololu.com/product/2779>])
- small Phillips screwdriver
- **USB A to Micro-B cable** [<https://www.pololu.com/product/2072>] to connect the board to your computer for programming and debugging

Optional tools

- small 2 mm slotted screwdriver for adjusting the LCD contrast
- **small pair of pliers** [<https://www.pololu.com/product/150>]
- **wire cutter and stripper** [<https://www.pololu.com/product/1923>], for adding wires for peripherals
- tape or small clamps (for holding parts together when soldering)

Optional accessories

You might also consider getting these for your Romi 32U4 Robot:

- an **8×2 character LCD** [<https://www.pololu.com/product/356>]
- a compatible Raspberry Pi (Model B+ or newer, including **Pi 3 Model B** [<https://www.pololu.com/product/2759>] and **Model A+** [<https://www.pololu.com/product/2760>])
- **sensors** [<https://www.pololu.com/category/7/sensors>], such as **optical** [<https://www.pololu.com/category/79/sharp-distance-sensors>] or **sonar range finders** [<https://www.pololu.com/category/78/sonar-range-finders>]
- **connectors and jumper wires** [<https://www.pololu.com/category/19/connectors>], for connecting additional sensors and components
- battery charger, if you are using rechargeable batteries; since the Romi just uses ordinary AA batteries, we recommend basic AA chargers (into which you stick the individual cells) available at most general electronics stores, though we carry a much fancier **iMAX-B6AC V2 balance charger/discharger** [<https://www.pololu.com/product/2588>] that can be also used for this

1.3. Supported operating systems

The Romi 32U4 Control Board can be programmed using Microsoft Windows 11, 10, 8.1, 8, 7, Vista, XP (with Service Pack 3), Linux, and macOS 10.11 or later.

2. Contacting Pololu

We would be delighted to hear from you about any of your projects and about your experience with the Romi 32U4 Control Board. You can **contact us** [<https://www.pololu.com/contact>] directly or post on our **forum** [<http://forum.pololu.com/>]. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

3. Romi 32U4 Control Board

3.1. Microcontroller

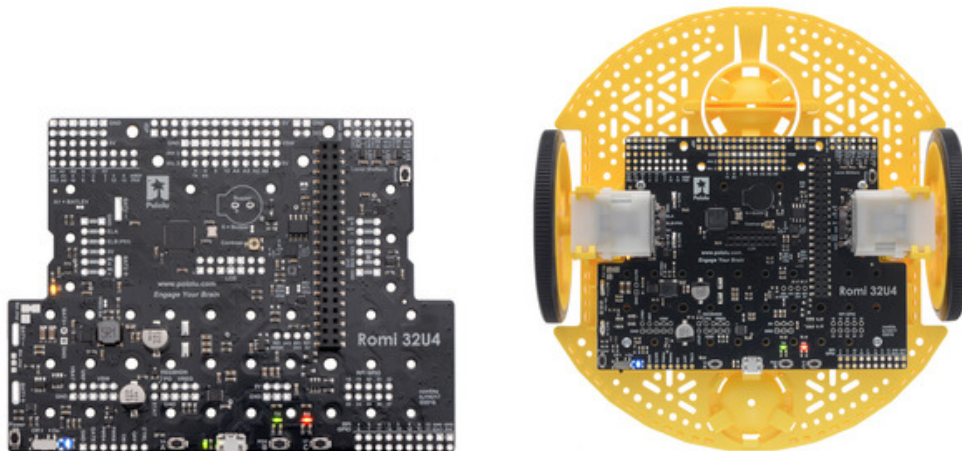
Like our **A-Star 32U4 programmable controllers** [<https://www.pololu.com/category/149/a-star-programmable-controllers>], the Romi 32U4 Control Board features an integrated, USB-enabled ATmega32U4 AVR microcontroller from Atmel, clocked by a precision 16 MHz crystal oscillator. This is the same microcontroller and clock frequency used in the **Arduino Leonardo** [<https://www.pololu.com/product/2192>] and **Arduino Micro** [<https://www.pololu.com/product/2188>].

The control board includes a USB Micro-B connector that can be used to connect to a computer's USB port via a **USB A to Micro-B cable** [<https://www.pololu.com/product/2072>] (not included). The USB connection can be used to transmit and receive data from the computer and program the board over USB. The USB connection can also provide power for the microcontroller and most of the other hardware on the board (but not motor power); see **Section 3.5** for more details.

The control board's ATmega32U4 comes preloaded with the Arduino-compatible **A-Star 32U4 USB bootloader** [<https://www.pololu.com/docs/0J69/8>], which allows it to be easily programmed using the Arduino IDE. For more information about programming the Romi 32U4 Control Board, see **Section 5**.

The board also has a 6-pin ISP header that allows it to be programmed with an external programmer, such as our **USB AVR programmer** [<https://www.pololu.com/product/3172>]. Pin 1 of the header is indicated with a small white dot and has an octagonal shape.

3.2. User interface



LEDs

The Romi 32U4 Control Board has five indicator LEDs.

- A **yellow** user LED is connected to Arduino digital pin 13, or PC7. You can drive this pin **high** in a user program to turn this LED on. The **A-Star 32U4 Bootloader** [<https://www.pololu.com/docs/0J69/8>] fades this LED on and off while it is waiting for a sketch to be loaded.
- A **green** user LED is connected to Arduino pin 30, or PD5, and lights when the pin is driven **low**. While the board is running the A-Star 32U4 Bootloader or a program compiled in the Arduino environment, it will flash this LED when it is *transmitting* data via the USB connection.
- A **red** user LED is connected to Arduino pin 17, or PB0, and lights when the pin is driven **low**. While the board is running the A-Star 32U4 Bootloader or a program compiled in the Arduino environment, it will flash this LED when it is *receiving* data via the USB connection.

The Romi32U4 library contains functions that make it easier to control the three user LEDs (see **Section 6**). All three user LED control lines are also LCD data lines, so you will see them flicker when you update the LCD. The green and red user LEDs also share I/O lines with pushbuttons (see below).

- A **blue** power LED next to the power switch indicates when the controller is receiving power from the Romi's batteries (the power switching circuit must be turned on).
- A **green** power LED on the bottom edge of the board near the USB connector indicates when the USB bus voltage (VBUS) is present.

Pushbuttons

The Romi 32U4 Control Board has five pushbuttons: a **power button** in the rear left corner, a **reset button** on the front right edge and **three user pushbuttons** located along the rear edge. The user pushbuttons, labeled A, B, and C, are on Arduino pin 14 (PB3), pin 30 (PD5), and pin 17 (PB0), respectively. Pressing one of these buttons pulls the associated I/O pin to ground through a resistor.

The three buttons' I/O lines are also used for other purposes: pin 14 is MISO on the SPI interface, pin 30 and pin 17 control the green and red user LEDs, and all three pins are LCD data lines. Although these uses require the pins to be driven by the AVR (or SPI slave devices in the case of MISO), resistors in the button circuits ensure that the Romi 32U4 Control Board will not be damaged even if the corresponding buttons are pressed at the same time, nor will SPI or LCD communications be disrupted. The functions in the Romi32U4 library take care of configuring the pins, reading and debouncing the buttons, and restoring the pins to their original states.

LCD

The Romi 32U4 Control Board has a set of through-holes in the center where a 2×7 header can be soldered to connect an **8×2 character LCD** [<https://www.pololu.com/product/356>] (or any other LCD with the common **HD44780 parallel interface** [<https://www.pololu.com/file/0J71/DMC50448N-AAE-AD.pdf>] (109k

pdf)). You can adjust the LCD contrast with the potentiometer on the top right of the LCD connector. We recommend using a 2 mm slotted screwdriver to adjust the contrast.

The Romi32U4 library provides functions to display data on a connected LCD. It is designed to gracefully handle alternate use of the LCD data lines by only changing pin states when needed for an LCD command, after which it will restore them to their previous states. This allows the LCD data lines to be used for other functions (such as pushbutton inputs and LED drivers).

Note that the control board is not designed to allow both an LCD and a Raspberry Pi to plug into it at the same time. However, having an LCD header soldered to the board should not interfere with mounting a Raspberry Pi.

Buzzer

The **buzzer** [<https://www.pololu.com/product/1484>] included with the Romi 32U4 Control Board can be soldered into the designated through-holes and used to generate simple sounds and music. By default, it is connected to digital pin 6 (which also serves as OC4D, a hardware PWM output from the AVR's 10-bit Timer4). If you alternate between driving the buzzer pin high and low at a given frequency, the buzzer will produce sound at that frequency. You can play notes and music with the buzzer using functions in the Romi32U4 library. If you want to use pin 6 for an alternate purpose, you can disconnect the buzzer circuit by cutting the surface-mount jumper next to the buzzer.

3.3. Motor drivers and encoders

Motor drivers

The Romi 32U4 Control Board has two Texas Instruments DRV8838 motor drivers that are used to power the Romi chassis's two **mini plastic gearmotors** [<https://www.pololu.com/product/1520>]. Four Arduino pins are used to control the drivers:

- **Digital pin 15**, or PB1, controls the **right motor direction** (LOW drives the motor forward, HIGH drives it in reverse).
- **Digital pin 16**, or PB2, controls the **left motor direction**.
- **Digital pin 9**, or PB5, controls the **right motor speed** with PWM (pulse width modulation) generated by the ATmega32U4's Timer1.
- **Digital pin 10**, or PB6, controls the **left motor speed** with PWM.

For more information about the drivers, see the **DRV8838 datasheet** [<https://www.pololu.com/file/0J806/drv8838.pdf.redirect>] (1k redirect). We also sell a **carrier board** [<https://www.pololu.com/product/2990>] for this driver.

The Romi32U4 library provides functions that allow you to easily control the motors (see **Section 6**).

The motor driver connections are brought out to two pairs of headers that are intended to interface with the **Romi Encoder Pair Kit** [<https://www.pololu.com/product/3542>]. A pair of low-profile female headers is included with the Romi 32U4 Control Board and can be soldered into either the outer or inner row of through-holes on each side. (Note that these headers must be soldered into the positions that match the male header installed on the encoder board.)



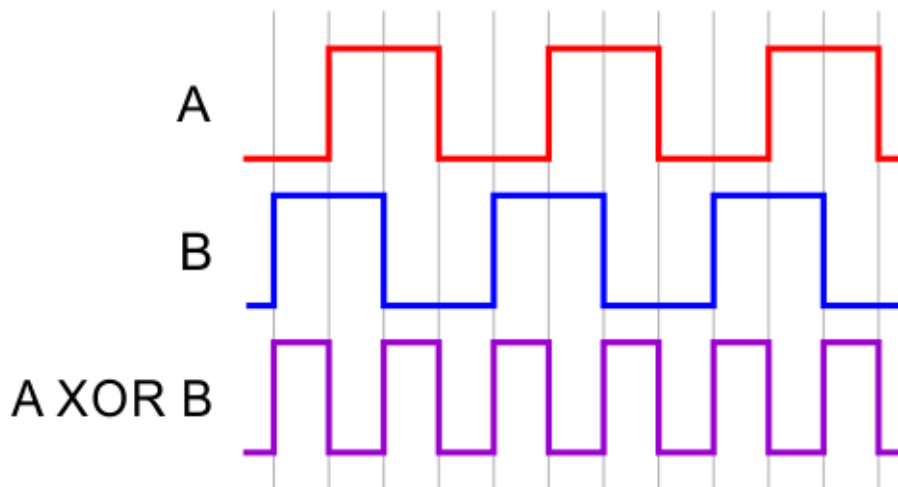
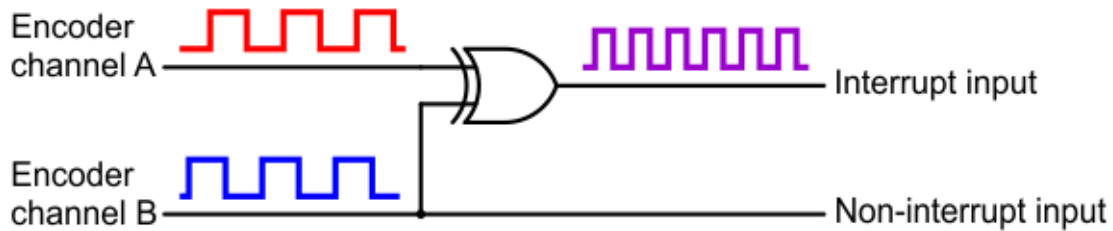
As your batteries run out, the voltage supplied to the motor drivers (VSW) will decrease, which will make the motors slower. It is possible to account for this in your code by monitoring the battery voltage (see **Section 3.5**) or using the encoders and other sensors to monitor the movement of the robot.

Quadrature encoders

The Romi 32U4 Control Board is configured to connect the quadrature encoder outputs from the Romi Encoder Pair Kit to the ATmega32U4 microcontroller. The encoders can be used to track the rotational speed and direction of the robot's drive wheels. They provide a resolution of 12 counts per revolution of the motor shaft when counting both edges of both channels, which corresponds to approximately 1440 counts per revolution of the Romi's wheels. For more information about the specifications of the Romi encoders, please see the **Romi Encoder Pair Kit product page** [<https://www.pololu.com/product/3542>].

Quadrature encoder transitions are often detected by monitoring both encoder channels directly. However, since transitions on the Romi's encoders can occur at high frequencies (several thousand per second) when its motors are running, it is necessary to use the AVR's pin change interrupts or external interrupts to read the encoders. To reduce the required number of interrupt pins, the Romi 32U4 Control Board XORs together both channels of each encoder and connects the resulting signal to an interrupt pin, while channel B of each encoder is connected to a non-interrupt pin:

- **Digital pin 7**, or PE6, reads the **right encoder XORed signal** using external interrupt INT6.
- **Digital pin 8**, or PB4, reads the **left encoder XORed signal** using pin change interrupt PCINT4.
- **Digital pin 23** (analog pin 5), or PF0, reads the **right encoder channel B**.
- Pin **PE2** reads the **left encoder channel B**.



The XORed signal and the channel B signal can be used to reconstruct the channel A signal by simply XORing them again: $(A \text{ XOR } B) \text{ XOR } B = A$. For both encoders, channel B leads channel A when the motor is rotating in the forward direction; that is, B rises before A rises and B falls before A falls. (The waveforms in the diagram above would be produced by forward rotation.) Note that this description designates the A and B signals as labeled on the control board itself, which puts A in front on both sides.

The Romi32U4 library provides appropriate interrupt service routines and functions for reading the encoders and keeping track of their counts (see **Section 6**).

3.4. Inertial sensors

The Romi 32U4 Control Board includes on-board inertial sensors connected to the ATmega32U4's I²C interface that can be used in advanced applications, such as helping detect collisions and determining the robot's orientation. These sensors are part of the ST **LSM6DS33** [<https://www.pololu.com/product/>

27361, which combines a 3-axis accelerometer and 3-axis gyro into a single package.

The inertial sensors are connected to the board's I²C bus by default, but they can be disconnected by cutting the surface-mount jumpers labeled "IMU SDA Jmp" and "IMU SCL Jmp". Level shifters built into the control board allow the ATmega32U4, operating at 5 V, to communicate with the 3.3 V sensors. If a Raspberry Pi is plugged into the control board, its I²C pins are connected to the 3.3 V side of the bus as well. (See **Section 3.7** for more information about the Raspberry Pi interface.)

We recommend carefully reading the **LSM6DS33 datasheet** [<https://www.pololu.com/file/0J1087/LSM6DS33.pdf>] (1MB pdf) to understand how these sensors work and how to use them.

Using the sensors

The Romi32U4 library (see **Section 6**) includes example programs that demonstrate how to use the sensors.

3.5. Power

The Romi 32U4 Control Board includes battery terminal connections that provide access to power from the Romi chassis's six-AA battery compartment. We recommend using rechargeable AA NiMH cells, which results in a nominal voltage of 7.2 V (1.2 V per cell). You can also use alkaline cells, which would nominally give you 9 V.

The negative battery voltage is connected to GND. The positive battery voltage is designated **VBAT**. VBAT feeds into a reverse protection circuit and then a power switching circuit controlled by the on-board pushbutton or slide switch. The output of the power switching circuit is designated **VSW**.

VSW provides power to the motors through the on-board DRV8838 motor drivers, so the motors can only operate if the batteries are installed and the power switch circuit is on.

The reverse protected and switched battery voltage on VSW can be monitored through a voltage divider that is connected to **analog pin 1** (PF6) by default. The divider outputs a voltage that is equal to one third of the battery voltage, which will be safely below the ATmega32U4's maximum analog input voltage of 5 V as long as the battery voltage is less than 15 V (though the maximum voltage for the board is still limited to 10.8 V by the DRV8838 motor driver). The `readBatteryMillivolts()` function in the Romi32U4 library can be used to determine the battery voltage from this reading. The surface-mount jumper labeled "A1 = BATLEV" can be cut to disconnect the voltage divider and free the pin for other uses.

Power switch circuit

The Romi 32U4 Control Board uses the patented latching circuit from the **Pololu pushbutton power switch** [<https://www.pololu.com/product/2808>], which provides a solid-state power switch for your robot

controlled with the on-board pushbutton. By default, this pushbutton can be used to toggle power: one push turns on power and another turns it off. Alternatively, a separate pushbutton can be connected to the **PWRA** and **PWRB** pins and used instead. Multiple pushbuttons can be wired in parallel for multiple control points, and each of the parallel pushbuttons, including the one on the board itself, will be able to turn the switch on or off. **The latching circuit performs some button debouncing, but pushbuttons with excessive bouncing (several ms) might not function well with it.**



For proper pushbutton operation, the board's slide switch should be left in its **Off** position. (Sliding the switch to the On position will cause the board power to latch on, and the switch must be returned to the Off position before the board can be turned off with the pushbutton.)

Alternatively, to disable the pushbutton, you can cut the button jumper labeled **Btn Jmp**; this transfers control of the board's power to the on-board slide switch instead. A separate slide or toggle switch can be connected to the **GATE** pin and used instead.

The power switch circuit also offers several alternate pushbutton connection options that result in push-on-only or push-off-only operation, and additional inputs enable further power control options like allowing your robot to turn off its own power. These advanced control options are available through the button connection pins and four control inputs:

PIN	Description
PWRA	Connect through momentary switch to pin “PWRB” for standard push-on/push-off operation. Connect through momentary switch to ground for on-only operation.
PWRB	Connect through momentary switch to pin “PWRA” for standard push-on/push-off operation.
ON	A high pulse (> 1 V) on this pin turns on the switch circuit. <i>This pin only functions when pushbutton operation is enabled (i.e. the button jumper has not been cut).</i>
OFF	A high pulse (> 1 V) on this pin turns off the switch circuit (e.g. allowing a powered device to shut off its own power). <i>This pin only functions when pushbutton operation is enabled.</i>
CTRL	With pushbutton operation enabled, this pin directly determines the state of the switch circuit. A high pulse (> 1 V) on this pin turns on the switch; a low pulse (e.g. driving the pin low with a microcontroller output line or pushing a button connected from this pin to ground) turns the switch off. Leave this pin disconnected or floating when not trying to set the switch state. <u>Note that this pin should not be driven high at the same time the “OFF” pin is driven high.</u>
GATE	With pushbutton operation <u>disabled</u> (button jumper cut), this pin controls the state of the switch circuit: driving it low turns the switch on, while letting it float turns the switch off. Connect through slide or toggle switch to ground for on/off operation. Leave this pin disconnected or floating for proper pushbutton operation. We recommend only ever driving this pin low or leaving it floating; <u>this pin should never be driven high while the slide switch is in the “On” position.</u>

5 V and 3.3 V regulators

VSW supplies power to a 5 V regulator, whose output is designated **VREG**. The battery voltage is regulated to 5 V by an MP4423H switching buck converter; although the regulator itself works with input voltages up to 36 V, the motor drivers limit the control board's maximum input voltage to 10.8 V. When available, VREG is generally used to supply logic power for the ATmega32U4, motor drivers, and encoders. The rest of the regulator's achievable output current, which depends on input voltage and ambient conditions, can be used to power other devices; this can include an attached Raspberry Pi (which typically draws a few hundred milliamps). Under typical conditions, up to 2 A of current is available from the VREG output. (We also make a **standalone regulator** [<https://www.pololu.com/product/2858>] based on this integrated circuit.)

The MP4423H regulator features an open-drain power good output, **PG**, which requires an external pull-up. PG drives low when the 5 V regulator's output voltage falls below around 85% of the nominal voltage and becomes high-impedance when the output voltage rises above around 90%. The regulator

circuit on the Romi 32U4 Control Board can be disabled by driving the regulator shutdown pin, **REGSHDN**, high; this will cause 5 V logic power for the control board to be sourced from USB instead if it is available.

The Romi 32U4 Control Board also contains a 3.3 V LDO that draws its power from the output of the logic power selection circuit described below. The output of the 3.3 V regulator is designated **3V3** and is used to supply the on-board inertial sensors and level shifters.

Logic power selection

The Romi 32U4 Control Board's power selection circuit uses the **TPS2113A power multiplexer** [<https://www.pololu.com/product/2596>] from Texas Instruments to choose whether its 5 V supply (designated **5V**) is sourced from USB or the batteries via the 5 V regulator, enabling the control board to safely and seamlessly transition between them. The TPS2113A is configured to select regulated battery power (VREG) unless the regulator output falls below about 4.5 V. If this happens, it will select the higher of the two sources, which will typically be the USB 5 V bus voltage if the control board is connected to USB.

Consequently, when the Romi 32U4 Control Board is connected to a computer via USB, it will receive 5 V logic power even when the power switch is off. This can be useful if you want to upload or test a program without drawing power from the batteries and without operating the motors. It is safe to have USB connected and battery power switched on at the same time.

The currently selected source is indicated by the **STAT** pin; this pin is an open-drain output that is low if the external power source is selected and high-impedance if the USB supply is selected. The current limit of the TPS2113A is set to about 1.9 A nominally. For more information about the power multiplexer, see the **TPS2113A datasheet** [<https://www.pololu.com/file/0J771/tps2113a.pdf.redirect>] (1k redirect).

The 5 V output of the selection circuit is used to supply the control board's ATmega32U4 microcontroller, logic power for the DRV8838 motor drivers, and the encoders; it also optionally powers an attached Raspberry Pi.

Raspberry Pi power

By default, the control board will provide power from its 5V line to an attached Raspberry Pi. In this situation, we recommend switching on the power circuit so that the Raspberry Pi receives power from the batteries through the control board's on-board switching regulator. Alternatively, you can use a **USB wall power adapter** [<https://www.pololu.com/product/1459>] to supply power through the control board's USB connector, although we have sometimes observed AVR brown-out resets occurring when a board powers the Raspberry Pi this way. A typical computer USB port might not be able to supply enough current to properly power the Romi 23U4 Control Board and an attached Raspberry Pi.

Power provided to the Raspberry Pi can be switched off by driving the Raspberry Pi shutdown pin, **RPISHDN**, to 5 V.

An ideal diode circuit on the control board prevents reverse current from flowing into the Romi 32U4 Control Board's 5 V supply if the Raspberry Pi is separately powered (for example, through its USB power receptacle). However, starting with the Raspberry Pi 3 Model B+, there is no corresponding ideal diode circuit on the Raspberry Pi's USB power input, so it is possible for the control board to backfeed a USB power adapter through the Raspberry Pi. As a result, we do not recommend connecting external USB power to the Raspberry Pi while it is powered through the control board.

Backfeeding is not an issue with older Raspberry Pi versions, which do have a diode circuit on the USB power input. With Raspberry Pi versions prior to the Pi 3 B+, it is safe to have any combination of control board USB power, battery power, and Raspberry Pi USB power connected to the system.

The **RPI5V** pin provides direct access to the Raspberry Pi's 5 V line, which comes from either the control board's 5V supply or the Raspberry Pi's USB power input (typically whichever is higher if both are connected). The 3.3 V output of the Raspberry Pi is also made available on the **RPI3V3** pin.

Note that the diode circuit prevents power from being shared in the reverse direction: the Raspberry Pi cannot supply 5 V logic power to the control board through the 40-pin connector.

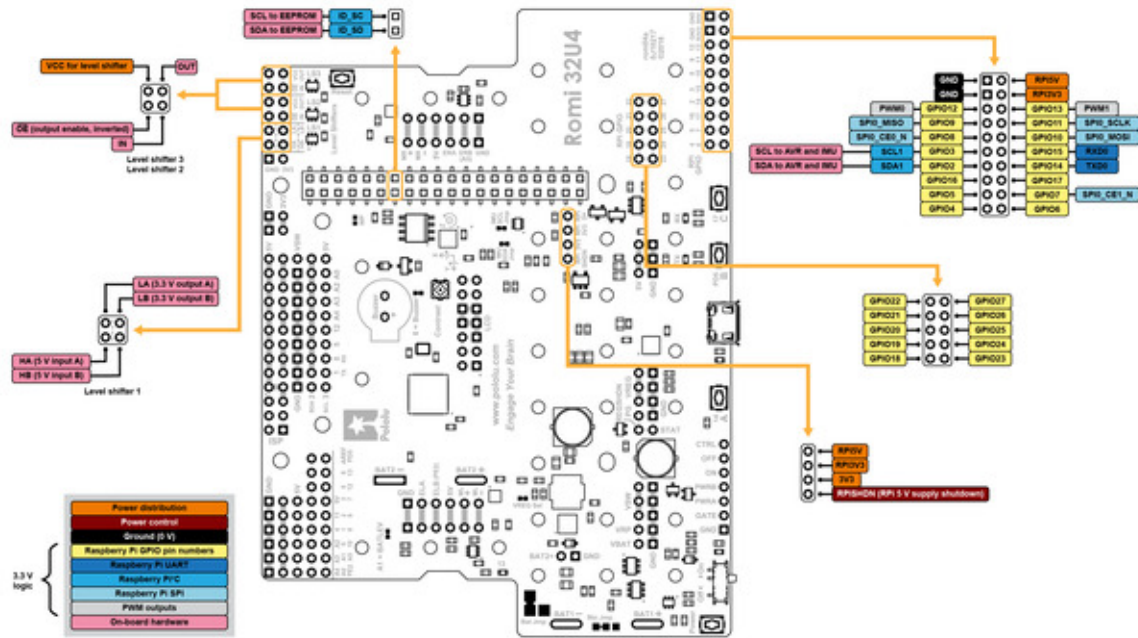
Power distribution

- **VBAT** is connected to the battery contact labeled **BAT1+** and provides a direct connection to the battery supply.
- **VRP** provides access to the battery voltage after reverse-voltage protection.
- **VSW** is the battery voltage after reverse protection and the power switch circuit.
- **VREG** is the output of the on-board 5 V regulator.
- **5V** is the output of the TPS2113A power multiplexer circuit which is connected to VREG by default, but switches to 5 V USB power if VREG is too low.
- **3V3** is the output of the 3.3 V LDO regulator.

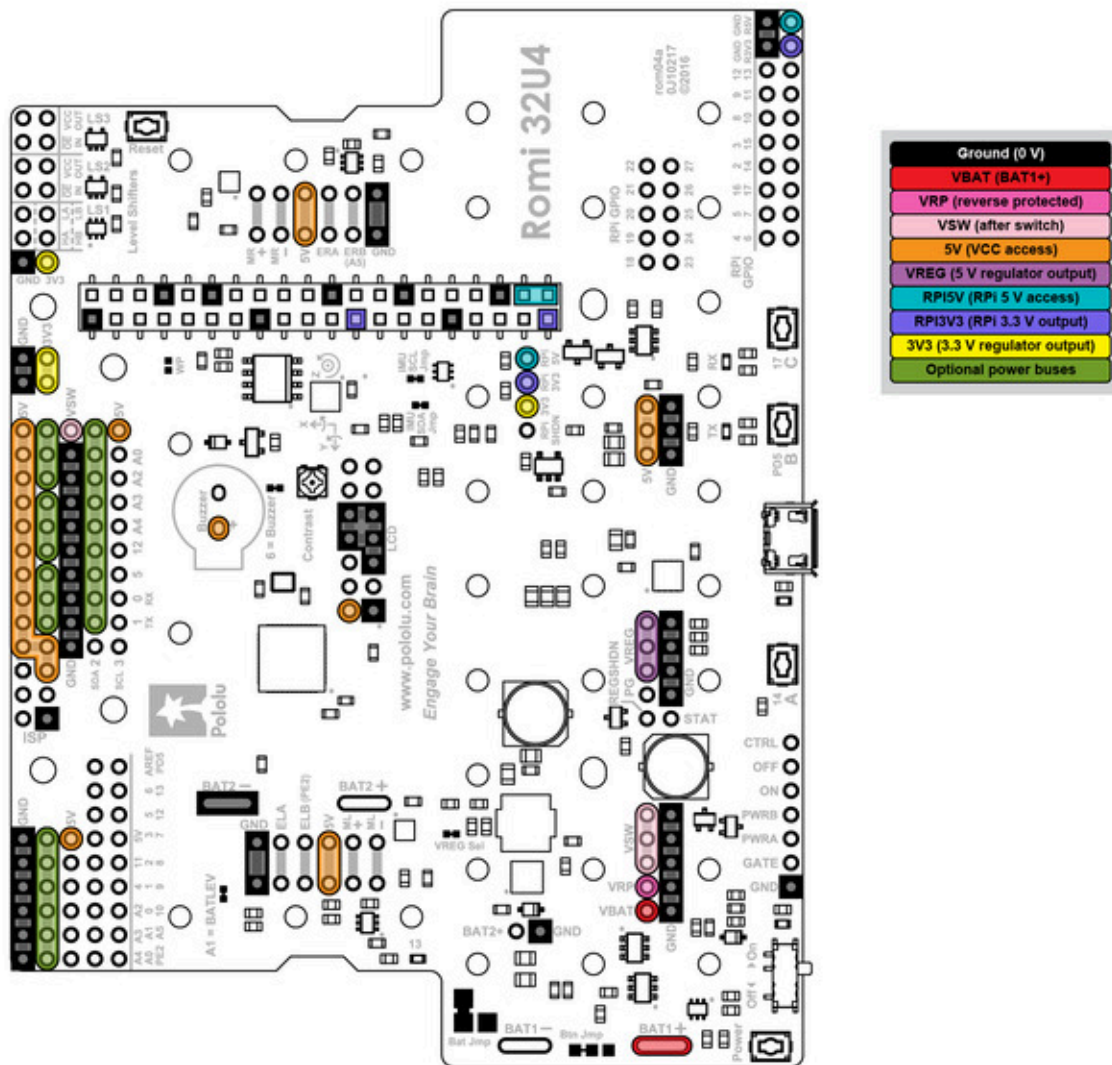
See **Section 3.6** for a diagram of the board's power distribution buses and access points.

3.6. Expansion areas

The Romi 32U4 Control Board has several expansion areas (primarily in three groups near the front left and middle and back right edges) that break out all of the general-purpose I/O lines from the ATmega32U4 microcontroller and the Raspberry Pi. The board also provides access to various power inputs, outputs, and control pins, and it makes a few stand-alone buses available to help you make connections. The following diagrams identify the locations of these pins and the hardware associated



Pinout diagram of the Romi 32U4 Control Board (Raspberry Pi pinout, peripherals, and level shifters).



Power distribution diagram of the Romi 32U4 Control Board.

3.7. Raspberry Pi interface and level shifters

The Romi 32U4 Control Board was designed to be easy to interface with a Raspberry Pi single-board computer to expand the Romi’s processing power. It has a connector and mounting holes matching the Raspberry Pi HAT (Hardware Attached on Top) specification and is designed to connect to the Model B+ and newer versions of the Raspberry Pi with 40-pin GPIO headers (including the **Raspberry Pi 3 Model B** [<https://www.pololu.com/product/2759>] and **Model A+** [<https://www.pololu.com/product/2760>]). A 2×20-pin 0.1" female header is soldered to the control board, and it ships with a set of four **standoffs** [<https://www.pololu.com/product/1952>], **screws** [<https://www.pololu.com/product/1968>], and **nuts** [<https://www.pololu.com/product/1967>].



I²C communication

When used with a Raspberry Pi, the control board is designed to serve as an auxiliary controller, communicating with the Raspberry Pi using an I²C interface (also known as 2-wire Serial Interface, or TWI). As such, the ATmega32U4 microcontroller's I²C data and clock lines (SDA and SCL) are connected to the corresponding lines on the Raspberry Pi's I²C bus 1 through on-board level-shifting circuits. These bidirectional level shifters convert between the AVR's 5 V logic level and the Raspberry Pi's 3.3 V logic level.

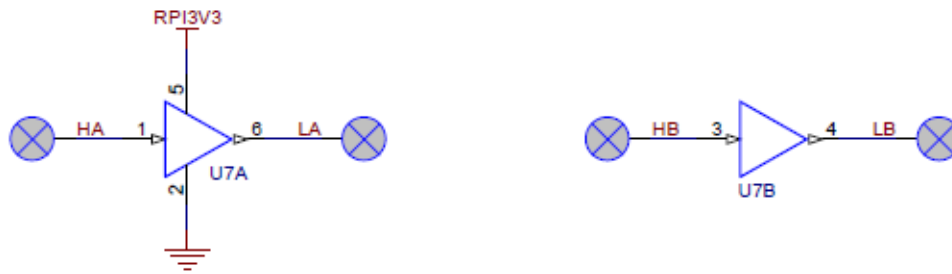
We have written an **Arduino library** [<https://github.com/pololu/pololu-rpi-slave-arduino-library>] for our 32U4 family of boards that lets them act as an I²C slave and provides a framework for communication between the ATmega32U4 and a Raspberry Pi master.

A **tutorial** [<https://www.pololu.com/blog/663>] on the Pololu blog demonstrates this library and its included example code, using them to make a robot that can be remotely controlled and monitored through a web server running on the Raspberry Pi.

General-purpose level shifters

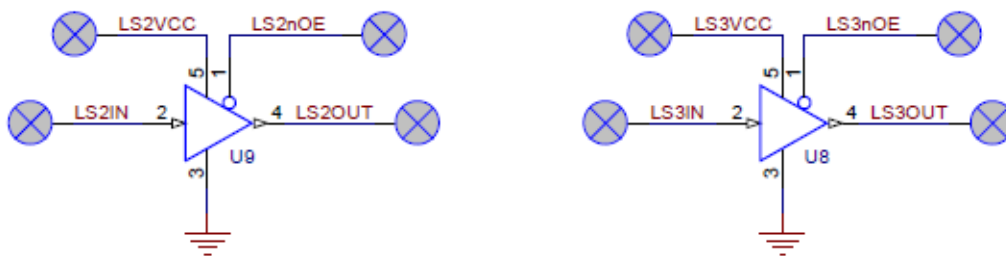
In addition to the dedicated I²C level shifters, the board also makes available a few general-purpose level shifters that are not connected to any signals by default.

LS1 is a dual-channel unidirectional level shifter that converts a pair of 5 V inputs (**HA** and **HB**) to a pair of corresponding 3.3 V outputs (**LA** and **LB**).



LS2 and **LS3** are each single-channel, tristatable, unidirectional level shifters. Each of these exposes four pins: $\overline{\text{OE}}$ (output enable), **IN** (input), **OUT** (shifted output), and **VCC** (logic supply voltage).

- When $\overline{\text{OE}}$ is high, **OUT** is in a high impedance state.
- When $\overline{\text{OE}}$ is low, **OUT** matches the state of **IN**, shifted to the voltage supplied on **VCC**.



For example, if you pull $\overline{\text{OE}}$ low, connect a 3.3 V signal to **IN**, and connect 5V to **VCC**, the signal will be shifted to 5 V logic level on **OUT**.

The input logic level can be 1.8 V to 5.5 V, while **VCC** (and the output logic level) can be 3 V to 5.5 V. The **IN** signal can have either a lower or higher logic level than the **VCC** voltage: you could connect a 5 V signal to **IN** and a 3.3 V to **VCC** or a 3.3 V signal to **IN** and a 5 V to **VCC**.

Powering the Raspberry Pi from the control board

The control board will provide 5 V power to an attached Raspberry Pi by default. See **Section 3.5** for more details about how power is shared and can be controlled between the two boards.

ID EEPROM

The Romi 32U4 Control Board includes a 32-kilobit (4096-byte) EEPROM that connects to the

Raspberry Pi's ID_SD and ID_SC pins. The EEPROM ships with its contents blank, but you can program it as an ID EEPROM in the format specified by the **Raspberry Pi HAT specifications** [<https://github.com/raspberrypi/hats>], using the utilities provided there. When suitably programmed, the EEPROM can help the Raspberry Pi identify and configure itself to work with the Romi 32U4 Control Board.

Write protection for the EEPROM can be enabled by using solder to bridge the surface-mount jumper labeled “WP” next to the EEPROM chip. (The EEPROM is not write-protected by default.)

3.8. Pin assignments

The table below lists the most important pin assignments for the ATmega32U4 on the Romi 32U4 Control Board. This table is helpful if you want to add your own electronics to the Romi 32U4, write your own low-level code for interfacing with the hardware, or just want to understand better how the Romi 32U4 works. Each row represents a physical pin on the ATmega32U4.

The “ATmega32U4 pin name” column shows the official name of the pin according to the **ATmega32U4 datasheet** [<https://www.microchip.com/en-us/product/ATmega32u4>].

The “Arduino pin names” column lists the names provided by the Arduino environment for the pin. These names can generally be used as arguments to any function that takes a pin number. However, there are some exceptions. For example, passing the number 4 to `analogRead` actually reads pin A4, not pin 4. Also, due to hardware limitations, some functions only work on a limited set of pins.

The “Romi 32U4 functions” column documents what the pin is used for on the Romi 32U4 Control Board. Many pins can serve multiple purposes concurrently by switching modes. For example, PB0 can read the state of button C when it is an input, and it can control the red LED and serve as an LCD data line when it is an output.

The “Note/alternate functions” column documents other features of the pin, although some of those features might be impractical to use.

ATmega32U4 pin name	Arduino pin names	Romi 32U4 Control Board functions	Notes/alternate functions
PB7	11	LCD control line (E)	Timer0 PWM output A (OC0A) Timer1 PWM output C (OC1C) UART flow control ($\overline{\text{RTS}}$) Pin-change interrupt (PCINT7)
PD4	4, A6, 24	LCD control line (RS)	Analog input (ADC8) Timer1 input capture pin (ICP1)
PB3	14, MISO	User pushbutton A LCD data line DB4	SPI Master Input/Slave Output (MISO) Pin-change interrupt (PCINT3) ISP programming line (PDO)
PB0	17, LED_BUILTIN_RX, SS	Red LED (RX) User pushbutton C LCD data line DB5	SPI slave select ($\overline{\text{SS}}$) Pin-change interrupt (PCINT0)
PC7	13, LED_BUILTIN	Yellow LED LCD data line DB6	Timer4 PWM output A (OC4A) Timer3 input capture pin (ICP3) Divided system clock output (CLKO)
PD5	30, LED_BUILTIN_TX	Green LED (TX) User pushbutton B LCD data line DB7	UART external clock (XCK1) UART flow control ($\overline{\text{CTS}}$)
PD7	6, A7, 25	Buzzer PWM	Analog input (ADC10) Timer4 PWM output D (OC4D) Timer0 counter source (T0)
PF6	A1, 19	Battery level input (VIN/3)	Analog input (ADC6) JTAG test data out (TDO)
PB6	10, A10, 28	Left Motor PWM	Analog input (ADC13)

ATmega32U4 pin name	Arduino pin names	Romi 32U4 Control Board functions	Notes/alternate functions
			Timer1 PWM output B (OC1B) Timer4 PWM output B (OC4B) Pin-change interrupt (PCINT6) SPI Master Output/Slave Input (MOSI)
PB2	16, MOSI	Left motor direction	Pin-change interrupt (PCINT2) ISP programming line (PDI)
PB5	9, A9, 27	Right motor PWM	Analog input (ADC12) Timer1 PWM output A (OC1A) Timer4 PWM output B ($\overline{OC4B}$) Pin-change interrupt (PCINT5)
PB1	15, SCK	Right motor direction	SPI Clock (SCK) Pin-change interrupt (PCINT1) ISP programming line (SCK)
PB4	8, A8, 26	Left encoder XORed input	Analog input (ADC11) Pin-change interrupt (PCINT4)
PE2	-	Left encoder input	Hardware bootloader select (\overline{HWPB})
PE6	7	Right encoder XORed input	Analog comparator negative input (AIN0) External interrupt source (INT6)
PF0	A5, 23	Right encoder input	Analog input (ADC0)
PD0	3, SCL	I ² C clock for Raspberry Pi communication and inertial sensors	Timer0 PWM output B (OC0B)

ATmega32U4 pin name	Arduino pin names	Romi 32U4 Control Board functions	Notes/alternate functions
			External interrupt source ($\overline{\text{INT0}}$)
PD1	2, SDA	I ² C data for Raspberry Pi communication and inertial sensors	External interrupt source ($\overline{\text{INT1}}$)
PD2	0	<i>Free I/O</i>	UART receive pin (RXD1) External interrupt source ($\overline{\text{INT2}}$)
PD3	1	<i>Free I/O</i>	UART transmit pin (TXD1) External interrupt source ($\overline{\text{INT3}}$)
PC6	5	<i>Free I/O</i>	Timer3 PWM output A (OC3A) Timer4 PWM output A ($\overline{\text{OC4A}}$)
PD6	12, A11, 29	<i>Free I/O</i>	Analog input (ADC9) Timer4 PWM output D ($\overline{\text{OC4D}}$) Timer1 counter source (T1)
PF7	A0, 18	<i>Free I/O</i>	Analog input (ADC7) JTAG test data in (TDI)
PF5	A2, 20	<i>Free I/O</i>	Analog input (ADC5) JTAG test mode select (TMS)
PF4	A3, 21	<i>Free I/O</i>	Analog input (ADC4) JTAG test clock (TCK)
PF1	A4, 22	<i>Free I/O</i>	Analog input (ADC1)
$\overline{\text{RESET}}$	-	Reset pushbutton	internally pulled high, active low
AREF	-	-	Analog reference

3.9. Adding electronics

This section gives tips for how additional electronics can be connected to the Romi 32U4 Control Board.

Free I/O pins

If you want your additional devices to send or receive information from the AVR, you will need to connect them to one or more of the AVR's I/O pins. The pin assignment table in **Section 3.8** lists all of these pins and how they are used. Many of the I/O pins are already being used for some other purpose on the board, but there are 8 free I/O pins: 0, 1, 5, 12, A0, A2, A3, and A4. Any of the free I/O lines can be used as basic digital inputs or outputs; each pin also has some special capabilities.

Pin 0 (PD2) and pin 1 (PD3) are the RX and TX lines of the AVR's TTL serial port.

Pin 5 (PC6) is a hardware PWM output and is usable with the Arduino `analogWrite()` function. Pin 12 (A11/PD6) can also be used as a PWM output, but it is not supported by `analogWrite()`, and using pin 12 for PWM might conflict with uses of pin 6 (which controls the buzzer by default) as these two pins are complementary outputs of Timer4 channel D.

Pins 12 (A11/PD6), A0 (18/PF7), A2 (20/PF5), A3 (21/PF4), and A4 (22/PF1) can be used as analog inputs.

Freeing up more I/O pins

If the free I/O pins are not sufficient for connecting the devices you want to connect, you might need to disable or disconnect some of the other features of the Romi 32U4 Control Board to free up more I/O pins.

If you do not need the AVR to be able to measure the battery voltage, you can use pin A1 (19/PF6) for other purposes. This pin can be used for digital input and output, as well as analog input. If you want to use this pin as a digital or analog input, you might need to cut the surface-mount jumper labeled "A1 = BATLEV" in order to disconnect it from the on-board voltage divider. If you only want to use A1 as an output, you might not need to cut that jumper.

If you are not connecting an LCD to the board's LCD connector, then pin 11 (PB7) and pin 4 (A6/PD4) are free. Both pins can be used for digital input and output. In addition, pin 11 can be used as a PWM output and a pin change interrupt and pin 4 can be used as an analog input.

If you are not connecting an LCD to the board's LCD connector, you can use the LCD contrast potentiometer for other purposes. The output of the potentiometer is a 0 V to 5 V signal which is accessible on the LCD connector. It can be connected to any free analog input if you want to read it from the AVR, or it might be useful to connect it to the other electronics that you are adding.

If you do not need to use the buzzer, you can free up pin 6 (A7/PD7) by cutting the surface-mount jumper labeled "6 = Buzzer". Pin 6 can be used as a PWM output, digital I/O line, or analog input. Disabling the buzzer also frees up Timer4, which has several PWM output pins. These pins can be used as PWM outputs if they are not needed for their normal tasks.

If you do not need encoders, you can free up pin 7 (PE6), pin 8 (A8/PB4), PE2, and pin A5 (23/PF0). Each of those pins has a corresponding surface mount jumper on the bottom of the board that can be cut in order to disconnect the encoder output from the AVR.

Be careful about connecting electronics to pin 13 (PC7), pin 17 (PB0), and pin 30 (PD5). These pins are used to control the LEDs on the Romi 32U4. All three of these pins are controlled as outputs by the bootloader. Pin 17 (PB0) and pin 30 (PD5) are used as RX and TX indicators, so if you are sending or receiving data over USB then the Arduino USB code will drive those pins in its interrupt service routines while your sketch is running.

I²C devices

It should be possible to attach additional I²C slave devices to the control board's I²C bus as long as the additional devices' slave addresses do not conflict with that of the LSM6DS33, which uses 7-bit address 1101011. The ATmega32U4's I²C pins (2 and 3) operate at 5 V. If you are connecting a 3.3 V device, you can connect it to the 3.3 V side of the bus instead (accessible through [Raspberry Pi GPIO pins 2](#), for SDA, and 3, for SCL, even if a Raspberry Pi is not connected). Separate level shifters might be necessary to interface with devices that use other voltages.

If you do not want to use the inertial sensors on the I²C bus, you can cut the surface-mount jumpers labeled "IMU SDA Jmp" and "IMU SCL Jmp". This frees up pin 2 (PD1) and pin 3 (PD0) for limited use as digital inputs and outputs as long as a Raspberry Pi is not attached to the control board. Note that the AVR's I²C pins will remain connected to the on-board I²C level shifters and will therefore still be pulled up to 5 V.

Power

The control board's power nodes are accessible in several areas on the board. If you power additional devices from VSW, then they will be powered whenever the control board's power is in ON, and they will receive whatever voltage the batteries are outputting. If you power them from VREG, they will get 5 V power whenever the batteries are installed and the power is on (but they cannot be powered from USB). If you power them from a 5V pin, then they will receive 5V power whenever the control board's logic components are powered. If you power them from 3V3, they will receive 3.3V power whenever the control board's logic components are powered. For more information about these power nodes and how much current they can provide, see **Section 3.5**.

It is also possible to add your own power switch to control power to the Romi 32U4 Control Board, as described in **Section 3.5**.

Ground

You should make sure that all the grounds in your system are connected. The Romi 32U4 Control Board's ground node can be accessed from pins labeled "GND". It should be connected to the ground

node of every other circuit board or device you add to the robot.

3.9.1. Controlling a servo

It is possible to modify the Servo library that comes with the Arduino IDE to use Timer 3 instead of Timer 1 with an ATmega32U4 based controller like the Romi 32U4. The modified Servo library does not interfere with Romi32U4Motors, making it possible to simultaneously control servos and the motors.

Warning: The modifications described here will affect any sketch for an ATmega32U4 based controller that uses the Servo library, including the Arduino Leonardo or A-Star.

1. First, you will need to locate the Arduino IDE's Servo library, and find the file inside it named `ServoTimers.h`. For the 1.6.x versions of the IDE, this file can be found in `libraries/Servo/src/avr/ServoTimers.h`. If you are using Mac OS X, you will need to right-click on the Arduino IDE icon and select "Show Package Contents" to see the files inside.
2. Open `ServoTimers.h` in a text editor.
3. Locate the following lines of code in `ServoTimers.h`:

```

1  #elif defined(__AVR_ATmega32U4__)
2  #define _useTimer1
3  typedef enum { _timer1, _Nbr_16timers } timer16_Sequence_t ;

```

4. The lower two lines of code specify that the library should use Timer 1. To use Timer 3 instead, just change `_useTimer1` to `_useTimer3` and `_timer1` to `_timer3`.
5. Save the file.

The Arduino IDE will automatically incorporate your modifications to the Servo library. The next time you compile a sketch for an ATmega32U4 based controller that uses the Servo library, it will use Timer 3 instead of Timer 1.

3.10. AVR timers

The ATmega32U4 has 4 timers: Timer0, Timer1, Timer3, and Timer4. Each timer has a different set of features, as documented in the datasheet.

- Timer0 is used by the Arduino environment for timing-related functions like `millis()`.
- Timer1 is used by the Romi 32U4 Control Board Arduino library for driving motors.
- Timer3 is *not* used by the Romi 32U4 Arduino library and can be freely used for your own purposes.

- Timer4 is used by the Romi 32U4 Arduino library for controlling the buzzer. The buzzer pin (digital pin 6, or PD7; Timer4 output OC4D) can be freed for other uses by cutting the surface-mount jumper labeled “6 = Buzzer”.

3.11. Schematics and dimensions

Schematics

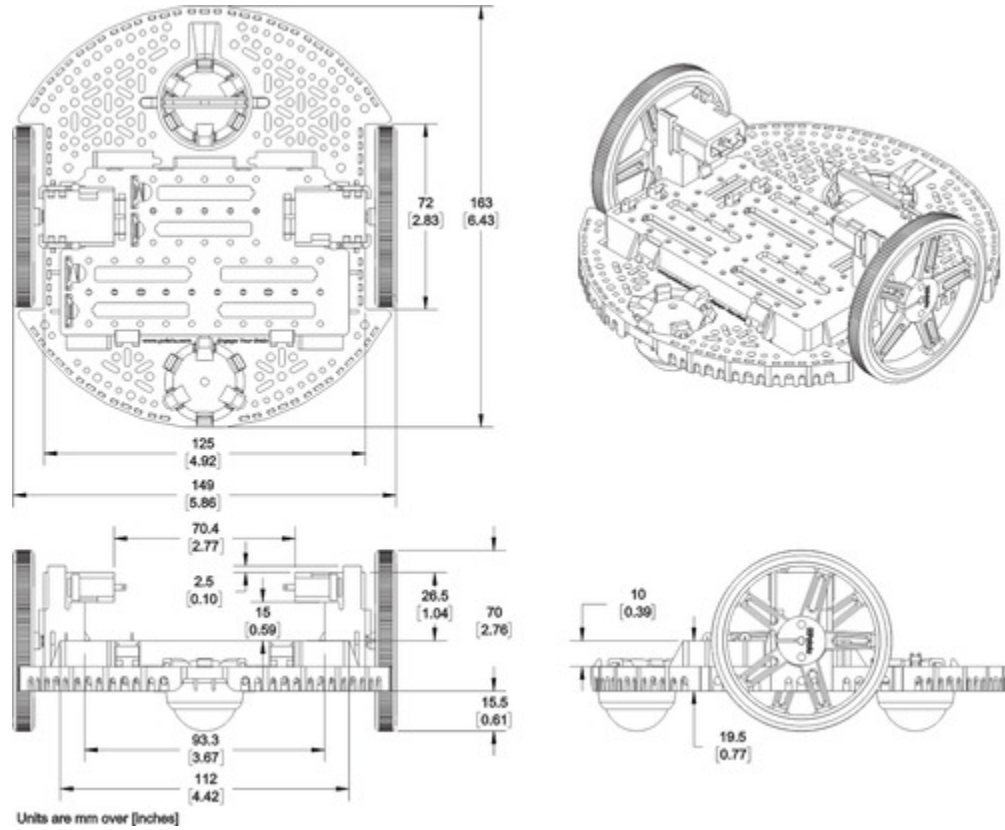
The schematic diagram for the Romi 32U4 Control Board is available as a PDF: **Romi 32U4 Control Board schematic diagram** [<https://www.pololu.com/file/0J1258/romi-32u4-control-board-schematic-diagram.pdf>] (646k pdf).

Dimensions

A basic dimension diagram for the Romi 32U4 Control Board is available as a PDF: **Romi 32U4 Control Board dimension diagram** [<https://www.pololu.com/file/0J1259/romi-32u4-control-board-dimensions.pdf>] (604k pdf).

Dimensions that are not included in the above diagram can be measured from the following DXF, which shows the main board outline along with the sizes and locations of all of the holes on the board: **Romi 32U4 Control Board drill guide** [<https://www.pololu.com/file/0J1260/rom04a-drill.dxf>] (346k dxf).

The following picture shows the approximate dimensions of the **Romi chassis** [<https://www.pololu.com/category/203/romi-chassis-kits>] which is intended to be used with the Romi 32U4 Control Board:



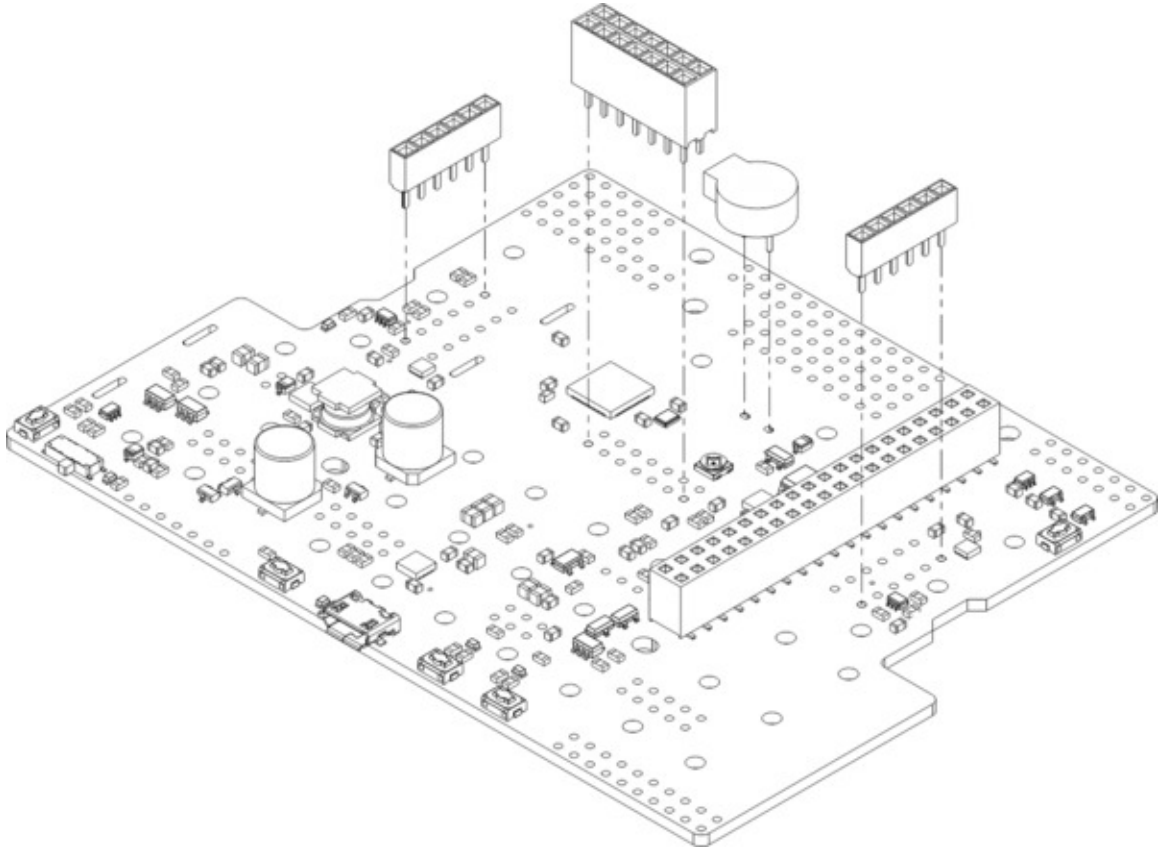
Basic dimension diagram of the Romi Chassis.

4. Assembling the Romi 32U4 Control Board

Control board additions

Most of the hardware on the Romi 32U4 Control Board consists of surface-mount components that are already soldered to the board, but there are a few through-hole parts that you need to solder yourself.

1. Solder the buzzer to the top of the control board, matching its orientation to the printed outline, then trim the excess length from the buzzer leads underneath the board.
2. Solder the two 1×6 low profile female headers for the encoders to the board. One female header should be soldered on each side in the set of through-holes that matches to the orientation you will use when soldering the corresponding male headers to the **Romi Encoder Pair Kit** [<https://www.pololu.com/product/3542>]. We recommend using the set closer to the edges of the Romi 32U4 Control Board.
3. **Optional:** If you plan to use an **LCD** [<https://www.pololu.com/product/356>] (not included) with the control board, solder either the 2×7 female or 2×7 male LCD header to the set of pins labeled “LCD” in the center of the control board. We generally recommend soldering the female connector to the board and the male connector to the LCD if you do not have some particular reason for doing it the other way.
4. **Optional:** If you plan to connect other headers or wires, consider soldering them now.



It is still possible to solder the buzzer and LCD header after the control board has been mounted on the chassis, but soldering them beforehand is easier and avoids the risk of inadvertently melting the chassis with your soldering iron. The control board must be removed from the chassis before the encoder headers can be soldered in.

The four battery contact terminals should be soldered to the control board **after** it is mounted on the chassis, as described in the chassis assembly instructions. You will be able to remove the board and battery contacts from the chassis as a single piece after soldering.

Assembling the chassis

Once the through-hole components are soldered to the Romi 32U4 Control Board, please follow the **instructions** [<https://www.pololu.com/docs/0J68/4>] given in the **Pololu Romi Chassis User's Guide** [<https://www.pololu.com/docs/0J68>] to finish assembling the chassis, mounting the control board, and soldering in the battery contacts.

5. Programming the Romi 32U4 Control Board

The Romi 32U4 Control Board is designed to be programmed over USB from the **Arduino IDE** [<https://www.arduino.cc/en/software/>]. It can be programmed from Windows, Linux, and Mac OS X. The ATmega32U4 on the control board comes preloaded with the same USB bootloader as the **A-Star 32U4 family** [<https://www.pololu.com/category/149/a-star-programmable-controllers>] of general-purpose programmable ATmega32U4 boards. The following sections will help you get started programming your Romi 32U4 Control Board.

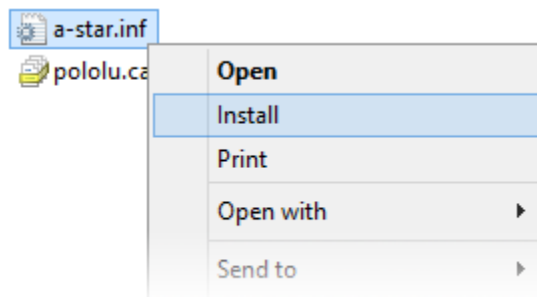
5.1. Installing Windows drivers



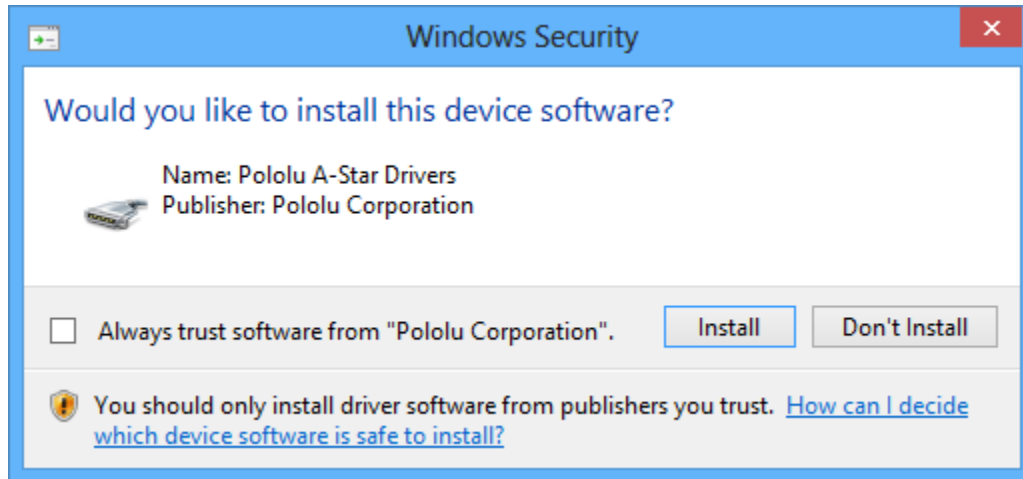
If you use Windows XP, you will need to have either Service Pack 3 or Hotfix KB918365 installed before installing the A-Star drivers. Some users who installed the hotfix have reported problems that were solved by upgrading to Service Pack 3, so we recommend Service Pack 3 over the hotfix.

Before you connect your Pololu A-Star 32U4 (or another of our 32U4 family of boards) to a computer running Microsoft Windows, you should install its drivers:

1. Download the **A-Star Windows Drivers** [<https://www.pololu.com/file/0J1240/a-star-windows-1.3.0.0.zip>] (7k zip) and extract the ZIP file to a temporary folder on your computer. (These files are also available in the “drivers” directory from the **A-Star repository on GitHub** [<https://github.com/pololu/a-star>].)
2. Open the “a-star-windows” folder. Right-click on “a-star.inf” and select “Install”.



3. Windows will ask you whether you want to install the drivers. Click “Install” (Windows 10, 8, 7, and Vista) or “Continue Anyway” (Windows XP).



4. Windows will not tell you when the installation is complete, but it should be done after a few seconds.

Windows 10, Windows 8, Windows 7, and Windows Vista users: After installing the drivers, your computer should automatically recognize the device when you connect it via USB. No further action from you is required. However, the first time you connect an A-Star device to your computer, Windows will take several seconds to recognize the device and configure itself properly. The first time you program the device, Windows will again take several seconds to recognize the A-Star USB bootloader, and this could cause the programming operation to fail the first time. Also, Windows will need to re-recognize the device and the bootloader if you connect the board to another USB port that it has not been connected to before.

Windows XP users: After installing the drivers, you will need to follow steps 5–9 for each new A-Star device you connect to your computer. You will also need to follow these steps the first time you attempt to program the device in order to make Windows recognize the bootloader, and when you connect the device to a different USB port that it has not been connected to before.

5. Connect the device to your computer's USB port.
6. When the "Found New Hardware Wizard" is displayed, select "No, not this time" and click "Next".
7. On the second screen of the "Found New Hardware Wizard", select "Install the software automatically" and click "Next".
8. Windows XP will warn you again that the driver has not been tested by Microsoft and recommend that you stop the installation. Click "Continue Anyway".
9. When you have finished the "Found New Hardware Wizard", click "Finish".

COM port details

After installing the drivers and plugging in an A-Star, in the “Ports (COM & LPT)” category of the Device Manager, you should see a COM port for the A-Star’s running sketch named “Pololu A-Star 32U4”.

You might see that the COM port is named “USB Serial Device” in the Device Manager instead of having a descriptive name. This can happen if you are using Windows 10 or later and you plugged the A-Star into your computer before installing our drivers for it. In that case, Windows will set up your A-Star using the default Windows serial driver (usbser.inf), and it will display “USB Serial Device” as the name for the port. The port will still be usable, but it will be hard to tell if it is the right one because of the generic name shown in the Device Manager. We recommend fixing the names in the Device Manager by right-clicking on each “USB Serial Device” entry, selecting “Update Driver Software...”, and then selecting “Search automatically for updated driver software”. Windows should find the drivers you already installed, which contain the correct name for the port.

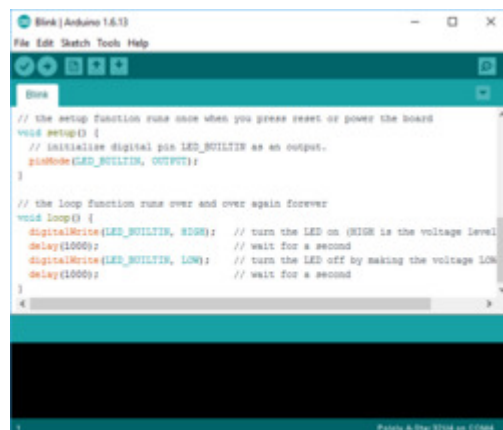
If you are using Windows 10 or later and choose not to install the drivers, the A-Star will still be usable. To tell which “USB Serial Device” in your Device Manager is the A-Star, double-click on each one and look at the “Hardware Ids” property in the “Details” tab. An A-Star running a sketch will have the ID `USB\VID_1FFB&PID_2300&MI_00`, while an A-Star in bootloader mode will have the ID `USB\VID_1FFB&PID_0101`.

If you want to change the COM port numbers assigned to your A-Star, you can do so using the Device Manager. Double-click a COM port to open its properties dialog, and click the “Advanced...” button in the “Port Settings” tab.

5.2. Programming using the Arduino IDE

Our 32U4 family of boards can be programmed from the popular Arduino integrated development environment (IDE). The Arduino IDE is a cross-platform, open source application that integrates a C++ code editor, the GNU C++ compiler, and a program upload utility. To get started programming your device with the Arduino IDE (version 1.6.4 or later), follow these steps:

1. Download the Arduino IDE from the **Arduino Download page** [<https://www.arduino.cc/en/software/>], install it, and start it.
2. In the Arduino IDE, open the **File** menu (Windows/Linux) or the **Arduino** menu (macOS) and select “Preferences”.

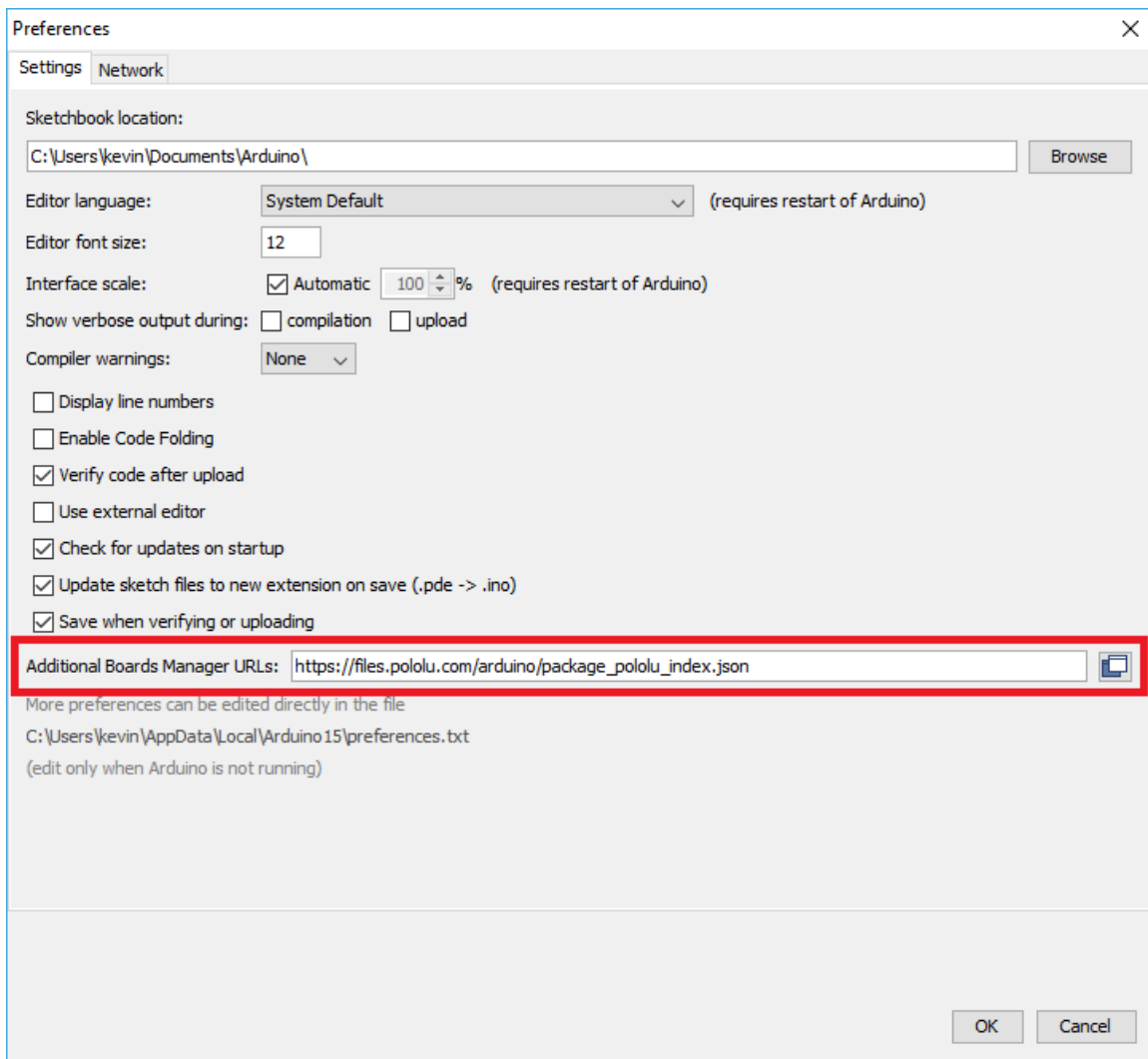


Programming the A-Star 32U4 from the Arduino IDE.

3. In the Preferences dialog, find the “Additional Boards Manager URLs” text box (highlighted in the picture below). Copy and paste the following URL into this box:

https://files.pololu.com/arduino/package_pololu_index.json

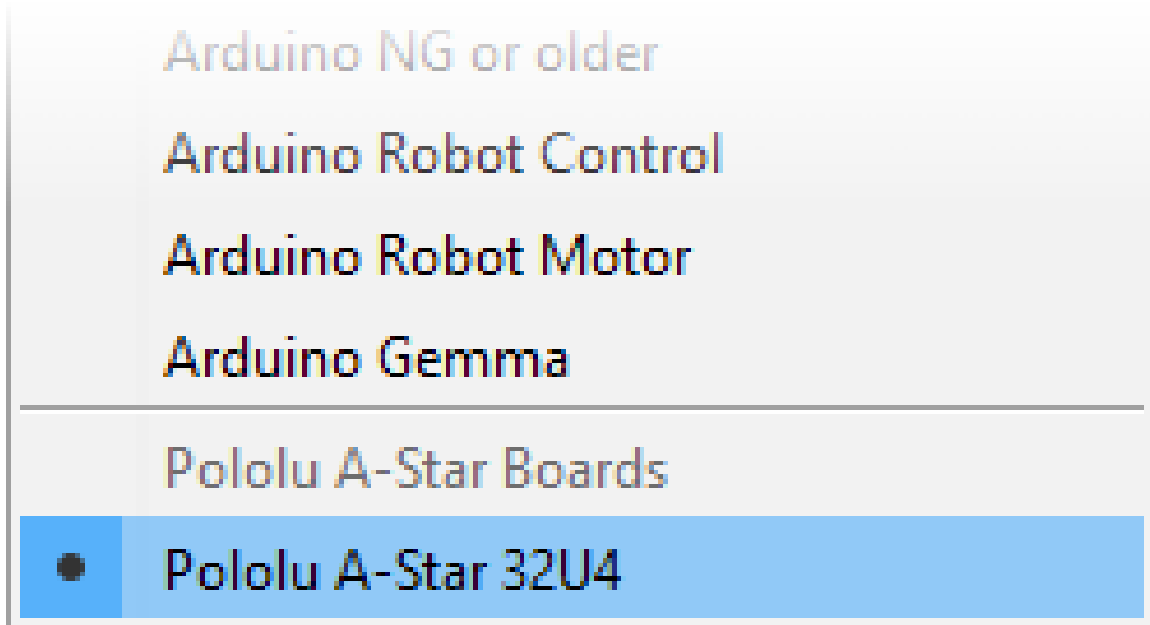
If there are already other URLs in the box, you can either add this one separated by a comma **or** click the button next to the box to open an input dialog where you can add the URL on a new line.



Adding a Boards Manager index for Pololu boards in the Arduino IDE's Preferences dialog.

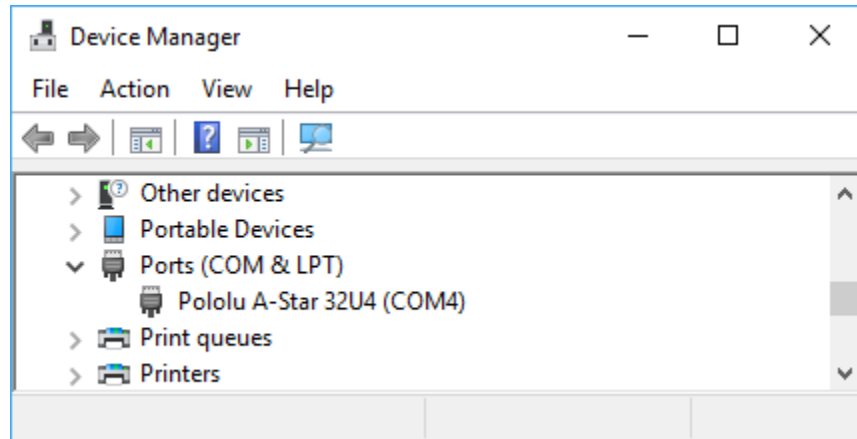
4. Click the “OK” button to close the Preferences dialog.

5. In the **Tools > Board** menu, select “Boards Manager...” (at the top of the menu).
6. In the Boards Manager dialog, search for “Pololu A-Star Boards”.
7. Select the “Pololu A-Star Boards” entry in the list, and click the “Install” button.
8. After the installation finishes, click the “Close” button to close the Boards Manager dialog.
9. In the **Tools > Board** menu, select the “Pololu A-Star 32U4” entry. If you do not see your device listed in the Board menu, try restarting the Arduino IDE.



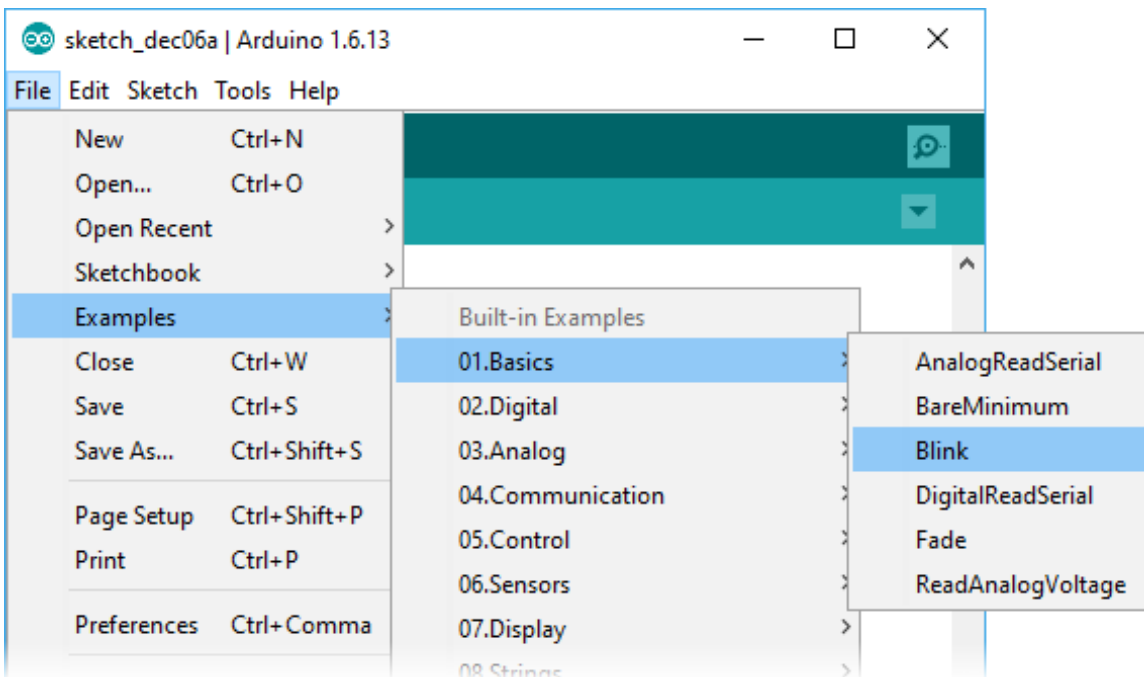
Selecting the Pololu A-Star 32U4 in the Boards menu.

10. In the **Tools > Port** menu, select the port for the device. On Windows you can determine what COM port the device is assigned to by looking at the “Ports (COM & LPT)” section of the Device Manager. On Linux, the port name will begin with “/dev/ttyACM”. On Mac OS X, the port name will begin with “/dev/tty.usbmodem”.



Windows 10 Device Manager showing the A-Star’s virtual COM port.

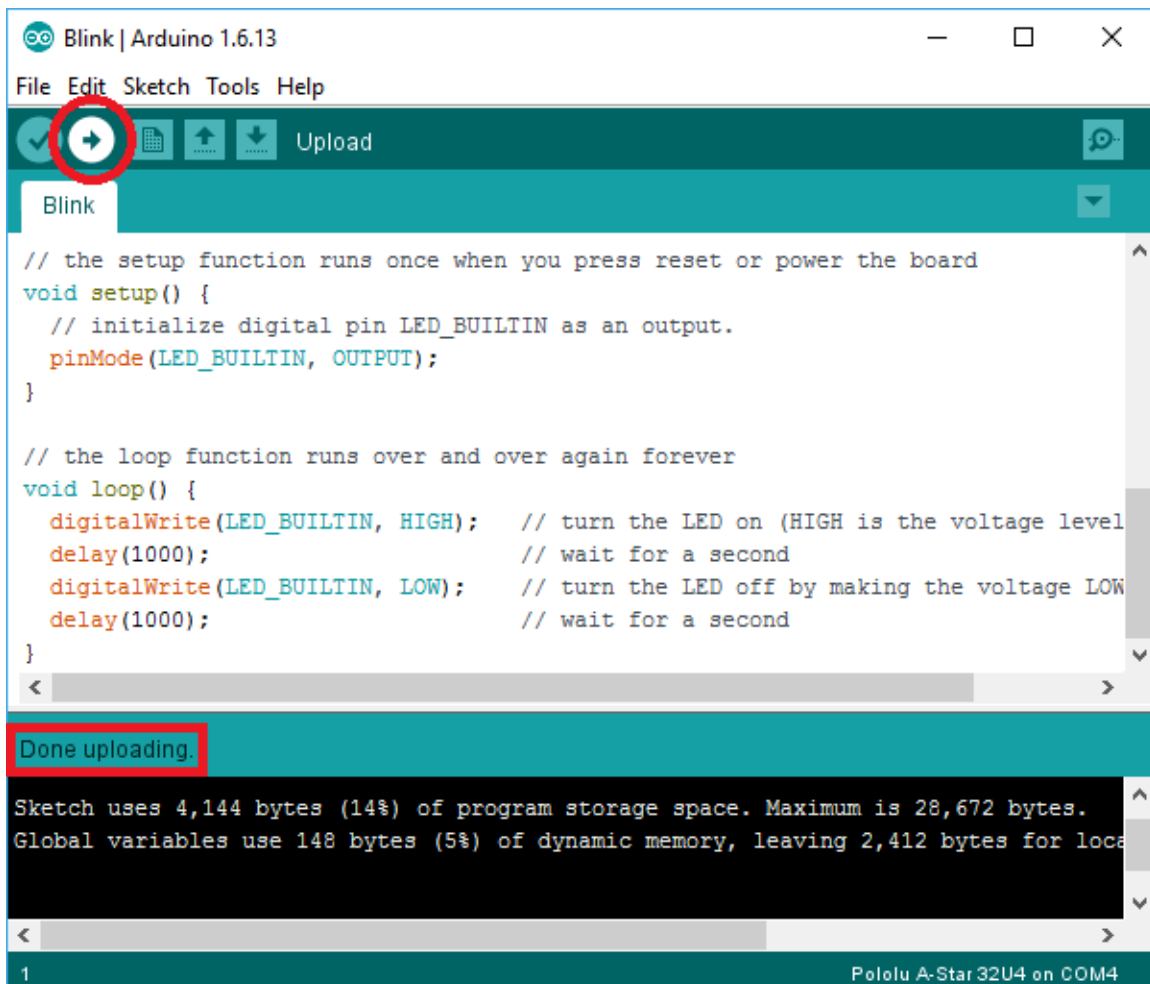
- Open up the “Blink” Arduino example, which can be found under **File > Examples > 01.Basics > Blink**. The code in this example will blink the yellow LED. When you select the Blink example, a new Arduino IDE window will open up. It is OK to close the first window.



Selecting the Blink example in the Arduino IDE.

- Press the “Upload” button to compile the sketch and upload it to the device. If everything goes correctly, you will see the message “Done uploading” appear near the bottom of the window. If you are using Windows and you have not previously programmed an A-Star device on this

USB port, then Windows might take several seconds to recognize the A-Star bootloader. The bootloader times out after 8 seconds and goes back to running the sketch, so the upload might fail if Windows does not recognize it quickly enough. If this happens, try again. If you are using Windows XP and have not programmed an A-Star on this USB port, you will have to go through the Found New Hardware Wizard again as described in the previous section, but the second time you try to upload it should work. If the Arduino IDE has trouble connecting to the port or using it, try unplugging the device, closing any programs that might be using the serial port, restarting the Arduino IDE, and then plugging the device back in.



Uploading a sketch to the A-Star using the Arduino IDE.

13. If you uploaded the Blink sketch, then the yellow LED should be blinking once every two seconds. However, we ship some A-Stars with that same example already programmed onto it, so you might not be convinced that anything has changed. Try changing the delay values in the sketch to something else and uploading again to see if you can change the speed of

the LED.



The A-Star 32U4 boards are similar enough to the Arduino Leonardo that you do not actually have to install the add-on. If you want to, you can just select the “Arduino Leonardo” board in the Arduino IDE. Note that if you upload a sketch to the device this way, your computer will then recognize it as a Leonardo (for example, its entry in the Windows Device Manager will display “Arduino Leonardo”).

After you succeed in programming your device from the Arduino IDE, there are many resources you can use to learn more:

- The Arduino IDE has many **examples** [<http://arduino.cc/en/Tutorial/HomePage>] that can run on A-Stars.
- The Arduino website has a **Language Reference** [<http://arduino.cc/en/Reference/HomePage>], a wiki called the **The Arduino Playground** [<http://playground.arduino.cc/>], and other resources.
- The A-Star 32U4 boards are similar to the **Arduino Leonardo** [<https://www.pololu.com/product/2192>] and **Arduino Micro** [<https://www.pololu.com/product/2188>], so you can search the Internet for relevant projects that use one of those boards.
- The Related Resources section lists many more resources.

Disabling ModemManager in Linux

If you are using Linux and have trouble following the instructions above, your issue might be caused by a program called ModemManager. This program automatically connects to serial ports and sends modem commands to them, interfering with other software using those ports. You can run `ps ax | grep -i Modem` to see if ModemManager is running. On Ubuntu, the command to permanently disable ModemManager is:

```
sudo systemctl disable ModemManager
```

5.3. Programming using avr-gcc and AVRDUDE

This section explains how to program our 32U4 family of boards using the avr-gcc toolchain and AVRDUDE. This section is intended for advanced users who do not want to use the Arduino IDE as described in the previous section.

Getting the prerequisites

If you are using Windows, we recommend downloading **WinAVR** [<http://winavr.sourceforge.net/>], which contains the avr-gcc toolchain and a command-line utility called **AVRDUDE** [<http://www.nongnu.org/>]

avrdude] that can be used to upload programs to the A-Star bootloader. If the version of GNU Make that comes with WinAVR crashes on your computer, we recommend using the **Pololu version of GNU Make** [<https://github.com/pololu/make/releases>].

If you are using macOS, we recommend first installing **Homebrew** [<https://brew.sh/>]. Then run the following commands to install AVRDUDE and **homebrew-avr** [<https://github.com/osx-cross/homebrew-avr/>]:

```
brew install avrdude
xcode-select --install
brew tap osx-cross/avr
brew install avr-gcc
```

If you are using Linux, you will need to install `avr-gcc`, `avr-libc`, and `AVRDUDE`. Ubuntu users can get the required software by running:

```
sudo apt-get install gcc-avr avr-libc avrdude
```

After you have installed the prerequisites, open a command prompt and try running these commands to make sure all the required utilities are available:

```
avr-gcc -v
avr-objcopy -v
make -v
avrdude
```

If any of those commands fail, make sure the desired executable is installed on your computer and make sure that it is in a directory listed in your `PATH` environment variable.

Compiling an example program

Copy the following code to a file named “main.c”:

```
1  #define F_CPU 16000000
2  #include <avr/io.h>
3  #include <util/delay.h>
4
5  int main()
6  {
7      DDRC |= (1 << DDC7);    // Make pin 13 be an output.
8      while(1)
9      {
10         PORTC |= (1 << PORTC7); // Turn the LED on.
11         _delay_ms(500);
12         PORTC &= ~(1 << PORTC7); // Turn the LED off.
13         _delay_ms(500);
14     }
15 }
```

In the same folder, create a file named “Makefile” with the following contents:

```
PORT=\\\\.\\GLOBALROOT\\Device\\USBSER000
```

```

MCU=atmega32u4
CFLAGS=-g -Wall -mcall-prologues -mmcu=$(MCU) -Os
LDFLAGS=-Wl,-gc-sections -Wl,-relax
CC=avr-gcc
TARGET=main
OBJECT_FILES=main.o

all: $(TARGET).hex

clean:
    rm -f *.o *.hex *.obj *.hex

%.hex: %.obj
    avr-objcopy -R .eeprom -O ihex $< $@

%.obj: $(OBJECT_FILES)
    $(CC) $(CFLAGS) $(OBJECT_FILES) $(LDFLAGS) -o $@

program: $(TARGET).hex
    avrdude -p $(MCU) -c avr109 -P $(PORT) -U flash:w:$(TARGET).hex

```

Make sure that the `PORT` variable in the Makefile is the name of the device's virtual serial port. In Windows, `\\\\.\\GLOBALROOT\\Device\\USBSER000` should work if the A-Star is the only USB device connected that is using the `usbser.sys` driver, but you can change it to be the actual name of the COM port (e.g. `COM13`).

In a command prompt, navigate to the directory with the Makefile and `main.c`. If you run the command `make`, the code should get compiled and produce a file named "main.hex".

Programming

To program the A-Star device, you will need to get it into bootloader mode first. One way to do this is to reset the AVR twice within 750 ms. Most of the boards in our 32U4 family have a reset button that can be used to reset the board. On any of our 32U4 family of boards, a pushbutton can be connected between the GND and RST pins to serve as a reset button, or you can use a wire. Once the device is in bootloader mode, quickly run the command `make program` to program it. If you wait longer than 8 seconds, the A-Star bootloader will exit and the AVR will go back to running the user program.

6. Romi 32U4 Arduino library

The Romi 32U4 Control Board can be programmed from the Arduino IDE as described in the preceding sections.

To help interface with all the on-board hardware on the control board, we provide the **Romi32U4 library**. The **Romi32U4 library documentation** [<https://pololu.github.io/romi-32u4-arduino-library/>] provides detailed information about the library, and the library comes with several example sketches.

If you are using version 1.6.2 or later of the Arduino software (IDE), you can use the Library Manager to install this library:

1. In the Arduino IDE, open the “Sketch” menu, select “Include Library”, then “Manage Libraries...”.
2. Search for “Romi32U4”.
3. Click the Romi32U4 entry in the list.
4. Click “Install”.

If this does not work, you can manually install the library:

1. Download the **latest release archive from GitHub** [<https://github.com/pololu/romi-32u4-arduino-library>] and decompress it.
2. Rename the folder “romi-32u4-arduino-library-master” to “Romi32U4”.
3. Move the “Romi32U4” folder into the “libraries” directory inside your Arduino sketchbook directory. You can view your sketchbook location by opening the “File” menu and selecting “Preferences” in the Arduino IDE. If there is not already a “libraries” folder in that location, you should make the folder yourself.
4. After installing the library, restart the Arduino IDE.

After you install the Romi32U4 library, you can learn more about it by trying the included example sketches and by reading the **Romi32U4 library documentation** [<https://pololu.github.io/romi-32u4-arduino-library/>].

If you are using the Romi 32U4 Control Board with a Raspberry Pi, you might also want to make use of our **Raspberry Pi slave library for Arduino** [<https://github.com/pololu/pololu-rpi-slave-arduino-library>], which sets up the A-Star as an I²C slave and helps establish communication with a Raspberry Pi master. See **Section 3.7** for more information about the Raspberry Pi interface.

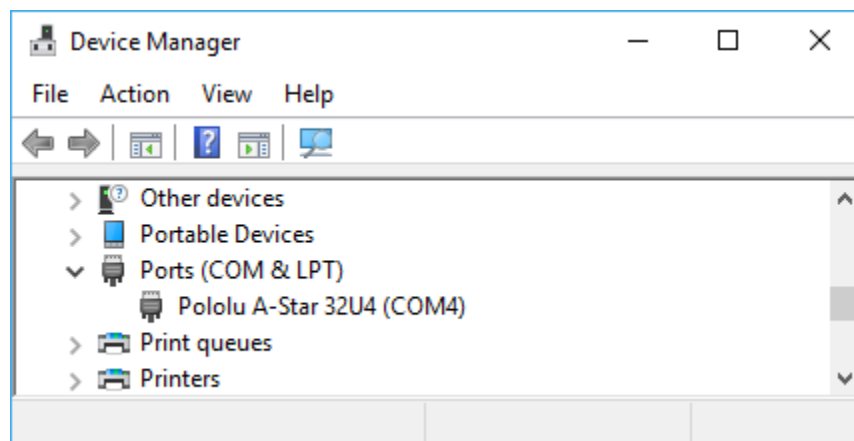
7. The Romi 32U4 USB interface

Our 32U4 family of boards are based on a single ATmega32U4 AVR microcontroller that runs the user program and also handles the USB connection to the computer. The AVR has a full-speed USB transceiver built into it and can be programmed to present almost any type of USB device interface to the computer.

USB is an asymmetric system that consists of a single “host” connected to multiple “devices”. The host is typically a personal computer. The ATmega32U4 can only act as a USB device, so an A-Star device cannot be connected to other USB devices like mice and keyboards; it can only be connected to a host such as your computer.

Programming an ATmega32U4 board using the Arduino IDE as described earlier will automatically configure it as a composite device with a single virtual serial port. If you program the microcontroller with an Arduino sketch that implements another USB device class, like HID or MIDI, you will see additional child devices as well.

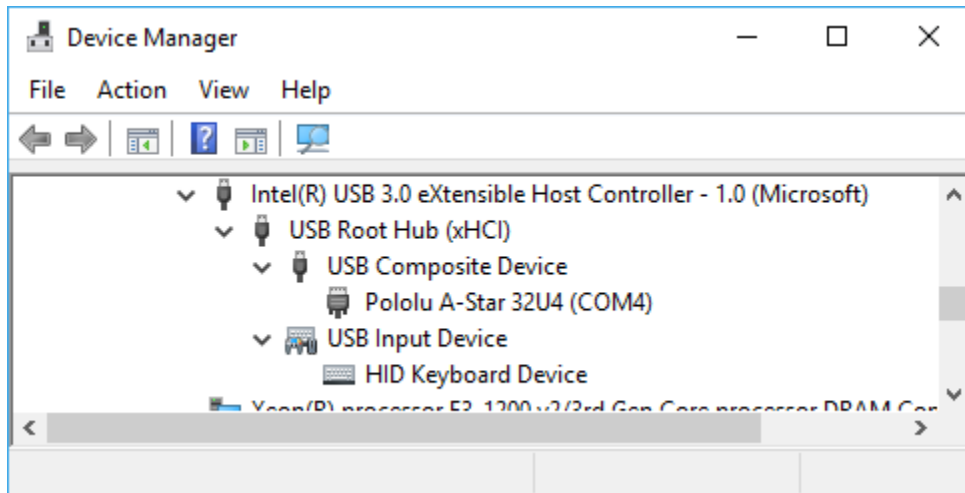
On a Windows computer, you can see the virtual serial port by going to your computer's Device Manager and expanding the “Ports (COM & LPT)” list. You should see a COM port labeled “Pololu A-Star 32U4”. In parentheses after the name, you will see the name of the port (e.g. “COM3” or “COM4”). Windows will assign a different COM port number to the device depending on what USB port you plug it into and whether it is in bootloader mode or not. If you need to change the COM port number assigned to the A-Star, you can do so using the Device Manager. Double-click on the COM port to open its properties dialog, and click the “Advanced...” button in the “Port Settings” tab. From this dialog you can change the COM port assigned to the device.



Windows 10 Device Manager showing the A-Star's virtual COM port.

On a Windows computer, you can see the rest of the USB interface by going to the Device Manager, selecting **View > Devices by connection**, and then expanding entries until you find the “Pololu A-Star

32U4" COM port. Near it, you should see the parent composite device.



The Windows 10 Device Manager in "Devices by connection" mode, showing that the A-Star is a composite device.

On a Linux computer, you can see details about the USB interface by running `lsusb -v -d 1fffb:` in a Terminal. The virtual serial port can be found by running `ls /dev/ttyACM*` in a Terminal.

On a Mac OS X computer, the virtual serial port can be found by running `ls /dev/tty.usbmodem*` in a Terminal.

You can send and receive bytes from the virtual serial port using any terminal program that supports serial ports. Some examples are the Serial Monitor in Arduino IDE, the **Pololu Serial Transmitter Utility** [<https://www.pololu.com/docs/0J23>], **Br@y Terminal** [<https://sites.google.com/site/terminalbpp/>], **PuTTY** [<https://www.chiark.greenend.org.uk/~sgtatham/putty/>], **TeraTerm** [<http://tssh2.osdn.jp/>], **Kermit** [<http://www.columbia.edu/kermit/ck80.html>], and **GNU Screen** [<http://www.gnu.org/software/screen/>]. Many computer programming environments also support sending and receiving bytes from a serial port.

8. The A-Star 32U4 Bootloader

Our 32U4 family of boards come with a USB bootloader that can be used in conjunction with the Arduino IDE or AVRDUDE to load new programs onto the device. This section documents some technical details of the bootloader for advanced users who want to better understand how it works. If you just want to get started using your device, it is fine to skip this section.

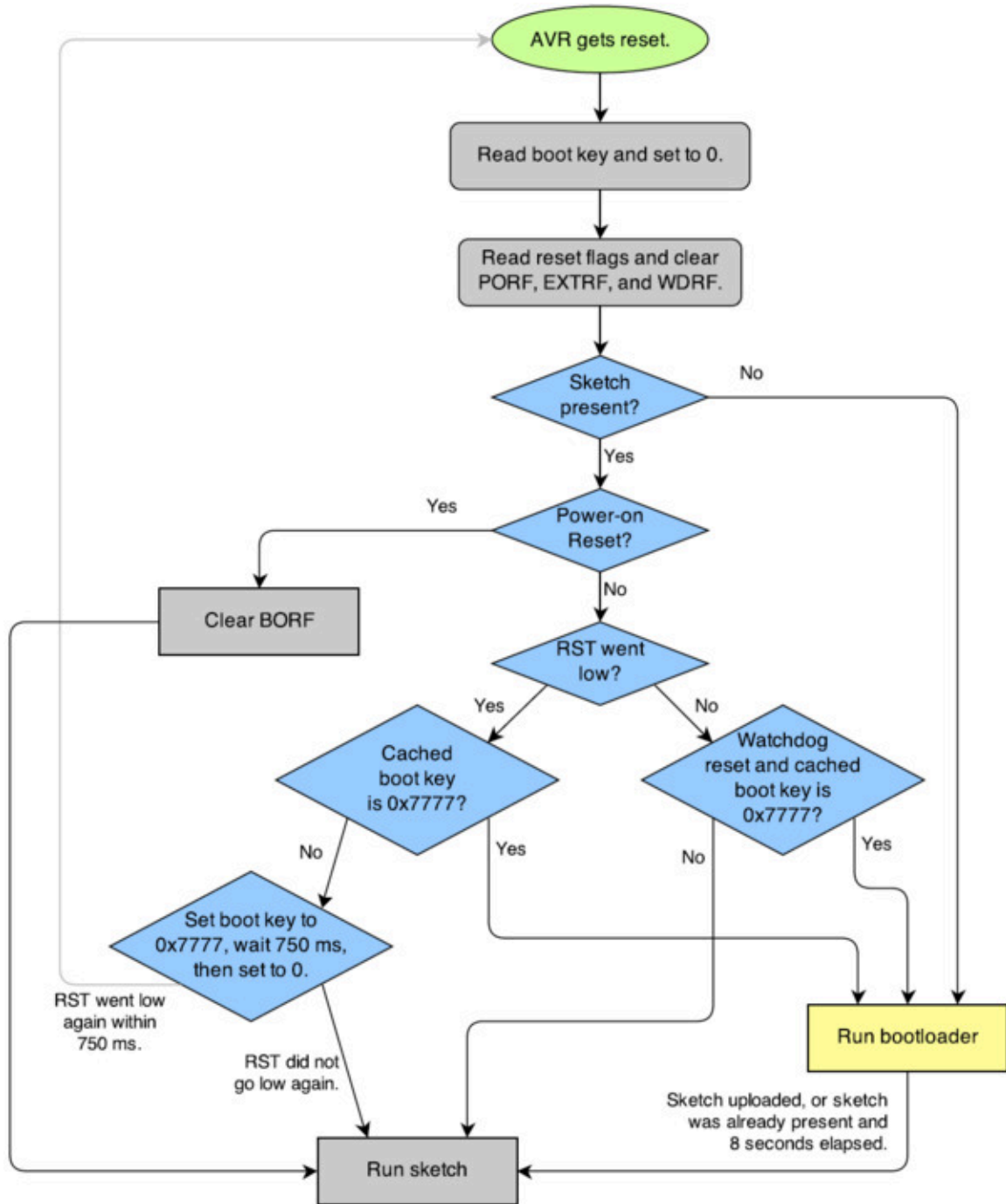
The A-Star 32U4 Bootloader is based on the **Caterina bootloader** [<https://github.com/arduino/Arduino/tree/master/hardware/arduino/avr/bootloaders/caterina>], which is the bootloader used on the **Arduino Leonardo** [<https://www.pololu.com/product/2192>], **Arduino Micro** [<https://www.pololu.com/product/2188>] and several other ATmega32U4 boards. The bootloader is open source and its **source code** [<https://github.com/pololu/a-star/tree/master/bootloaders/caterina>] is available on GitHub. The bootloader occupies the upper four kilobytes of the ATmega32U4's program memory, leaving 28 KB for the user program. The bootloader's USB interface consists of a single virtual serial port that accepts the programming commands defined in **AVR109** [<https://www.microchip.com/en-us/application-notes/an1644>]. The bootloader always runs first immediately after the AVR is reset.

Startup logic

The main difference between the A-Star 32U4 Bootloader and Caterina is in the startup logic. This is the part of the bootloader that runs immediately after the AVR is reset, and it decides whether to run the user program or run the rest of the bootloader. The startup logic of the Caterina bootloader is designed so that when the RST line goes low, the bootloader will run. This means that if you want to restart your program using the RST line, it will take 8 seconds before the bootloader times out waiting for an upload and the sketch starts.

The A-Star 32U4 Bootloader has different startup logic that allows you to use the RST line to reset the board with a smaller delay. If the RST line goes low once, the user program will run after a 750 ms delay. If the RST line goes low twice within 750 ms, then the bootloader will run. (This behavior is the same as on boards like SparkFun's Pro Micro.)

The start-up logic of the A-Star 32U4 Bootloader is shown in the flowchart below:



The startup logic for the A-Star 32U4 bootloader.

Brown-out detection

Unlike many other ATmega32U4 boards, our 32U4 family of boards have brown-out detection enabled. The brown-out threshold is 4.3 V, and if the voltage on VCC goes below this then the AVR will reset.

The bootloader was designed so that the user program can detect brown-out resets. To do so, check to see if the BORF bit in the MCUSR register is set, and then clear it later. Here is some example code you could put in your `setup` function for detecting brown-out resets:

```
1 pinMode(13, OUTPUT);
2 if (MCUSR & (1 << BORF))
3 {
4     // A brownout reset occurred. Blink the LED
5     // quickly for 2 seconds.
6     for(uint8_t i = 0; i < 10; i++)
7     {
8         digitalWrite(13, HIGH);
9         delay(100);
10        digitalWrite(13, LOW);
11        delay(100);
12    }
13 }
14 MCUSR = 0;
```


9. Reviving an unresponsive Romi 32U4

In order to load a new program onto your A-Star 32U4 device, you will need to get it into bootloader mode and send programming commands to it over its virtual serial port using appropriate software. If you are programming the device from the Arduino IDE, the sketch loaded onto the device will generally support a special USB command for putting it in bootloader mode, and the Arduino IDE sends that command automatically when you click the Upload button. However, you might find yourself in a situation where the device is unresponsive and that method will not work. This can happen for two reasons:

- You accidentally loaded a malfunctioning program onto the device that is incapable of responding to the special USB command. For example, your program might be stuck in an infinite loop with interrupts disabled.
- You loaded a program which uses a non-standard type of USB interface or no USB interface.

The following sections provide different procedures you can use to revive your device.

9.1. Reviving using the Arduino IDE

This section explains two special methods for programming an A-Star (or another of our 32U4 family of boards) using the Arduino IDE in case your usual method of programming is not working. These instructions were developed for the Arduino IDE versions 1.0.5-r2 and 1.6.0, and they might need to be modified for future versions.

Reset button

If you have an A-Star 32U4 Micro, you should connect a **momentary pushbutton** [<https://www.pololu.com/product/1400>] between the GND and RST pins to serve as a reset button. Other boards in our 32U4 family have a reset button you can use. Alternatively, you can use a wire to temporarily connect GND and RST together instead of using a reset button.

Resetting the board twice within 750 ms makes the board go into bootloader mode. The bootloader will exit after 8 seconds and try to run the sketch again if it has not started receiving programming commands. To revive the device, you need to make sure you start sending it programming commands before the 8-second period is over.

In bootloader mode, the yellow LED (the one labeled *LED 13*) fades in and out. It is useful to look at this LED so you can know what mode the microcontroller is in. Also, we recommend enabling verbose output during upload using the Arduino IDE's "Preferences" dialog. Looking at the LED and looking at the verbose output during the following procedures will help you understand what is going on.

The uploading-before-bootloader method

The goal of the uploading-before-bootloader method is to select a non-existent serial port in the Arduino IDE and then make sure the Arduino IDE enters the uploading phase before the microcontroller goes into bootloader mode. This method has been tested on Arduino 1.0.5-r2 and 1.6.0. This method does not work on Arduino 1.5.6-r2 or Arduino 2.0.0 because those versions of the IDE gives a fatal error message if the selected serial port is not present at the beginning of the uploading phase (e.g. “Board at COM7 is not available.” or “Upload error: Failed uploading: no upload port provided”).

1. Connect the device to your computer via USB.
2. In the “Tools” menu, open the “Board” sub-menu, and select “Pololu A-Star 32U4”.
3. In the “Tools” menu, open the “Port” sub-menu, and check to see if any ports are selected. If the “Port” menu is grayed out or no ports in it are selected, that is good, and you can skip to step 6.
4. Reset the board twice to get the board into bootloader mode. While the board is in bootloader mode, quickly select the new serial port that corresponds to the bootloader in the “Port” menu.
5. After 8 seconds, the bootloader will exit and attempt to run the sketch again. Wait for the bootloader to exit. Verify that either the “Port” menu is grayed out or no ports in it are selected.
6. Click the Upload button. The Arduino IDE will compile your sketch and start uploading it.
7. As soon as the large status bar near the bottom of the IDE says “Uploading...”, reset the board twice to get into bootloader mode.

The Arduino IDE will stay in the uploading phase for 10 seconds, waiting for a new serial port to appear. Once the serial port of the bootloader appears, the Arduino IDE will connect to it and send programming commands.

The bootloader-before-uploading method

The goal of the bootloader-before-uploading method is to select the bootloader’s virtual serial port in the Arduino IDE and then make sure the board is in bootloader mode at the time when the Arduino IDE enters the uploading phase.

1. Connect the device to your computer via USB.
2. In the “Tools” menu, open the “Board” sub-menu and check to see if the “Pololu A-Star 32U4 (bootloader port)” entry is visible. If this entry is visible, you can skip to step 6.
3. Using a text editor, open the file named boards.txt that provides the “Pololu A-Star” board entries. In Windows, you can typically find this file in `%LocalAppData%\Arduino15\`

`packages\pololu-a-star\hardware\avr\`. In Linux, you can typically find this file in `~/.arduino15/packages/pololu-a-star/hardware/avr/`. On macOS, you can typically find this file in `~/Library/Arduino15/packages/pololu-a-star/hardware/avr/`. If you installed the boards manually instead of using the Board Manager, you can find it in **[sketchbook location]/hardware/pololu/avr**.

4. In the `boards.txt` file that you opened, find the lines at the bottom of the file that start with `#a-star32U4bp`. Uncomment each of those lines by deleting the “#” character, and then save the file.
5. Close the Arduino IDE and restart it.
6. In the “Tools” menu, open the “Board” sub-menu and select “Pololu A-Star 32U4 (bootloader port)”. This entry is configured so that the Arduino IDE will send programming commands directly to selected serial port, instead of trying to send a special USB command to the port to get it into bootloader mode and then waiting for the new port to appear. By selecting this entry, the timing of the programming process below becomes easier, especially on Windows.
7. Prepare the computer to show you a list of its virtual serial ports. If you are using Windows, this means you should open the Device Manager. If you are on Linux or macOS, this means you should open a Terminal and type the command `ls /dev/tty*` but do not press enter until the board is in bootloader mode in the next step.
8. Reset the board twice to get the board into bootloader mode. While it is in bootloader mode, quickly look at the list of serial ports provided by your operating system in order to determine what port the bootloader is assigned to.
9. Reset the board twice to get the board into bootloader mode again. While the board is in bootloader mode, quickly select the serial port of the bootloader in the Arduino IDE. The port can be selected in the “Port” sub-menu under “Tools”.
10. In the Arduino IDE, click the “Verify” button to compile your sketch. This could make the timing easier during the next step.
11. Press the reset button twice to get the board into bootloader mode again. As soon as you see the yellow LED fading in and out, press the Upload button.

The Arduino IDE will compile your sketch and then upload it to the selected serial port.

If the compilation of the sketch takes longer than 8 seconds, then this procedure will fail because the bootloader will time out and start trying to run the malfunctioning sketch again. If that happens, try the procedure again using a simpler sketch such as the Blink example that can be found under **File > Examples > 01.Basics > Blink**.

After reviving your device, be sure to change the Board setting back to “Pololu A-Star 32U4” and select the right Port.

9.2. Reviving using AVRDUDE

This section explains a special method for reviving an A-Star (or another of our 32U4 family of boards) using the command-line utility **AVRDUDE** [<http://www.nongnu.org/avrdude/>] in case your usual method of programming is not working. AVRDUDE stands for “AVR Downloader/UplodaDEr”, and it is compatible with the A-Star bootloader.

If you have an A-Star 32U4 Micro, you should connect a **momentary pushbutton** [<https://www.pololu.com/product/1400>] between the GND and RST pins to serve as a reset button. Other boards in our 32U4 family have a reset button you can use. Alternatively, you can use a wire to temporarily connect GND and RST together instead of using a reset button.

1. Connect the device to your computer via USB.
2. Prepare the computer to show you a list of its virtual serial ports. If you are using Windows, this means you should open the Device Manager. If you are on Linux or Mac OS X, this means you should open a Terminal and type the command `ls /dev/tty*` but do not press enter until the board is in bootloader mode in the next step.
3. Press the reset button twice within 750 ms to make the AVR go into bootloader mode. You should see the yellow LED fading in and out when the AVR is in bootloader mode. While it is in bootloader mode, quickly look at the list of serial ports provided by your operating system in order to determine what port the bootloader is assigned to.
4. Type the following command in your terminal and replace COM4 with the name of the bootloader's serial port, but do not press enter yet. This command will erase the malfunctioning program on the device but preserve the bootloader.

```
avrdude -c avr109 -p atmega32U4 -P COM4 -e
```

5. Press the reset button twice within 750 ms to make the AVR go into bootloader mode.
6. Quickly run the command you typed previously. The command needs to be run within 8 seconds of starting the bootloader, or else the bootloader will exit and try to run the malfunctioning program again.

By following the instructions above, the malfunctioning program on the device will be erased and the device will stay in bootloader mode indefinitely. You can now load another program onto it using the Arduino IDE or AVRDUDE.

10. Related resources

To learn more about using the Romi 32U4 Control Board, see the following list of resources:

- The Arduino IDE has many **examples** [<http://arduino.cc/en/Tutorial/HomePage>] that can run on the Romi 32U4 Control Board (although note that the control board's on-board hardware might conflict with some of these examples).
- The Arduino website has a **Language Reference** [<http://arduino.cc/en/Reference/HomePage>], a wiki called the **The Arduino Playground** [<http://playground.arduino.cc/>], and other resources.
- The Romi 32U4 Control Board uses the same microcontroller as the **Arduino Leonardo** [<https://www.pololu.com/product/2192>] and **Arduino Micro** [<https://www.pololu.com/product/2188>], so you can search the Internet for relevant projects and code examples that use one of those boards.
- **Atmel's ATmega32U4 documentation** [<https://www.microchip.com/en-us/product/ATmega32u4>] has the ATmega32U4 datasheet and many related documents.
- **AVR Libc Home Page** [<http://www.nongnu.org/avr-libc/>]: this page documents the standard library of functions that you can use with GNU C and C++ compilers for the AVR.
- **Romi 32U4 Arduino library** [<https://github.com/pololu/romi-32u4-arduino-library>]
- **Romi 32U4 library documentation** [<https://pololu.github.io/romi-32u4-arduino-library/>]
- **LUFA – the Lightweight USB Framework for AVRs** [<http://www.fourwalledcubicle.com/LUFA.php>]
- **WinAVR** [<http://winavr.sourceforge.net/>]
- **Microchip Studio for AVR and SAM Devices** [<https://www.microchip.com/en-us/tools-resources/develop/microchip-studio>]
- **AVRDUDE** [<http://www.nongnu.org/avrdude/>]
- **AVR Freaks** [<https://www.avrfreaks.net/>]

Datasheets for some of the components found on the Romi 32U4 Control Board are available below:

- **ATmega32U4 documentation** [<https://www.microchip.com/en-us/product/ATmega32u4>]
- **Texas Instruments DRV8838 motor driver datasheet** [<https://www.pololu.com/file/0J806/drv8838.pdf.redirect>] (1k redirect)
- **ST LSM6DS33 3D accelerometer and 3D gyro module datasheet** [<https://www.pololu.com/file/0J1087/LSM6DS33.pdf>] (1MB pdf)
- **Texas Instruments TPS2113A power multiplexer datasheet** [<https://www.pololu.com/file/>]

[0J771/tps2113a.pdf.redirect](#) (1k redirect)

Finally, we would like to hear your comments and questions on the **Pololu Robotics Forum** [<https://forum.pololu.com/>]!