Intel Development Board

Shield Testing Report for the Arduino 101* Board

January 2017

Revision 001

This document contains information on products, services, and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications, and roadmaps.

The products and services described may contain defects or errors known as errata, which may cause deviations from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a nonexclusive, royalty-free license to any patent claim thereafter drafted that includes subject matter disclosed herein.

Forecasts: Any forecasts of requirements for goods and services are provided for discussion purposes only. Intel will have no liability to make any purchase pursuant to forecasts. Any cost or expense you incur to respond to requests for information or in reliance on any forecast will be at your own risk and expense.

Business Forecast: Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

Copies of documents that have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting *www.intel.com/design/literature.htm*.

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication, or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the government, in accordance with the software license agreement as defined in FAR 52.227-7013.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

The code names presented in this document are only for use by Intel to identify products, technologies, or services in development that have not been made commercially available to the public, i.e., announced, launched, or shipped. They are not "commercial" names for products or services and are not intended to function as trademarks.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com/design/literature.htm.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

Intel and the Intel logo are trademarks of Intel Corporation in the US and other countries.

Revision History

Revision	Description	Date
001	Initial public release.	January 2017

Contents

Figures

Tables

Example sketches

Testing Overview

0.1 Introduction to the testing process

This document provides guidance for using shields and sensors installed with an Intel® Curie™-based development board, namely, the Arduino* 101 board. The document is based on tests performed to evaluate hardware and library compatibility with the x86 architecture and presents the results, describing any known issues. Results are expected to the same for other boards that are based on the Arduino* 101 design, such as the tinyTILE* board, assuming that some type of interposer or adapter is used.

For comparison, each test was first performed on an Arduino* Uno R3 board before executing it on the Arduino 101 production board.

The testing for each device was carried out as follows:

- Examine documentation related to the shield and its companion library.
- Analyze the description and explanation of the shield's functionality in the documentation or elsewhere.
- Study the hardware specification of the shield and check for compatibility with Arduino 101 boards before attempting to connect.
- Examine the general structure of the companion library and check for AVR-specific code.
- Test on the Arduino Uno R3 board.
- Compile against x86 architecture and attempt to fix any errors.
- Upload and perform functional testing.
- Attempt to resolve problems and report the best results obtained for key features of the shield.

The following sections of this overview chapter present some terminology and references, and then cover important information with respect to setting up the shields for testing and use. The rest of this report provides the results of our testing.

0.2 Terminology

Table 1 **Terminology**

Term	Definition
ADC	Analog-to-digital converter
BLE	Bluetooth* Low Energy
BOM	Bill of materials
CTS	Clear to send
FW	Firmware
IDE	Integrated Development Environment - A software application where it is possible to edit source code, compile source code, debug and start the application created. The Arduino* IDE also enables the upload of the code to different boards and monitor program output via serial monitor.
NFC	Near Field Communication
OS	Operating system
OTP	One-time programmable
PWM	Pulse width modulation
RAM	Random access memory
SW	Software
UART	Universal asynchronous receiver transmitter.

0.3 References

The following documents provide useful information about

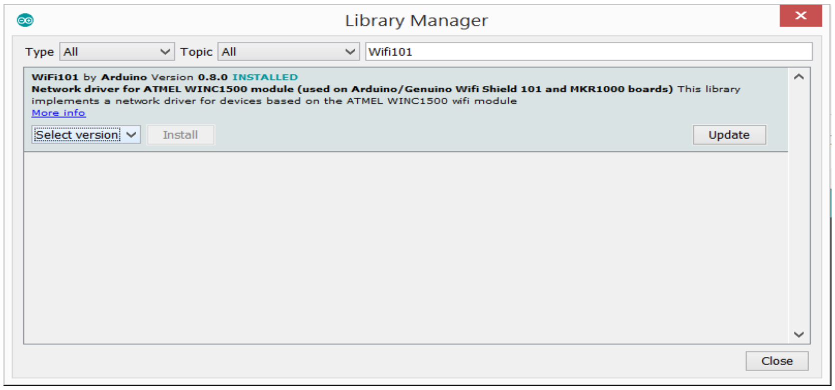
- Intel® Curie™ Module Datasheet
- Intel® Curie™ Module Design Guide

- Intel® Curie™ Open Developer Kit (ODK) (online)

0.4 Adding third party libraries

Beginning with Version 1.6.8 of the Arduino* IDE, it is advisable to use the Library Manager rather than download third party libraries directly from the vendor's website. This is preferable because the latest version of a library may not always work for a specific board and its version of the corelibs. The Arduino project maintains multiple versions of libraries while the vendor may not, making it necessary to search for a version tag in the vendor's repository or in GitHub. We recommend that you first use the Library Manager to search for a particular library and then go down in version if the latest does not compile or has run time issues. Only if you cannot find the library through the Library Manager should you get it from the vendor's website or recommended URL. To get to the Library Manager go to *Sketch Menu -> Include library -> Manage Libraries*. You can type the first letters of the library name or scroll down to the list to find a desired library.

Figure 1 **Library Manager. The graphic shows an example search for *Wifi101*. It shows that Version 0.8.0 is installed even though the newest is 0.9.1**



Wifi101

WiFi101 by **Arduino** Version **0.8.0** INSTALLED
Network driver for ATMEL WINC1500 module (used on Arduino/Genuino Wifi Shield 101 and MKR1000 boards) This library implements a network driver for devices based on the ATMEL WINC1500 wifi module
More info

0.5 SD card library

The Arduino* 101 board supports shields that have SD card sockets. However, the Arduino 101 board itself does not have an SD card socket onboard. If SD functionality is needed, the *SD.h* file needs to be included in the sketch.

0.6 Wi-Fi library

For the Arduino* 101 board, the Wi-Fi Library and the Arduino* Uno Wi-Fi shield (*https://www.arduino.cc/en/Main/ArduinoWiFiShield*) have been retired, but are still functional. The new alternative is the Wi-Fi 101 shield (*https://www.arduino.cc/en/Main/ArduinoWiFiShield101*) but it has not been integrated with the IDE and the library has to be downloaded as a custom library from this GitHub project (*https://github.com/arduino-libraries/WiFi101*).

0.7 Inter-Integrated Circuit (I^2C)

The Arduino* 101 platform supports I^2C for Arduino using the Wire Library. I^2C is used on the Arduino 101 board to write to the EEPROM, interface shields, etc. An Arduino sketch communicates over I^2C via the A4/A5 analog pins. I^2C can also communicate over SCL/SDA which is the same A4/A5 analog pins. However, these pins are not always routed up to a shield.

Serial Peripheral Interface (SPI)

The Arduino* 101 board can interface a SPI product that uses either the LSB or MSB format mode.

The Arduino 101 board can use any pin as the Chip Select (CS or Slave Select/SS) pin.

0.8 IOREF voltage

The Arduino* 101 board only supports 3.3 V signals to the GPIO pins. A level shifter can be used as a workaround for pins that require 5 V. The pins are, however, 5 V tolerant and in cases where the logical 1 is triggered below 3.3 ,V, the board might work. Consequently, most shields that generate 5V to the pins are supported. Most shields tested worked fine without a level shifter. If there are any issues, a level shifter chip that has two-way shifting should be used for the pins that are being used. The *analogRead()* function used with the analog pins will only support up to 3.3V.

0.9 Test hardware

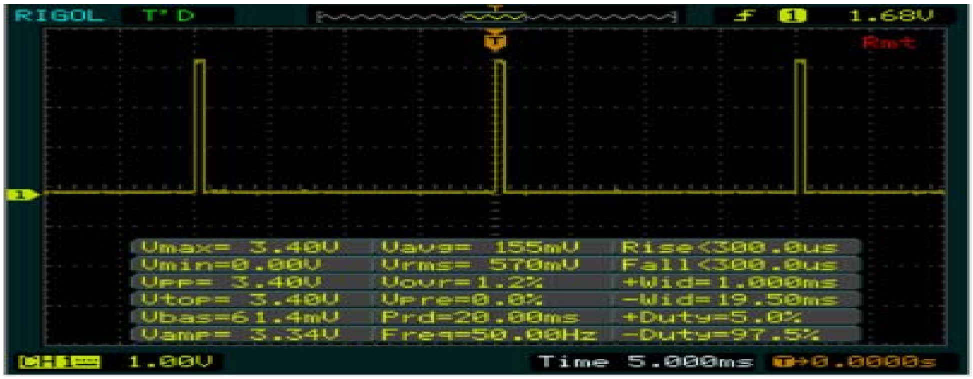
Some signals were verified using an oscilloscope. A DC power supply was used to produce external power. The table below lists the hardware used in the tests for this report.

Key Info	Description / Links
Variable DC Power Supply	*http://www.amazon.com/gp/product/B000CSQK5E?ref_=oh_details_o00_s00_i00&redirect=true&psc=1&pldnSite=1* Primarily used for motor testing. This generated external power for specific shields.
Oscilloscope	*http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,842,1018&Prod=ANALOG-DISCOVERY&CFID=4899125&CFTOKEN=d6665606d1992697-11280C15-5056-0201- 02608083BE4E9864* Analog Discovery USB Oscilloscope. Many of the oscilloscope screen shots were done with this software as was verification of pin activity.
Saleae Logic Analyzer, Logic Pro 8	*https://www.saleae.com* Used this logic analyzer to verify packet (I2C, SPI, UART, etc.) data

0.10 PWM (pulse width modulation)

DC and servo motors most often use pulse width modulation (PWM) for speed control. On Arduino* 101 boards, the supported PWM channels are 3, 5, 6 and 9. There are some motor shields which use pin 11 (by default) for PWM. One option is to bend/cut the pin on the motor shield and re-direct the pin (from the header) with jumper wire to one of the remaining PWM pins and update the sketch for that pin. Another option is to use software PWM support using the *CurieTimerOne* PWM functions. This feature enables the sketch to allocate any digital pin to support PWM. Only one pin at a time is supported. As an added benefit the period/frequency is customizable from within a sketch. The minimum size for *CurieTimerOne* pulses is 10 microseconds (positive or negative duty). The *CurieTimerOne* > *CurieTimer1PWM* example sketch provides an example for the usage of the functions. The report on the *Arduino* Motor Shield R3* provides an example of a sketch that uses this API. *Figure 2* shows an oscilloscope screenshot of a servo motor pulse. The frequency for PWM is normally set to either 50 or 60 Hz. For servo motors, the range of the positive duty cycle of the pulse is between 1 and 2 ms. The servo motor should stop when the pulse width is at 1.5 ms, rotate in one direction if it is between 1.0 and 1.5 ms, and rotate in the other direction when the pulse width is between 1.5 and 2.0 ms.

Figure 2 **Servo PWM oscilloscope output for an Intel® Curie™ board**



```
Vmax= 3.40V    Vavg= 155mV      Rise<300.0us
Vmin=0.00V     Vrms= 570mV      Fall<300.0us
Vpp= 3.40V     Vovr=1.2%        +Wid=1.000ms
Vtop= 3.40V    Vpre=0.0%        -Wid=19.50ms
Vbas=61.4mV    Prd=20.00ms      +Duty=5.0%
Vamp= 3.34V    Freq=50.00Hz     -Duty=97.5%
```

0.11 *CurieTimerOne* interrupts

Arduino* 101 boards have support for configurable timed callback functions using the *CurieTimerOne* API. This collection of functions also includes the PWM functions that were described in the previous section. A sketch can define a single function that is called back in intervals. The minimum time for accuracy is around 10 microseconds. The *CurieTimerOne > CurieTimer1Interrupt* example sketch illustrates the usage of the functions.

0.12 USB cables and sketch uploads

The Arduino* 101 sketch upload mechanism involves uploading working firmware using *dfu-util* with the sketch embedded in it. We have found that the quality of the USB cable is important. Any interference can affect the sketch upload process. We recommend the shortest possible USB cable, and one with a ferrite choke. The following is an example that can be obtained through various online suppliers:

> *http://www.amazon.com/Tripp-Lite-Hi-Speed-Ferrite-U023-006/dp/B003MQ29B2/ref=sr_1_cc_1?s=aps&srs=13989227011&ie=UTF8&qid=1458065247&sr=8-1-catcorr&keywords=ferrite+usb+-mini+-Micro*.

Figure 3 **USB cable with Ferrite Choke**

0.13 Test motors

DC motors with gears were used to reduce the noise levels. Before you apply power from an external power source, verify the supported voltage level.

Table 2 **DC motors**

#	Motor	Voltage	Gear ratio/torque	Link
DC1	Standard gear motor	3 to 12 VDC	18:1	*https://www.sparkfun.com/products/12144*
DC2	Micrometal gear motor	6 VDC	100:1	*https://www.sparkfun.com/products/retired/8910*

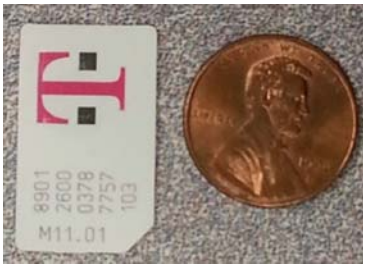
Table 3 **Servo motors**

#	Motor	Voltage	Gear ratio/torque	Link
SV1	Analog miniservo	4.8 to 6 VDC		*http://www.SeeedStudio.com/depot/EMAX-9g-ES08A-High-Sensitive- Mini-Servo-p-760.html?gclid=CPzajsTKqb8CFYhafgodCKsA7A*
SV2	Metal gear servo	4.8 to 6 VDC		*https://www.adafruit.com/products/1142*
SV3	Digital servo	9 to 12 VDC	254:1	*http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_ actuator.htm*
SV4	Digital servo	6.5 to 8.4 VDC		*http://www.dfrobot.com/index.php?route=product/product&filter_name=ser0026&product_id=373*
SV5	Continuous rotation, parallax	6 VDC max	Torque 3.4 kg-cm / 47 oz-in	*http://www.frys.com/product/5230077*
SV6	Digital servo LS- 8101F	6 to 7.2 VDC	Stall torque 12 kb-cm / 13 kg-cm	*http://www.frys.com/product/7027291*
SV7	Feedback servo	6 V	25 oz*in / 1.8 kg*cm	*https://www.adafruit.com/products/1450*

Table 4 **Stepper motors**

#	Motor	Voltage	Gear ratio/torque	Link
ST1	Bipolar	12 VDC 350 mA/ phase	200 steps/rev 1.8 degrees	*https://www.adafruit.com/products/324* Red/Yellow Green/Gray
ST2	Unipolar 5 wire	5 V	32 steps/rev 11.25 degrees 1/64 Gearing	*https://www.adafruit.com/products/858* Pink/Orange/Red Yellow/Blue/Red For bipolar, ignore red.
ST3	Unipolar 5 wire	5 to 12 VDC	32 steps/rev 1/16 Gearing	*https://www.adafruit.com/products/918* Pink/Orange/Red Yellow/Blue/Red For bipolar, ignore red.
ST4	Unipolar 6 wire	3 VDC 2A/Phase	200 steps/rev	*https://www.sparkfun.com/products/10847* Black/Yellow/Green Red/White/Blue For bipolar, ignore yellow/white.

0.14 SIM card for GSM/GPRS shields



For testing all the GSM/GPRS shields it was necessary to have an unlocked SIM card from a communications provider. The communications provider must either provide GSM coverage in the area of testing or have a roaming agreement with a company that provides GSM coverage in the test locality.

The SIM card used in this testing was from T-Mobile*. A prepaid plan with US $10 refills was purchased. It was necessary to change the plan to a pay by day usage model so that GPRS service could be provided for Internet testing. The service on these days provided unlimited calls/text/data although the data bandwidth was throttled. The SIM card used for all the testing in this document is the Mini-SIM (2FF) 25 × 15 mm.

0.15 Hardware serial ports

For the Arduino* 101 board, 'Serial1' is mapped to D0/D1 for transmit and receive. D0/D1 is a HardwareSerial port. SoftwareSerial is also available and can communicate with 'Serial1' for most baud rates.

Table 5 **Sketch serial ports**

Board	# Ports	Serial1	Serial2	Software Serial
Arduino* 101	1	DO/D1	N/A	Supported

0.16 On-board clock

The Intel® Curie™ has an RTC (Real Time Clock) that allows a sketch to keep track of long timelines like a watch. There is no battery to keep the clock running if an Arduino* 101board is disconnected from power. There are DS1307 Real Time Breakout Boards that can be connected which will keep the time using a battery even when the board is disconnected from power.

Arduino* 101 firmwareThe Arduino* 101 firmware is set in the factory. The procedure for updating the firmware is available at: *https://downloadmirror.intel.com/25832/eng/Readme.pdf*.

0.17 Official firmware sources

Arduino* 101: *https://downloadcenter.intel.com/download/25832*

0.18 Arduino* direct port manipulation

On the Arduino* Uno board, software access to the direct ports (analog and digital pins) is abstracted by means of APIs (digitalRead and digitalWrite). The advantage of using these APIs is the compatibility of the sketches between microcontrollers. The disadvantage of using these APIs is that the API may introduce overhead that may slow down the pin access. To overcome the performance loss, library and sketch writers uses a means of directly manipulating the microcontroller's registers. Unfortunately, registers are not compatible between microcontrollers. These access types may cause compile errors. Sketches or Libraries accessing the ports directly can be determined by the presents of headers files referencing the "avr" directory. One example is the *Arduino* TFT Screen*, which uses the TFT library.

Example of header files used for Direct Port Manipulation

<avr/pgmspace.h>
<avr/sleep.h>
void setup() {}
void loop() {}

0.19 Software serial ports

On the Arduino* Uno board, for devices that need additional serial port(s) for transfer or communication, there is a SoftwareSerial library that can be used.For Arduino* 101, SoftwareSerial is supported, however, only one port is configurable on any pair of digital pin. Since only one is supported, this means that SoftwareSerial to SoftwareSerial communication is not possible. However, HardwareSerial to SoftwareSerial is functional for most baud rates.

0.20 PROGMEM / F( ) macro

On the Arduino* Uno board, it is possible to store data in flash (program) memory using the 'PROGMEM' statement. There is also another macro call "F() macro", that indirectly does the same thing. These are routinely found in Arduino sketches. Eliminating these macros can be time consuming; however, the following example can eliminate the use of this special macro with one line. If a shield library contains F() macros, then this workaround should be placed as the first #define statement to the main header file of the library as removing the F() macros can be tedious and error prone.

Example 1 **Removing F() macro sketch**

Before:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("setup");
}
void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(F("Loop"));
}
```

After:

```
#define PSTR(arg)(arg)
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("setup");
}
void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(F("Loop"));
```

0.21 Installing board-specific tools and libraries using the Arduino* IDE

Starting from version 1.6.4 of the Arduino* IDE from *arduino.cc* (not to be confused with the IDE from *Arduino.org*), additional board packages are installed by using the board manager tool in the IDE. There is no longer a need for an Intel specific IDE. Arduino* 101 installable board support came with version 1.6.7. Installable board support means that although the board manager lists the Intel boards. the user will have to initiate their installation, or their boards will not appear in the board list. An Internet connection is required for these additional board support packages to be downloaded.

Figure 4 **Default 1.6.7 IDE With No Intel or additional boards Installed**

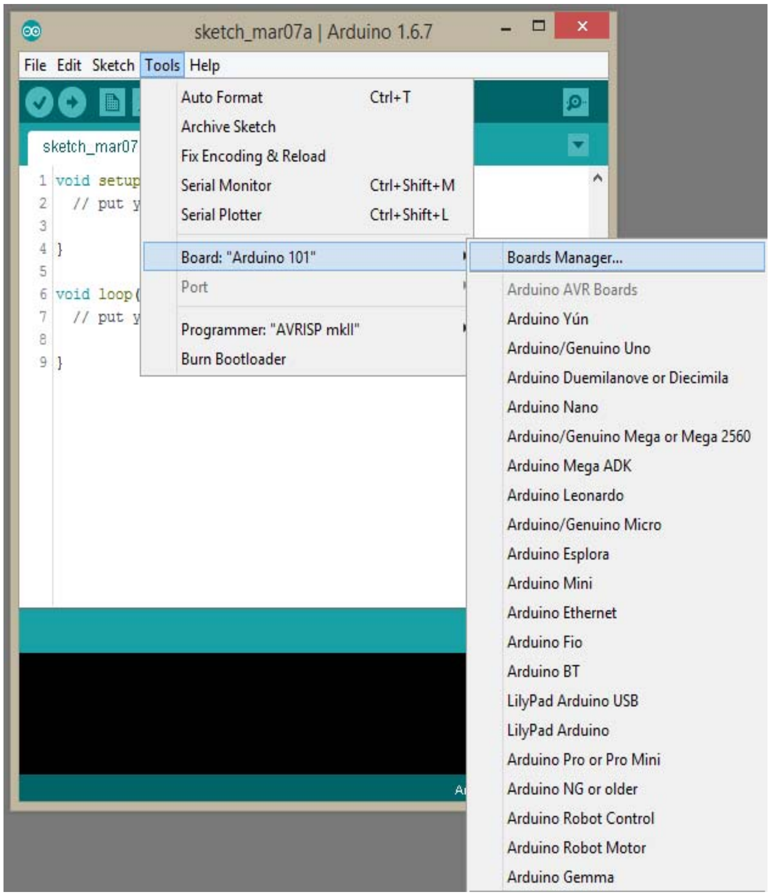
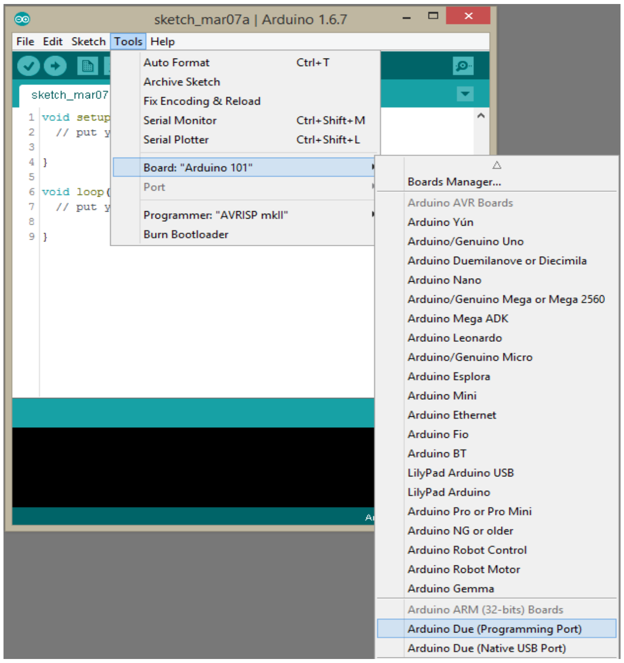


Figure 5 **IDE with Additional boards Added**



The board support package, also know as core-libs, are installed in a different path from the default built-in boards which are installed inside the folder where the IDE package is installed. For additional boards, the files are installed in different locations depending on the operating system. These locations cannot be changed.

For Linux it is in the {USER HOME}/.arduino15 (ex: /home/bob/.arduino15. For Mac OSX it is at {USERHOME}/Library/Arduino15 (ex: /Users/bob/Library/Arduino15). For Windows it is at %LOCALAPPDATA%\Arduino15 (ex: C:\Users\bob\AppData\Local\Arduino15). This is often referred to as the Arduino15 folder.

The board support packages for Intel maker boards are located in the Arduino15 folder/packages/Intel/hardware and Arduino15 folder/packages/Intel/tools. Arduino 101 packages are in arc-elf32. Each version is within another folder so for the 1.6.7 version it will be in a folder named like so 1.6.7+1.0. Tools which include compilers and upload tools are in Arduino15 folder/packages/Intel/tools but do not necessarily follow the naming conventions above. The hardware (core-libs,) and tools folders may have different version numbers and naming conventions and do not necessarily increment together.

0.21.1 Install and Compile issues

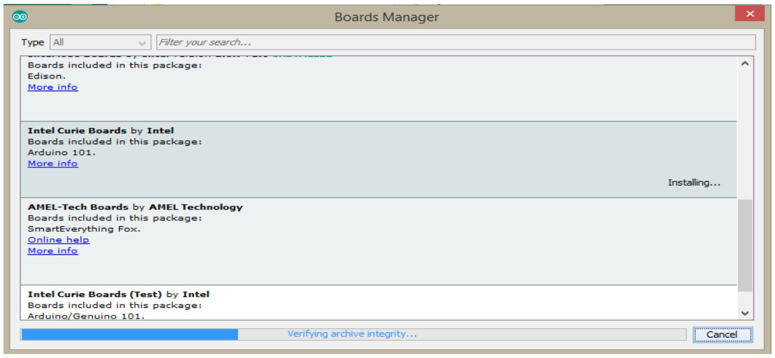
Many board support issues like compilation, sketch upload, serial monitoring and compatibility can be resolved by deleting files in the Arduino15 directory. Uninstalling a board can be done by the board manager but this does not always work perfectly. It is not recommended to install more than one version of a board support package even if the IDE allows this. A restart of the IDE is important whenever installing boards, uninstalling boards, deleting files in the Arduino15 directory or adding 3rd party libraries. Folders should not be copied from another Arduino15 library from another user or computer. The board manager is the only method allowed to install board support.

0.21.2 Installing board support for Arduino* 101

To install Arduino* 101 board support, open Board Manager and scroll down until the option for Intel® Curie™ boards is found. This selection will only be available while using IDE version 1.6.7 or greater. There may or may not be a version drop down. Click on Install.

After installation the appropriate board will be available for uploading sketches. Installable board support will only be available on the Arduino* IDE downloaded from *https://www.arduino.cc/en/Main/Software*.

Figure 6 **Installing Arduino* 101 boards**



Boards included in this package:
Edison.
More info

Intel Curie Boards by **Intel**
Boards included in this package:
Arduino 101.
More info
Installing...

AMEL-Tech Boards by **AMEL Technology**
Boards included in this package:
SmartEverything Fox.
Online help
More info

Intel Curie Boards (Test) by **Intel**
Boards included in this package:
Arduino/Genuino 101.

Verifying archive integrity...

If there are any compile or upload errors, or if the installation happens very quick (under 5 seconds), remove the Intel folder inside Arduino15\packages and also clear out the Arduino15\staging folder. The Arduino IDE downloader is slow by design.

0.21.3 Dealing with network errors

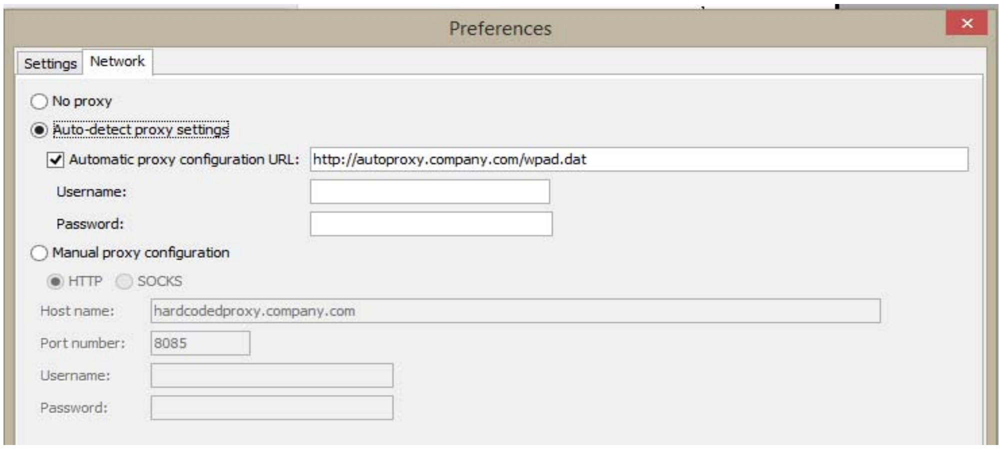
One of the symptoms that the IDE cannot connect with the Arduino repository on the Internet is that when opening Board Manager it will show an error in the bottom in red. “Error downloading http://downloads.arduino.cc/packages/package_index.json”. Copy the URL below and paste it in the browser. If the Internet connection is good, a text file in XML format is displayed in the browser or the browser will download a file with a json extension.

If behind a proxy the IDE should be set with similar proxy settings as the browser or as set by the network administrator. Proxy can be set by auto-detecting the system settings. In Windows this is accomplished by getting the setting in the Internet Explorer browser and in Linux and Mac by the Global or system-wide settings. There is also an auto-detect method and an option to enter an automatic configuration URL. The other way is to use manual settings where the host name or IP of the proxy server and port is entered.

It is best to use the manual setting rather than using the auto-detect method. In Linux and Mac, the manual settings seems to be the only way to get the IDE to communicate on some corporate proxies.

Since the Arduino* IDE is a Java application, it is possible that a Java application, user or administrator has set an environmental variable called JAVA_OPTIONS in Windows and JAVA_TOOL_OPTIONS in Linux and set values could also control the proxy behavior of the IDE. It is recommended to clear out http_proxyHost and http_proxyPort settings in these variables.

Figure 7 **Proxy Settings in the IDE preferences' network tab**



○ No proxy
● Auto-detect proxy settings
☑ Automatic proxy configuration URL: http://autoproxy.company.com/wpad.dat
Username:
Password:
○ Manual proxy configuration
● HTTP ○ SOCKS
Host name: hardcodedproxy.company.com
Port number: 8085
Username:
Password:

Another possible cause of network issues is that some firewall and anti-virus applications are really stringent. It might be necessary to manually create an exception to the Java process which is bundled with the IDE. In Windows it might be required to make an exception to {Arduino_Install_Folder}\java\bin\java.exe and javaws.exe. In Windows, after running an IDE installation for the first time, there will be a Windows security dialog asking if it should allow Java (remember the IDE is Java application) to use the networks. If entering no, then this will have an impact on the IDE's network connectivity. Also note that if installing the IDE multiple times, there will be an exception for each installation because Java is bundled in the IDE.

0.21.4 Board connectivity

Download the proper USB and serial drivers for Arduino* 101 if using the Windows OS.

Arduino* 101 drivers are installed as part of the Arduino 101 board installation from the IDE's Board Manager.

0.21.5 Driver installation issues in Windows 7

If Windows 7 has just been installed and there have not been any major Windows updates, it is possible that there will be driver installation issues. This is because the root certificates have not been updated yet. Since Windows drivers need to be signed and Intel drivers are using current root certificates, Windows will prevent driver installations. Please update the OS and retry installing the driver again.

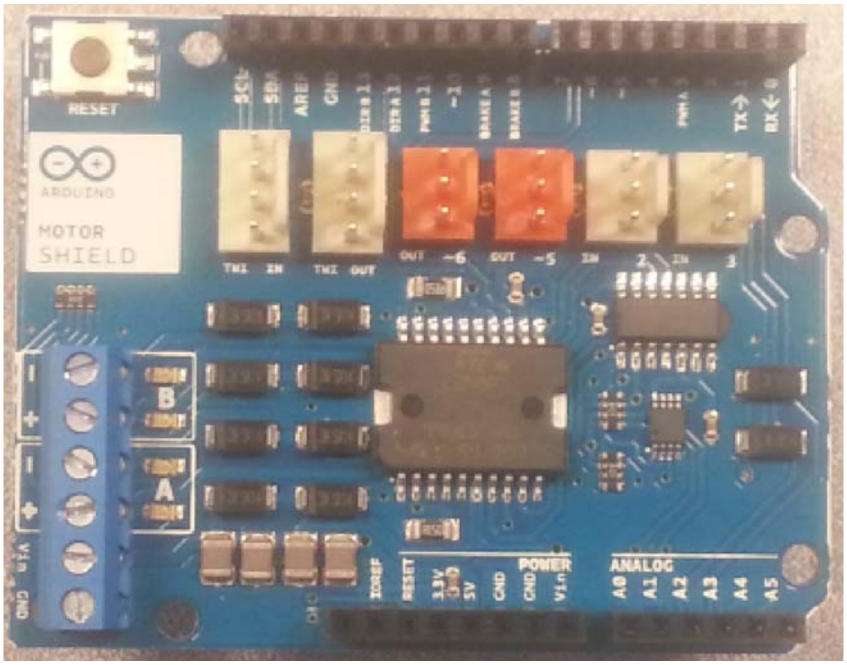
1 Arduino* Motor Shield R3

1.1 Use case

The Arduino* Motor Shield is mainly used to drive DC motors and less frequently, stepper motors. With its H-bridge (L298) the motor shield can drive two DC motors up to 0.75 A each. The shield has two separate channels that use four Arduino* 101 pins each to drive, set direction, brake or sense the motor current. It is possible to use each channel separately to drive two DC motors or combine them to drive one bipolar stepper motor. If the current sensing and brake features are not needed and more pins for an application are needed, the features can be disabled by cutting the respective jumpers on the backside of the shield.

Key Info	Links
URL	*http://arduino.cc/en/Main/ArduinoMotorShieldR3*
Tutorial	*http://arduino.cc/en/Tutorial/DueMotorShieldDC*
Library	None.

Figure 8 **Arduino* Motor Shield**

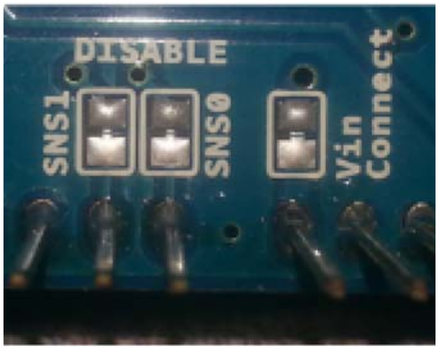


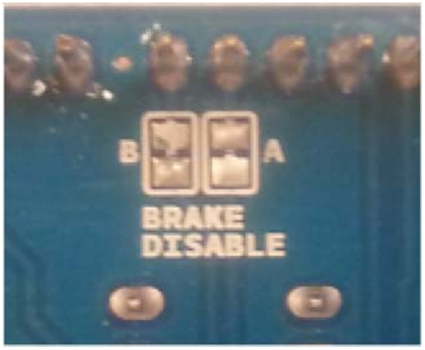
1.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 to 12 V
Maximum current	2 A per channel or 4 A max (with external power supply).
IOREF	3.3 or 5 V. (See *Section 0.8 IOREF voltage*.)
Use VIN for power source	Yes or option of using external power supply if over 5 V needed.
Motor controller	L298P drives two DC motors or one stepper motor. The controller has two separate power connections, one for logic and one for the motor supply driver. *http://www.st.com/web/en/catalog/sense_power/FM142/CL851/SC1790/SS1555/PF63147*
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.7
Current sensing	1.65 V/A
Motors	DC1, DC2

Jumpers:

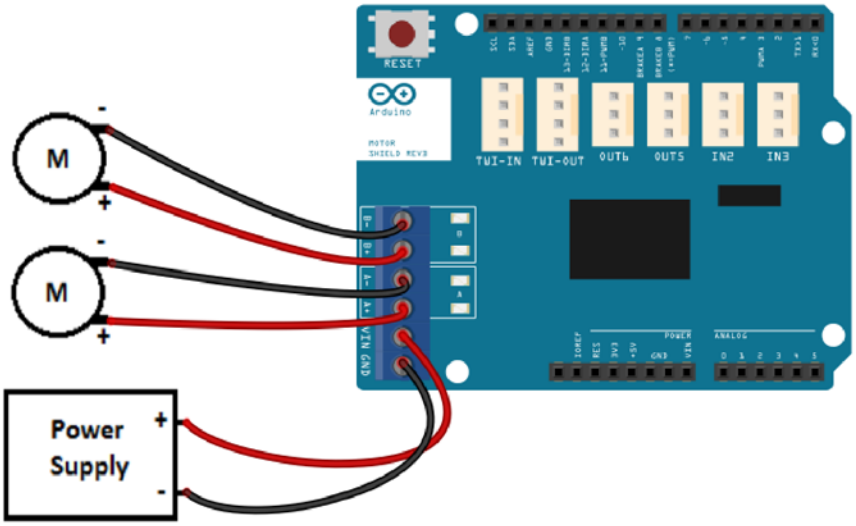
- **Arduino* 101 boards only**: If using an external power supply and using the barrel jack power supply for board logic then cut the “Vin Connect” jumper.
- If you don’t need brake and current sensing, you can free the pins by cutting these TinkerKit* pin jumpers on the shield:
  - Two connectors for two analog inputs (in white), connected to A2 and A3.
  - Two connectors for two analog outputs (in orange, in the middle), connected to PWM outputs on pins D5 and D6.
  - Two connectors for the TWI interface (in white, with 4 pins), one for input and the other for output.





Wire the motor as shown in *Figure 9*.

Figure 9 **Connect two DC motors with an external power supply**



The sketch provided runs one motor at a time. To run motor A (bottom), change variables to the following:

Pin Name	Motor A	Motor B	Function for motor
Direction	D12	D13	Set direction of motor
PWM for speed	D03	D11	Pulse width modulated (PWM) speed for motor. Arduino* 101 boards don't have analogWrite() PWM support on pin 11 so the pin can be bent out and jumpered to pin 5, 6 or 9 or CurieTimerOne PWM can be used on any pin. The sketch must be updated
Brake	D09	D08	Brake for motor
Current sensing	A0	A1	Current sensing for motor – Analog

The shield can work in both 5 V and 3.3 V mode. The current sensing pins (1.65 A/V) range from 0 to 3.3 V maximum, so it is compatible with the 3.3 V configuration also

1.3 Companion library

Not required.

1.4 Compile and upload

The following sketch can be used to set different speeds, directions, and braking patterns.

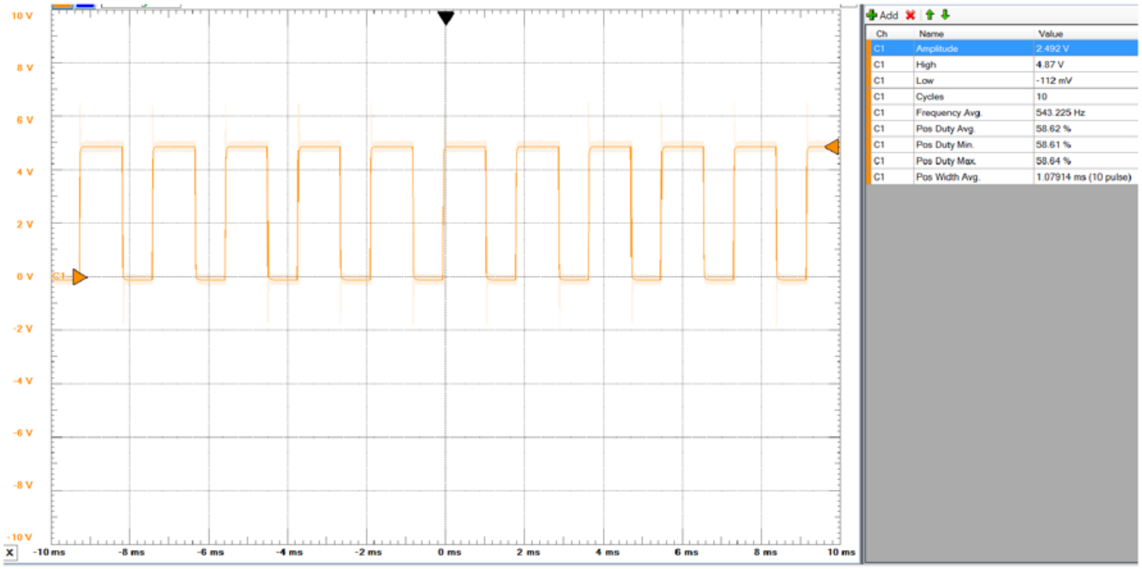
Example 2 **DC motor sketch**

```
// DC motor test
#define ARDUINO101  // comment this line if not using Arduino101

#include "CurieTimerOne.h"
#define FREQUENCY   980 // need to specify a freqency for TimerOne

int DIR_A   = 12;
int PWM_A   = 3;
int BRAKE_A = 9;
int SNS_A   = A0;

int DIR_B   = 13;
int PWM_B   = 11;
int BRAKE_B = 8;
int SNS_B   = A1;
int SEC     = 5;

void setup() {
  // setPwmSwizzler(3, 5, 10, 11);

  // Configure the A output
  pinMode(BRAKE_A, OUTPUT);  // Brake pin on channel A
  pinMode(DIR_A, OUTPUT);    // Direction pin on channel A

  // Configure the b output
  pinMode(BRAKE_B, OUTPUT);  // Brake pin on channel B
  pinMode(DIR_B, OUTPUT);    // Direction pin on channel B

  // Open Serial communication
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Motor shield DC motor Test:\n");
}

void loop() {

// Set the outputs to run the motor forward

  digitalWrite(BRAKE_A, LOW);  // setting brake LOW disable motor brake
  digitalWrite(DIR_A, HIGH);   // setting direction to HIGH the motor will spin forward

  digitalWrite(BRAKE_B, LOW);  // setting brake LOW disable motor brake
  digitalWrite(DIR_B, HIGH);   // setting direction to HIGH the motor will spin forward

  analogWrite(PWM_A, 128);     // Set the speed of the motor, 127 half speed (max 255)
  Serial.print("map(128, 0, 255, 0, 1023): ");
  Serial.println(map(128, 0, 255, 0, 1023));
#ifdef ARDUINO101
  CurieTimerOne.pwmStart(PWM_B, (int)map(128, 0, 255, 0, 1023), 1000000/FREQUENCY);
#else
   analogWrite(PWM_B, 128);
#endif
  delay(1000*SEC);                 // hold the motor at full speed for 5 seconds
  Serial.print("current consumption at full speed: ");
  Serial.println(analogRead(SNS_A));
  Serial.println(analogRead(SNS_B));

  // Brake the motor
  Serial.println("Start braking\n");
  // raising the brake pin the motor will stop faster than the stop by inertia
  digitalWrite(BRAKE_A, HIGH);  // raise the brake
  digitalWrite(BRAKE_B, HIGH);  // raise the brake

  delay(1000*SEC);

// Set the outputs to run the motor backward

  Serial.println("Backward");
  digitalWrite(BRAKE_A, LOW);  // setting again the brake LOW to disable motor brake
  digitalWrite(DIR_A, LOW);    // now change the direction to backward setting LOW the DIR_A
pin
  digitalWrite(BRAKE_B, LOW);  // setting again the brake LOW to disable motor brake
  digitalWrite(DIR_B, LOW);    // now change the direction to backward setting LOW the DIR_B
pin

  analogWrite(PWM_A, 255);     // Set the speed of the motor
  Serial.print("map(255, 0, 255, 0, 1023): ");
  Serial.println(map(255, 0, 255, 0, 1023));
#ifdef ARDUINO101
  CurieTimerOne.pwmStart(PWM_B, (int)map(255, 0, 255, 0, 1023), 1000000/FREQUENCY); // 100%
#else
   analogWrite(PWM_B, 255);     // Set the speed of the motor
#endif
  delay(1000*SEC);
  Serial.print("current consumption backward: ");
  Serial.println(analogRead(SNS_A));
  Serial.println(analogRead(SNS_B));

  // now stop the motor by inertia, the motor will stop slower than with the brake function
  analogWrite(PWM_A, 0);       // turn off power to the motor
  digitalWrite(BRAKE_A, HIGH); // raise the brake
#ifdef ARDUINO101
  CurieTimerOne.pwmStart(PWM_B, (int)map(0, 0, 255, 0, 1023), 1000000/FREQUENCY); // 100%
// turn off power to the motor
#else
 analogWrite(PWM_B, 0);       // turn off power to the motor
#endif
  digitalWrite(BRAKE_B, HIGH); // raise the brake

  Serial.print("current brake: ");
  Serial.println(analogRead(SNS_A));
  Serial.println(analogRead(SNS_B));
  Serial.println("End of the motor shield test with DC motors. Thank you!");

  while(1);
}
```

DC motor: works, PWM directions and brakes, both channels.

Figure 10 shows the pulse width running with a DC motor with the speed set at 150 (out of 255). The motor is using 6 V from an external power supply. The reading is taken from pin D3 of an Arduino 101 board.

Figure 10 **PWM with a DC motor using a 6 V external power supply at 100 speed**



Ch	Name	Value
C1	Amplitude	2.492 V
C1	High	4.87 V
C1	Low	-112 mV
C1	Cycles	10
C1	Frequency Avg.	543.225 Hz
C1	Pos Duty Avg.	58.62 %
C1	Pos Duty Min.	58.61 %
C1	Pos Duty Max.	58.64 %
C1	Pos Width Avg.	1.07914 ms (10 pulse)

1.5 Results

Arduino* 101 compatible.

§

2 XBee* S2 Module w/Arduino* Wireless Proto Shield

2.1 Use case

The XBee modules are small radio devices implementing various protocols all using the physical layer of the ZigBee protocol. They are widely used by the Arduino* user community because they are versatile and are configurable with the software provided by Digi* (called X-CTU).

Figure 11 **XBee* module**

Key Info	Links
URL	*http://arduino.cc/en/Main/ArduinoWirelessProtoShield*
Library	Not required.
Guide	*http://arduino.cc/en/Guide/ArduinoWirelessShieldS2*
Firmware upgrade	*http://arduino.cc/en/Guide/ArduinoWirelessShieldS2* API Mode: *https://code.google.com/p/xbee-arduino*
Configure software	*http://www.digi.com/support/kbase/kbaseresultdetl?id=2125* Run the X-CTU setup program as administrator Configure procedure: *http://arduino.cc/en/Guide/ArduinoWirelessShieldS2*

The XBee* Series 2 modules have two main modes of operation. The basic mode involves configuring the module for point-to-point networks. When configured in point-to-point mode the XBee* acts like a virtual serial link so no library is required.

The Arduino* XBee protoshield where the XBee modules are mounted are not compatible with Arduino* 101 for serial pass through AT programming that the XCTU or AT command set. Use the Arduino* Uno for programming

Creating a complex mesh network (API mode) requires updating the XBee* firmware. To reduce complexity, use a library to operate in API mode.

2.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Use VIN as power source	No
Source	from 5 V pin
ICSP	Yes

Pin Name	Function
D0 (Rx)	Serial1 Rx pin
D1 (Tx)	Serial1 Tx pin

Each XBee* S2 module came from the factory configured as a router. For point-to-point communication, the module needs to be configured (firmware changed) as a controller and the other an end device. The following section gives some details on setting the shields up for programming or transmitting; however, the details for upgrading the firmware is not discussed.

The process of updating the firmware is not reliable and repeated attempts are necessary. Problems encountered during the process included loss of communication with the board.

Serial select switch

There is a switch on the board whose task is to cross the Rx and Tx pin of the hardware serial on pins D0 and D1. This switch is designed to let the shield work with boards that use pins D0 and D1 (such as the Arduino* Uno board) for programming without removing the shield. The switch is called “Serial Select”. It determines how the XBee's serial communication connects to the serial communication between the microcontroller and USB-to-serial chip on the Arduino Uno board.

When ‘Serial Select’ is in the ‘Micro’ position, the DOUT pin of the wireless module is connected to the Rx pin of the microcontroller; and DIN is connected to Tx pin. The wireless module will then communicate with the microcontroller. Note that the Rx and Tx pins of the microcontroller are still connected to the Tx and Rx pins (respectively) of the USB-to-serial converter. Data sent from the microcontroller will then be transmitted to the computer (via USB) as well as being sent wirelessly by the wireless module. The microcontroller will not be programmable (via USB) in this mode.

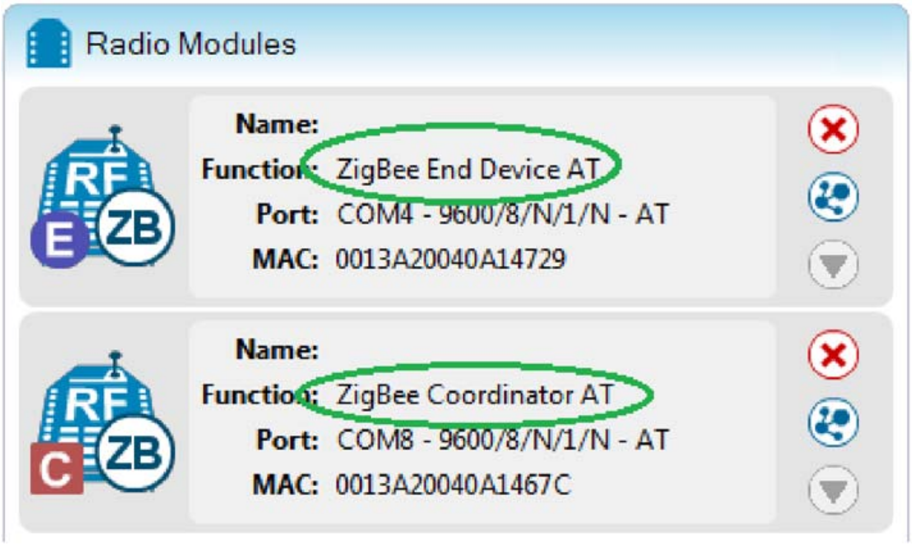
When ‘Serial Select’ is in the ‘USB’ position, the DOUT pin of the wireless module is connected to the Rx pin of the USB-to-serial converter, and DIN pin on the wireless module is connected to the Tx pin of the USB-to-serial converter. This means that the module can communicate directly with the computer. The microcontroller on the board will be bypassed. The shield will be set into this mode so that the modules can be queried to determine their configuration. Before querying, you must program the microcontroller with the ‘EmptySketch’ program, then selecting the ‘discovery’ option in the XCTU program.

Sketch: EmptySketch

void **setup**() { }
void **loop**() { }

All known com ports on the PC will be displayed. Select the com ports related to the Arduino shields. The XCTU program will query the module through the COM port. The following example shows that XBee* is indeed configured as point-to-point.

Figure 12 **Radio modules**



Radio Modules

Name:
Function: ZigBee End Device AT
Port: COM4 - 9600/8/N/1/N - AT
MAC: 0013A20040A14729

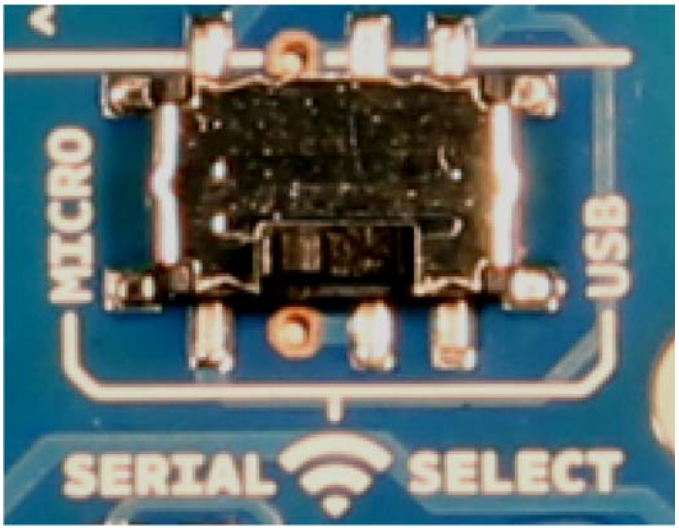
Name:
Function: ZigBee Coordinator AT
Port: COM8 - 9600/8/N/1/N - AT
MAC: 0013A20040A1467C

IMPORTANT: The modules will not communicate if the PAN IDs are not the same. Use XCTU and an Arduino 101 to make sure the PAN IDs are the same for both Coordinator and End-point radios. We use the ID AF with no issues.

Normal operation

For normal operation on the Arduino* 101 board, the serial select switch shall be in the “Micro” position. **This means that the XBee S2 modules cannot be programmed (firmware altered, etc.).**

Figure 13 **Serial switch set to micro**



2.3 There are no other pins involved in the shield. The **Compile and upload**

Two XBee S2 modules configured for point-to-point communication. Remember that each XBee S2 module came from the factory configured as a router. For point-to-point communication, configured as a Controller and the other an End Device.

To test, the end point module was placed on the Arduino* Uno board and the controller module was placed on the Arduino* 101 board. Two example sketches are based on the configuration of controller or endpoint.

1. Set switch to “USB”.
2. Download the “Receiver” sketch to the Arduino 101 board. This sketch receives a message repetitively over the Serial1 (XBee) and echoes everything received on “Serial1” and prints on the “Serial” port.
3. Set switch to “MICRO” and open serial terminal.

Example 3 **Receiver sketch**

```
// R E C E I V E R
// This code demonstrates how to run the shield on Arduino and Arduino* 101 boards.
// A R D U I N O
HardwareSerial* gSerialStdPtr = &Serial; // Arduino Uno, Tx(D1) Rx(D0)
HardwareSerial* gSerialTwoPtr = &Serial; // Arduino Uno, Tx(D1) Rx(D0)
bool            gGalileo      = false;

// G A L I L E O
//TTYUARTClass*   gSerialStdPtr = &Serial;  // Arduino* 101, /dev/ttyGSO, Tx pin
//TTYUARTClass*   gSerialTwoPtr = &Serial1; // Arduino* 101, /dev/ttySO,  Rx pin
//bool            gGalileo      = true;

bool qData;
void setup()
{
  qData = false;              // Initialize on reset
  gSerialStdPtr->begin(9600); // Receiver
  gSerialTwoPtr->begin(9600); // Sender
  waitForUser(5);             // Give usr time to open serial terminal
  gSerialStdPtr->println("XBee-Receiver-setup");
}

void loop()
{
  // Give indication that no data has been received
  if(qData == false) gSerialStdPtr->println("XBee-Receiver-waiting");
  // Get data from Sender and print to Receiver serial port
  while(gSerialTwoPtr->available())
  {
    char c= gSerialTwoPtr->read();  // Read  XBee data
    gSerialStdPtr->write(c);        // Write local
    qData = true;
  }
  if(qData == false) delay(1000*1); // Slow down until data is rec
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
}
```

1. Set switch to “USB”.
2. Download the “Sender” sketch to the Arduino 101 board. This sketch sends a message repetitively over the serial (XBee) port. It will be received on the controller.
3. Set switch to “MICRO” and open serial terminal.

Example 4 **Sender sketch**

```
// S E N D E R
// This code demonstrates how to run the shield on Arduino and Arduino* 101.
// A R D U I N O
HardwareSerial* gSerialStdPtr = &Serial; // Arduino 101/Uno, Tx(D1) Rx(D0)
HardwareSerial* gSerialTwoPtr = &Serial1; // Arduino 101/Uno, Tx(D1) Rx(D0)
//bool            gGalileo      = false;
```

```
void setup()
{
  gSerialStdPtr->begin(9600); // Sender IDE
  gSerialTwoPtr->begin(9600); // Receiver
  waitForUser(5);             // Give usr time to open serial terminal
  gSerialStdPtr->println("XBee-Sender-setup");
}
void loop()
{
  // Send data in 1 sec increments
  gSerialTwoPtr->println("EP> Hello from XBee-Sender");
  if(gGalileo) gSerialStdPtr->println("Hello from XBee-Sender");
  delay(1000*1);
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
}
```

Both serial monitors were opened and data from one XBee is received on the other XBee. This confirms that the two XBee modules are communicating.

These shields are compatible without using a library.

2.4 Results

Arduino* 101 compatible.

§

3 Adafruit* RFID/NFC Shield

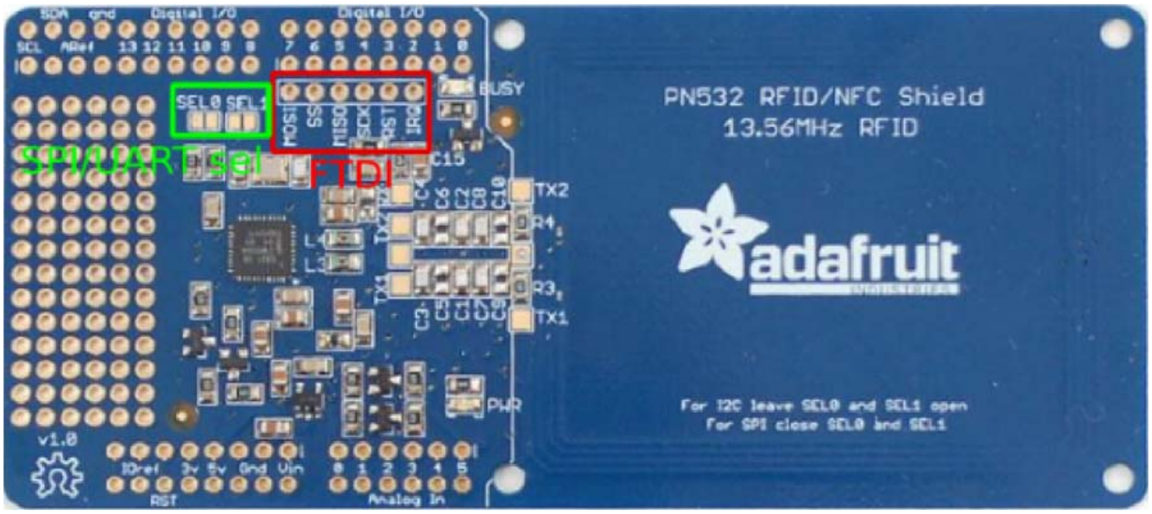
3.1 Use case

NFC (Near Field Communications) is a way for two devices very close to each other to communicate. It is similar to a very short range Bluetooth* that does not require authentication. It is an extension of RFID, so anything you can do with RFID you can do with NFC. The Adafruit* NFC shield was the first to use the NXP PN532 chipset (the most popular NFC chip on the market) and is embedded in a majority of phones or devices that do NFC.

Key Info	Links
URL	*http://www.adafruit.com/products/789*
Library	*https://github.com/adafruit/Adafruit-PN532/ dated 4/12/2016*
Guide	*http://learn.adafruit.com/adafruit-pn532-rfid-nfc*
About NFC	*https://learn.adafruit.com/adafruit-pn532-rfid-nfc/about-nfc*

This chipset is very powerful and can pretty much do it all, such as read and write to tags and cards, communicate with phones (say for payment processing), and “act” like an NFC tag. While the controller has many capabilities, the Adafruit* Arduino* library currently only supports reading/writing tags, and does not support phone-to-shield communication, tag emulation (which requires an external “secure element” only available from NXP), or other more advanced features at this time. This shield exists as an Arduino shield or as a breakout board. By default, it is designed to work with the I^{2}C bus.

Figure 14 **Adafruit* RFID/NFC shield**



3.2 Hardware summary

Key Info	Description/Links
IDE	Arduino 1.6.8
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Operating voltage	Designed for 5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No
ICSP	No
Datasheet	*http://www.adafruit.com/datasheets/pn532ds.pdf*
Schematics	*https://github.com/adafruit/Adafruit-PN532-RFID-NFC-Shield*

The Adafruit NFC shield is designed to use I^{2}C by default. I^{2}C only uses two pins (Analog 4 and 5, which are fixed in hardware and cannot be changed) to communicate and one pin as an “interrupt” pin (Digital 2 - can be changed). I^{2}C is a “shared” bus—unlike SPI and TTL serial—so you can put as many sensors as needed on the same two pins as long as their addresses do not collide/conflict. The Interrupt pin is an advantage because instead of constantly asking the NFC shield “Is there a card in view yet? What about now?” the chip will alert us when a NFC target comes into the antenna range.

Pin Name	I^{2}C Function (No wires required)
A4	I^{2}C, SDA (serial data line) data
A5	I^{2}C, SCL (serial clock line) clock
Device address	I^{2}C, header file 36d defines as: #define PN532_I2C_ADDRESS (0x48 >> 1)
D2	IRQ, Interrupt pin (excluded by default, close the onboard jumper to activate)

Note: The IRQ pin is meant to be connected to an interrupt pin on a microcontroller/ processor.

Figure 15 **MiFare* 1K card**

3.3 Companion library

The library is “Adafruit-PN532-RFID-NFC-Shield” that comes with seven example sketches. There is another NFC library that has a readMifare example that uses SPI so be sure the example is coming from the proper library.

The “readMifare” example was tested. By default, it reads Mifare RFID tags. A few lines can be uncommented and/or modified to allow data writes to the card. This functionality was successfully tested on both the Arduino* Uno board and Arduino* 101 boards and with *Mifare-One RFID Tag* and the RFID card pictured above. All tests worked as expected.

```
/**************************************************************************/
/*!
    @file     readMifare.pde
    @author   Adafruit Industries
    @license  BSD (see license.txt)

    This example will wait for any ISO14443A card or tag, and
    depending on the size of the UID will attempt to read from it.

    If the card has a 4-byte UID it is probably a Mifare
    Classic card, and the following steps are taken:

    - Authenticate block 4 (the first block of Sector 1) using
      the default KEYA of 0XFF 0XFF 0XFF 0XFF 0XFF 0XFF
    - If authentication succeeds, we can then read any of the
      4 blocks in that sector (though only block 4 is read here)

    If the card has a 7-byte UID it is probably a Mifare
    Ultralight card, and the 4 byte pages can be read directly.
    Page 4 is read by default since this is the first 'general-
    purpose' page on the tags.


    This is an example sketch for the Adafruit PN532 NFC/RFID breakout boards
    This library works with the Adafruit NFC breakout
      ----> https://www.adafruit.com/products/364

    Check out the links above for our tutorials and wiring diagrams
    These chips use I2C to communicate

    Adafruit invests time and resources providing this open source code,
    please support Adafruit and open-source hardware by purchasing
    products from Adafruit!
*/
/**************************************************************************/
#include <Wire.h>
#include <Adafruit_NFCShield_I2C.h>

#define IRQ   (2)
#define RESET (3)  // Not connected by default on the NFC Shield

Adafruit_NFCShield_I2C nfc(IRQ, RESET);

void setup(void) {
  Serial.begin(115200);
  Serial.println("Hello!");

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // configure board to read RFID tags
  nfc.SAMConfig();

  Serial.println("Waiting for an ISO14443A Card ...");
}


void loop(void) {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;                        // Length of the UID (4 or 7 bytes depending on
ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

  if (success) {
    // Display some basic information about the card
    Serial.println("Found an ISO14443A card");
    Serial.print("  UID Length: ");Serial.print(uidLength, DEC);Serial.println(" bytes");
    Serial.print("  UID Value: ");
    nfc.PrintHex(uid, uidLength);
    Serial.println("");

    if (uidLength == 4)
    {
      // We probably have a Mifare Classic card ...
      Serial.println("Seems to be a Mifare Classic card (4 byte UID)");
      // Now we need to try to authenticate it for read/write access
      // Try with the factory default KeyA: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
      Serial.println("Trying to authenticate block 4 with default KEYA value");
      uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

      // Start with block 4 (the first block of sector 1) since sector 0
      // contains the manufacturer data and it's probably better just
      // to leave it alone unless you know what you're doing
      success = nfc.mifareclassic_AuthenticateBlock(uid, uidLength, 4, 0, keya);

      if (success)
      {
        Serial.println("Sector 1 (Blocks 4..7) has been authenticated");
        uint8_t data[16];

        // If you want to write something to block 4 to test with, uncomment
        // the following line and this text should be read back in a minute
        //memcpy(data, (const uint8_t[]){ 'a', 'd', 'a', 'f', 'r', 'u', 'i', 't', '.', 'c',
'o', 'm', 0, 0, 0, 0 }, sizeof data);
        //success = nfc.mifareclassic_WriteDataBlock (4, data);

        // Try to read the contents of block 4
        success = nfc.mifareclassic_ReadDataBlock(4, data);

        if (success)
        {
          // Data seems to have been read ... spit it out
          Serial.println("Reading Block 4:");
          nfc.PrintHexChar(data, 16);
          Serial.println("");

          // Wait a bit before reading the card again
          delay(1000);
        }
        else
        {
          Serial.println("Ooops ... unable to read the requested block.  Try another key?");
        }
      }
      else
      {
        Serial.println("Ooops ... authentication failed: Try another key?");
      }
    }

    if (uidLength == 7)
    {
      // We probably have a Mifare Ultralight card ...
      Serial.println("Seems to be a Mifare Ultralight tag (7 byte UID)");

      // Try to read the first general-purpose user page (#4)
      Serial.println("Reading page 4");
      uint8_t data[32];
      success = nfc.mifareultralight_ReadPage (4, data);
      if (success)
      {
        // Data seems to have been read ... spit it out
        nfc.PrintHexChar(data, 4);
        Serial.println("");

        // Wait a bit before reading the card again
        delay(1000);
      }
      else
      {
        Serial.println("Ooops ... unable to read the requested page!?");
      }
    }
  }
}

Sketch Output:  readMifare
Hello!
Found chip PN532
Firmware ver. 1.6
Waiting for an ISO14443A Card ...Found an ISO14443A card
  UID Length: 4 bytes
  UID Value: 0xD5 0xEE 0xC6 0x24

Seems to be a Mifare Classic card (4 byte UID)
Trying to authenticate block 4 with default KEYA value
Ooops ... authentication failed: Try another key?Found an ISO14443A card
  UID Length: 4 bytes
  UID Value: 0x6A 0xFE 0x68 0xA1

Seems to be a Mifare Classic card (4 byte UID)
Trying to authenticate block 4 with default KEYA value
Sector 1 (Blocks 4..7) has been authenticated
Reading Block 4:
44 50 4C 20 46 6C 65 78 20 54 65 73 74 00 00 00  DPL Flex Test...
```

3.4 Results

Arduino* 101: The library dated 4/12/16 passes.

§

4 Adafruit* Ultimate GPS Logger Shield

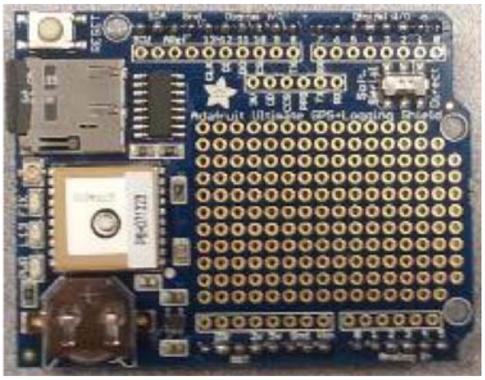
4.1 Use case

This GPS shield is designed to log data to an SD card or for use with a geocaching project, but the card has a connection for an external antenna so it is great for putting inside an enclosure.

Key Info	Links
URL	*http://www.adafruit.com/products/1272*
Library	*https://github.com/adafruit/Adafruit-GPS-Library* *https://github.com/adafruit/RTClib* Dated: 11/29/15, 4/10/16

4.2 Hardware summary

Figure 16 **Adafruit* Ultimate GPS logger shield**



Key Info	Description/Links
Operating voltage	Designed for 5 V. Should work at 3.3 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
IDE	Arduino* 1.6.8
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*https://learn.adafruit.com/adafruit-ultimate-gps-logger-shield/downloads*
Antenna	Internal patch antenna. Adapter connection for attaching an external antenna.
GPS module	*GTPA013* Uses the MTK33x9 chipset.
RTC	DS1307. **NOTE:** If the battery is not preset, strange behavior will occur and a possibility of hanging the board.
Battery	Coin Cell, CR1220 ***Note:*** The battery must be present in the shield at all times (even if it is dead).
SD card	Micro SD card to be formatted at 16FAT/32FAT.

The following pins are used for the RTC.

Pin Name	I^{2}C Function (No wires required)
A4	I^{2}C, SDA (Serial Data Line) Data
A5	I^{2}C, SCL (Serial Clock Line) Clock
Device Address	I^{2}C, RTClib.cpp defines the address as: #define DS1307_ADDRESS 0x68

The following pins used for the SD Card are as shown in the table below.

The process to list files on the SD is documented in the '*Adafruit Data Logging Shield*' section. The sketch used is called 'List-Files'.

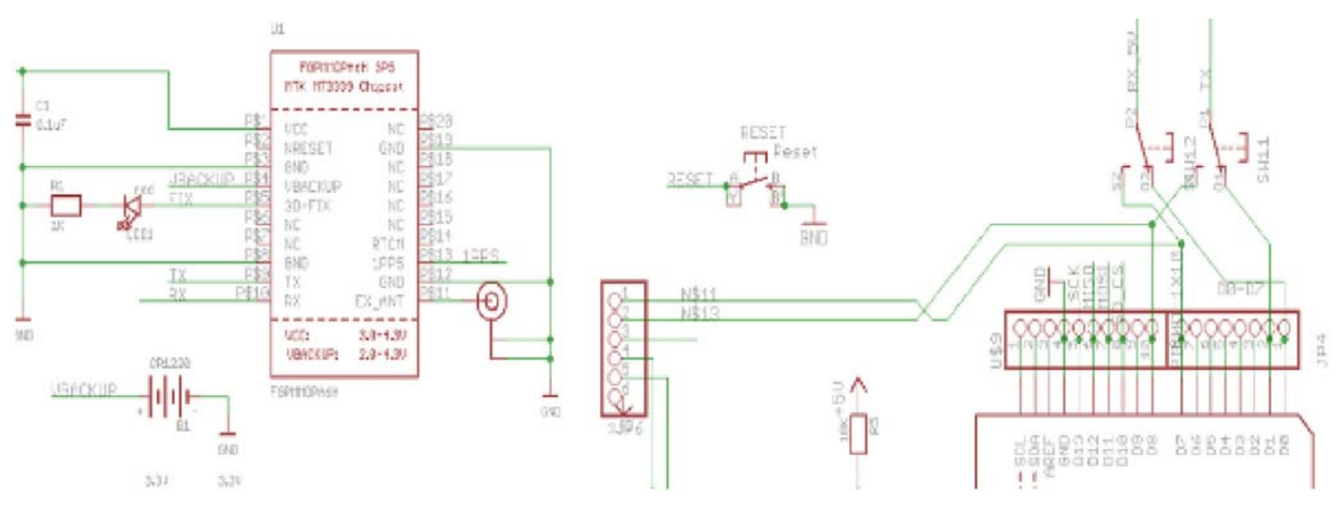
Shield Pin Name	SPI Function (no wires required)
D11	SPI, MOSI (Master Out Slave In)
D12	SPI, MISO (Master In Slave Out)
D13	SPI, Clock (CLK)
CS	SPI, Chip Select (CS), The sketch defines as: *SD.begin(10)*
D10	Must be left as an output (event if it is not used) in order for the SD to work.

The following pins are used for hardware and software serial.

Pin Name	Shield Pin/Function
D0	Hardware Tx
D1	Hardware Rx
D7	Connect to the GPS Rx pin on the Shield
D8	Connect to the GPS Tx pin on the Shield

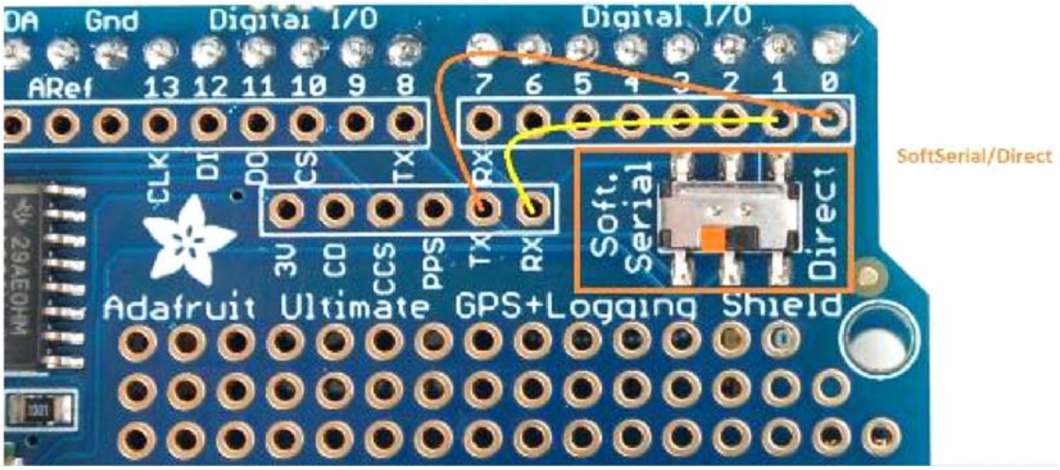
The shield is designed for working with the software serial library; however, Rx and Tx signals are broken out to a header (on the shield) so the GPS serial can be rewired to the hardware Serial pins.

Figure 17 **Adafruit* Ultimate GPS logger shield schematic**



Place the switch in the "Soft. Serial" position and connect D1 to GPS_RX and D0 to GPS_TX.

Figure 18 **Adafruit* Ultimate GPS logger shield Tx/Rx connections**



4.3 Compile and upload

The GPS requires no library. GPS data is simply transmitted as serial data from the shield. Since the GPS automatically sends NMEA strings, the data reception was tested with a simple sketch that reads the GPS stream from the HW serial and prints back over the USB port.

`Simple GPS sketch`

```
boolean bWaiting = true;
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);
  waitForUser(3);
}


void loop() {
 if(Serial1.available())
 {
   char c = Serial1.read();
   Serial.write(c);
   bWaiting = false;
 }
 if(bWaiting == true)
 {
   Serial.println("...Waiting");
   delay(1000*1);
 }
}
```

```
void waitForUser(unsigned int aSec) {
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
  Serial.println("");
```

}The data displayed is raw and needs to be parsed.

```
$GPRMC,000502.399,V,,,,,0.00,0.00,060180,,,N*46
$GPGGA,000502.600,,,,,0,0,,,M,,M,,*49
$GPRMC,000502.600,V,,,,,0.00,0.00,060180,,,N*43
$GPGGA,000502.800,,,,,0,0,,,M,,M,,*47
$GPRMC,000502.800,V,,,,,0.00,0.00,060180,,,N*4D
$GPGGA,000503.000,,,,,0,0,,,M,,M,,*4E
$GPRMC,000503.000,V,,,,,0.00,0.00,060180,,,N*44
$GPGGA,000503.199,,,,,0,0,,,M,,M,,*4F
$GPRMC,000503.199,V,,,,,0.00,0.00,060180,,,N*45
$GPGGA,000503.399,,,,,0,0,,,M,,M,,*4D
$GPRMC,000503.399,V,,,,,0.00,0.00,060180,,,N*47
$GPGGA,000503.600,,,,,0,0,,,M,,M,,*48
$GPRMC,000503.600,V,,,,,0.00,0.00,060180,,,N*42
$GPGGA,000503.800,,,,,0,0,,,M,,M,,*46
$GPRMC,000503.800,V,,,,,0.00,0.00,060180,,,N*4C
$GPGGA,000504.000,,,,,0,0,,,M,,M,,*49
$GPRMC,000504.000,V,,,,,0.00,0.00,060180,,,N*43
$GPGGA,000504.199,,,,,0,0,,,M,,M,,*48
$GPRMC,000504.199,V,,,,,0.00,0.00,060180,,,N*42
$GPGGA,000504.399,,,,,0,0,,,M,,M,,*4A
$GPRMC,000504.399,V,,,,,0.00,0.00,060180,,,N*40
```

4.4 Companion library

The companion library was functional on the Arduino* Uno board; however, using the hardware serial had unknown issues when connecting the Rx wire.

.

4.5 Results

Arduino* 101 GPS passes, RTC fails (saw same failure using the RTC on this shield with Intel® Galileo boards). SD card passes.

§

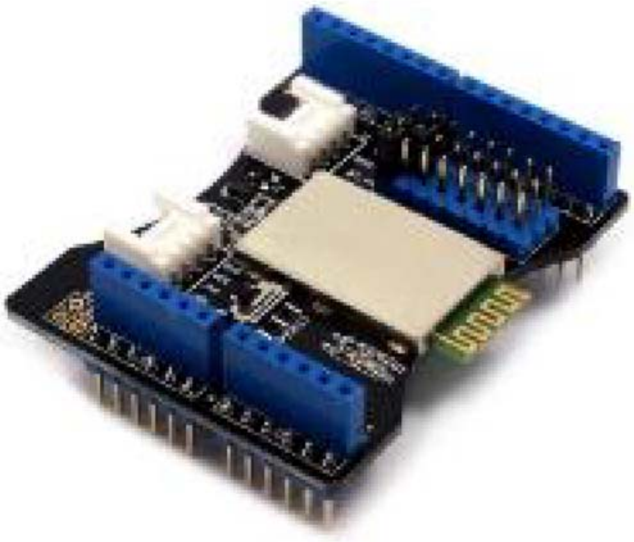
5 Seeed Studio* Bluetooth* Shield v1.1

5.1 Use case

This shield is an inexpensive Bluetooth (BT) module that is easily accessible on the market. The shield is based on HC-05 and HC-06. This version uses a custom firmware that differs from the standard AT commands. This module communicates with the microcontroller through the serial interface. Configuration of the shield is not intuitive; therefore, pay close attention to the setup.

Key Info	Links
Product	*http://www.SeeedStudio.com/depot/bluetooth-shield-p-866.html*
Library/example	*https://github.com/Seeed-Studio/Bluetooth_Shield_Demo_Code, dated 05/02/2016* Provided, however, did not use.
Guide	*http://forum.arduino.cc/index.php?topic=120113.0*
Commands	*http://www.SeeedStudio.com/wiki/images/e/e8/BTSoftware_Instruction.pdf*
Phone software	https://play.google.com/store/apps/details?id=jp.side2.apps.btterm It may be possible to use the Bluetooth on the same PC as the IDE. However, there are cases where the Arduino* IDE will hang if Bluetooth is turned on (in cases where the BT is on the chipset). An alternative solution is to use a USB Bluetooth adapter. The best method is to use a device that is independent from the IDE. In this case, we are using an Android phone.

Figure 19 **Seeed Studio* Bluetooth* shield v1.1**

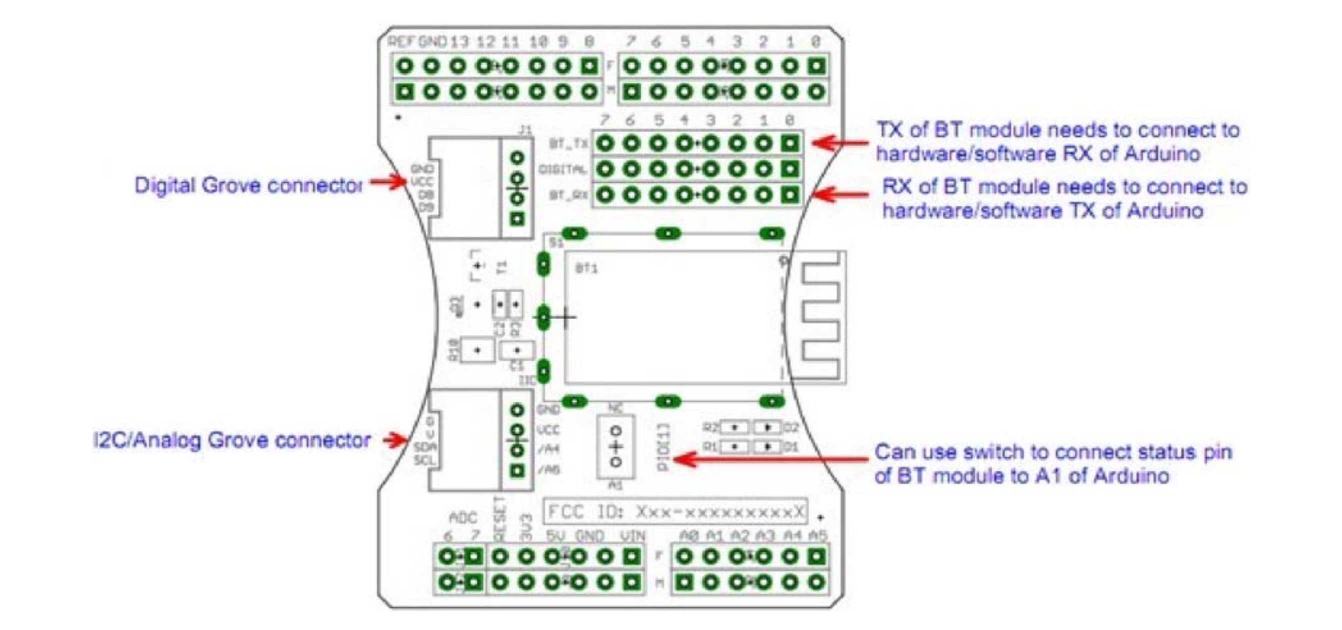


5.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IOREF	No
Use VIN as power source	No
Operating voltage	5 V
Schematics	*http://www.SeeedStudio.com/wiki/images/c/c3/BT_shield_eagle_files.zip*

Shield Pin Name	Function
BT_Tx	Shield, selectable pins for Tx (D0 through D7). Arduino* Uno, set to D7 (default)
BT_Rx	Shield, selectable pins for Rx (D0 through D7). Arduino Uno, set to D6 (default)
PIO [1]	Shield, selectable for A0 and NC. By default set to A0
Passcode	"0000", default passcode. Will ask on the very first pairing.

Figure 20 **Seeed Studio* Bluetooth* shield connectors**


Digital Grove connector
I2C/Analog Grove connector
TX of BT module needs to connect to hardware/software RX of Arduino
RX of BT module needs to connect to hardware/software TX of Arduino
Can use switch to connect status pin of BT module to A1 of Arduino
BT_TX
DIGITAL
BT_RX
GND
VCC
D8
D9
SDA
SCL
BT1
PIO[1]
FCC ID: Xxx-xxxxxxxxxX
RESET
3V3
5V GND VIN
A0 A1 A2 A3 A4 A5
REF GND 13 12 11 10 9 8
7 6 5 4 3 2 1 0

On the shield, for softserial the pins are selectable for the serial communication. Typically, BT shields are pairable with no sketch uploaded; after that point, the firmware is modified to desired settings. With this shield, the BT module needs to be set up in every sketch, then paired. Due to the difficulty in pairing, the Arduino* Uno version of pairing will be demonstrated with simple printouts. Notice that this example uses two serial ports. On the Arduino SoftSerial is used as the second port. On the Arduino* 101 board, */dev/ttySO* is used as the second serial port. These settings have different jumper settings on the shield.

On the Arduino Uno board, upon power-up, the D1 LED has a fast steady blink. Ensure that the jumper for the shield (noted in the table above) are set properly.

- Download the sketch below and bring up the serial terminal.
- The sketch gives the users 5 seconds to bring up the serial monitor; the sketch will then state that the BT is "inquirable".
- When "inquirable", D1and D2 will flashing back and forth as red and green. At this point, pair the device with the phone.
- The BT scan will show that this device is labeled as "SeeedBTSlave".
- Pair with the default passcode (if needed).
- Once paired, the D1

- LED will have a slow blink.

```
Pairing Bluetooth for Arduino
// This code demonstrates how to run the shield on Arduino and Galileo.
// A R D U I N O
//   Uses two serial port (IDE Serial, Softserial)
#include <SoftwareSerial.h>
#define RxD 6 // Set Shield BT_Rx jumper to 6
#define TxD 7 // Set Shield BT_Tx jumper to 7
SoftwareSerial  btSerial(RxD,TxD);
HardwareSerial* gSerialStdPtr = &Serial;   // Arduino Uno, Rx pin 0, Tx pin 1
SoftwareSerial* gSerialTwoPtr = &btSerial; // Arduino Uno, Second serial port (Software
Serial)



#define DEBUG_ENABLED  1
void setup()
{
  gSerialStdPtr->begin(9600);
  delay(1000*5);
  gSerialStdPtr->println("...Setup 1");
  setupBlueToothConnection();
}
void loop()
{
  char recvChar;
  while(1){
    if(gSerialTwoPtr->available()){//check if there's any data sent from the remote bluetooth
shield
      recvChar = gSerialTwoPtr->read();
      gSerialStdPtr->print(recvChar);
    }
    if(gSerialStdPtr->available()){//check if there's any data sent from the local serial
terminal, you can add the other applications here
      recvChar  = gSerialStdPtr->read();
      gSerialTwoPtr->print(recvChar);
    }
  }
}
void setupBlueToothConnection()
{
  gSerialStdPtr->println("...Setup 2");
  gSerialTwoPtr->begin(38400); //Set BluetoothBee BaudRate to default baud rate 38400
  gSerialTwoPtr->print("\r\n+STWMOD=0\r\n"); //set the bluetooth work in slave mode
  gSerialTwoPtr->print("\r\n+STNA=SeeedBTSlave\r\n"); //set the bluetooth name as
"SeeedBTSlave"
  gSerialTwoPtr->print("\r\n+STOAUT=1\r\n"); // Permit Paired device to connect me
  gSerialTwoPtr->print("\r\n+STAUTO=0\r\n"); // Autoconnection should be forbidden here
  delay(2000); // This delay is required.
  gSerialTwoPtr->print("\r\n+INQ=1\r\n"); //make the slave bluetooth inquirable
  gSerialStdPtr->println("The slave bluetooth is inquirable!");
  delay(2000); // This delay is required.
  gSerialTwoPtr->flush();
}
```

The output for pairing and connection is as follows:

```
Arduino pairing output
...Setup 1
...Setup 2
The slave bluetooth is inquirable!
```

```
+BTSTATE:3

CONNECT:OK

+BTSTATE:4
```

5.3 Companion library

No library required.

5.4 Compile and upload

For Arduino* 101, the same sketch is used. The sketch is constructed such that Arduino* Uno-specific code can be commented out and Arduino 101-specific code can be commented back in. Unfortunately, the sketch utilizes pointers to make this transition easier; however, it might not be easier to the reader.

Example 5 **Change to Sketch for Galileo (for clarity, comments at EOL have been removed)**

Before

```
#include <SoftwareSerial.h>
#define RxD 6
#define TxD 7
SoftwareSerial  btSerial(RxD,TxD);
HardwareSerial* gSerialStdPtr = &Serial;
SoftwareSerial* gSerialTwoPtr = &btSerial;

// G A L I L E O
//TTYUARTClass*   gSerialStdPtr = &Serial;
//TTYUARTClass*   gSerialTwoPtr = &Serial1;
```

After

```
//#include <SoftwareSerial.h>
//#define RxD 6
//#define TxD 7
//SoftwareSerial  btSerial(RxD,TxD);
//HardwareSerial* gSerialStdPtr = &Serial;
//SoftwareSerial* gSerialTwoPtr = &btSerial;
```

The pairing LED works the same as it did on the Arduino* Uno board. However, there seems to be more output on the Intel Curie compared to the Arduino* Uno board. The shield has been demonstrated to be compatible for both boards.

```
Galileo Pairing output
...Setup 1
...Setup 2
The slave bluetooth is inquirable!

+STWMOD=0

+STNA=SeeedBTSlave

+S
OK

OK

OK

OK

WORK:SLAVER

+BTSTATE:0

+BTSTATE:1

+INQ=1

OK

+BTSTATE:2
```

```

+BTSTATE:3

CONNECT:OK

+BTSTATE:4
```

5.5 Results

Arduino* 101

§

6 Cooking Hacks* eHealth Shield v2.0

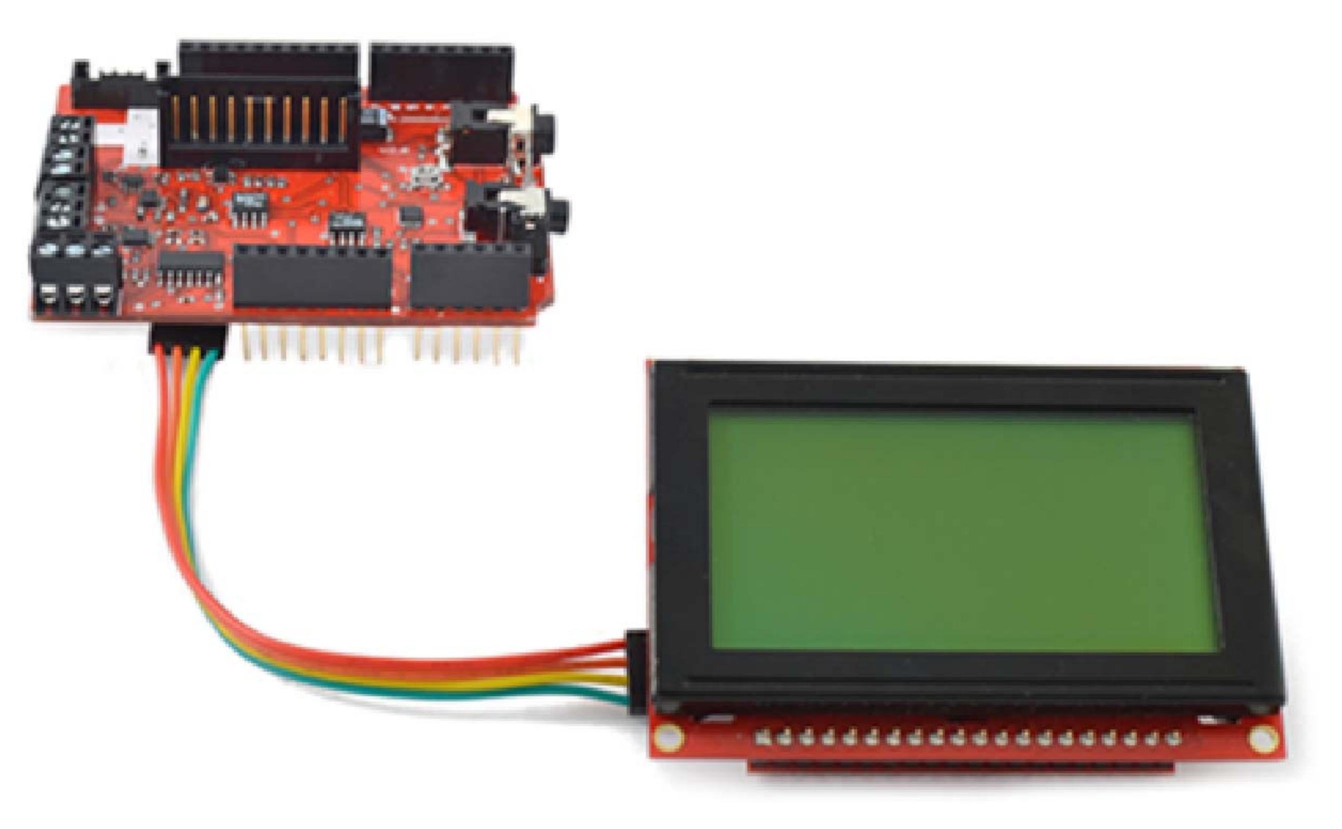
6.1 Use case

This shield collects data from many health sensors in one shield and one library. There is an e-Health Sensor Platform Complete Kit which includes all the sensors supported by the shield. The e-Health Sensor has been designed to help researchers, developers and artists to measure biometric sensor data for experimentation, fun and test purposes. The platform does not have medical certifications and cannot be used to monitor critical patients who need accurate medical monitoring or those whose conditions must be accurately measured for an ulterior professional diagnosis.

Key Info	Links
URL	*http://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical*
Library	*https://www.cooking-hacks.com/media/cooking/images/documentation/tutorial_intel_galileo/eHealth_library_for_Intel_Galileo.zip*
Temperature Sensor	*http://www.cooking-hacks.com/body-temperature-sensor-ehealth-medical*

The shield supports 10 different sensors: pulse, oxygen in blood (SPO2), airflow (breathing), body temperature, ECG, glucometer, galvanic skin response, blood pressure, patient position and EMG. Version 2.0 has been improved with new features such as new muscle sensor, new blood pressure sensor, upgraded glucometer and new connection possibilities. Much of the sensor data can be accessed by the serial monitor, but it is also possible to use a Wi-Fi module to perform direct communications with iPhone* and Android* devices without the need of an intermediate router by creating an ad hoc network between them.

Figure 21 **Cooking Hacks* eHealth shield v2.0**



The Cooking Hacks website has instructions for assembling and programming the different sensors as well as smartphone connections. The e-Health library includes functions to manage a graphic LCD for visualizing data. The serial Graphic LCD backpack is soldered to the 128x64 Graphic LCD and provides the user with a simple serial interface.

6.2 Hardware summary

Key Info	Description/Links
Operating Voltage	5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No
Schematic	*http://skin.cdn-libelium.com/frontend/default/cooking/images/catalog/documentation/e_health_v2/e-Health_v2.0.pdf*
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8

Figure 22 **Cooking Hacks* eHealth shield connectors**

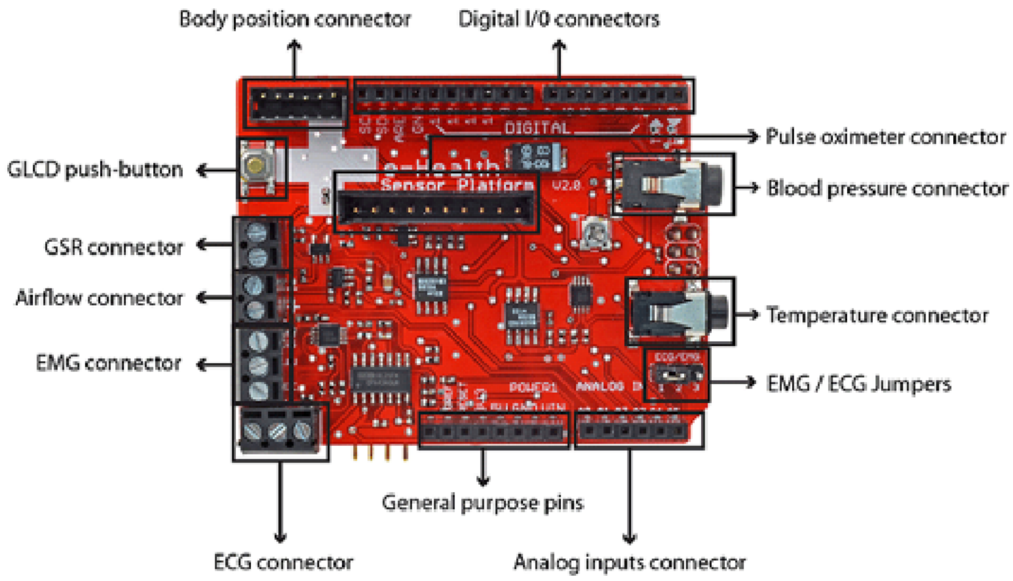


Figure 23 **Glucometer connector and graphic LCD connector**

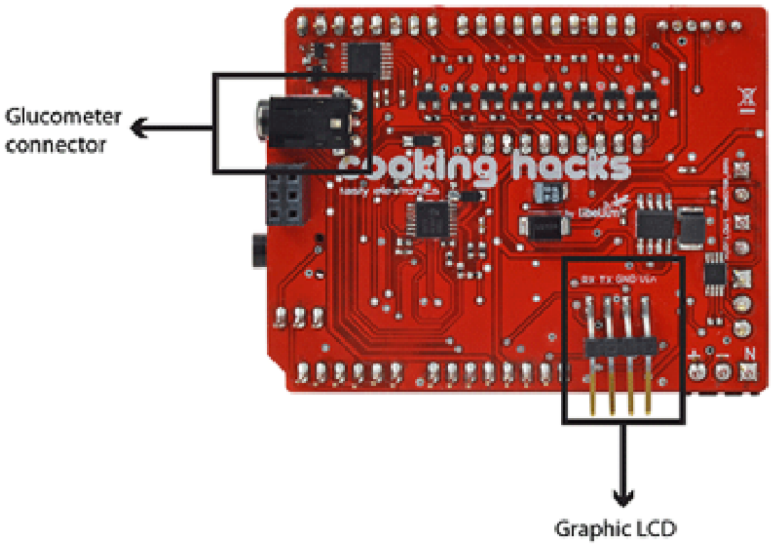
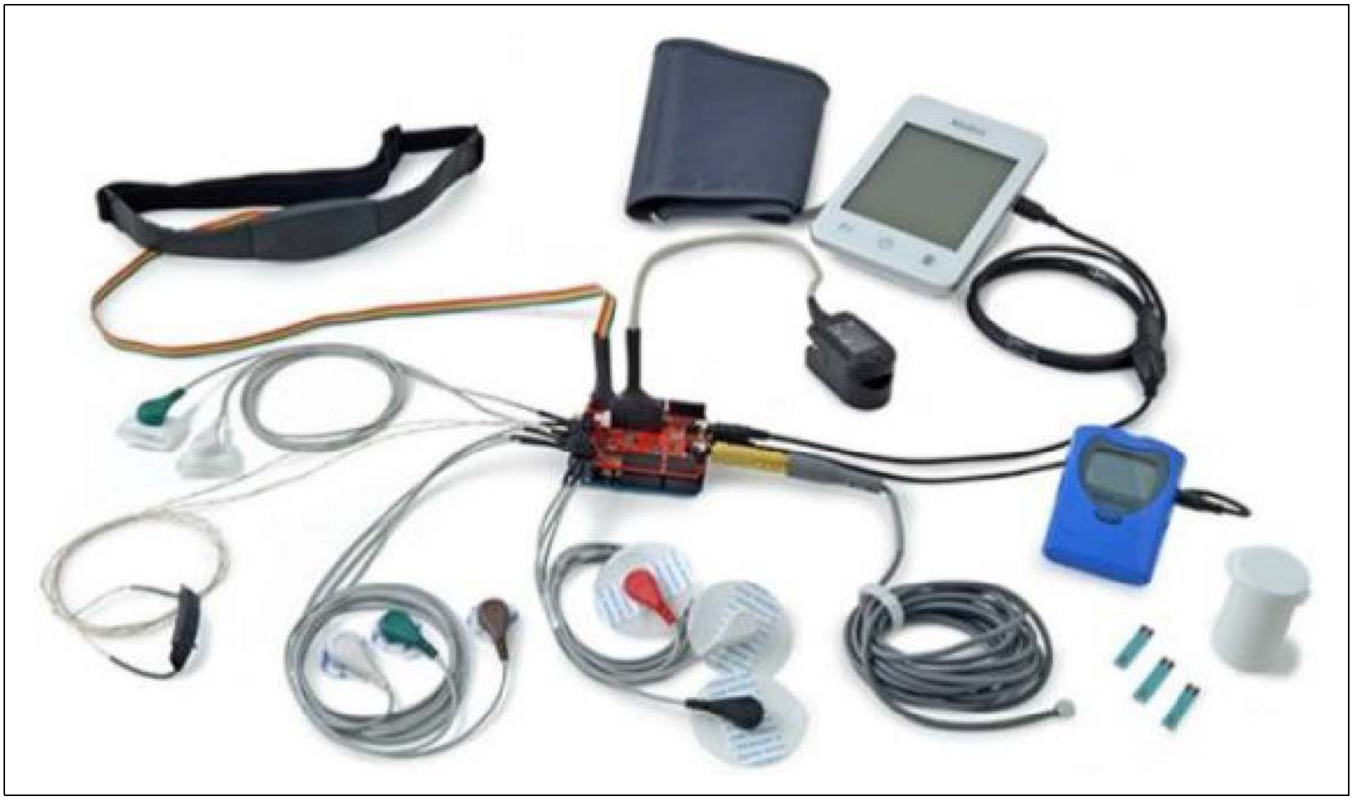


Figure 24 shows a photo of the sensors included when ordering the complete platform kit.

Figure 24 **Cooking Hacks* eHealth shield sensors**



Pin Name	Function
D6 to D13	Pulse and oxygen in blood (SPO2)
A0	Electrocardiogram (ECG)
A1	Airflow: breathing sensor
A3	Body temperature
D5, A4, A5	Blood pressure
D2, D3, SDA, SCL	Patient position and falls
A2	Galvanic skin response (GSR)
Serial D0, D1	Glucometer

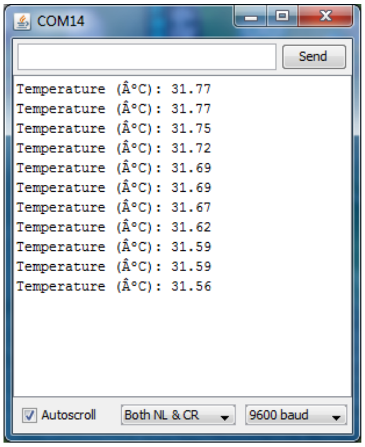
6.3 Companion library

The ‘eHealth’ library provided above has been rewritten by Cooking Hacks to provide functionality for both the Arduino* Uno board and Arduino* 101 13 sketches are provided, however, only one sensor has been purchased for test. The sketch used is called ‘TemperatureExample’.

6.4 Compile and upload

The 'TemperatureExample' sketch was compiled and uploaded with no changes. The temperature sensor was purchased separately. This sensor plugs into the 'Temperature connector' port. Output to the serial port is as follows:

Figure 25 **Temperature connector output**



```
Temperature (Â°C): 31.77
Temperature (Â°C): 31.77
Temperature (Â°C): 31.75
Temperature (Â°C): 31.72
Temperature (Â°C): 31.69
Temperature (Â°C): 31.69
Temperature (Â°C): 31.67
Temperature (Â°C): 31.62
Temperature (Â°C): 31.59
Temperature (Â°C): 31.59
Temperature (Â°C): 31.56
```

6.5 Results

Arduino* 101 compatible.

§

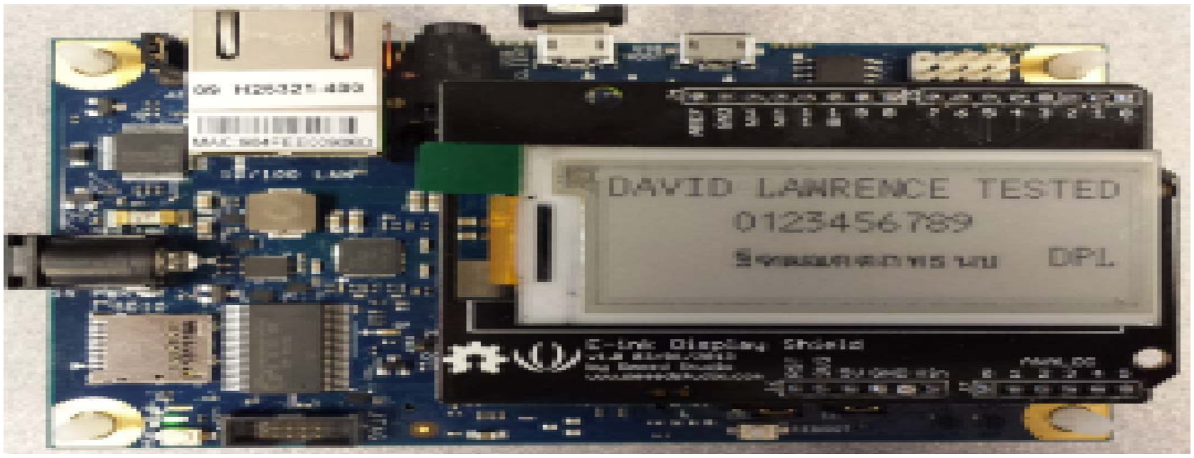
7 Seeed Studio* E-Ink Shield

7.1 Use case

These displays are relatively new. Seeed Studio was probably the first company who designed a shield using E-Ink displays. This product has raised much interest in the maker community. This shield is a good candidate for testing because the library contains very few AVR-specific instructions and uses the Arduino* SPI library.

Key Info	Links
Order/product Info	*http://www.epictinker.com/E-Ink-Display-Shield-p/sld01093p.htm*
Guide	*http://www.SeeedStudio.com/wiki/E-ink_Display_Shield*
Library	*http://www.SeeedStudio.com/wiki/File:SeeedEink.zip* (Date 05/30/13)

Figure 26 **Seeed Studio* E-ink shield**



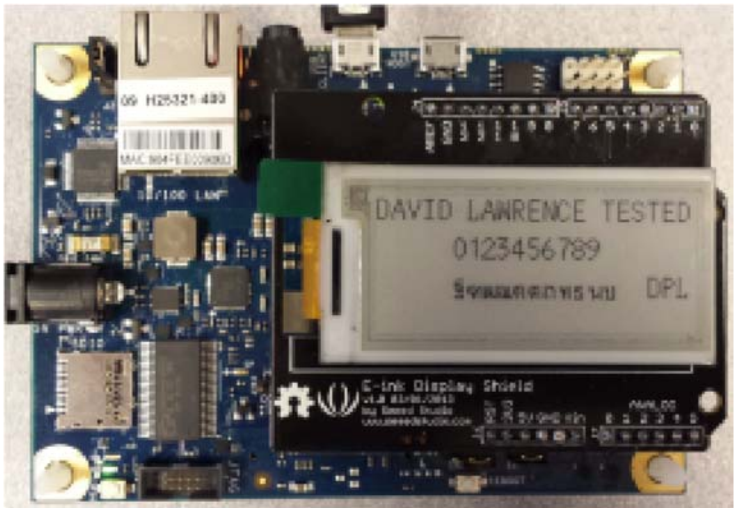
7.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
IOREF	No
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635

Pin Name	Function
D3	VPP, power supply for OTP programming.
D4	GT_CS, GT20L16P1Y select input pin.
D5	Eink_D/C, E-ink Data/Command control pin.
D6	Eink_CS, E-ink select input pin.
D7	BUSY, E-ink Device Busy Signal, When Busy is High, the operation of the chip should not be interrupted, and command should not be sent. Pins Used for SPI Interface:
CS	GT_CS on D4 for GT20L16P1Y select input pin EInk_CS on D6 for E-ink select input pin EInk_DC on D5 E-ink Data/Command control pin

Pin Name	SPI Function (No wires required)
D11	SPI, MOSI (Master Out Slave In)
D12	SPI, MISO (Master In Slave Out)
D13	SPI, Clock (CLK)
D10	SPI, Chip Select(CS)
CS	GT_CS on D4 for GT20L16P1Y select input pin EInk_CS on D6 for E-ink select input pin EInk_DC on D5 E-ink Data/Command control pin

Figure 27 **Seeed Studio* E-ink shield on an Intel® Galileo first generation board**



The shield adopts the pin layout used before version 1.0 of the Arduino* platform, which means that it probably is not compatible with the 3.3 V logic. There is no documentation concerning this, only an FAQ that warns you to use it on the Due.

7.3 Companion library

The library is called “SeedEink” and uses the Arduino* SPI library. There are two example sketches called ‘displayCharacter’ and ‘displayChinese’.

Example 6 **Change Eink.h as follows:**

Before

```
#else

#define Eink_CS1_LOW  {DDRD |= 0x40;PORTD &=~ 0x40;}
#define Eink_CS1_HIGH {DDRD |= 0x40;PORTD |= 0x40;}
#define Eink_DC_LOW   {DDRD |= 0x20;PORTD &=~ 0x20;}
#define Eink_DC_HIGH  {DDRD |= 0x20;PORTD |= 0x20;}
#define GT_CS2_LOW    {DDRD |= 0x10;PORTD &=~ 0x10;}
#define GT_CS2_HIGH   {DDRD |= 0x10;PORTD |= 0x10;}
```

After

```
#else

#define Eink_CS1_LOW  pinMode(6,OUTPUT); digitalWrite(6,LOW)
#define Eink_CS1_HIGH pinMode(6,OUTPUT); digitalWrite(6,HIGH)
#define Eink_DC_LOW   pinMode(5,OUTPUT); digitalWrite(5,LOW)
#define Eink_DC_HIGH  pinMode(5,OUTPUT); digitalWrite(5,HIGH)
#define GT_CS2_LOW    pinMode(4,OUTPUT); digitalWrite(4,LOW)
#define GT_CS2_HIGH   pinMode(4,OUTPUT); digitalWrite(4,HIGH)
```

This change works properly with the Arduino* Uno board, however, calling the ‘pinMode’ and ‘digitalWrite’ is slower than AVR specific instructions.

7.4 Compile and upload

The included example sketch ‘displayCharacter’ compiles. It runs with no problem on the Arduino* Uno board.

7.5 Results

Arduino* 101 not tested since shield is not working.

8 Seeed Studio* NFC Shield V1.0

8.1 Use case

This shield uses the NXP* PN532 NFC/RFID controller that appeared originally on the *Adafruit RFID/NFC shield*. This chipset is very powerful, and can pretty much do it all, such as read and write to tags and cards, communicate with phones (for example, payment processing), and "act" like a NFC tag.

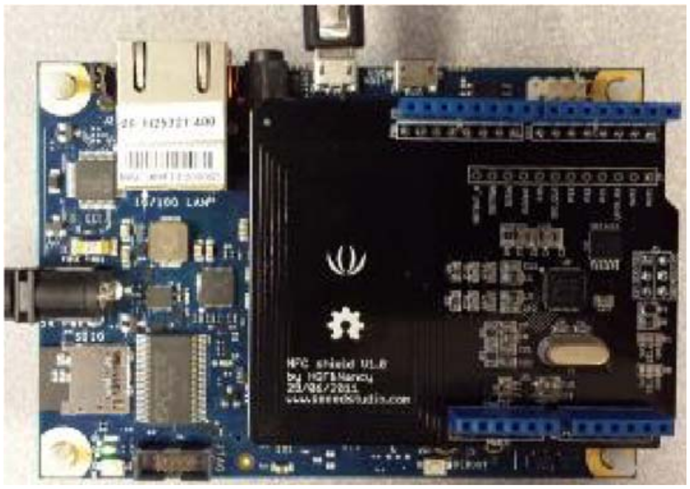
Key Info	Links
Order/product info	*http://www.SeeedStudio.com/wiki/NFC_Shield_V1.0*
Library	*http://www.SeeedStudio.com/wiki/File:PN532_SPI_V2.zip* Date 12/20/12

8.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Use VIN as power source	No.
Power from	5 V pin (can use 3.3 or 5 V).
Schematics	*http://www.SeeedStudio.com/wiki/File:NFC_Shield_Schematic.pdf*

- Typical current: 100 mA
- SPI interface using the ICSP connection hence, most Arduino* Uno pins are available for other applications.
- Built in PCB Antenna.
- Supports both 3.3 and 5 V operation using the TI* TxB0104 level translator.
- Socket to connect other shields.
- The maximum communication range of this NFC Shield is about 5 cm.
- Not able to read/write ultralightC chip - can only read its ID.
- Pinout: None provided.

Figure 28 **Seeed Studio* NFC shield, shown here on an Intel® Galileo first generation board**



8.3 Companion library

The library is PN532 that comes with eleven example sketches.

Arduino* Uno Board:

Download the ‘readAllMemoryBlocks’ sketch provided with the library. Bringing up the IDE serial port, the output will confirm if the shield is detected (in bold below). Run the RFID tag across the shield, this will signal the RFID read. The remaining output shown below:

Example 7 **Sketch output: readAllMemoryBlocks**

```
Hello!
Found chip PN532
Firmware ver. 1.6
Supports 7
Found 1 tags
Sens Response: 0x4
Sel Response: 0x8
 0xD5 0xEE 0xC6 0x24Read card #3589195300

D5 EE C6 24 D9 8  4  0  62 63 64 65 66 67 68 69 | Block  0 | Manufacturer Block
```

The following sketch is a modified version of the “readAllMemoryBlocks” example included in the library. It reads all MIFARE card memory blocks from 0x00 to 0x63 and should be tested with a MIFARE 1K card. The sketch was modified to display the text in block 4. While testing *Adafruit RFID/NFC Shield*, data was written to Block 4 of the test card. The same test card was used to test this shield.

Example 8 **Modified sketch: readAllMemoryBlocks**

```
#include <PN532.h>
#include <SPI.h>

#define PN532_CS 10

PN532 nfc(PN532_CS);

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Hello!");
  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata)
  {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5");
  Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. ");
  Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.');
  Serial.println((versiondata>>8) & 0xFF, DEC);
  Serial.print("Supports ");
  Serial.println(versiondata & 0xFF, HEX);
  // configure board to read RFID tags and cards
  nfc.SAMConfig();
}

void loop(void)
{
  uint32_t id;
  // look for MiFare type cards
  id = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A);

  if (id != 0)
  {
    Serial.print("Read card #");
    Serial.println(id);
    Serial.println();
    uint8_t keys[]= { 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF };// def key of a fresh card
    for(uint8_t blockn=0;blockn<64;blockn++)
    {
      if(nfc.authenticateBlock(1, id ,blockn,KEY_A,keys)) //auth block blockn
      {
        //if authentication successful
        uint8_t block[16];
        String theStr;  // string to store block 4 chars

        //read memory block blockn
        if(nfc.readMemoryBlock(1,blockn,block))
        {
          //if read operation is successful
          for(uint8_t i=0;i<16;i++)
          {
            //print memory block
            Serial.print(block[i],HEX);

            if (blockn == 4) // Save off block 4 chars
            {
              theStr += (char) block[i];
            }

            if(block[i] <= 0xF) //Data arrangement / beautify
            {
              Serial.print("  ");
            }
            else
            {
              Serial.print(" ");
            }
          }

          Serial.print("| Block ");
          if(blockn <= 9) //Data arrangement / beautify
          {
            Serial.print(" ");
          }
          Serial.print(blockn,DEC);
          Serial.print(" | ");

          if(blockn == 0)
          {
            Serial.println("Manufacturer Block");
          }
          else
          {
            if(((blockn + 1) % 4) == 0)
            {
              Serial.println("Sector Trailer");
            }
            else
            {
              Serial.println("Data Block");
            }
          }

          if (blockn == 4)  // DPL
          {
            Serial.print("Chars from block 4: ");
            Serial.println(theStr);
          }
        }
      }
    }
  }
  delay(2000);
}
```

8.4 Compile and upload

Following the same Arduino* Uno procedure above, use the standard version of the sketch (that came with the library) to test.

8.5 Results

Arduino* 101 compatible

§

9 EMAX* ES08A Analog Servo

9.1 Use case

Used for simple RC and robotics applications. The servo library is an Arduino* Core library. They are easy to use and the base of many robots.

Key Info	Links
URL	*http://www.yinyanmodel.com/En/ProductView.asp?ID=106*
Library	NA
Sketch	Sketch initially came from the Arduino* Uno board version 1.0.5 r2.

Figure 29 **EMAX* ES08A analog servo**

9.2 Hardware summary

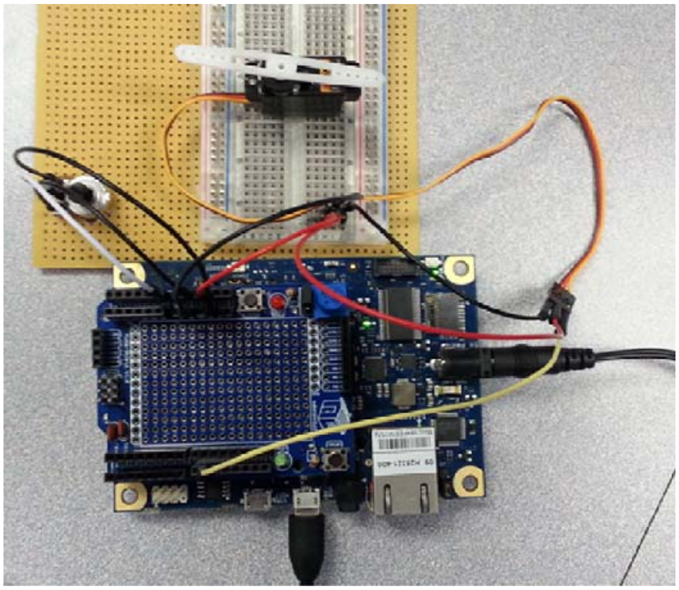
Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Operating Voltage	5 V
Use VIN as power source	Depends on the motor shield, for this sketch power was from 5 V pin
Weight	8.0 g
Dimensions	23.0 × 11.5 × 24.0 mm
Stall Torque	At 4.8 V: 1.5 kg/cm, at 6.0 V: 1.8 kg/cm
Speed	At 4.8 V: 0.12 sec/60 degrees, 0.10 sec/60 degrees
Pulse width range	1.5 to 1.9 ms

Servo and potentiometer connections:

Servo Wire	Arduino* 101 Pin and Function
Orange	Connect to D9 (Pulse control)
Red	Connect to 5 V (Power)
Brown	Connect to GND (Ground)

Potentiometer wire	Arduino* 101 Pin and Function
Left pin	Connect to GND
Right pin	Connect to 3.3 V
Center	Connect to A0

Figure 30 **EMAX* ES08A analog servo on an board**



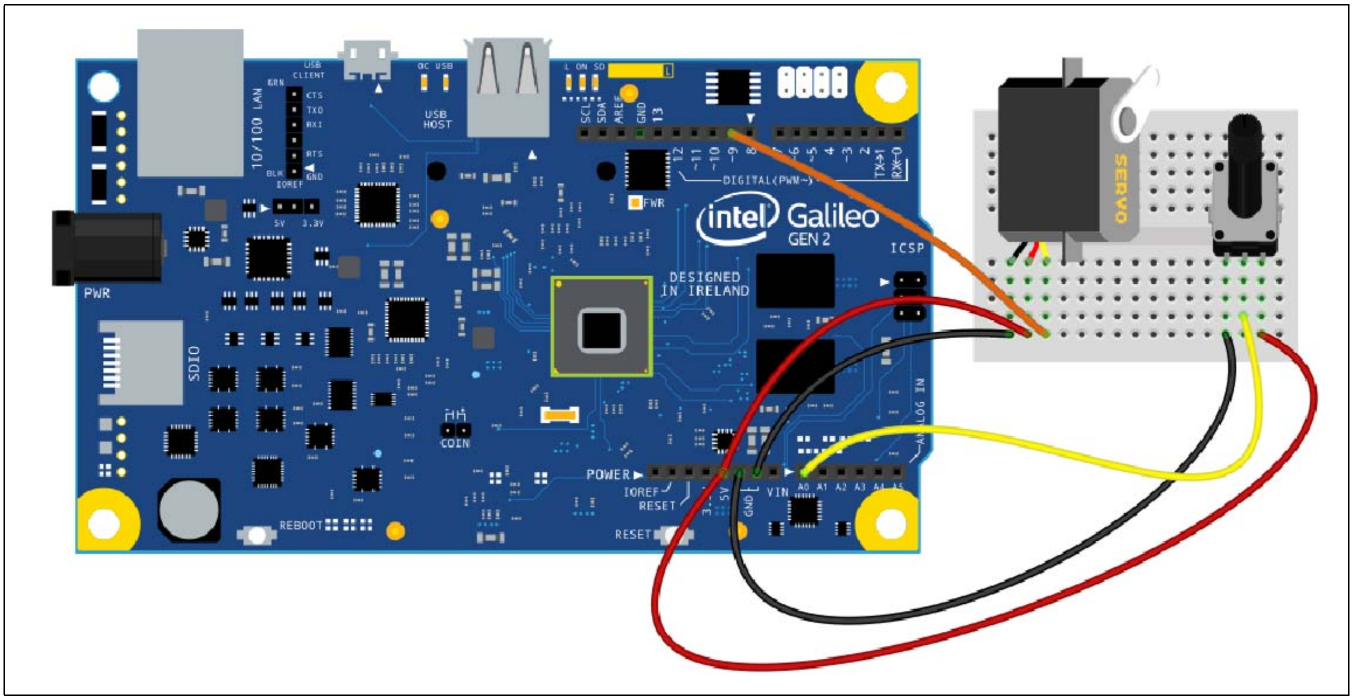
9.3 Companion library

No library required.

9.4 Compile and upload

Connect the Servo to the Arduino* 101 and run the example sketch below. For Arduino* 101 use barrel jack or VIN external power.

Figure 31 **Connecting an analog servo EMAX* ES08A to an Intel® Galileo second generation board**



Example 9 shows a simple sketch that demonstrates the servomotor being controlled by turning the potentiometer (10 kohm). The motor will rotate a little over 180 degrees when changing the pot from 0 ohm to 10 kohm.

Example 9 **Servomotor sketch**

```
// Controlling a servo position using a potentiometer (variable resistor)
// by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>
#include <Servo.h>
Servo myservo;  // create servo object to control a servo

int dPin    = 9;  // digital pin to drive servo
int aPinPot = 0;  // analog  pin connect to the potentiometer
void setup()
{
  myservo.attach(dPin);  // attaches the servo on pin 9 to the servo object
}
void loop()
{
  int val;    // variable to read the value from the analog pin
  val = analogRead(aPinPot); // read value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179); // scale it for use with servo (between 0 and 180)
  myservo.write(val);      // sets the servo position according to the scaled value
  delay(15); // waits for the servo to get there
}
```

9.5 Results

- Arduino* 101 compatible
- Motor also works with a few of the servo shields in this report.

§

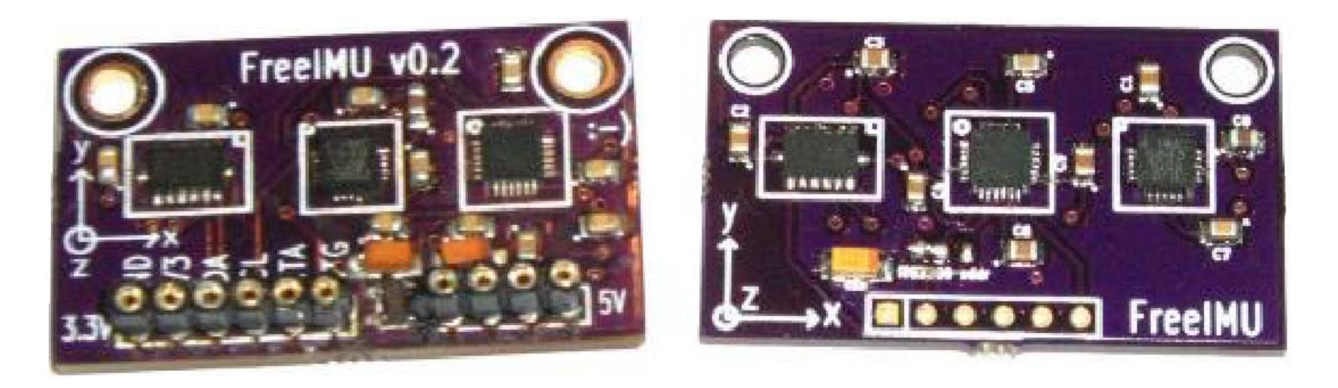
10 FreeIMU*

10.1 Use case

FreeIMU is an open source inertial measurement unit that is an ongoing research project. The power of this library is its compatibility with many other IMU present on the market. It uses multiple libraries, one for each sensor compatible with the FreeIMU library that contains the sensor fusion algorithms.

Key Info	Links
URL	*http://www.varesano.net/projects/hardware/FreeIMU*
Library	*https://github.com/Fabio-Varesano-Association/freeimu*

Figure 32 **FreeIMU***



The compatible sensors work mainly using I²C. The library seems to have very little AVR-specific code, which would make it a good candidate for either the Intel® Galileo or Arduino* Uno boards.

FreeIMU is widely used in quadcopters, motion tracking, and odometry. FreeIMU has been used for various projects, such as a sensor for cameras to adjust lens distortion.

10.2 Hardware summary

The sensors all use the I²C interface; no other connections are needed for basic usage. It also has two interrupt pins that are the outputs of the Gyroscope+accelerometer (MPU6050) and the magnetometer (HMC5883).

Pin Name	Function
SDA, SCL	I²C bus

10.3 Companion library

It has been reported that the library is structured as a collection of libraries for different sensors, which makes the FreeIMU library compatible with other IMU boards.

Here the list of the compatible IMUs:

The FreeIMU library also supports the following 3rd parties' boards:

- SparkFun* IMU Digital Combo Board - 6 Degrees of Freedom ITG3200/ADXL345 SEN-10121
- SparkFun 9 Degrees of Freedom - Razor IMU SEN-10736
- SparkFun 9 Degrees of Freedom - Sensor Stick SEN-10724
- SparkFun 9 Degrees of Freedom - Sensor Stick SEN-10183
- DIYDrones
- * ArduIMU+ V3
- Generic MPU6050 Breakout boards (eg: GY-521, SEN-11028 and other MPU6050 that have the MPU6050 AD0 pin connected to GND.)

Because this board is no longer available, no additional analysis could be done.

10.4 Results

Inconclusive, since the board is not available.

§

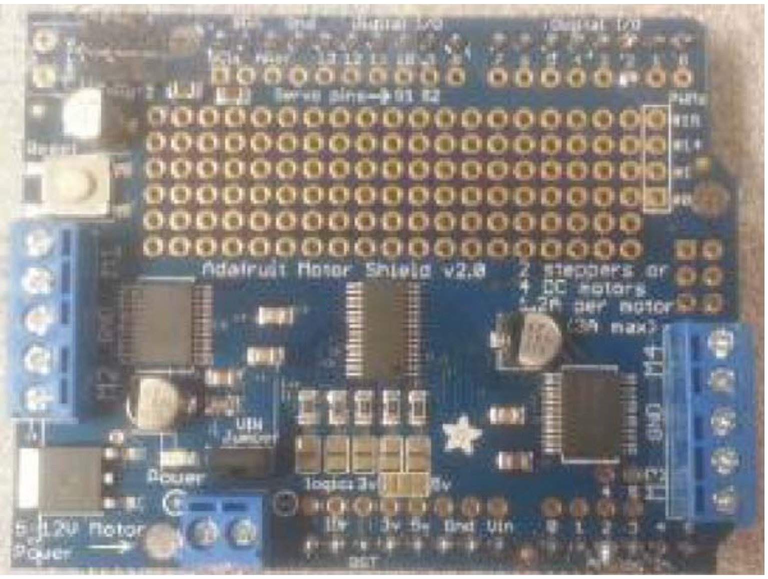
11 Adafruit* Motor Shield v2.0

11.1 Use case

The Adafruit* motor shield v2.0 has the ability to drive four DC motors or two stepper motors. The shield has a fully dedicated PWM driver chip onboard. This chip handles all the motor and speed controls over I^2C. Only two pins (SDA and SCL) are required to drive the multiple motors. Since it is I^2C, it is possible to connect any other I^2C devices or shields to the same pins. There are five address select pins, so it supports 32 stackable shields, which maps to 64 steppers or 128 DC motors.

Key Info	Links
URL	*http://www.adafruit.com/products/1438*
Guide	*https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview*
Library	*https://github.com/ladyada/Adafruit_Motor_Shield_V2_Library/archive/master.zip 05/03/2016*

Figure 33 **Adafruit* motor shield v2.0**



11.2 Hardware summary

Key Info	Function
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IOREF	3.3 or 5 V solderable jumper (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	
Power for motors	5 to 12 V via external power source. Shield supports 2 Steppers or 4 DC motors. 1.2 A per motor (3 A Max)
Power for logic	3.3 or 5 V selectable through a soldering jumper.
Schematic	*https://learn.adafruit.com/assets/9536*
TB6612FNG	DC motor driver datasheet *http://www.adafruit.com/datasheets/TB6612FNG_datasheet_en_20121101.pdf*
Motors Used	DC1, DC2, ST1, ST2, ST3

Shield Pin Name	Function (no wiring required for digital/analog pins)
I^{2}C bus	Connect to SDA and SCL pins. It is possible to stack multiple shields, however, each shield must have a unique I^{2}C address: Board 0: Address = 0x60 Offset = binary 0000 (no jumpers required) Board 1: Address = 0x61 Offset = binary 0001 (bridge A0) Board 2: Address = 0x62 Offset = binary 0010 (bridge A1, to the left of A0) Board 3: Address = 0x63 Offset = binary 0011 (bridge A0 & A1, two rightmost jumpers) Board 4: Address = 0x64 Offset = binary 0100 (bridge A2, middle jumper)
D9	Only if servo is used.
D10	Only if servo is used.

The shield has a jumper to select power from the Vin power source or power from the screw terminals on the shield.

11.3 Compile and upload

Adafruit_MotorShield.cpp and Adafruit_PWMServoDriver.cpp will need a change in order to compile. In the sections that define WIRE, put in code that will leave WIRE defined as ‘Wire’ instead of ‘Wire1’ such as:

Example 10

DC Motor Test:

Connections:

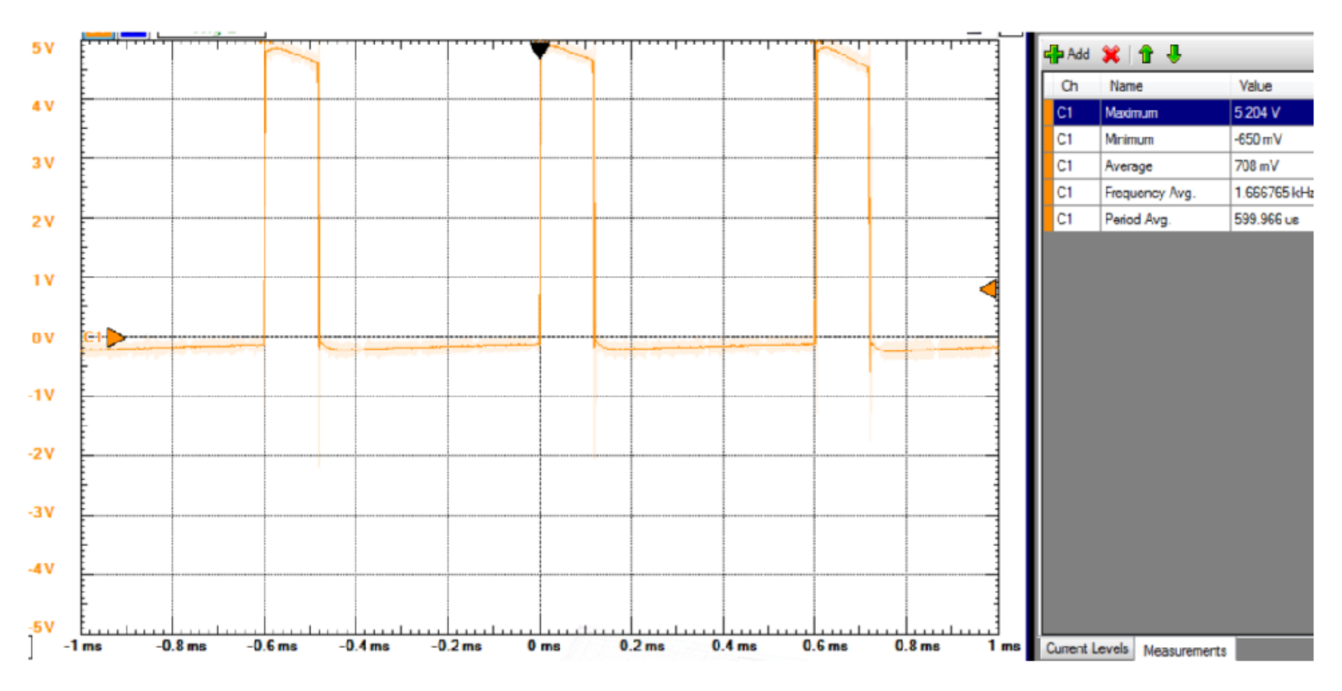
- Connect a DC motor to each of the terminals and test the DC motor sketch
- The 5 V DC motor can be used without the external power supply

The DC Motor Test tests one DC motor per execution. The DC motor number is changed to see that the shield will power the motor from 0 to 255 for all four DC ports separately. The following sketches were taken from the examples that were installed with the library.

Example 11 **Sketch: DC motor test**

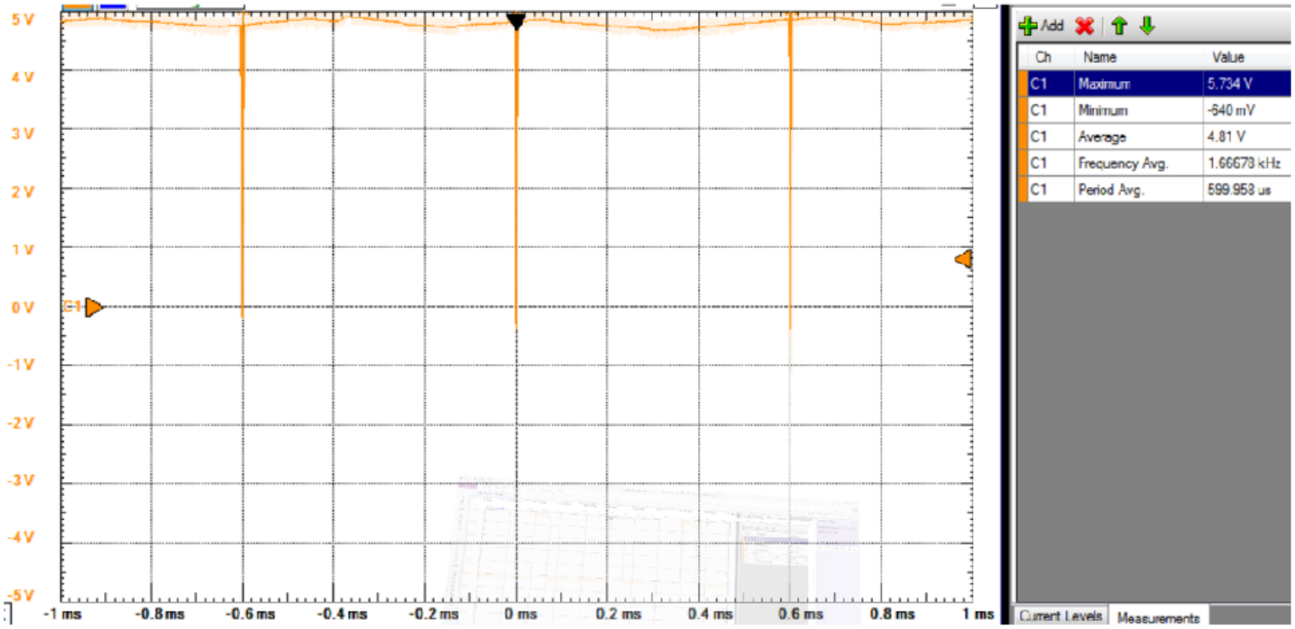
```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Or, create it with a different I2C address (say for stacking)
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
// You can also make another motor on port M2
//Adafruit_DCMotor *myOtherMotor = AFMS.getMotor(2);

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Adafruit Motorshield v2 - DC Motor test!");

  AFMS.begin();  // create with the default frequency 1.6KHz
  //AFMS.begin(1000);  // OR with a different frequency, say 1KHz

  // Set the speed to start, from 0 (off) to 255 (max speed)
  myMotor->setSpeed(150);
  myMotor->run(FORWARD);
  // turn on motor
  myMotor->run(RELEASE);
}

void loop() {
  uint8_t i;

  Serial.print("tick");

  myMotor->run(FORWARD);
  for (i=0; i<255; i++) {
    myMotor->setSpeed(i);
    delay(10);
  }
  for (i=255; i!=0; i--) {
    myMotor->setSpeed(i);
    delay(10);
  }

  Serial.print("tock");

  myMotor->run(BACKWARD);
  for (i=0; i<255; i++) {
    myMotor->setSpeed(i);
    delay(10);
  }
  for (i=255; i!=0; i--) {
    myMotor->setSpeed(i);
    delay(10);
  }

  Serial.print("tech");
  myMotor->run(RELEASE);
  delay(1000);
}
```

Figure 34 **Pulse width while connected to DC motor speed set at 50**



Ch	Name	Value
C1	Maximum	5.204 V
C1	Minimum	-650 mV
C1	Average	708 mV
C1	Frequency Avg.	1.666765 kHz
C1	Period Avg.	599.966 us

Figure 35 **Pulse width while connected with DC motor speed set at 255**

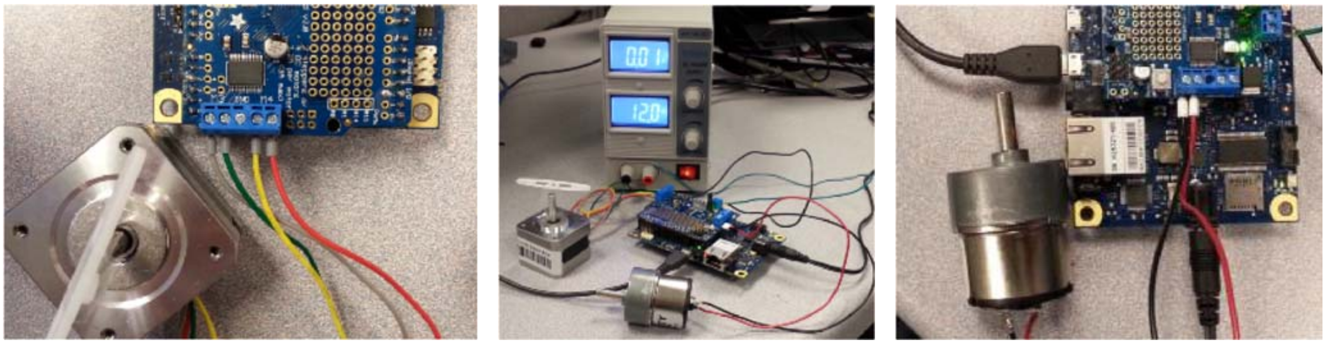


Ch	Name	Value
C1	Maximum	5.734 V
C1	Minimum	-640 mV
C1	Average	4.81 V
C1	Frequency Avg.	1.66678 kHz
C1	Period Avg.	599.958 us

Stepper motor test:

Connect a DC motor to M1 and a stepper to M3/M4 for this sketch. Test can also handle a servo connected to servo1 if desired. The stepper is a bipolar, 200x, 12 V motor. The DC motor is a 12 V geared motor. Use external power supply 12 V.

Figure 36 **Connections for a stepper motor test on Intel® Galileo first generation board**



```
Stepper Test
/*
This is a test sketch for the Adafruit assembled Motor Shield for Arduino v2
It won't work with v1.x motor shields! Only for the v2's with built in PWM
control

For use with the Adafruit Motor Shield v2
---->      http://www.adafruit.com/products/1438
*/


#include <Wire.h>
```

```
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Or, create it with a different I2C address (say for stacking)
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// motor port #1 (M1 and M2)
// motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2); // Steps, Port

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Stepper test!");

  AFMS.begin();  // create with the default frequency 1.6KHz
  //AFMS.begin(1000);  // OR with a different frequency, say 1KHz

  myMotor->setSpeed(10);  // 10 rpm
}

void loop() {
  Serial.println("Single coil steps");
  myMotor->step(100, FORWARD, SINGLE);
  myMotor->step(100, BACKWARD, SINGLE);

  Serial.println("Double coil steps");
  myMotor->step(100, FORWARD, DOUBLE);
  myMotor->step(100, BACKWARD, DOUBLE);

  Serial.println("Interleave coil steps");
  myMotor->step(100, FORWARD, INTERLEAVE);
  myMotor->step(100, BACKWARD, INTERLEAVE);

  Serial.println("Microstep steps");
  myMotor->step(50, FORWARD, MICROSTEP);
  myMotor->step(50, BACKWARD, MICROSTEP);
}
```

11.4 Results

- Arduino* 101 compatible for DC and stepper motors.

§

12 Arduino* GSM Shield

12.1 Use case

The Arduino* GSM Shield connects to the Internet using the GPRS wireless network. Plug in a SIM card from an operator offering GPRS coverage, and the shield is ready to start using the Internet. The shield also supports making and receiving voice calls and text messaging. The shield uses a Quectel* radio modem M10 and supports communication with the board using AT commands. For voice calls, it is necessary to add a speaker and microphone to hear and speak to the other party. This will require soldering.

Key Info	Links
Product info	*http://arduino.cc/en/Main/ArduinoGSMShield*
Reference	*http://arduino.cc/en/Reference/GSM*
Guide	*http://arduino.cc/en/Guide/ArduinoGSMShield*
Library	Library installed by default with the Arduino* Uno board.
AT commands	*http://arduino.cc/en/uploads/Main/Quectel_M10_AT_commands.pdf*

Figure 37 **Arduino** GSM Shield**

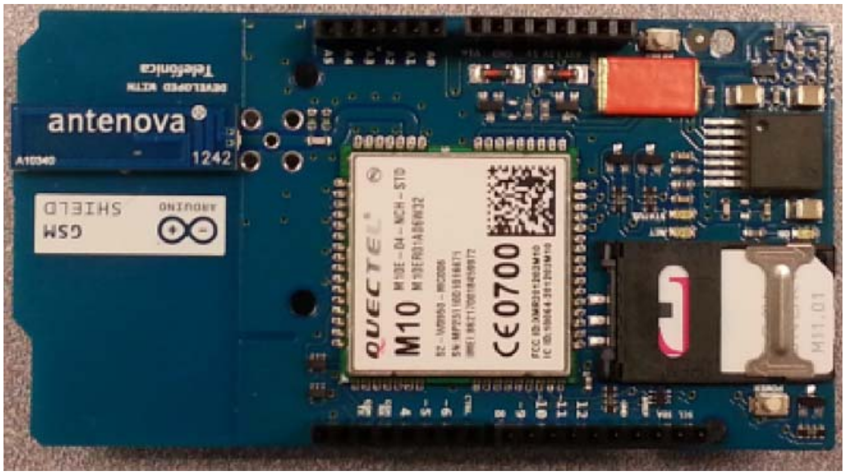
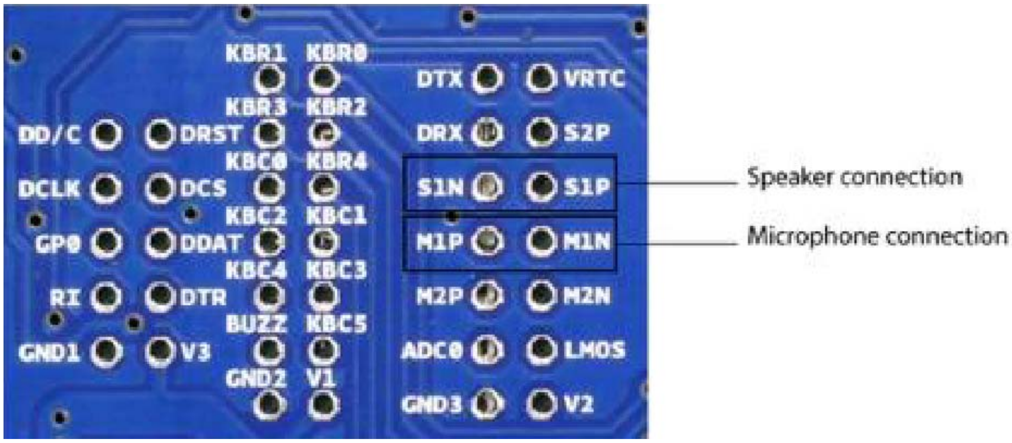


Figure 38 **Speaker and microphone connectors**



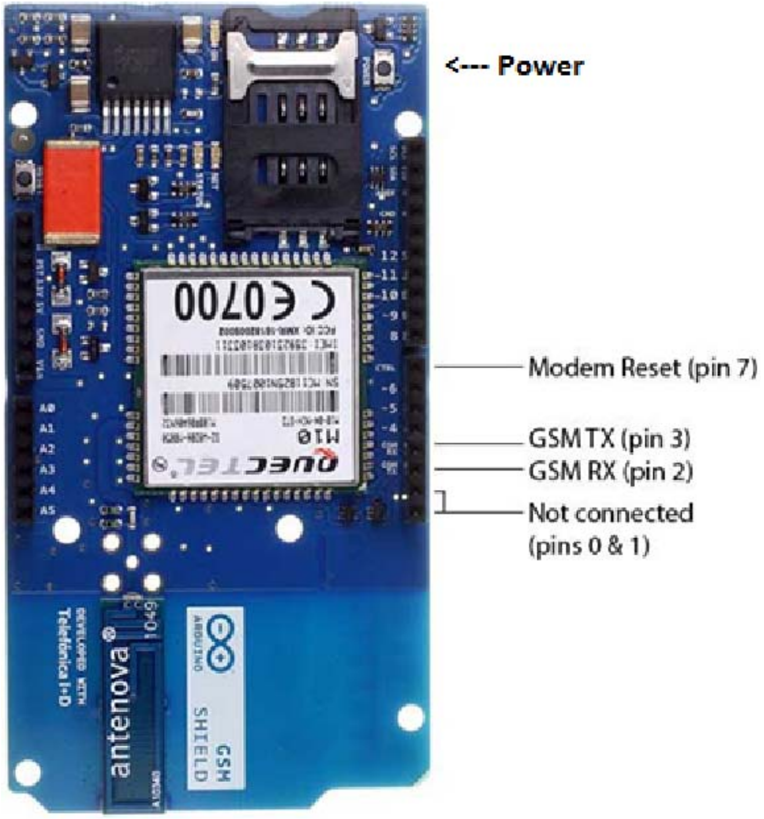
12.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	Yes
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematic	*http://arduino.cc/en/uploads/Main/arduino-gsm-shield-06-reference-design.zip*

Pin Name	Function
GSM Rx	Connected to D2 (UART Rx)
GSM Tx	Connected to D3 (UART Tx)
CTRL	Connected to D7 (Modem Reset)

There is a jumper on the back side of the shield to select the pin D7 as CTRL pin, which controls the power supply to the Quectel M10 module. The jumper must be inserted to use the shield with the library.

Figure 39 **Modem, Tx/Rx connectors and power**



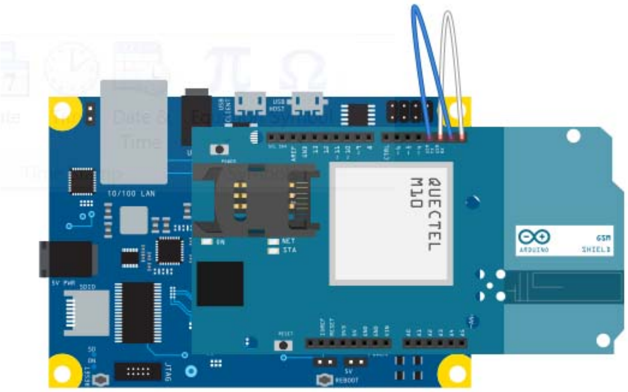
12.3 Companion library

The GSM library is included with Arduino* IDE 1.0.4 and later. The library uses the software serial library for communication between the modem and the Arduino Uno board.

12.4 Compile and upload

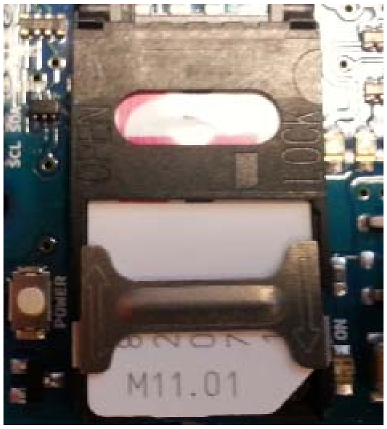
A sketch below was used to use hardware serial. A few pins needed to be re-routed as show below. The transmit of the shield (D2) was connected to the D0 (receive) pin of the Arduino* 101and the receive of the shield (D3) was connected to the D1 (transmit) pin of the Arduino 101. A sketch was uploaded that is successful with voice calls and texting using AT commands on other GSM/GPRS shields. Install the shield to the Arduino 101 and wire the transmit and receive connections.

Figure 40 **Wiring the Arduino* GSM Shield**



Insert an unlocked SIM card. Power up and connect the Arduino 101 USB to the PC. Push the shield power button until the status led and ON led is illuminated. The net led should also blink.

Figure 41 **Unlocked SIM card inserted**



- Update the following sketch to input a valid telephone number into the following lines of code. For this example, this is a US call using a 1 followed by a fictitious area code 555 and phone number 5555555.

```
char* gPhoneNumber = "15555555555";
```

- Upload the following sketch that will enable sending text messages and making calls.
- Open the serial terminal from the IDE
- at 9600 baud.
- From the serial terminal, input one of the following:
  - **a:** Answer a voice call. (The ring should appear in the serial terminal and the ring tone is played through the headset if connected.)
  - **d:** Dial a voice call. (Phone number already set in the sketch.)
  - **g:** Set up an IP session.
  - **r:** Display all received text messages.
  - **t:** Send a text message. (Text and phone number already set in the sketch.)

Example 12 **Sketch: Issue AT commands over serial link**

```
// G A L I L E O
TTYUARTClass*   gSerialStdPtr = &Serial;  // Galileo, /dev/ttyGSO, Tx pin
TTYUARTClass*   gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO,  Rx pin

char   data[256];
String gPhoneNumber = "15555555555"; // 15555555555
char*  gTextLine1   = "G2>";
char*  gTextLine2   = "Hello from Shield 12";
#define BAUDRATE1 9600

void setup()
{
  Serial1.begin(BAUDRATE1);
  Serial.begin(BAUDRATE1);    // the GPRS baud rate
  waitForUser(9);
}
void loop()
{
  if (Serial.available() > 0)
  {
    switch(Serial.read())
    {
    case 't':
      SendTextMessage();
      break;
    case 'r':
      ReceiveTextMessage();
      break;
    case 'd':
      DialVoiceCall();
      break;
    case 'a':
      AnswerVoiceCall();
      break;
    case 'g':
      GPRS();
      break;
    case 'h':
      HangupCall();
      break;
    case 'x':
      Check();
      break;
    } // switch
  } // if
  if (Serial1.available())
    Serial.write(Serial1.read());
}
void Check()
{
  Serial1.println("AT");
  delay(100);
  ShowSerialData();

  Serial1.println("ATE1");
  delay(100);
  ShowSerialData();
}
void AnswerVoiceCall()
{
  Serial1.println("ATA");
  delay(100);
  ShowSerialData();
}
void HangupCall()
{
  Serial1.println("ATH1"); // H, hang up (1=hang up, 1=pickup)
  delay(100);
  ShowSerialData();
}
void SendTextMessage()
{
  Serial1.println("AT+CMGF=1");    //Because we want to send the SMS in text mode
  delay(100);
  ShowSerialData();

  Serial1.print("AT+CMGS=\"");
  Serial1.print(gPhoneNumber);
  Serial1.println("\"");
  Serial1.print(gTextLine1);
  Serial1.println(gTextLine2);
  Serial1.println((char)26);  //the ASCII code of the ctrl+z is 26 (0x1A)
  delay(1000);
  ShowSerialData();
}
void ReceiveTextMessage()
{
  Serial1.println("AT+CMGF=1");       // We want to receive the SMS in text mode
  delay(100);
  ShowSerialData();

  Serial1.println("AT+CPMS=\"SM\"");  // read first SMS
  delay(100);
  ShowSerialData();

  Serial1.println("AT+CMGL=\"ALL\""); // show message
  delay(100);
  ShowSerialData();
}
void DialVoiceCall()
{
  Serial1.print("ATD+");             // dial the number
  Serial1.print(gPhoneNumber);
  Serial1.println(";");
  delay(1000);
  ShowSerialData();
}
void ShowSerialData()
{
  while(Serial1.available()>0)
    Serial.write(Serial1.read());
}
void GPRS()
{
  Serial1.println("AT+CPIN?");    // Is SIM ready to use?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+CGREG?");    // Is device registered?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+COPS?");    // Does SIM info match network?
  delay(1000);
  ShowSerialData();

  Serial.println("Check signal quality");
  Serial1.println("AT+CSQ");    // Check signal quality
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+cgatt=1");    // GPRS attach
  delay(1000);
  ShowSerialData();

  // define a PDP context with IP connection, ID is 1
  Serial1.println("AT+CGDCONT=1,\"IP\",\"fast.t-mobile.com\"");
  delay(1000);
  ShowSerialData();

  // list PDP contexts that are defined
  Serial1.println("at+cgdcont?");
  delay(3000);
  ShowSerialData();

  // setup the session using the appropriate PDP context
  Serial1.println("AT+CGACT=1,1");
  delay(1000);
  ShowSerialData();

  Serial.println("session is setup delay 5 seconds");
  delay(5000);

  // deactivate the PDP context
  Serial1.println("AT+CGACT=0,1");
  delay(1000);
  ShowSerialData();

  // detach from GPRS newtork
  Serial1.println("AT+CGATT=0");
  delay(1000);
  ShowSerialData();
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
}
```

12.5 Results

Arduino* 101 not tested

§

13 Arduino* TFT Screen

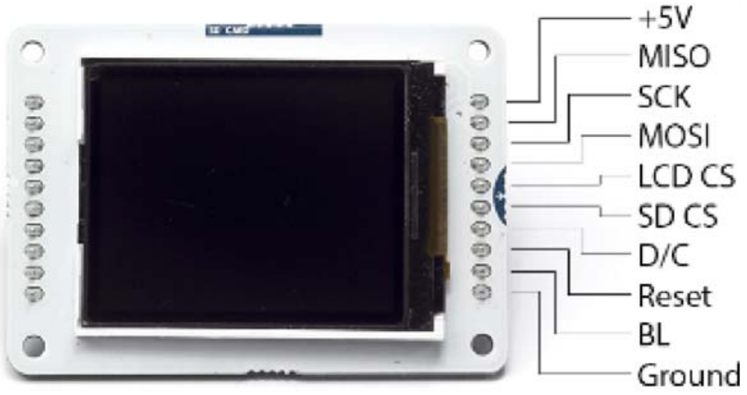
13.1 Use case

The Arduino* TFT (thin-film-transistor) screen is a backlit LCD screen with headers. You can draw text, images, and shapes to the screen with the TFT library. There is an onboard micro-SD card slot on the back of the screen that can, among other things, stores bitmap images for the screen to display. This TFT screen was originally conceived as an add-on for the Arduino Robot or the Esplora, because only these two products have the specific connector.

Key Info	Links
Product info	*http://arduino.cc/en/Main/GTFT*
Reference	*http://arduino.cc/en/Reference/TFTLibrary*
Guide	*http://arduino.cc/en/Guide/TFT* *http://arduino.cc/en/Guide/TFTtoBoards*
Library	Use the library from corelibs.

Because it is a color display with a built-in SD card slot, and the library has few extra methods with a processing-like syntax, it becomes quite useful as a breakout board with all the other Arduino boards (including the Arduino* Due). The library for this shield compiles for all the Arduino boards.

Figure 42 **Arduino* TFT Screen pins**



13.2 Hardware summary

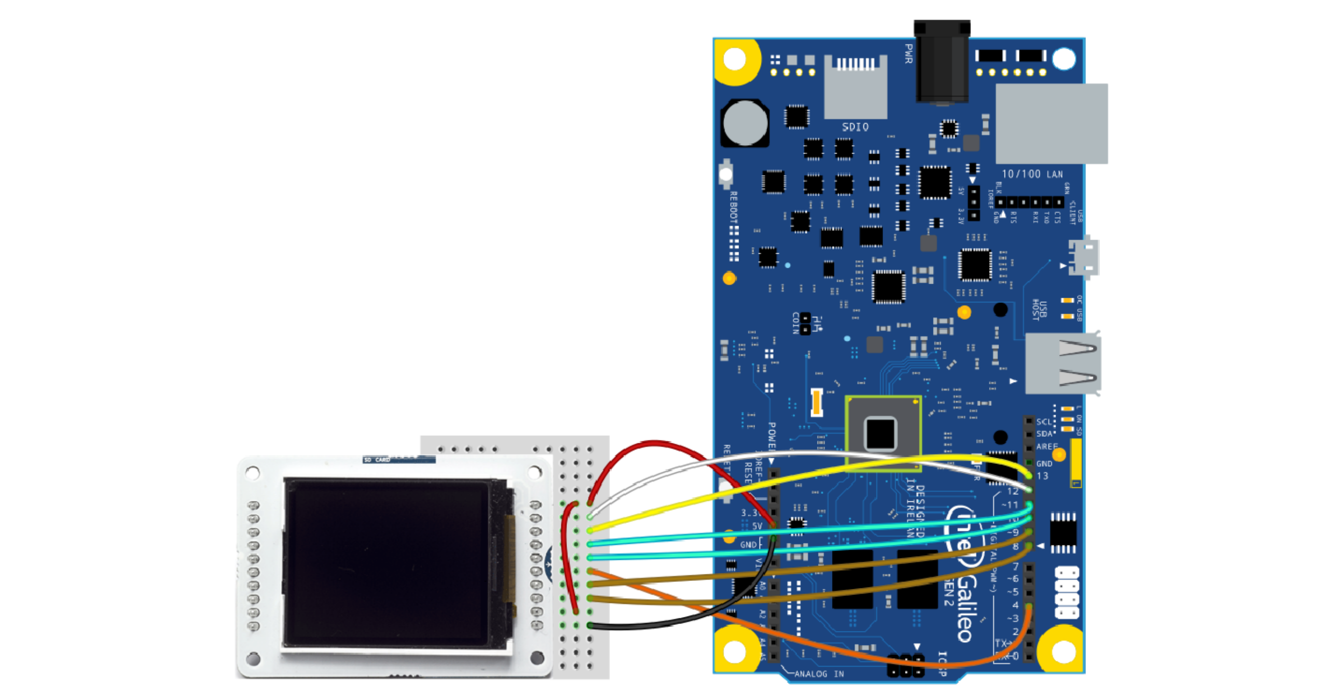
Key Info	Description/Links
Operating voltage	5 V
IOREF	No
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*http://arduino.cc/en/uploads/Main/LCD_Rev-4.zip*

Table 6 **Pin connections**

TFT Board	MC Board
+5 V	5 V
MISO	pin 12
SCK	pin 13
MOSI	pin 11
LCD CS	pin 10

TFT Board	MC Board
SD CS	pin 4
D/C	pin 9
RESET	pin 8
BL	+5 V
GND	GND

Figure 43 **Connecting an Arduino** TFT Screen to an Intel® Galileo second generation board**



The screen is 1.77" diagonal, with 160 × 128-pixel resolution. The display uses the SPI interface to communicate with the Arduino* Uno board.

13.3 Companion library

The library comes with the IDE. For Arduino* 101 remove the TFT library from the Arduino sketch folder and use the one from the Arduino 101 board installation.

13.4 Compile and upload

Example 13 is a modified version of the Arduino TFT bitmap logo example. The file shall be named “arduino.bmp”. The file size is 7 KB. If the file format is incorrect, an error will display in the serial terminal. (Step by step instructions are in the sketch’s header.)

Example 13 **Modified Arduino* TFT bitmap logo sketch**

```
/* This example reads an image file from a micro-SD card
 and draws it on the screen, at random locations.
 In this sketch, the Arduino logo is read from a micro-SD card.
 There is a .bmp file included with this sketch.
 - open the sketch folder (Ctrl-K or Cmd-K)
 - copy the "arduino.bmp" file to a micro-SD
 - put the SD into the SD slot of the Arduino TFT module.

 http://arduino.cc/en/Tutorial/TFTBitmapLogo  */

// include the necessary libraries
#include <SPI.h>
#include <SD.h>
#include <TFT.h>  //Arduino LCD library

// pin definition for the Uno
```

```
#define sd_cs  4
#define lcd_cs 10
#define dc     9
#define rst    8

// pin definition for the Leonardo
//#define sd_cs  8
//#define lcd_cs 7
//#define dc     0
//#define rst    1

TFT TFTscreen = TFT(lcd_cs, dc, rst);

// this variable represents the image to be drawn on screen
PImage logo;


void setup() {
  // initialize the GLCD and show a message
  // asking the user to open the serial line
  TFTscreen.begin();
  TFTscreen.background(255, 255, 255);

  TFTscreen.stroke(0, 0, 255);
  TFTscreen.println();
  TFTscreen.println("Arduino TFT Bitmap Example");
  TFTscreen.stroke(0, 0, 0);
  TFTscreen.println("Open serial monitor");
  TFTscreen.println("to run the sketch");

  // initialize the serial port: it will be used to
  // print some diagnostic info
  Serial.begin(9600);
  while (!Serial) { // wait for serial line to be ready
  }

  // clear the GLCD screen before starting
  TFTscreen.background(255, 255, 255);

  // try to access the SD card. If that fails (e.g.
  // no card present), the setup process will stop.
  Serial.print("Initializing SD card...");
  if (!SD.begin(sd_cs))
  {
    Serial.println("failed!");
    return;
  }
  Serial.println("OK!");

  // initialize and clear the GLCD screen
  TFTscreen.begin();
  TFTscreen.background(255, 255, 255);

  // now that the SD card can be access, try to load the
  // image file.
  logo = TFTscreen.loadImage("arduino.bmp");
  if (!logo.isValid())
  {
    Serial.println("error while loading arduino.bmp");
  }
}
```

```
void loop()
{ // don't do anything if the image wasn't loaded correctly.
  if (logo.isValid() == false)
  {
    return;
  }

  Serial.println("drawing image");

  TFTscreen.background(255, 255, 255);  // clear the screen

  // place random colored squares in each corner DPL
  TFTscreen.stroke(0,0,255); // outline the rectangles with a blue line
  TFTscreen.fill(random(255), random(255), random(255));
  TFTscreen.rect(0,0,10,10);
  TFTscreen.fill(random(255), random(255), random(255));
  TFTscreen.rect(TFTscreen.width() -10,0,TFTscreen.width(),10);
  TFTscreen.fill(random(255), random(255), random(255));
  TFTscreen.rect(0,TFTscreen.height() - 10,10,TFTscreen.height());
  TFTscreen.fill(random(255), random(255), random(255));
  TFTscreen.rect(TFTscreen.width() - 10,TFTscreen.height() - 10,
TFTscreen.width(),TFTscreen.height());

  // get a random location where to draw the image.
  // To avoid the image to be draw outside the screen,
  // take into account the image size.
  int x = random(TFTscreen.width() - logo.width());
  int y = random(TFTscreen.height() - logo.height());

  // draw the image to the screen
  TFTscreen.image(logo, x, y);

  // wait a little bit before drawing again
  delay(1500);
}
```

13.5 Results

Arduino* 101 passes.

Worked with the Arduino* Uno board.

§

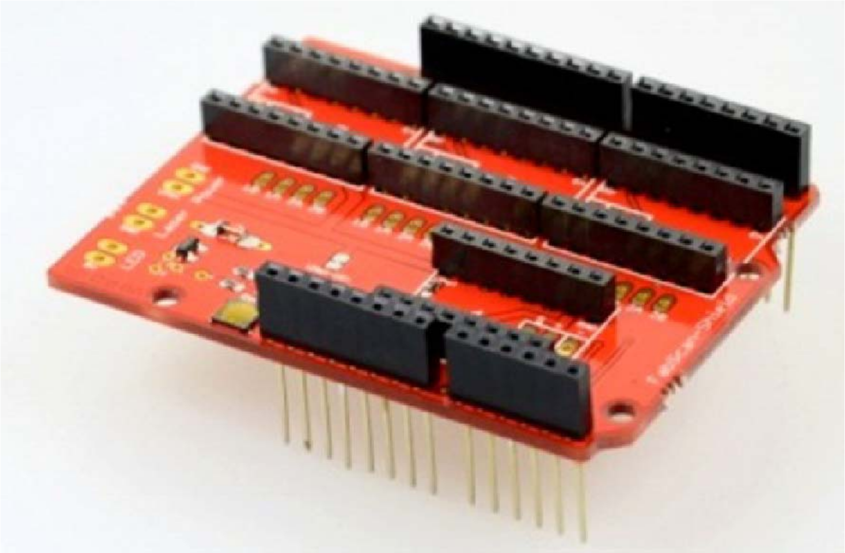
14 FabScan* 3D Scanner

14.1 Use case

FabScan is an open source, do-it-yourself 3D laser scanner. FabScan was featured on *Thingiverse* and it is popular on *GitHub*. This shield is the result of the cooperation between multiple enthusiasts. The shield is also a universal motor driver as it can connect up to four *Pololu* A4988 stepper motor drivers*. It is possible to build a complete 3D Laser Scanner with this *BOM*, which will require a motor-driver shield, a 2.5 to 5.0 mW laser, a stepper motor, a webcam, and some connecting hardware.

Key Info	Links
Product Info	*http://www.3dotmatrix.com/shop/fabscan-shield-copy/* *http://hci.rwth-aachen.de/fabscan*
Library	N/A
Example Sketch	*https://raw.githubusercontent.com/francisengelmann/FabScan/master/arduino/FabScanArduinoFirmware.pde*

Figure 44 **FabScan* 3D scanner**



14.2 Hardware summary

Key Info	Description/Links
Operating Voltage	Logic 5 V; motors and laser can operate at voltage from external supply
Use VIN as power source	Yes or external supply if connected
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*https://github.com/watterott/FabScan-Shield/tree/master/pcb*
PWM	D3
Motors	SV5, SV7

14.3 Companion library

N/A

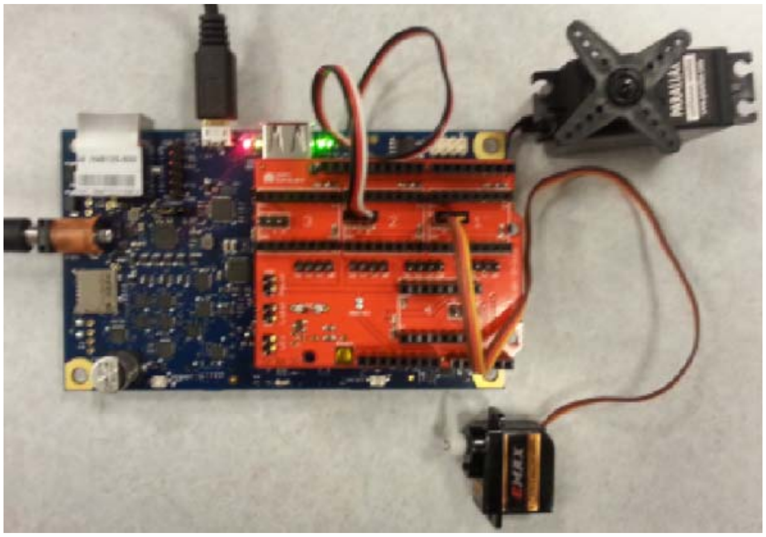
14.4 Compile and upload

Two sketches (from the link above) were used to test the shield. The following sketch was tested without stepper motors since the stepper driver breakout board was not used. Connect the shield to the Arduino* 101 board and upload the following sketch. To test with stepper motors follow the instructions for using the *stepper motor carrier*.

```
Example Sketch
// FabScan - http://hci.rwth-aachen.de/fabscan
//
// R. Bohne 30.12.2013
// This sketch tests all four stepper drivers on the FabScan-Shield V1.1
// It also turns on the light and the laser
// This sketch is not the real FabScan firmware, but just a test script for people who want to
test their hardware!
// at startup, the sketch blinks all leds, the light and the motor a few times and after that,
it starts to spin the motors.

#define LIGHT_PIN 17
#define LASER_PIN 18
#define MS_PIN    19

//Stepper 1 as labeled on Shield, Turntable
#define ENABLE_PIN_0  2
#define STEP_PIN_0    3
#define DIR_PIN_0     4

//Stepper 2, Laser Stepper
#define ENABLE_PIN_1  5
#define STEP_PIN_1    6
#define DIR_PIN_1     7

//Stepper 3, currently unused
#define ENABLE_PIN_2  11
#define STEP_PIN_2    12
#define DIR_PIN_2     13

//Stepper 4, currently unused
#define ENABLE_PIN_3  14
#define STEP_PIN_3    15
#define DIR_PIN_3     16

void setup()
{
  pinMode(LASER_PIN, OUTPUT);
  pinMode(LIGHT_PIN, OUTPUT);

  pinMode(MS_PIN, OUTPUT);
  digitalWrite(MS_PIN, HIGH);  //HIGH for 16microstepping, LOW for no microstepping

  pinMode(ENABLE_PIN_0, OUTPUT);
  pinMode(DIR_PIN_0, OUTPUT);
  pinMode(STEP_PIN_0, OUTPUT);

  pinMode(ENABLE_PIN_1, OUTPUT);
  pinMode(DIR_PIN_1, OUTPUT);
  pinMode(STEP_PIN_1, OUTPUT);

  pinMode(ENABLE_PIN_2, OUTPUT);
  pinMode(DIR_PIN_2, OUTPUT);
  pinMode(STEP_PIN_2, OUTPUT);

  pinMode(ENABLE_PIN_3, OUTPUT);
  pinMode(DIR_PIN_3, OUTPUT);
  pinMode(STEP_PIN_3, OUTPUT);

  //enable turntable and laser steppers
  digitalWrite(ENABLE_PIN_0, LOW);  //HIGH to turn off
  digitalWrite(ENABLE_PIN_1, LOW);  //HIGH to turn off
  digitalWrite(ENABLE_PIN_2, LOW);  //LOW to turn on
  digitalWrite(ENABLE_PIN_3, LOW);  //LOW to turn on

  //blink all leds, lights ans the laser 10 times
  for(int i=0; i<10; i++)
  {
    digitalWrite(ENABLE_PIN_0, HIGH);  //HIGH to turn off
    digitalWrite(ENABLE_PIN_1, HIGH);  //HIGH to turn off
    digitalWrite(ENABLE_PIN_2, HIGH);  //LOW to turn on
    digitalWrite(ENABLE_PIN_3, HIGH);  //LOW to turn on
    digitalWrite(LIGHT_PIN, 0); //turn light off
    digitalWrite(LASER_PIN, 0); //turn laser off
    delay(120);
    digitalWrite(ENABLE_PIN_0, LOW);  //HIGH to turn off
    digitalWrite(ENABLE_PIN_1, LOW);  //HIGH to turn off
    digitalWrite(ENABLE_PIN_2, LOW);  //LOW to turn on
    digitalWrite(ENABLE_PIN_3, LOW);  //LOW to turn on
    digitalWrite(LIGHT_PIN, 1); //turn light on
    digitalWrite(LASER_PIN, 1); //turn laser on
    delay(120);
  }

}

//loop just steps all four steppers. Attached motors should turn!
void loop()
{
  digitalWrite(STEP_PIN_0, HIGH);
  digitalWrite(STEP_PIN_1, HIGH);
  digitalWrite(STEP_PIN_2, HIGH);
  digitalWrite(STEP_PIN_3, HIGH);
  delay(1);
  digitalWrite(STEP_PIN_0, LOW);
  digitalWrite(STEP_PIN_1, LOW);
  digitalWrite(STEP_PIN_2, LOW);
  digitalWrite(STEP_PIN_3, LOW);
  delay(1);

}
```

The following sketch was based on the sweep sketch from the Servo library that is installed with the IDE. Connect two servo motors and upload the sketch. The servo motors should rotate back and forth. The voltage of the servo motors should be ~5 V, since there was no external power supply needed for these motors. The motors used were EMAX* ES08A, Tower Pro* MG995R, Parallex Continuous.

Figure 45 **FabScan* 3D scanner on Intel® Galileo second generation board**

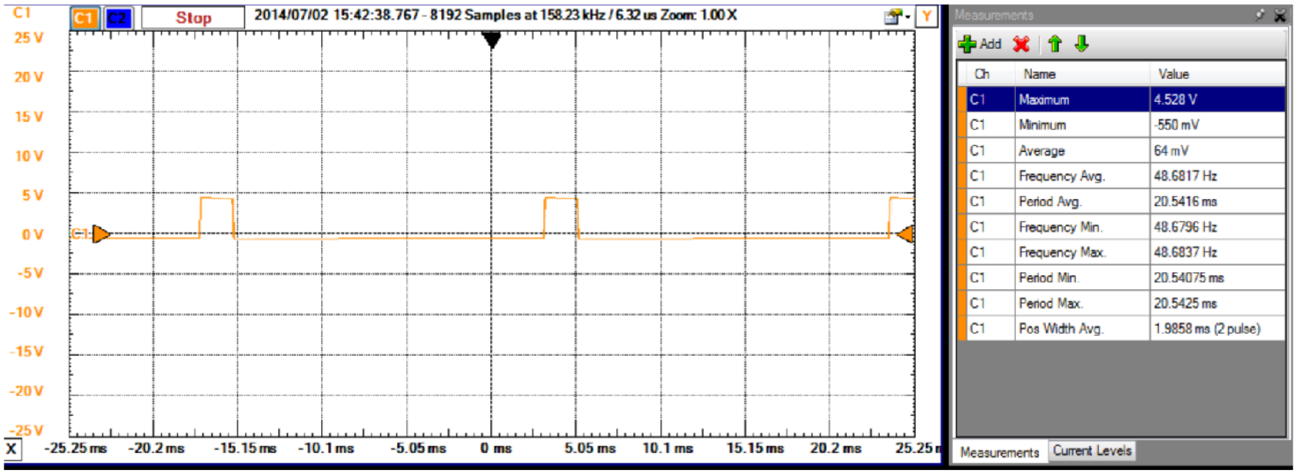


Example 14 **Example servo sketch**

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.


#include <Servo.h>

// map the pins for the servos
#define SERVO1 3
#define SERVO2 6

Servo myservo1;  // create servo object to control a servo, a maximum of eight servo objects
can be created
Servo myservo2;

int pos = 0;    // variable to store the servo position

void setup()
{
  myservo1.attach(SERVO1);  // attach the servos
  myservo2.attach(SERVO2);
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo1.write(pos);              // tell servo to go to position in variable 'pos'
    myservo2.write(pos);
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    myservo1.write(pos);              // tell servo to go to position in variable 'pos'
    myservo2.write(pos);
    delay(15);                       // waits 15ms for the servo to reach the position
    }
}
```

Arduino* 101: The servo is able to provide smooth operation with the pulse width and frequency provided. Below is a snapshot of the oscilloscope on the control pin (D3) of one of the servo motors.

Figure 46 **Intel® Galileo second generation: Pulse width and frequency graph on D3**



2014/07/02 15:42:38.767 - 8192 Samples at 158.23 kHz / 6.32 us Zoom: 1.00 X

Measurements

Ch	Name	Value
C1	Maximum	4.528 V
C1	Minimum	-550 mV
C1	Average	64 mV
C1	Frequency Avg.	48.6817 Hz
C1	Period Avg.	20.5416 ms
C1	Frequency Min.	48.6796 Hz
C1	Frequency Max.	48.6837 Hz
C1	Period Min.	20.54075 ms
C1	Period Max.	20.5425 ms
C1	Pos Width Avg.	1.9858 ms (2 pulse)

14.5 Results

Tested using a sketch without connecting the stepper motor driver and lasers. Another sketch to control servo motors was tested. To test with steppers follow the instructions for using the *stepper motor carriers*.

Arduino* 101 Compatible

The shield appears compatible as the voltages for the motor and laser were being supplied when running the sketch. The servo motors on ports one and two were working. According to the BOM of some FabScan kits, it would be possible to create a 3D scanner with one stepper motor.

§

15 RedBearLabs* Nordic BLE Shield

15.1 Use case

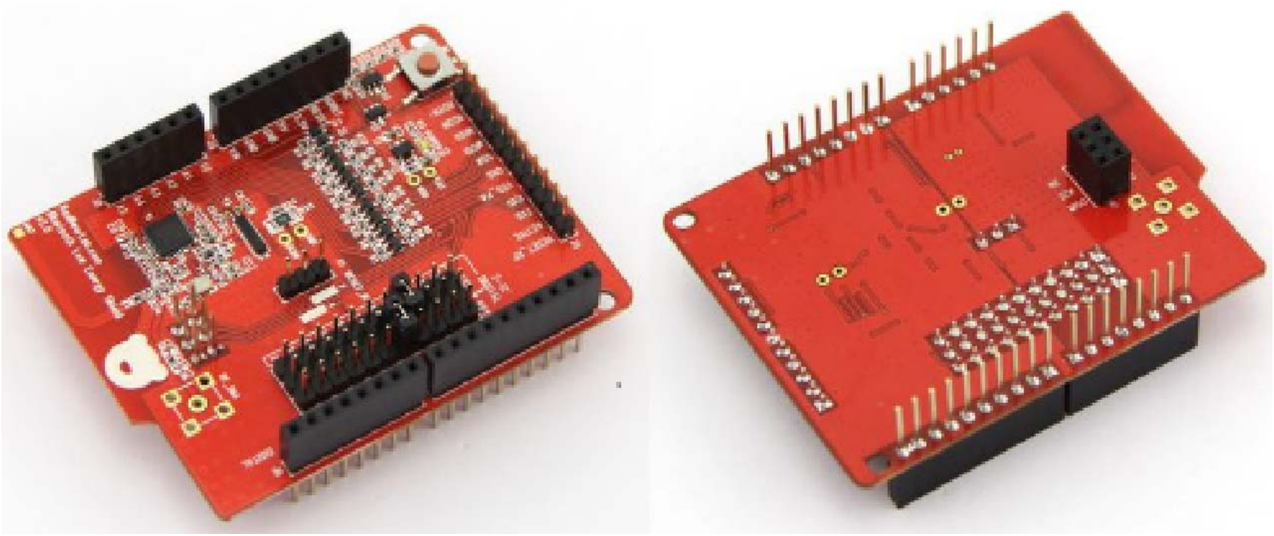
The Bluetooth* Low Energy (BLE) shield is designed to work with Arduino* Uno boards and allows you to connect your Arduino* Uno board with other BLE Central devices, like smartphones and tablets. The BLE shield can operate with 3.3 or 5 V; therefore, it should work with a lot of Arduino Uno-compatible boards. This version of the board enables the attachment of an antenna for greater range. Many BLE modules are becoming common in the market. This test will determine if it could be used with the Intel® Galileo board.

Key Inf0	Links
Product info	http://www.makershed.com/Bluetooth_Low_Energy_BLE_Shield_for_Arduino_2_0_p/mkrbl1.htm
Library	Library 1: Nordic Bluetooth low energy SDK for Arduino*, (052914) https://github.com/NordicSemiconductor/ble-sdk-arduino Library 2: RedBearLab nRF8001 Library version (063014) https://github.com/RedBearLab/nRF8001
Phone app	BLE Controller (for iOS) https://itunes.apple.com/app/ble-controller/id855062200 BLE Sensor Tag (for Android) https://play.google.com/store/apps/details?id=sample.ble.sensortag&hl=en
Guide	http://redbearlab.com/getting-started-bleshield/ http://www.makershed.com/Bluetooth_Low_Energy_BLE_Shield_for_Arduino_2_0_p/mkrbl1.htm http://boriskourt.com/2013/12/09/setting-up-the-ble-mini-from-red-bear-lab/

Using the Arduino Uno board and BLE shield together allows you to:

- Control your Arduino Uno pins with our/your own mobile App
- Send sensor data from your Arduino Uno board to an app for processing
- Use your mobile device as an
- Internet gateway for your Arduino Uno board and much more!

Figure 47 **RedBearLabs* Nordic BLE shield**



15.2 Hardware summary

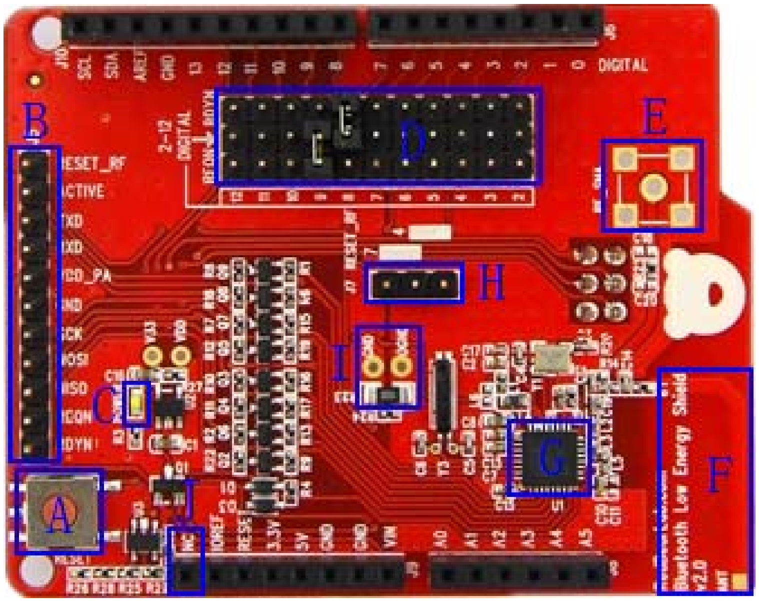
Key Info	Description/Links
Operating voltage	Designed for 3.3 and 5 V
IOREF	Present but not used
Use VIN as power source	No
Power	5 V or VIN, automatically selected.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*https://github.com/RedBearLab/BLEShield/commit/306dd34feef285cdebdb75dd9724e8decc9a66c9*
Antenna	Onboard PCB antenna and with an optional SMA connector for external antenna.
BLE connectivity IC	*http://www.nordicsemi.com/eng/Products/Bluetooth-R-low-energy/nRF8001*

Shield layout:

A. nRF8001 and Arduino* Uno board reset button
B. Factory testing pins
C. Power-on LED
D. Flexible REQN and RDYN pins from pin 2 to 12
E. External antenna
F. Onboard antenna
G. Nordic nRF8001
H. Optional pin to control the shield's reset
I. Power consumption measurement of the nRF8001 chip
J. Power monitor on oscilloscope

Signal	Arduino* Uno	nRF8001	Description
MISO	Input	Output	SPI: Master In Slave Out
MOSI	Output	Input	SPI: Master Out Slave In
SCK	Output	Input	SPI: Serial data Clock
REQN	Output D8 (On Shield)	Input	Application controller to nRF8001 handshake signal
RDYN	Input D9 (On Shield)	Output	nRF8001 to application controller handshake signal

Figure 48 **RedBearLabs* Nordic BLE shield layout**



15.3 Companion library

Compiling on an Arduino* Uno board requires importing the BLE libraries into the Arduino IDE. The BLE library is in the Nordic Bluetooth low-energy SDK for the Arduino Uno board. There are 19 example sketches; however, this test uses the 'ble_A_Hello_World_Program' sketch.

Arduino* Uno board:

The exact procedure to get this running on the Arduino Uno board is a little confusing. Some sites reference outdated libraries called 'BLEShield' and others reference 'nRF8001'. The "BLE" library works as is on the Arduino Uno board . This test requires another device that the shield can perform BLE communication with. This example uses an Android phone with the "BLE Sensor Tag" app.

When the sketch loaded, the shield is immediately detected (in bold below). The shield then waits for a device to connect to the shield, so you will need to start the app on the phone. The app will begin to scan, and a device labeled as 'Hello' will appear on the phone. Select this device (on the phone), and a series of services become accessible. Disconnection of the device will display in the IDE serial monitor. The output below is the output for a successful connection using the Arduino Uno board.

Example 15 **Sketch output: ble_A_Hello_World_Program**

```
Arduino setup
Set line ending to newline to send data from the serial monitor
Set up done
Evt Device Started: Setup
Evt Device Started: Standby
Advertising started : Tap Connect on the nRF UART app
Evt Connected
Evt Pipe Status
Evt Disconnected/Advertising timed out
Advertising started. Tap Connect on the nRF UART app
```

Intel® Galileo/Intel® Edison:

Because compiling the library generates a lot of compile errors, we created a bare sketch to help focus on modification of the library first. Place the following sketch in the IDE, but do not compile right away.

Example 16 **Sample Sketch for compiling library**

```
#include <SPI.h>
#include <lib_aci.h>
#include <aci_setup.h>
#include "uart_over_ble.h"
#include "services.h"
void setup(void){
}
void loop() {
}
```

One of the major compile issues is that a series of typedefs is not available to the Intel® Galileo board. Edit the "hal_plaform.h" file that is part of the library. There are board-dependent definitions, but not one for the Intel Galileo board. Add an "else" and include the Arduino header file and compile. This will bring in the "typedefs" the compiler is looking for; however, more changes are required.

Example 17 **Modification for hal_plaform.h**

```
    //Redefine the function for reading from flash in ChipKit
    #define memcpy_P        memcpy
#endif
    //Redefine the function for reading from flash in ChipKit
    #define memcpy_P        memcpy
#else
    #include "Arduino.h"
#endif
```

Edit the following implementation file and remove the AVR specific header file cited below.

Example 18 **Modifications for hal_aci_tl.cpp**

```
#include <SPI.h>
```

```
#include "hal_platform.h"
#include "hal_aci_tl.h"
#include "aci_queue.h"
#include <avr/sleep.h>
#include <SPI.h>
#include "hal_platform.h"
#include "hal_aci_tl.h"
#include "aci_queue.h"
//#include <avr/sleep.h>
```

On the Arduino Uno board, if memory in SRAM is tight, a typical technique is to move constant data from SRAM to FLASH memory. The “PROGMEM” keyword performs this functionality for the Atmel processor, but it is not available in for the Intel Galileo board (since the processor is not Atmel). The Intel Galileo board sets a ‘#define’ for the keyword that enables a successful compile, but it will not work. The Arduino Uno board gives very little space for program space and data; however, the Intel Galileo board has a much larger SRAM space that is shared with Linux*.

In this implementation file, the mention ‘PROGMEM’ keyword is wrapped in a macro call the “F” macro. This “F” macro needs to be removed throughout the program. Currently this includes other library files (dfu.cpp, etc.).

Example 19 **Remove F macros from hal_aci_tl.cpp**

```
  for (i=0; i<=length; i++)
  {
    Serial.print(p_data->buffer[i], HEX);
    Serial.print(F(", "));
  }
  Serial.println(F(""));
}
  for (i=0; i<=length; i++)
  {
    Serial.print(p_data->buffer[i], HEX);
    Serial.print(", ");
  }
  Serial.println("");
}
```

The library should successfully compile with all the above changes. Load the “ble_A_Hello_World_Program” that is included with the BLE library. This example requires that all of the F macros be removed.

15.4 Compile and upload

Arduino* Uno board:

Compiling on the Arduino Uno board was successful. Importing the BLE libraries into the Arduino IDE brought in new example sketches. These sketches compiled and uploaded successfully. The companion app for iOS and Android, *BLE Controller*, is required.

Intel® Galileo/Intel® Edison:

An attempt to make this board functional on the Arduino Uno board has not yet succeeded. All of the compile errors were resolved, but what is next is to resolve the Serial-vs-Serial1 calls for Arduino Uno board. Due to time constraints, we did not complete this.

Since the Nordic BLE module makes use of the SPI interface to communicate with the microcontroller board, there is a good chance that issues exist (due to lack of SPI performance) related to the communication with the Intel Galileo board.

15.5 Results

Arduino* 101

§

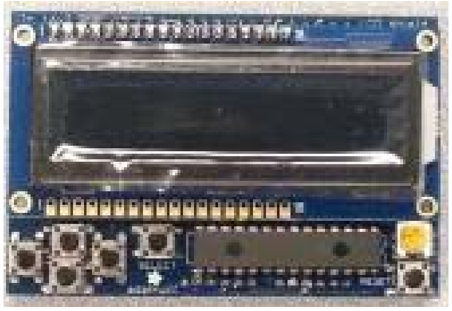
16 Adafruit* I²C RGB LCD Shield Kit with 16 × 2 Display

16.1 Use case

The LCD is a 16 × 2 character LCD, with up to three backlight pins and five keypad pins, using only two I²C pins on the board. This shield provides a quick way to add a display without all the wiring. This is a good display to use when you want to build a standalone project with its own user interface.

Key Info	Links
Product info	*https://www.adafruit.com/product/714*
Library	*https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library. Date: 12/21/2015*
Guide	*https://learn.adafruit.com/rgb-lcd-shield/using-the-rgb-lcd-shield*

Figure 49 **Adafruit* I²C RGB LCD shield kit with 16 × 2 display**

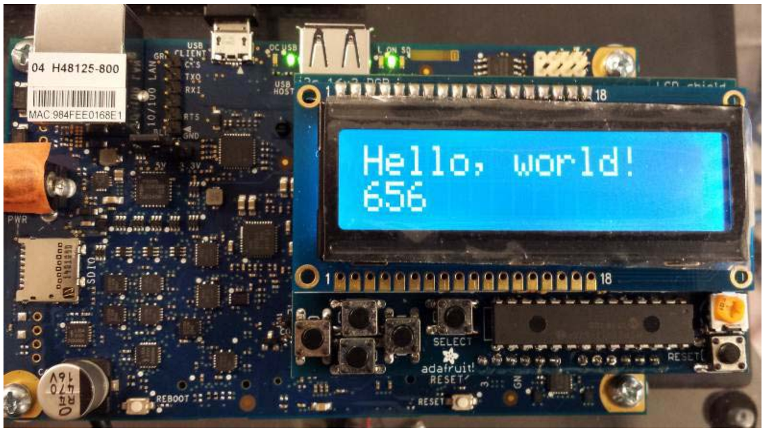


16.2 Hardware summary

Pin Name	Function
Use VIN as power source	No
Operating voltage	5 V
Power for logic	5 V, 3.3 V
Schematic	*https://github.com/adafruit/Adafruit-RGB-LCD-shield*
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635

Pin Name	I²C Function (No wires required)
A4	I²C, SDA (serial data line) data
A5	I²C, SCL (serial clock line) clock
Device address	I²C, address defined in Adafruit_MCP23017.h header file as 32d defines as: #define MCP23017_ADDRESS 0x20
GND	Ground
Power	5 V, 3.3 V

Figure 50 **Adafruit* I²C RGB LCD shield kit with 16 × 2 display on Intel® Galileo second generation**



16.3 Companion library

The shield and LCD were tested using the 'hello world' demo that installed with the library called "Adafruit_RGBLCDShield". The shield worked on the all the boards.

16.4 Compile and upload

The *hello world* demo compiles and runs.

- Plug the LCD directly onto the board.

Following is a simple sketch that should demonstrate the LCD to display 'hello world' and allowing changes the display back light to five different colors using the buttons.

Example 20 **Sketch: hello world**

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>
#include <Adafruit_RGBLCDShield.h>

// The shield uses the I2C SCL and SDA pins. On classic Arduinos
// this is Analog 4 and 5 so you can't use those for analogRead() anymore
// However, you can connect other I2C sensors to the I2C bus and share
// the I2C bus.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

// These #defines make it easy to set the backlight color
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7

void setup() {
  // Debugging output
  Serial.begin(9600);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);

  // Print a message to the LCD. We track how long it takes since
  // this library has been optimized a bit and we're proud of it :)
  int time = millis();
  lcd.print("Hello, world!");
  time = millis() - time;
  Serial.print("Took "); Serial.print(time); Serial.println(" ms");
  lcd.setBacklight(WHITE);
}

uint8_t i=0;
void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis()/1000);

  uint8_t buttons = lcd.readButtons();

  if (buttons) {
    lcd.clear();
    lcd.setCursor(0,0);
    if (buttons & BUTTON_UP) {
      lcd.print("UP ");
      lcd.setBacklight(RED);
    }
    if (buttons & BUTTON_DOWN) {
      lcd.print("DOWN ");
      lcd.setBacklight(YELLOW);
    }
    if (buttons & BUTTON_LEFT) {
      lcd.print("LEFT ");
      lcd.setBacklight(GREEN);
    }
    if (buttons & BUTTON_RIGHT) {
      lcd.print("RIGHT ");
      lcd.setBacklight(TEAL);
    }
    if (buttons & BUTTON_SELECT) {
      lcd.print("SELECT ");
      lcd.setBacklight(VIOLET);
    }
  }
}
```

16.5 Results

Arduino* 101 compatible.

§

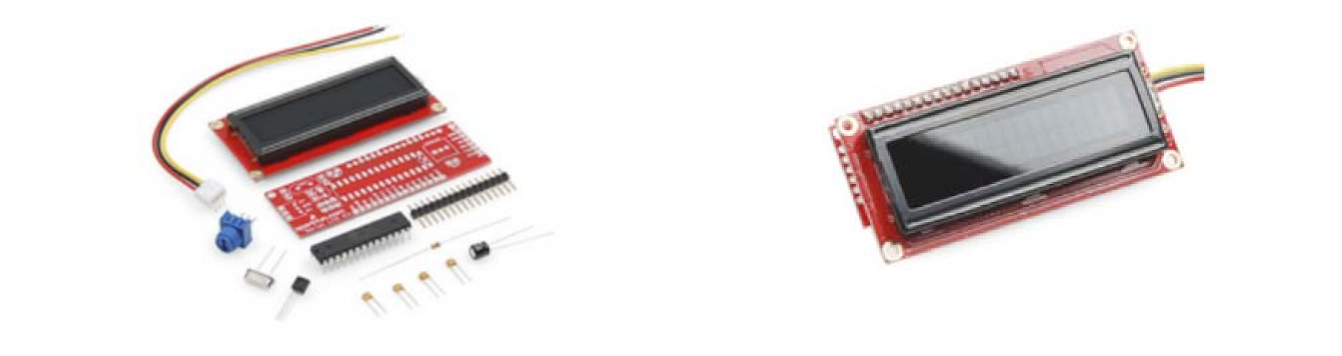
17 SparkFun* Serial-Enabled LCD Shield

17.1 Use case

This breakout board allows you to write on an LCD screen using only one pin of the board. This add-on board runs firmware that allows you to communicate with the display using a serial protocol. The kit is a good choice when you want to add a display in your project. The kit will help you to save the controller board pins, and can simplify the code because the display refresh is decoupled from the sketch.

Key Info	Links
Product Info	*https://www.sparkfun.com/products/10097*
Library	No library
Guide/Tutorial	*https://www.sparkfun.com/tutorials/289* Sample programs: *https://github.com/jimblom/Serial-LCD-Kit/wiki/Serial-Enabled-LCD-Kit-Datasheet*
Schematic	*https://www.sparkfun.com/datasheets/Kits/Serial-LCD-Kit-v11.pdf*

Figure 51 **SparkFun* serial-enabled LCD shield**



17.2 Hardware summary

Pin Name	Function
Use VIN as power source	No
Operating voltage	5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8

On the add-on board, there are also pins for connecting to an FTDI cable (or compatible) to change the firmware. The pins of the Atmel* ATmega328 that are not used by the display are broken out on one side of the PCB.

The connection requires three wires: red to provide the power supply (5 V only), black provide the ground, and cyan for the serial port receiver pin, which must be connected to the transmit on the controller board. You have the option to use the pins on the left or right (FTDI pins).

Shield Pin	Function w/Connections
GND	Ground, connect to Intel® Galileo GND
5 V	Power, connect to Intel Galileo 5 V
RxI	Serial port Receiver pin, connected to the Intel® Galileo first generation/Intel Galileo Tx pin (D1) Since Intel® Galileo second generation has another serial pin, connect to Tx pin (D3).
TxO	Serial port Transmit pin (not used)
DTR	Reset (not used)

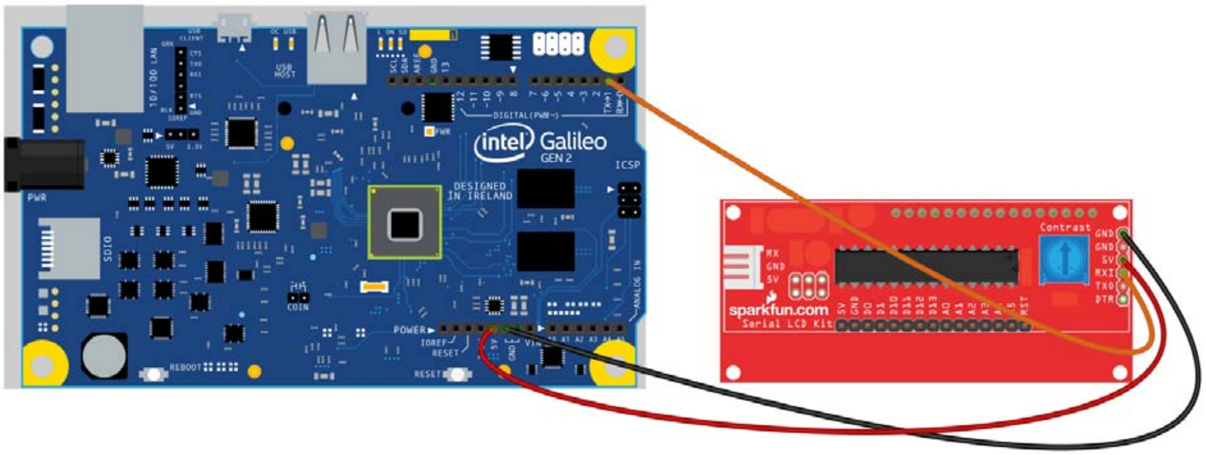
17.3 Companion library

No library required.

17.4 Compile and upload

The example sketch used for this test is from the Guide/Tutorial page. It uses most of the available serial commands. Connect 5 V, GND and Rx to the board. Rx is connected to Tx (D1). The only thing to be changed is the 'lcd' declaration that uses the SoftwareSerial library for the Arduino* Uno board. For the board, the hardware serial is used.

Figure 52 **Connecting a SparkFun* serial-enabled LCD shield**



Example 21 **Sample code's serial port was changed as follows**

Before	After

`#include <SoftwareSerial.h>`
`SoftwareSerial lcd(2, 3);`
	`#define lcd Serial1`

Example 22 **Test sketch**

```
#define lcd Serial1  // Galileo 1 or Galileo 2
//#define lcd Serial2  // Only Galileo 2
int year = 11;  // Enter the current year here, 11 = 2011
int month = 6;  // Enter the current month here, 6 = June
int date = 20;  // Enter the current date here
int day = 1;  // 0 = Sunday, 6 = Saturday
int hours = 23;  // Enter the hours here
int minutes = 59;  // Enter the minutes here
int seconds = 50;  // Enter the seconds here
void setup()
{
  lcd.begin(9600);  // Start the LCD at 9600 baud
  clearDisplay();  // Clear the display
  setLCDCursor(2);  // Set cursor to the 3rd spot, 1st line
  lcd.print("Hello, world");
  setLCDCursor(16);  // Set the cursor to the beginning of the 2nd line
  lcd.print("Running clock...");
  // Flash the backlight:
  for (int i=0; i<3; i++)
  {
    setBacklight(0);
    delay(250);
    setBacklight(255);
    delay(250);
  }
}
void loop()
{
  if (!(millis() % 1000))  // If it's been 1 second
  {
    checkTime();  // this function increases the time and date if necessary
    clearDisplay();  // Clear the display
    setLCDCursor(1);  // set cursor to 2nd spot, 1st row
    printDay();  // print the day
    lcd.print(" ");
    lcd.print(month);  // print the date:
    lcd.print("/");
    lcd.print(date);
    lcd.print("/");
    lcd.print(year);
    setLCDCursor(20);  // set the cursor to the 5th spot, 2nd row
    lcd.print(hours);  // print the time:
    lcd.print(":");
    lcd.print(minutes);
    lcd.print(":");
    lcd.print(seconds);
  }
}
void setBacklight(byte brightness)
{
  lcd.write(0x80);  // send the backlight command
  lcd.write(brightness);  // send the brightness value
}
void clearDisplay()
{
  lcd.write(0xFE);  // send the special command
  lcd.write(0x01);  // send the clear screen command
}
void setLCDCursor(byte cursor_position)
{
  lcd.write(0xFE);  // send the special command
  lcd.write(0x80);  // send the set cursor command
  lcd.write(cursor_position);  // send the cursor position
}
void printDay()
{
  switch(day)
  {
    case 0:
      lcd.print("Sun.");
      break;
    case 1:
      lcd.print("Mon.");
      break;
    case 2:
      lcd.print("Tue.");
      break;
    case 3:
      lcd.print("Wed.");
      break;
    case 4:
      lcd.print("Thur.");
      break;
    case 5:
      lcd.print("Fri.");
      break;
    case 6:
      lcd.print("Sat.");
      break;
  }
}
void checkTime()
{
  seconds++;  // increase seconds
  if (seconds == 60)  // If it's been a minute
  {
    seconds = 0;  // start over seconds
    minutes++;  // Increase minutes
    if (minutes == 60)  // If it's been an hour
    {
      minutes = 0;  // start over minutes
      hours++;  // increase hours
      if (hours == 24)  // If it's been a day
      {
        hours = 0;  // start the day over
        day++;  // increase the day
        if (day == 7)  // if it's been a week
          day = 0;  // start the week over
        date++;  // increase the date
        checkDate();  // this function increases the date/month/year if necessary
      }
    }
  }
}
void checkDate()
{
  // 30 days has sept. apr. jun. and nov.
  if (((month == 9)||(month == 4)||(month == 6)||(month == 11))&&
      (date > 30))
  {
    date = 1;
    month++;
  }
  else if ((month == 2)&&(date > 28))
  {
    date = 1;
    month++;
  }
  else if (date > 31)
  {
    date = 1;
    month++;
    if (month > 12)
    {
      month = 1;
      year++;  // happy new year!
      clearDisplay();
      lcd.print("Happy New Year!");
      delay(5000);
      seconds+=5;
    }
  }
}
```

All the other serial commands in the list have been tested and work as expected.

17.5 Results

Arduino* 101 Compatible.

§

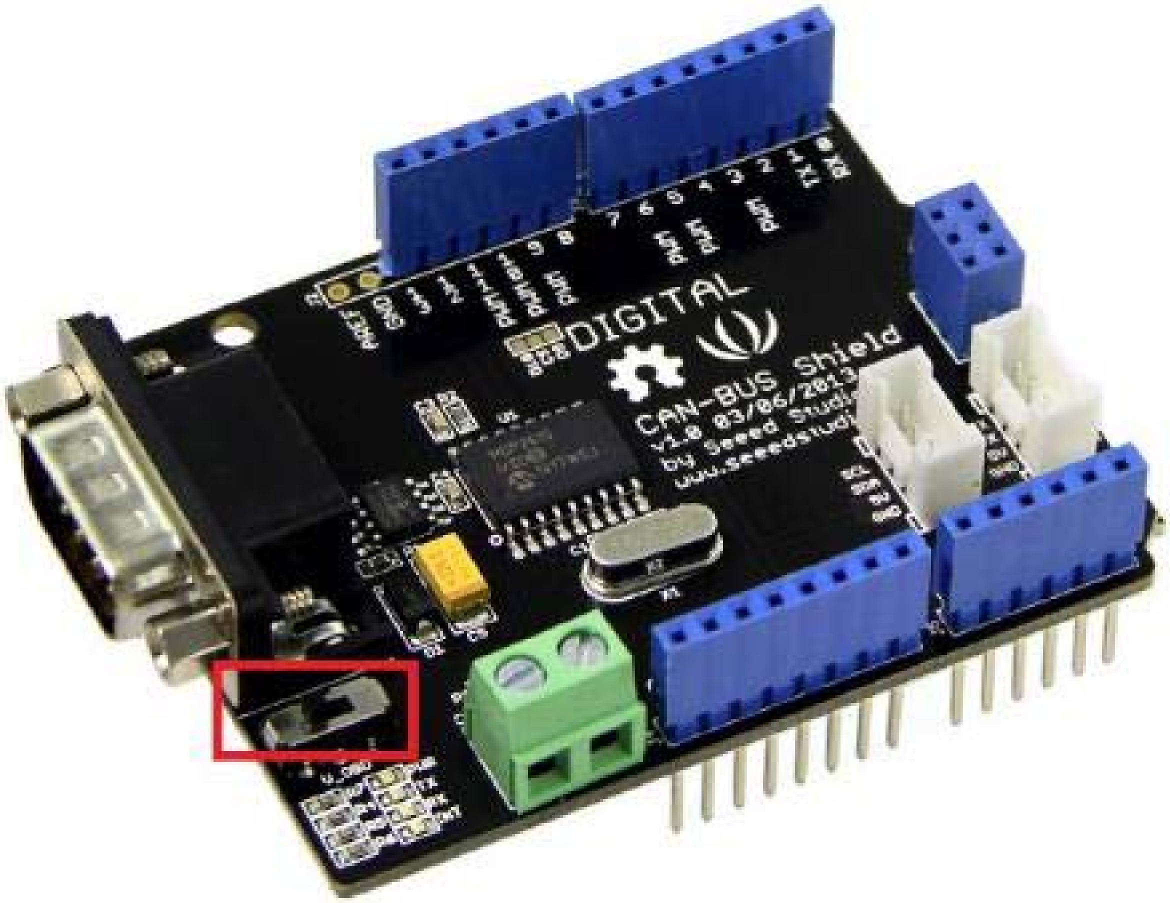
18 Seeed Studio* CAN Bus Shield

18.1 Use case

CAN bus, as a standard in the industrial field, features highly reliable long-distance transmissions with a medium communication speed. Many industrial sensors that implement the CAN bus protocol are becoming cheap and familiar in the DIY and makers markets, and it is the standard for the automotive diagnostic through the OBD-II port. Many of the new microcontrollers are integrating the CAN bus peripheral onboard. For example, the Arduino* Uno board. The Arduino Due board has it inside the SAM3x processor.

Key Info	Links
Product	*http://www.SeeedStudio.com/depot/CANBUS-Shield-p-1240.html* *http://www.SeeedStudio.com/wiki/CAN-BUS_Shield*
Library	*http://www.SeeedStudio.com/wiki/images/5/55/CAN_BUS_Shield.zip dated 8/13/12*
Guide	*http://upverter.com/Seeed Studio/c71b8e78aef6dce5/CAN-BUS-Shield--v10/*

Figure 53 **Seeed Studio* CAN bus shield**



18.2 Hardware summary

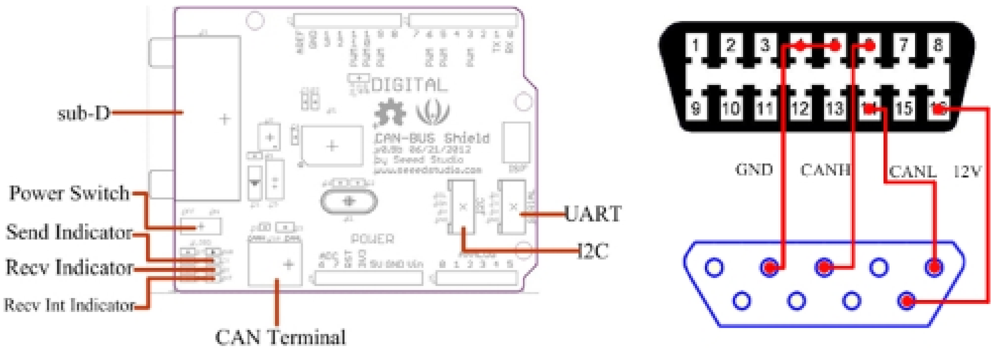
Key Info	Description/Links
Operating voltage	Designed for 5 V
IOREF	3.3 or 5 V (Intel® Galileo first generation board only operates at 5 V setting. See also *Section 0.8 IOREF voltage*.)
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8
Schematics	Unable to find v1.0 schematics *http://www.SeeedStudio.com/wiki/images/7/78/CAN-BUS_Shield_v0.9b.pdf*
Shield version	v1.0, 03/06/2013
CAN bus controller w/SPI	MCP2515 *http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010406*
CAN bus transceiver	MCP2551 *http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010405*

V-ODB switch (Power Switch): Default is OFF; the shield has a sub-D connector where power can be provided using the Vin pin and setting the V-ODB switch to the ON position.

The CANH and CANL lines are accessible through the 2-pin screw terminals (CAN Terminal).

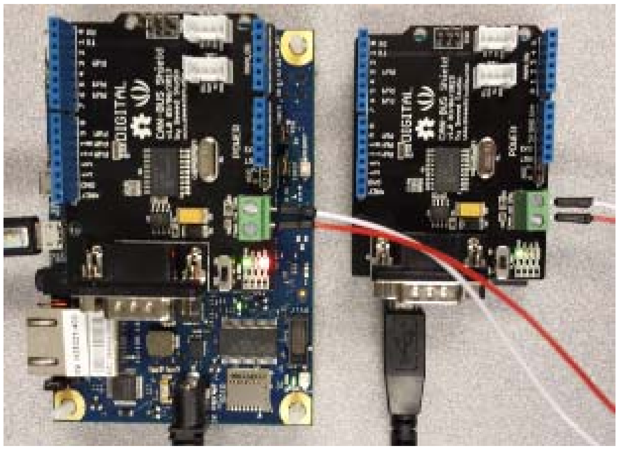
There are four LEDs: one indicating the presence of the power supply (PWR); two for the communication (Rx and Tx); the fourth LED is connected to the interrupt pin and indicates data is being received by the shield.

Figure 54 **Seeed Studio* CAN bus shield layout**



Pin Name	Function
13 (SCK)	SPI SCK
12 (MISO)	SPI MISO
11 (MOSI)	SPI MOSI
10 (CS) default	Default CS pin used by the library.
9 (CS - optional)	Selectable CS pin through an add-on jumper (instead of pin 10).
3 (INT)	Interrupt

Figure 55 **Seeed Studio* CAN Bus Shield shown here on an Intel® Galileo first generation board and and Arduino* Uno board**



18.3 Companion library

The library uses the Arduino* Core SPI library. It compiled without modification on both the Arduino* Uno board and the Arduino* 101 board.

18.4 Compile and upload

The example sketches “send” and “receive” included in the “mcp_can” library were used to test.

Arduino* Uno-to-Arduino Uno Test:

Two Arduino Uno boards communicating with each other via CAN-BUS. One board with the send sketch loaded, and the other with the receive sketch loaded.

Everything works as expected. The Tx and Rx LEDs on the “send shield” flash repeatedly, and the Rx and INT LEDs on the “receive shield” flash repeatedly. The following data displays over and over on the receiver’s serial monitor:

```
CAN_BUS GET DATA!
data len = 8
0       1         2       3         4                   567
```

Arduino* 101 “send” to Arduino Uno “receive” Test:

One Arduino 101 and one Arduino Uno board communicating with each other via CAN bus. The Arduino 101 board had the ‘send’ sketch loaded and the Arduino Uno board had the receive sketch loaded.

The data shown above is displayed twice on the Arduino Uno board’s serial monitor and then nothing else appears. The Tx and Rx LEDs on the Arduino 101 “send shield” flash repeatedly. The Rx LED on the Arduino Uno board “receive shield” flashes repeatedly, but the INT LED stays a constant red. It appears that one or two interrupt signals are generated by the MCP2515 in this configuration. An interrupt signal is generated for every packet when the devices are communicating correctly.

Arduino 101 “receive” to Arduino Uno “send” Test:

One Arduino 101 and one Arduino Uno board communicating with each other via CAN bus. The Arduino 101 board had the receive sketch loaded and the Arduino* Uno board had the send sketch loaded. Setup messages appear on the Arduino 101 serial monitor and then nothing else. The Tx and Rx LEDs on the Arduino Uno board “send shield” flash repeatedly. The Rx LED on the Arduino 101 “receive shield” flashes repeatedly, but the INT LED stays a constant red. It appears that no interrupt signals are generated by the MCP2515 in this configuration. An interrupt signal is generated for every packet when the devices are communicating correctly.

The MCP2515_ISR() function sets a flag which is later used in a conditional in the loop to print received data. This function is called by the attachInterrupt() function in setup. In the Arduino 101 SPI library, attachInterrupt is rewritten with the following line:

```
trace_error("SPI slave mode is not currently supported\n");
```

Because the flag is not set, data received is not printed. The loop rewritten as follows allows the received data to be printed.

Example 23 **Change receive.ino as follows**

Before

```


void loop()
{
    if(Flag_Recv)                      // check
if get data
    {
      Flag_Recv = 0;                   // clear
flag
      CAN.readMsgBuf(&len, buf);    // read
data,  len: data length, buf: data buf
      Serial.println("CAN_BUS GET DATA!");
      Serial.print("data len =
");Serial.println(len);
      for(int i = 0; i<len; i++)    // print
the data
      {

Serial.print(buf[i]);Serial.print("\t");
      }
      Serial.println();
    }
}
```

After

```
int n = 0;

void loop()
{
    //if(Flag_Recv) //Flag conditional
commented out
    {
      Flag_Recv = 0;                   // clear
flag
      CAN.readMsgBuf(&len, buf);    // read
data,  len: data length, buf: data buf
      Serial.println("CAN_BUS GET DATA!");
      Serial.print("data len =
");Serial.println(len);
      for(int i = 0; i<len; i++)    // print
the data
      {

Serial.print(buf[i]);Serial.print("\t");
      }
      Serial.println(n++);
    }
}
```

18.5 Results

Arduino* 101 compatible.

§

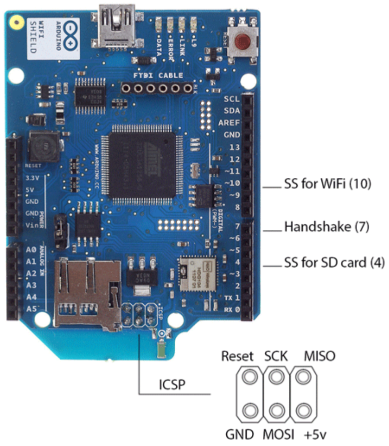
19 Arduino* Wi-Fi Shield

19.1 Use case

This test goal is to determine if it is easier to plug a shield instead of following multiple steps to get the miniPCI Wi-Fi working. The Intel® Galileo board has a miniPCI slot where a Wi-Fi module can be placed. This method of Wi-Fi will give you great performance, like on a normal computer; however, the Intel Galileo board needs to boot the Linux* system from the SD card. This might not be good for a beginner.

Key Info	Links
URL	*http://arduino.cc/en/Main/ArduinoWiFiShield*
Library	None. A Wi-Fi library exists in the existing IDE.
Guide	*http://arduino.cc/en/Guide/ArduinoWiFiShield*

Figure 56 **Arduino* Wi-Fi Shield**



19.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V
IOREF	Used, the shield can operate either at 3.3 and 5 V. (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Power	5 V or VIN, automatically selected.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*http://arduino.cc/en/uploads/Main/arduino-wifi-shield-reference-design.zip*
Shield firmware	The MicroUSB is used for updating the Atmel* AVR32UC3.
Microcontroller	AVR32UC3 *http://pdf1.alldatasheet.com/datasheet-pdf/view/392560/ATMEL/AVR32UC.html*
Wi-Fi 802.11b	HDG204, *http://pub.ucpros.com/download/1451_hdg204_datasheet_pa4.pdf?osCsid=mcrh728ovgeg6ub4ka6mccrso5*

The AVR32UC3 and the HDG204 both work with 3.3 V that is generated by the onboard voltage regulator.

The pins involved in communication with the controller board are protected by level shifters.

The handshake signal on pin digital 7 tells to the controller board when the shield is ready to communicate. It is not implemented with hardware interrupts but rather by polling the shield pin. The latency on the GPIO could affect the performance of the other code in the sketch.

Pin Name	Function (no wires required)
D4	Slave select for SD card
D7	Handshake pin. Tells when the shield is ready for communication.
D10	Slave select for Wi-Fi.
ICSP header	SPI interface through 6-pin ICSP header. SS (slave connect)
Jumper	Leave unconnected. This is used for shield firmware update.

19.3 Compile and upload

This shield works with no problems. See *Overview Section* for information on Wi-Fi and SD cards on a shield.

19.4 Results

Arduino* 101 compatible

§

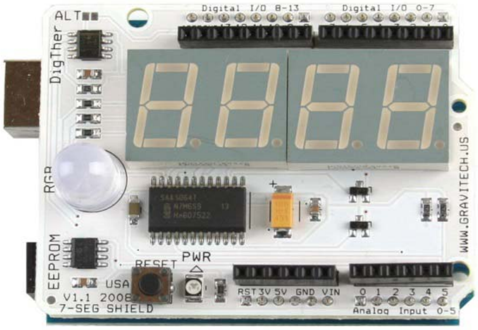
20 Gravitech* 7-Segment Shield

20.1 Use case

This shield is ideal for Intel® Galileo and Arduino* Uno boards experimenting, temperature monitoring, etc. It is ideal for I^{2}C basic electronic practice because the example code utilizes no libraries. There are also three I^{2}C devices on the board. Unfortunately, the shield is not stackable and may require some lead clipping to place a shield underneath.

Key Info	Links
Product	*http://store.gravitech.us/7segmentshield.html*
Library/example	No library required Example 1, RGB Example (see code in the Guide linked below) Example 2, Segment Display and Temperature *http://site.gravitech.us/Arduino/SHIELD7/SEG7_SHIELD.zip* This code required modification.
Guide	*http://tronixstuff.com/2011/08/24/review-gravitech-7-segment-arduino-shield*

Figure 57 **Gravitech* 7-segment shield**



Features of the board include:

- I^{2}C four-digit, 7-segment driver (10 mm diameter)
- I^{2}C temperature sensor
- I^{2}C EEPROM
- PWM RGB LED
- Blue PWR LED
- Reset
- button
- All pins breakout

20.2 Hardware summary

Pin Name	Function
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.7
Use VIN as power source	Not present.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Power	5 V only
Schematics	*http://site.gravitech.us/Arduino/SHIELD7/7SEG-SHIELD_schematic_v1_1.pdf*

The shield mounts a digital temperature sensor (TMP75), an I^{2}C EEPROM (24LC128), 4-digit display I^{2}C controller, and RGB LED. The shield does not follow the pinout of the latest Arduino* Uno board (version 1.0), so it does not have the IOREF and the dedicated pins for the I^{2}C interface. Mechanically, this is not a stackable shield. On the shield, there are 4.7 k pull-up resistors for the I^{2}C bus. Although the temperature sensor is functional, the heat of the shield and the Intel® Galileo board interferes with the temperature readings.

Pin Name	I^{2}C Function (No wires required)
A4	I^{2}C, SDA (Serial Data Line) Data
A5	I^{2}C, SCL (Serial Clock Line) Clock
Device Address 1	I^{2}C, defines 7-SEG address as: #define _7SEG (0x38)
Device Address 2	I^{2}C, defines digital thermometer as: *#define THERM (0x49)*
Device Address 3	I^{2}C, defines EEPROM address as: *#define EEP (0x50)*

Pin Name	Function
D3	PWM RED led
D5	PWM GREEN led
D6	PWM BLUE led
D2	TMP75 ALT pin - optional, to enable a solder jumper must be closed

20.3 Compile and upload

The shield does not come with a companion library, but two example sketches exist. The first example simply varies the brightness of the RBG LED using PWM. The second example allows you to read the temperature of the sensor and write it on the 7-segment display. Additionally the RGB LED changes its color depending on the temperature.

The second example sketch itself is outdated and requires modification. All ‘send’ commands were replaced with the ‘write’ command, and all ‘receive’ commands were replaced with the ‘read’ command.

Example 24 **Companion sketch modified as noted above**

```
/***************************************************************************

                     Copyright 2008 Gravitech
                        All Rights Reserved

***************************************************************************/

/***************************************************************************
 File Name: I2C_7SEG_Temperature.pde

 Hardware: Arduino Diecimila with 7-SEG Shield

 Description:
   This program reads I2C data from digital thermometer and display it on 7-Segment

 Change History:
   03 February 2008, Gravitech - Created

***************************************************************************/
```

```
#include <Wire.h>

#define BAUD (9600)    /* Serial baud define */
#define _7SEG (0x38)   /* I2C address for 7-Segment */
#define THERM (0x49)   /* I2C address for digital thermometer */
#define EEP (0x50)     /* I2C address for EEPROM */
#define RED (3)        /* Red   color pin of RGB LED */
#define GREEN (5)      /* Green color pin of RGB LED */
#define BLUE (6)       /* Blue  color pin of RGB LED */

#define COLD (23)      /* Cold temperature, drive blue LED (23c) */
#define HOT (26)       /* Hot temperature, drive red LED (27c) */

const byte NumberLookup[16] =   {0x3F,0x06,0x5B,0x4F,0x66,
                                 0x6D,0x7D,0x07,0x7F,0x6F,
                                 0x77,0x7C,0x39,0x5E,0x79,0x71};

/* Function prototypes */
void Cal_temp (int&, byte&, byte&, bool&);
void Dis_7SEG (int, byte, byte, bool);
void Send7SEG (byte, byte);
void SerialMonitorPrint (byte, int, bool);
void UpdateRGB (byte);

/***************************************************************************
 Function Name: setup

 Purpose:
   Initialize hardware.
****************************************************************************/

void setup()
{
  Serial.begin(BAUD);
  Wire.begin();        /* Join I2C bus */
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
  delay(500);          /* Allow system to stabilize */
}

/***************************************************************************
 Function Name: loop

 Purpose:
   Run-time forever loop.
****************************************************************************/

void loop()
{
  int Decimal;
  byte Temperature_H, Temperature_L, counter, counter2;
  bool IsPositive;

  /* Configure 7-Segment to 12mA segment output current, Dynamic mode,
     and Digits 1, 2, 3 AND 4 are NOT blanked */

  Wire.beginTransmission(_7SEG);
  Wire.write(0);
  Wire.write(B01000111);
  Wire.endTransmission();
  /* Setup configuration register 12-bit */

  Wire.beginTransmission(THERM);
  Wire.write(1);
  Wire.write(B01100000);
  Wire.endTransmission();

  /* Setup Digital THERMometer pointer register to 0 */

  Wire.beginTransmission(THERM);
  Wire.write(0);
  Wire.endTransmission();

  /* Test 7-Segment */
  for (counter=0; counter<8; counter++)
  {
    Wire.beginTransmission(_7SEG);
    Wire.write(1);
    for (counter2=0; counter2<4; counter2++)
    {
      Wire.write(1<<counter);
    }
    Wire.endTransmission();
    delay (250);
  }

  while (1)
  {
    Wire.requestFrom(THERM, 2);
    Temperature_H = Wire.read();
    Temperature_L = Wire.read();

    /* Calculate temperature */
    Cal_temp (Decimal, Temperature_H, Temperature_L, IsPositive);

    /* Display temperature on the serial monitor.
       Comment out this line if you don't use serial monitor.*/
    SerialMonitorPrint (Temperature_H, Decimal, IsPositive);

    /* Update RGB LED.*/
    UpdateRGB (Temperature_H);

    /* Display temperature on the 7-Segment */
    Dis_7SEG (Decimal, Temperature_H, Temperature_L, IsPositive);

    delay (1000);        /* Take temperature read every 1 second */
  }
}

/***************************************************************************
 Function Name: Cal_temp

 Purpose:
   Calculate temperature from raw data.
****************************************************************************/
void Cal_temp (int& Decimal, byte& High, byte& Low, bool& sign)
{
  if ((High&B10000000)==0x80)    /* Check for negative temperature. */
    sign = 0;
  else
    sign = 1;
  High = High & B01111111;      /* Remove sign bit */
  Low = Low & B11110000;        /* Remove last 4 bits */
  Low = Low >> 4;
  Decimal = Low;
  Decimal = Decimal * 625;      /* Each bit = 0.0625 degree C */

  if (sign == 0)                /* if temperature is negative */
  {
    High = High ^ B01111111;    /* Complement all of the bits, except the MSB */
    Decimal = Decimal ^ 0xFF;   /* Complement all of the bits */
  }
}

/***************************************************************************
 Function Name: Dis_7SEG

 Purpose:
   Display number on the 7-segment display.
****************************************************************************/
void Dis_7SEG (int Decimal, byte High, byte Low, bool sign)
{
  byte Digit = 4;                 /* Number of 7-Segment digit */
  byte Number;                    /* Temporary variable hold the number to display */

  if (sign == 0)                  /* When the temperature is negative */
  {
    Send7SEG(Digit,0x40);         /* Display "-" sign */
    Digit--;                      /* Decrement number of digit */
  }

  if (High > 99)                  /* When the temperature is three digits long */
  {
    Number = High / 100;          /* Get the hundredth digit */
    Send7SEG (Digit,NumberLookup[Number]);     /* Display on the 7-Segment */
    High = High % 100;            /* Remove the hundredth digit from the TempHi */
    Digit--;                      /* Subtract 1 digit */
  }

  if (High > 9)
  {
    Number = High / 10;           /* Get the tenth digit */
    Send7SEG (Digit,NumberLookup[Number]);     /* Display on the 7-Segment */
    High = High % 10;            /* Remove the tenth digit from the TempHi */
    Digit--;                      /* Subtract 1 digit */
  }

  Number = High;                  /* Display the last digit */
  Number = NumberLookup [Number];
  if (Digit > 1)                  /* Display "." if it is not the last digit on 7-SEG */
  {
    Number = Number | B10000000;
  }
  Send7SEG (Digit,Number);
  Digit--;                        /* Subtract 1 digit */

  if (Digit > 0)                  /* Display decimal point if there is more space on 7-SEG */
  {
    Number = Decimal / 1000;
    Send7SEG (Digit,NumberLookup[Number]);
    Digit--;
  }
  if (Digit > 0)                 /* Display "c" if there is more space on 7-SEG */
  {
    Send7SEG (Digit,0x58);
    Digit--;
  }

  if (Digit > 0)                 /* Clear the rest of the digit */
  {
    Send7SEG (Digit,0x00);
  }
}

/***************************************************************************
 Function Name: Send7SEG

 Purpose:
   Send I2C commands to drive 7-segment display.
****************************************************************************/

void Send7SEG (byte Digit, byte Number)
{
  Wire.beginTransmission(_7SEG);
  Wire.write(Digit);
  Wire.write(Number);
  Wire.endTransmission();
}

/***************************************************************************
 Function Name: UpdateRGB

 Purpose:
   Update RGB LED according to define HOT and COLD temperature.
****************************************************************************/

void UpdateRGB (byte Temperature_H)
{
  digitalWrite(RED, LOW);
  digitalWrite(GREEN, LOW);
  digitalWrite(BLUE, LOW);        /* Turn off all LEDs. */

  if (Temperature_H <= COLD)
  {
    digitalWrite(BLUE, HIGH);
  }
  else if (Temperature_H >= HOT)
  {
    digitalWrite(RED, HIGH);
  }
  else
  {
    digitalWrite(GREEN, HIGH);
  }
}

/***************************************************************************
 Function Name: SerialMonitorPrint

 Purpose:
   Print current read temperature to the serial monitor.
****************************************************************************/
void SerialMonitorPrint (byte Temperature_H, int Decimal, bool IsPositive)
```

```
{
    Serial.print("The temperature is ");
    if (!IsPositive)
    {
      Serial.print("-");
    }
    Serial.print(Temperature_H, DEC);
    Serial.print(".");
    Serial.print(Decimal, DEC);
    Serial.print(" degree C");
    Serial.print("\n\n");
}
```

20.4 Results

Arduino* 101 compatible.

§

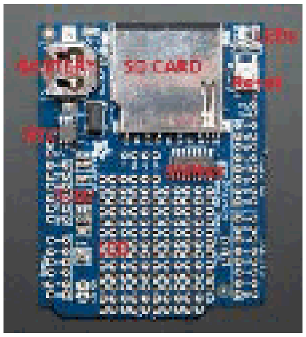
21 Adafruit* Data Logging Shield for Arduino*

21.1 Use case

This shield makes it easy to save data to files on a FAT16/FAT32 SD card. The data can be read later for analysis. There exists 3.3 V level shifter circuitry that prevents damage to the SD card. This shield includes a real-time clock (RTC) that can timestamp the data with the current time. A coin-cell battery is included so that the time continues to run even if the Arduino* Uno or Arduino* 101 board is unplugged.

Key Info	Links
Order/product info	*https://www.adafruit.com/product/1141*
Guide	*https://learn.adafruit.com/adafruit-data-logger-shield*
RTC library	*https://github.com/adafruit/RTClib/archive/master.zip* Dated GitHub 02/26/16
SD library	None.

Figure 58 **Adafruit* data logging shield for Arduino***



21.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
IDE	1.5.8-Intel.1.0.51..6,8
Schematics	N/A
RTC	DS1307, *http://datasheets.maximintegrated.com/en/ds/DS1307.pdf* ***Note:*** If the battery is not present, strange behavior will occur and a possibility of hanging the Arduino* Uno board.
Battery	Coin Cell, CR1220. ***Note:*** The battery must be present in the shield at all times (even if it is dead).

The following pins were used for the RTC clock. These defaults were used.

Pin Name	I^{2}C Function (no wires required)
A4	I^{2}C, SDA (Serial Data Line) Data
A5	I^{2}C, SCL (Serial Clock Line) Clock
Device Address	I^{2}C, RTClib.cpp defines the address as: `#define DS1307_ADDRESS 0x68`

The pins used for the SD card are as follows for the Arduino* Uno board.

Pin Name	SPI Function (No wires required)
D11	SPI, MOSI (Master Out Slave In)
D12	SPI, MISO (Master In Slave Out)
D13	SPI, Clock (CLK)
CS	SPI, Chip Select (CS), The sketch defines as: **SD.begin(10)**
D10	Must be left as an output (even if it is not used) in order for the SD to work on the Arduino* Uno board.

21.3 Companion library

The RTC library contains the example sketches (DateCal, DS1307, SoftRTC). See the example notes for details on what the sketch is doing. All programs may require changing some or all of the following:

- The **serial port** baud rate to 9600. This is the IDE default.
- Adding a **delay** right after the serial initialization. This gives the user time to open up the serial terminal to see the output.

The outputs for a few selected sketches are:

Example 25 **Sketch output: DateCalc**

```
dt0 2000/1/1 0:0:0 = 946684800s / 10957d since 1970
dt1 2001/1/1 0:0:0 = 978307200s / 11323d since 1970
dt2 2009/1/1 0:0:0 = 1230768000s / 14245d since 1970
dt3 2009/1/2 0:0:0 = 1230854400s / 14246d since 1970
dt4 2009/1/27 0:0:0 = 1233014400s / 14271d since 1970
dt5 2009/2/27 0:0:0 = 1235692800s / 14302d since 1970
dt6 2009/12/27 0:0:0 = 1261872000s / 14605d since 1970
dt7 2009/12/27 1:0:0 = 1261875600s / 14605d since 1970
dt8 2009/12/28 0:0:0 = 1261958400s / 14606d since 1970
dt9 2010/1/3 0:0:0 = 1262476800s / 14612d since 1970
```

The time used for the sketch is the timestamp on the sketch itself. There is a minor difference between DS1307 and SoftRTC. The primary difference is highlighted.

Example 26 **Sketch output: DS1307**

```
2014/5/13 13:44:0
 since midnight 1/1/1970 = 1399988640s = 16203d
 now + 7d + 30s: 2014/5/20 13:44:30

2014/5/13 13:44:3
 since midnight 1/1/1970 = 1399988643s = 16203d
 now + 7d + 30s: 2014/5/20 13:44:33

2014/5/13 13:44:6
 since midnight 1/1/1970 = 1399988646s = 16203d
     now + 7d + 30s: 2014/5/20 13:44:36
```

Example 27 **Sketch output: SoftRTC**

```
2014/5/13 13:45:40
 seconds since 1970: 1399988740
 now + 7d + 30s: 2014/5/20 13:46:10

2014/5/13 13:45:43
 seconds since 1970: 1399988743
 now + 7d + 30s: 2014/5/20 13:46:13

2014/5/13 13:45:46
 seconds since 1970: 1399988746
           now + 7d + 30s: 2014/5/20 13:46:16
```

21.4 Compile and upload

Real-time clock test: On the Arduino* Uno board, all examples in the RTC library worked. See examples in the previous section.

SD card test: There is no special library for the SD card (they are included with the IDE). The standard SD Library includes the following examples:

- *CardInfo (Arduino Only).* Change the ‘chipSelect’ from 4 to 10
- Datalogger
- DumpFile
- Files
- Listfiles
- ReadWrite

The output below comes from a couple of examples.

Example 28 **Sketch output: files**

```
Initializing SD card...initialization done.
example.txt doesn't exist.
Creating example.txt...
example.txt exists.
Removing example.txt...
        example.txt does not exist.
Sketch Output: ListFiles
Initializing SD card...initialization done.
boot/
        grub/
                 grub.conf        801
bzImage          2113856
core-image-minimal-initramfs-clanton.cpio.gz1441609
image-full-clanton.ext3314572800
done!
```

21.5 Results

- *the* RTC and SD are compatible with Arduino* 101

§

22 LinkSprite* 3G + GPS Shield for Arduino**

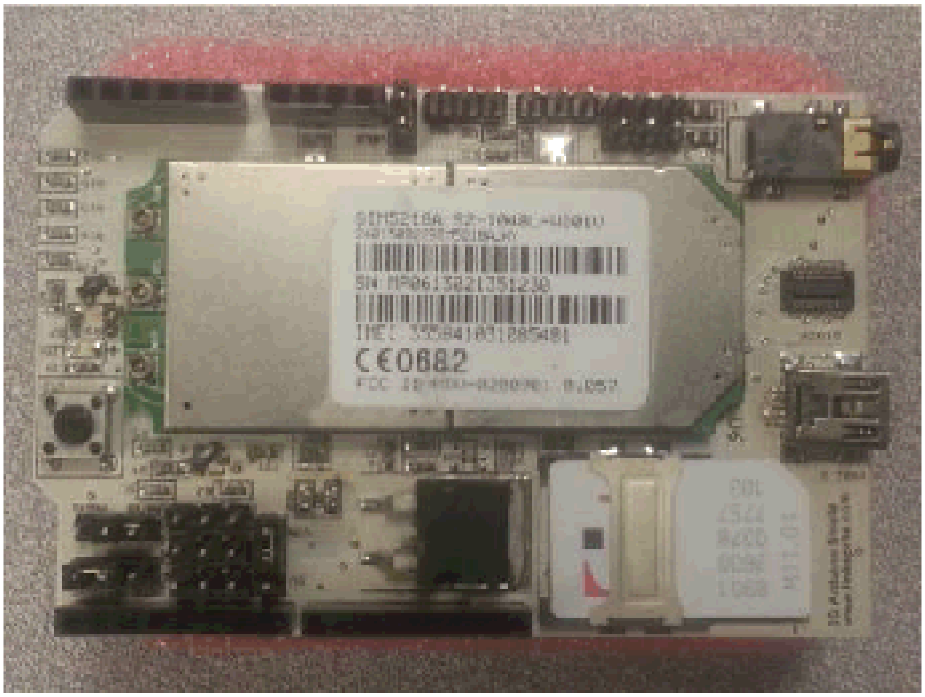
22.1 Use case

This shield enables connectivity to high-speed WCDMA and HSPA cellular networks, enabling the creation of the next level of the “Internet of Things”. The module has an internal GPS combined with standard NMEA frames with mobile cell ID triangulation using both assisted mobile (A-GPS) and mobile-based (S-GPS) modes so the device can be tracked indoors and outdoors. With the SD card socket, it is possible to handle complete FAT16 file systems and store up to 32 GB of information. The 3G module can work at full speed (~7.2 Mbps download, ~5.5 Mbps upload) when working with SD files directly without the need of the Arduino* 101 board for data or files management. The 3G module supports using AT commands.

Other interesting accessories that can be connected to the module are a video camera that enables the recording of video at 640 × 480 resolution, an audio kit including microphone, speaker, hands-free and headphone sets, and a SD socket to save all the data from the 3G network or recorded from the video camera. It is also possible for the shield to talk directly to web servers by HTTP/HTTPS, upload/download files directly by FTP/FTPS, and send/receive emails by POP3/SMTP.

Key Info	Links
Product info	*http://linksprite.com/wiki/index.php5?title=3G_%2B_GPS_Shield_for_Arduino*
Library	No library needed.
AT commands	*http://en.wikipedia.org/wiki/AT_command_set*

Figure 59 **3G + GPS Shield for Arduino***



22.2 Hardware summary

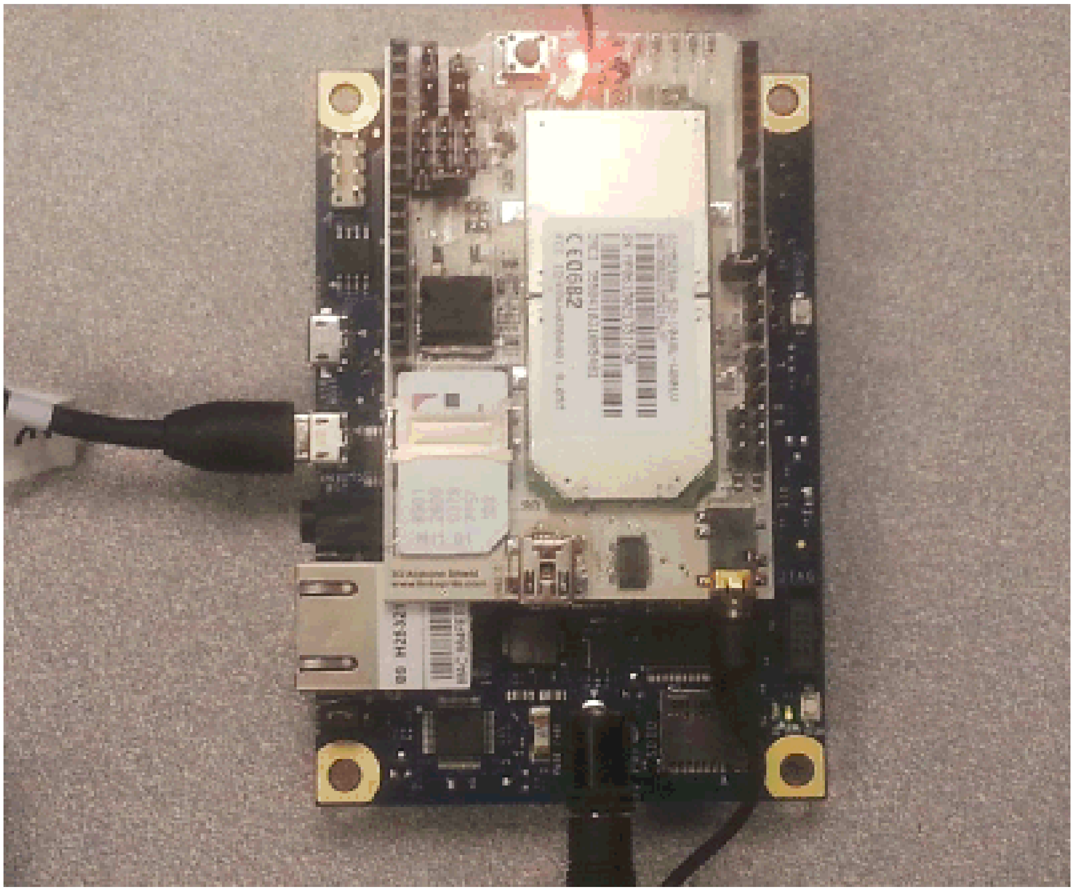
Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	No.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Audio interface	2-in-1 *TTRS Headset*
SIM	Mini *SIM card* interface.
GSM module	SIMCOM* SIM5218 – quad band GSM/GPRS/EDGE and UMTS engine.
Antenna	Connection port for external antenna.
IDE	1.6.8
Schematics	*http://www.cooking-hacks.com/index.php/documentation/tutorials/arduino-3g-gprs-gsm-gps*

22.3 Companion library

No library required.

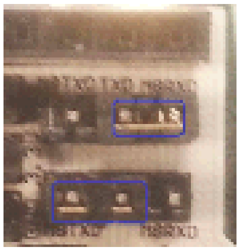
22.4 Compile and upload

Figure 60 **3G + GPS Shield for Arduino** on an Intel® Galileo first generation board**

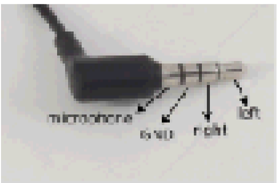


To test the GSM capability, do the following:

1. Insert unlocked SIM (mini SIM) card.
2. Connect cellular antenna with connection to the main antenna port.
3. Use two jumpers to set the serial settings as shown.



4. Connect a 2-in-1 stereo headset to the headset connector.
5. Attach the shield to the Arduino* 101.
6. Power up the Arduino* 101.
7. Connect Arduino* 101 USB to the computer.



8. Power up the shield by pressing the power button on the shield for two seconds.
9. Update the following sketch to input a valid telephone number into the following lines of code. For this example, this is a US call using a 1 followed by a fictitious area code 555 and phone number 5555555. Translated format is 1-555-555-5555

```
char* gPhoneNumber = "15555555555";
```

10. Upload the following sketch that will enable sending text messages and making calls.
11. Open the serial terminal from the IDE.
12. From the serial terminal, input one of the following:
    - **a:** Answer a voice call. (The ring should appear in the serial terminal and the ring tone is played through the headset if connected.)
    - **d:** Dial a voice call. (Phone number already set in the sketch.)
    - **g:** Set up an IP session.
    - **r:** Display all received text messages.
    - **t:** Send a text message. (Text and phone number already set in the sketch.)

Example 29 **Test voice calls in/out, text in/out, IP session**

```
char  data[256];
char* gPhoneNumber = "15555555555"; // 15555555555
char* gTextLine1   = "G2>";
char* gTextLine2   = "Hello from Shield 22";

void setup()
{
  Serial.begin(115200);
  Serial1.begin(115200);
  delay(1000);
  Serial1.println("AT+CSDVC=2");
  delay(1000);
  Serial1.println("AT+CSDVC?");
  delay(1000);

  waitForUser(9);
  Serial.println("...Powerup should complete before countdown completes");
  Serial.println("t=text, r=receive text, d=dial, a=answer, g=setup IP session");
}
void loop()
{
    if (Serial.available())
       switch(Serial.read())
      {
       case 't':
          SendTextMessage();
          break;
       case 'r':
          ReceiveTextMessage();
          break;
       case 'd':
          DialVoiceCall();
          break;
       case 'a':
          AnswerVoiceCall();
          break;
       case 'h':
          HangupVoiceCall();
          break;
       case 'g':
          GPRS();
          break;
   }
  if (Serial1.available())
    Serial.write(Serial1.read());
}
void AnswerVoiceCall()
{
  Serial1.println("ATA");
}
void HangupVoiceCall()
{
  //Serial1.println("ATH1"); // H, hang up (1=hang up, 0=pickup)
  //delay(100);
  //Serial1.println();
}
void SendTextMessage()
{
  Serial1.print("AT+CMGF=1\r");    //We want to send the SMS in text mode
  delay(1000);
  Serial1.print("AT+CMGS=\"");
  Serial1.print(gPhoneNumber);
  Serial1.println("\"");
  delay(1000);

  Serial1.print(gTextLine1);
  Serial1.println(gTextLine2);
  delay(1000);

  Serial1.println((char)26);  //the ASCII code of the ctrl+z is 26 (0x1A)
  delay(1000);
  Serial1.println();
}
void ReceiveTextMessage()
{
  Serial1.println("AT+CMGF=1"); //We want to receive the SMS in text mode
  delay(1000);
  Serial1.println("AT+CPMS=\"SM\"");      // read first SMS
  delay(1000);
  Serial1.println("AT+CMGL=\"ALL\""); // show message
}
void DialVoiceCall()
{
  Serial1.print("ATD + ");//dial the number
  Serial1.print(gPhoneNumber);
  Serial1.println(";");
  delay(1000);
  Serial1.println();
}
void ShowSerialData()
{
  while(Serial1.available()!=0)
      Serial.write(Serial1.read());
}
void GPRS()
{
  Serial1.println("AT+CPIN?");    // Is SIM ready to use?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+CGREG?");    // Is device registered?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+COPS?");    // Does SIM info match network?
  delay(1000);
  ShowSerialData();

  Serial.println("Check signal quality");
  Serial1.println("AT+CSQ");    // Check signal quality
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+cgatt=1");    // GPRS attach
  delay(1000);
  ShowSerialData();

  // define a PDP context with IP connection, ID is 1
  Serial1.println("AT+CGDCONT=1,\"IP\",\"fast.t-mobile.com\"");
  delay(1000);
  ShowSerialData();
  // list PDP contexts that are defined
  Serial1.println("at+cgdcont?");
  delay(3000);
  ShowSerialData();

  // setup the session using the appropriate PDP context
  Serial1.println("AT+CGACT=1,1");
  delay(1000);
  ShowSerialData();

  Serial.println("session is setup delay 5 seconds");
  delay(5000);

  // deactivate the PDP context
  Serial1.println("AT+CGACT=0,1");
  delay(1000);
  ShowSerialData();

  // detach from GPRS newtork
  Serial1.println("AT+CGATT=0");
  delay(1000);
  ShowSerialData();
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
Serial.println("");
} // waitForUser
```

22.5 Results

Arduino* 101 not tested

§

23 LinkSprite* ATWIN Quadband GPRS/GSM Shield for Arduino*

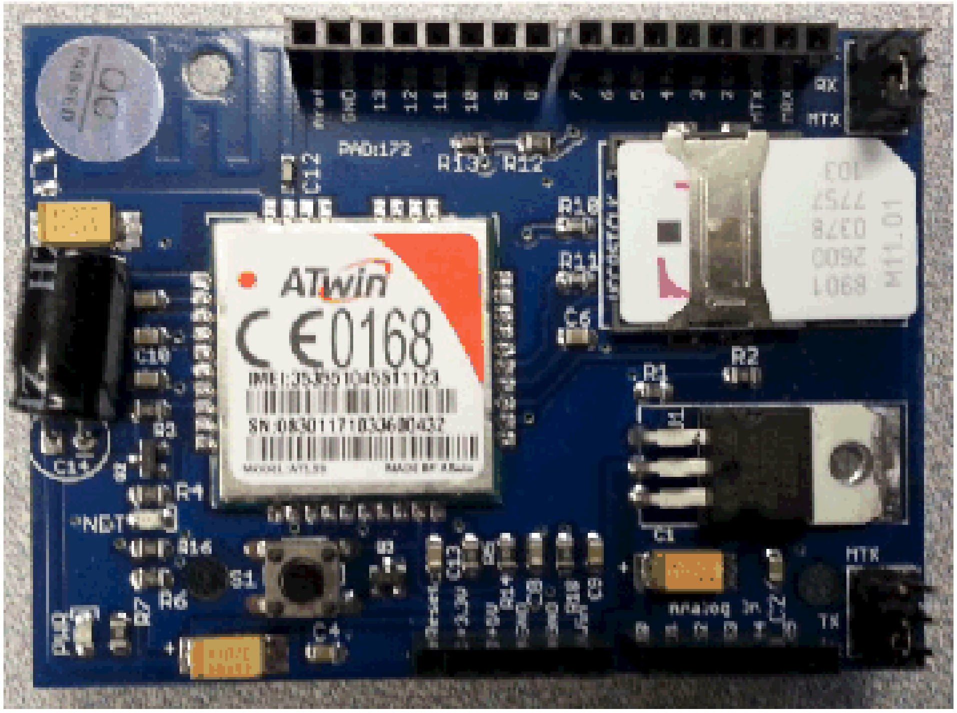
23.1 Use case

This shield makes it possible to hook the Arduino* 101 up to the GSM/GPRS cellular telephone network. It is possible to make and receive calls, or send and receive text messages using AT commands.

The shield has a PCB antenna, so there is no need to hook up an external antenna. This is a low power consumption quadband (AT139) and dual-band (AT139D) GSM/GPRS module. It can support voice, SMS, fax, and data applications.

Key Info	Links
Product info and wiki	*http://linksprite.com/wiki/index.php5?title=ATWIN_Quad-band_GPRS/GSM_Shield_for_Arduino*
Library	No library needed.
AT Command Set	*http://en.wikipedia.org/wiki/AT_command_set*

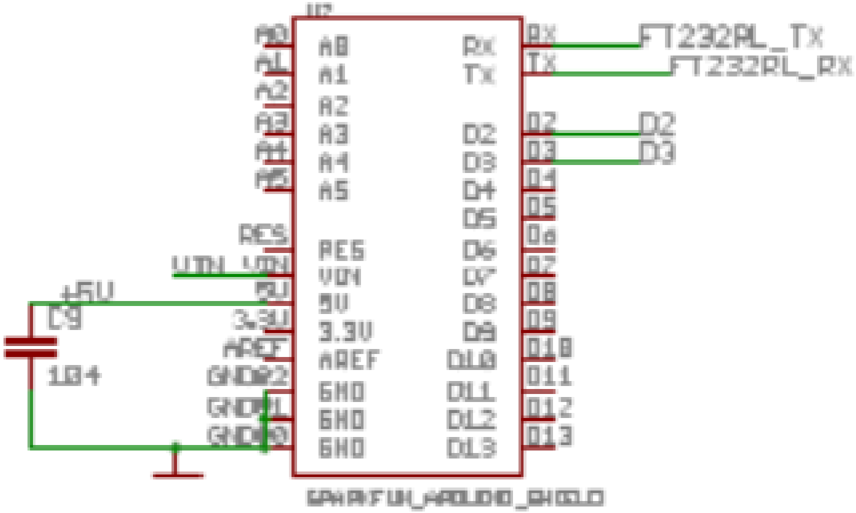
Figure 61 **LinkSprite* ATWIN Quadband GPRS/GSM Shield for Arduino****



23.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 or 5 V – jumper selectable.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	Yes
Audio Interface	2 channels input/output.
SIM	Mini *SIM card* interface.
GSM module	ATWin* AT139. *http://www.openhacks.com/uploadsproductos/at139_hardware_design_manual_v1.3.pdf*.
Antenna	Built-in PCB antenna.
IDE	1.6.8
Schematics	*https://s3.amazonaws.com/linksprite/Shields/ATWIN/schematic_of_ATWIN.rar*

Figure 62 **LinkSprite* ATWIN quadband GPRS/GSM shield layout**



Pin Name	Function
Rx of hardware serial port	Connected to D0 (Input from Arduino* 101)
Tx of hardware serial port	Connected to D1 (Output from Arduino* 101)
5 V	Connected to 5 V
GND	Connected to GND
D2	Soft serial – selecting jumper option that doesn't use this.
D3	Soft serial – selecting jumper option that doesn't use this.

23.3 Companion library

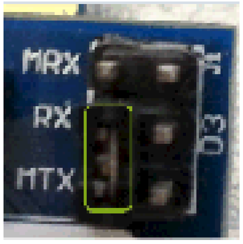
No library required.

23.4 Compile and upload

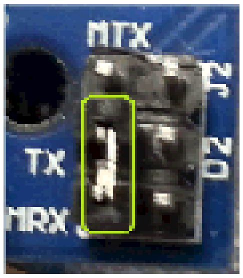
Do the following:

1. Insert an unlocked SIM card. The SIM is a mini-SIM or 2FF size.
2. Attach the shield to the Intel® Galileo board. There is no extra wiring necessary.

3. Set the Serial port select jumpers to the Hardware Serial position:
   a. Set J1 so that Rx is connected to MTx.



   b. Set J2 so that Tx is connected to MRx.



4. Power up the Arduino* 101 and connect to computer with the USB.
5. Update the following sketch to input a valid telephone number into the following lines of code. For this example, this is a US call using a 1 followed by a fictitious area code 555 and phone number 5555555.

```
char* gPhoneNumber = "15555555555";
```

6. Upload the following sketch that will enable sending text messages and making calls.
7. Open the serial terminal from the IDE and set rate to 9600.
8. From the serial terminal, input one of the following:
   - **a:** Answer a voice call. (The ring should appear in the serial terminal and the ring tone is played through the headset if connected.)
   - **d:** Dial a voice call. (Phone number already set in the sketch.)
   - **g:** Set up an IP session.
   - **r:** Display all received text messages.
   - **t:** Send a text message. (Text and phone number already set in the sketch.)

Example 30 **Issue AT commands over serial link**

```
char  data[256];
char* gPhoneNumber = "15555555555"; // 15555555555
char* gTextLine1   = "G2>";
char* gTextLine2   = "Hello from Shield 23";

void setup()
{
  Serial.begin(9600);    // the GPRS baud rate
  Serial1.begin(9600);
  waitForUser(9);
  Serial.println("t=text, r=receive text, d=dial, a=answer, h=hangup, g=set ip session");
}
void loop()
{
    if (Serial.available())
       switch(Serial.read())
      {
        case 't':
          SendTextMessage();
          break;
        case 'r':
          ReceiveTextMessage();
          break;
       case 'd':
          DialVoiceCall();
          break;
       case 'a':
          AnswerVoiceCall();
          break;
       case 'h':
          HangupVoiceCall();
          break;
       case 'g':
          GPRS();
          break;
   }
  if (Serial1.available())
    Serial.write(Serial1.read());
}
void AnswerVoiceCall()
{
  Serial1.println("ATA");
}
void HangupVoiceCall()
{
//  Serial1.println("ATH1"); // H, hang up (1=hang up, 1=pickup)
//  delay(100);
//  Serial1.println();
}
void SendTextMessage()
{
  Serial1.print("AT+CMGF=1\r");    //Because we want to send the SMS in text mode
  delay(1000);

  Serial1.print("AT+CMGS=\"");
  Serial1.print(gPhoneNumber);
  Serial1.println("\"");
  delay(1000);

  Serial1.print(gTextLine1);
  Serial1.println(gTextLine2);
  delay(1000);

  Serial1.println((char)26);  //the ASCII code of the ctrl+z is 26 (0x1A)
  delay(1000);
  Serial1.println();
}
void ReceiveTextMessage()
{
  Serial1.println("AT+CMGF=1");    //Because we want to receive the SMS in text mode
  delay(1000);
  Serial1.println("AT+CPMS=\"SM\"");      // read first SMS
  delay(1000);
  Serial1.println("AT+CMGL=\"ALL\""); // show message
}
void DialVoiceCall()
{
  Serial1.print("ATD + ");//dial the number
  Serial1.print(gPhoneNumber);
  Serial1.println(";");
  delay(100);
  Serial1.println();
}
void ShowSerialData()
{
  while(Serial1.available()!=0)
      Serial.write(Serial1.read());
}
void GPRS()
{
  Serial1.println("AT+CPIN?");    // Is SIM ready to use?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+CGREG?");    // Is device registered?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+COPS?");    // Does SIM info match network?
  delay(1000);
  ShowSerialData();

  Serial.println("Check signal quality");
  Serial1.println("AT+CSQ");    // Check signal quality
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+cgatt=1");    // GPRS attach
  delay(1000);
  ShowSerialData();

  // define a PDP context with IP connection, ID is 1
  Serial1.println("AT+CGDCONT=1,\"IP\",\"fast.t-mobile.com\"");
  delay(1000);
  ShowSerialData();

  // list PDP contexts that are defined
  Serial1.println("at+cgdcont?");
  delay(3000);
  ShowSerialData();

  // setup the session using the appropriate PDP context
  Serial1.println("AT+CGACT=1,1");
  delay(1000);
  ShowSerialData();

  Serial.println("session is setup delay 5 seconds");
  delay(5000);

  // deactivate the PDP context
  Serial1.println("AT+CGACT=0,1");
  delay(1000);
  ShowSerialData();

  // detach from GPRS newtork
  Serial1.println("AT+CGATT=0");
  delay(1000);
  ShowSerialData();
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
Serial.println("");
} // waitForUser
```

23.5 Results

Arduino* 101 not tested

§

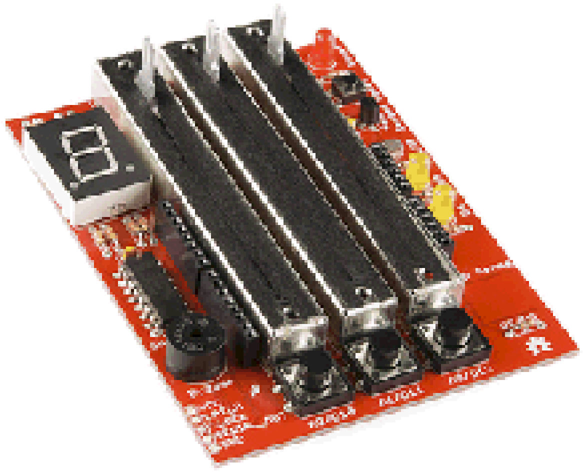
24 SparkFun* Danger Shield

24.1 Use case

The SparkFun Danger shield mounts on top of your board and equips it with a variety of fun and useful inputs and outputs.

Key Info	Links
URL	*https://www.sparkfun.com/products/11649*
Library	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/Danger_shield_V16_CapSense.zip* Contains “CapSense” and “Danger_shield” libraries.

Figure 63 **SparkFun* danger shield**



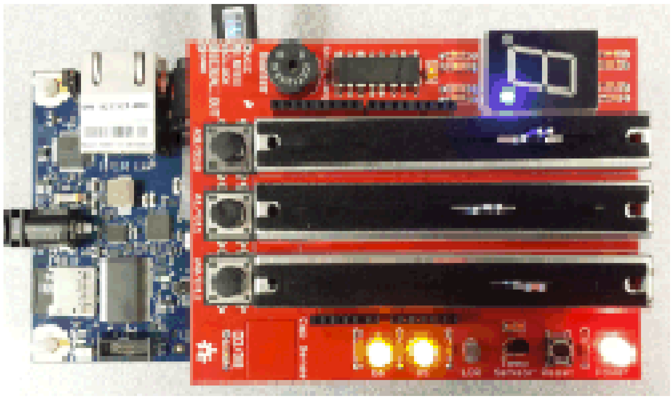
The shield’s features include:

- Three linear slide potentiometers connected to the Arduino* Uno board analog pins 0 through 2.
- Red and yellow LEDs connected to digital pins 5 and 6 - PWM pins - so you can easily vary their brightness.
- Three momentary push buttons connected up to the Arduino* Uno digital pins 10 through 12. Those lines will go low when the buttons are pressed.
- A photocell and a temperature sensor, both with analog outputs, are connected to the analog pins 3 and 4, respectively.
- An 8-bit shift register set up to control a blue 7-segment LED.
- A buzzer - so you can get sound.
- Capacitive touchpad - Using the CapSense library to sense touch.

24.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
IDE	1.6.8
74HC595N	8-bit shift register: *http://bildr.org/?s=74hc595*
TMP36	Temperature sensor.
CET-12A3.5	Piezo buzzer/speaker.
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/Danger_Shield-v17.pdf*

Figure 64 **SparkFun* Danger shield on an Intel® Galileo first generation board**



24.3 Compile and upload

Arduino* Uno board:

The 'Danger_shield' sketch was run on the Arduino* Uno board. This sketch tests all of the devices on the Danger Shield, one at a time. It is not clear how the capacitive touchpad is working, but this is what appears to be occurring:

- Download the sketch.
- Bring up the IDE serial port.
- Press button 1 (D10) to cycle through the different component tests. The test output is displayed on the serial terminal running at 9600 baud.
  - Press button 1, Adjust 'Sliders' bars.
  - Press button 1, Will sound buzzard, press again to turn off.
  - Press button 1, Capacitive test (not sure what this performs).
  - Press button 1, Displays the 'Temp' value is 153.
  - Press button 1, Display photocell 'Light' values.
  - Press button 1, Enable Button 2 and Button 3 tests that will turn off LED-D5 and LED-D6.
  - Press button 1, Enables the 'Seven segment display' test.
- Reset the board to restart the tests.

Arduino* 101:

The *Danger_shield* sketch was modified to function with the Arduino* 101. The example test below has all references to the capacitive test removed. The capacitive test is referencing AVR registers. A few delays were added (serial port startup), and delays were adjusted.

Example 31 **Modified *Danger_shield* sketch**

```
//#include <CapSense.h>
/*
 * Danger Shield Example Sketch
 * Copyright (c) 2010 SparkFun Electronics.  All right reserved.
 * Written by Chris Taylor
 *
 * This code was written to demonstrate the Danger Shield from SparkFun Electronics
 *
 * This code will test all of the devices on the Danger Shield one at a time.
 * Press button 1 (D10) to cycle through the different tests. View their output on
 * a terminal running at 9600 baud.
 *
 * http://www.sparkfun.com
 */

//#include "AdvancedIO.h"

// Shift register bit values to display 0-9 on the seven-segment display
const byte ledCharSet[10] = {


B00111111,B00000110,B01011011,B01001111,B01100110,B01101101,B01111101,B00000111,B01111111,B01
101111
};

// Global variables
int val = 0;
int state = 0;
int x = 0;
int i = 0;

// Pin definitions
#define SLIDER1  0
#define SLIDER2  1
#define SLIDER3  2

#define KNOCK    5

#define BUTTON1  10
#define BUTTON2  11
#define BUTTON3  12

#define LED1  5
#define LED2  6

#define BUZZER   3

#define TEMP  4

#define LIGHT  3

#define LATCH 7
#define CLOCK 8
#define DATA 4

// State machine values
#define SLIDER_TEST 1
#define BUZZER_TEST 2
#define CAPSENSE_TEST  3
#define TEMP_TEST  4
#define LIGHT_TEST 5
```

```
#define BUTTON_TEST 6
#define SEVENSEG_TEST 7

//CapSense   cs_9_2 = CapSense(9,2);   //Initializes CapSense pins

void setup()
{
  Serial.begin(9600);
  for(i=5; i>0; i--)
  {
  delay(1000*1);
  Serial.println(i);
  }

  // cs_9_2.set_CS_AutocaL_Millis(0xFFFFFFFF); // Calibrates CapSense pin timing

  pinMode(BUTTON1,INPUT);
  pinMode(BUTTON2,INPUT);
  pinMode(BUTTON3,INPUT);

  digitalWrite(BUTTON1,HIGH);
  digitalWrite(BUTTON2,HIGH);
  digitalWrite(BUTTON3,HIGH);


  pinMode(BUZZER, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  digitalWrite(LED1,HIGH);
  digitalWrite(LED2,HIGH);
  pinMode(LATCH, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(DATA,OUTPUT);

  Serial.println("Danger Shield Component Test");
  Serial.println("Press Button 1 to begin.");
}
void loop()
{
  if(!(digitalRead(BUTTON1))) // Change state
  {
    delay(1); // Debounce
    state++;
    if(state > 7){
      state = 1;
    }
    while(!(digitalRead(BUTTON1)));
  }

  if(state == SLIDER_TEST) // Displays values of sliders
  {
    Serial.print("Sliders: ");
    val = analogRead(SLIDER1);
    Serial.print(" ");
    Serial.print(val);
    val = analogRead(SLIDER2);
    Serial.print("  ");
    Serial.print(val);
    val = analogRead(SLIDER3);
    Serial.print("  ");
    Serial.println(val);
    delay(300);
  }

  if(state == BUZZER_TEST) // Activates buzzer
  {
    for(int x = 0; x < 100; x++)
    {
      digitalWrite(BUZZER, HIGH);
      delay(2);
      digitalWrite(BUZZER, LOW);
      delay(2);
    }
  }

  if(state == CAPSENSE_TEST) // Tests CapSense pad
  {
    Serial.println("Skip-CapSense");
    //RM4GALILEO, long start = millis();
    //RM4GALILEO, long total1 =  cs_9_2.capSense(30);
    //RM4GALILEO, Serial.println(total1);
    delay(10);
  }

  if(state == TEMP_TEST) // Displays temp sensor values
  {
    val = analogRead(TEMP);
    Serial.print("Temp: ");
    Serial.println(val);
  }

  if(state == LIGHT_TEST) // Displays light sensor values
  {
    val = analogRead(LIGHT);
    Serial.print("Light: ");
    Serial.println(val);
  }

  if(state == BUTTON_TEST) // Toggles LED's depending on Buttons 2 and 3
  {
    Serial.println("Button 2 & 3 Test");
    if(digitalRead(BUTTON3))
    {
      digitalWrite(LED1,HIGH);

    }
    else
    {
      digitalWrite(LED1,LOW);

    }
    if(digitalRead(BUTTON2))
    {
      digitalWrite(LED2,HIGH);
    }
    else
    {
      digitalWrite(LED2,LOW);
    }
  }

  if(state == SEVENSEG_TEST) // Cycles through 0-9 on seven-segment
  {
    i = 0;
    Serial.println("Seven segment display test");
    while(1)
    {
      digitalWrite(LATCH,LOW);
      shiftOut(DATA,CLOCK,MSBFIRST,~(ledCharSet[i]));
      digitalWrite(LATCH,HIGH);
      i++;
      if(i==10){
        i = 0;
      }
      delay(500);
    }
  }
}

/*
 **************************************************************************************
 * This function is part of wiringPi:
 *      https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 * wiringPi is free software: you can redistribute it and/or modify it under the terms of
 * the GNU Lesser General Public License as published by the Free Software Foundation,
 * either version 3 of the License, or (at your option) any later version.
 *
 * wiringPi is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
 * PURPOSE.  See the GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with wiringPi.  If not, see <http://www.gnu.org/licenses/>.
 **************************************************************************************
 * shiftOut:
 *      Shift data out to a clocked source
 */
void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val)
{
  int8_t i;

  if (order == MSBFIRST)
    for (i = 7 ; i >= 0 ; --i)
    {
      digitalWrite (dPin, val & (1 << i));
      digitalWrite (cPin, HIGH);
      digitalWrite (cPin, LOW);
    }
  else
    for (i = 0 ; i < 8 ; ++i)
    {
      digitalWrite (dPin, val & (1 << i));
      digitalWrite (cPin, HIGH);
      digitalWrite (cPin, LOW);
    }
}
```

24.4 Results

Arduino* 101 compatible. The temperature value is in the 240s before conversion (0 - 255 analogRead(). It was 153 in our report with other shields. This is because the 101 has a scale of 0 – 3.3V for analogRead(). Ignore temperature because of the scale.

No capability for CapSense due to AVR calls.

§

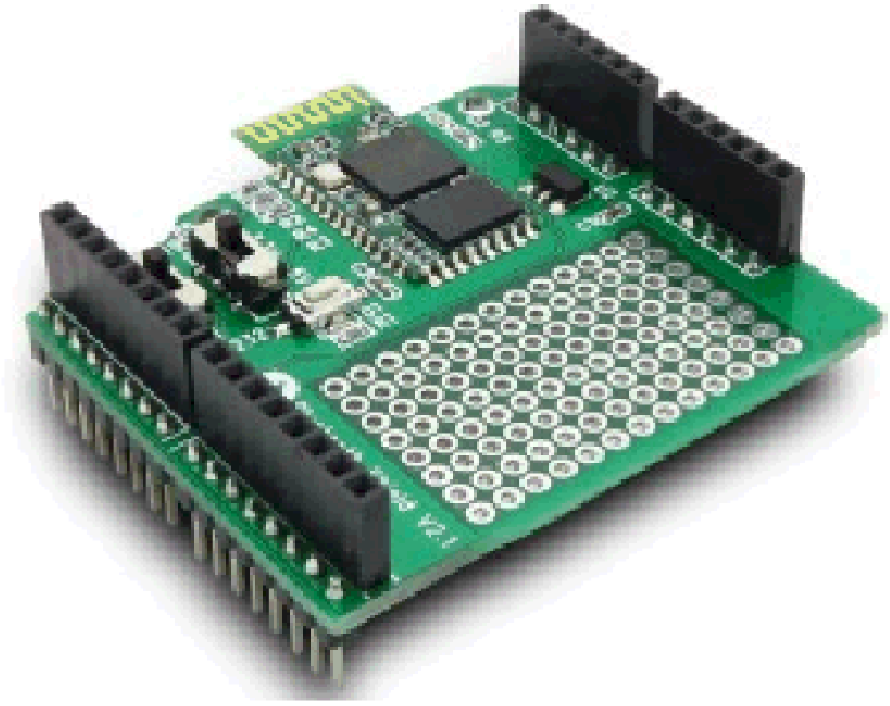
25 ITEAD* Bluetooth* Shield (Slave)

25.1 Use case

This Bluetooth* (BT) shield (slave) is an HC-06 serial port Bluetooth module breakout board for the Arduino* Uno board. You can directly use this BT shield with the Arduino* Uno UART port for Bluetooth communication. This shield cannot be a master Bluetooth device.

Key Info	Links
URL	*http://imall.iteadstudio.com/im120417006.html*
Library	No library required.
Phone software	*https://play.google.com/store/apps/details?id=jp.side2.apps.btterm* It may be possible to use the Bluetooth on the same PC as the IDE. However, there are cases where the Arduino* IDE will hang if Bluetooth is turned on (in cases where the BT is on the chipset). Alternative solution is to use a USB Bluetooth adapter. The best method is to use a device that is independent from the IDE. In this case, we are using an Android phone.

Figure 65 **ITEAD* Bluetooth* shield (slave)**



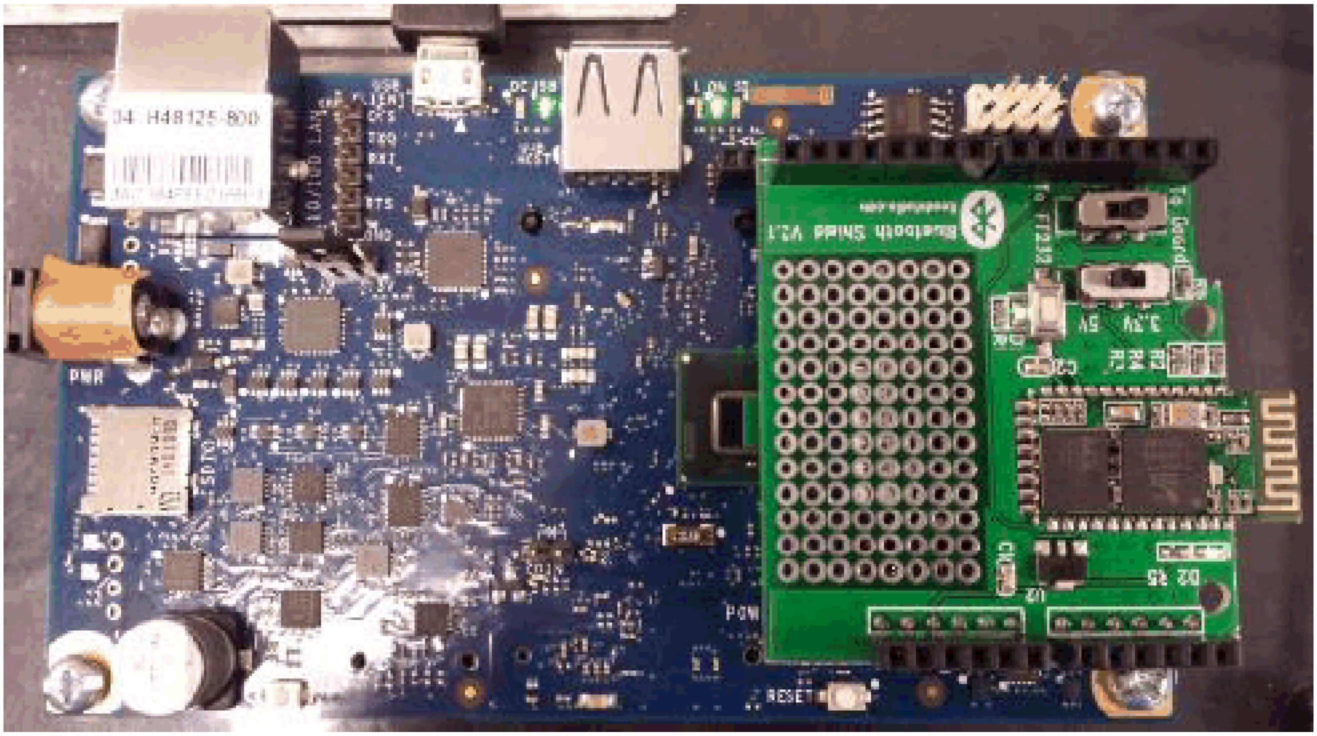
- Arduino* Uno board compatible
- Up to10 m communication distance in house without obstacle
- UART interface (TTL) with programmable baud rate (SPP firmware installed)
- Default baud rate: 9600, data bits: 8, stop bit: 1, Parity: No parity
- Default PINCODE:
- “1234”
- A full set of configuration commands
- Onboard PCB antenna
- FCC ID certified

25.2 Hardware summary

Key Info	Description/Links
IDE	1.6.8
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No
Operating voltage	3.3 to 5 V
Datasheet	*ftp://imall.iteadstudio.com/IM120417006_BT_Shieldv2.1/DS_IM120417006_BT_Shield_Slave.pdf*

The shield has two switches. One switch toggles between 3.3 and 5 V. The other switch toggles between FT232 and the board. When in FT232 mode, the Arduino* Uno board and any sketch that is running on it is ignored and USB serial input is passed straight through to the Bluetooth module.

Figure 66 **ITEAD* Bluetooth* shield on an Intel® Galileo second generation board**



25.3 Compile and upload

For the test, two Bluetooth devices are required. An Android* smartphone running an app called “Bluetooth SPP Pro” was used as the second Bluetooth device (master). The shield is the first Bluetooth device (slave).

- Ensure that the shield is set to 3.3 V for the duration of this test.
- Before downloading the sketch, pair (from the phone) the phone and shield together. The password is indicated above. Pairing only needs to happen once. If shield is powered down, only a connection request is required.
- Connect the phone and shield from the phone software.
- Set the phone software and the serial port to send ‘\r\n’ with send request. Excluding these characters will delay transmission.

At this point, the phone and shield are connected. We need a sketch to send data. The following settings are for sketch download.

Shield toggle (for sketch download)	Description
To board / To FT232	Set this to 'To Board'. When set to 'FT232', a sketch was not allowed to upload.
3.3 V / 5 V	Set to '5 V'. When set to '3.3 V', Arduino* was not allowed to upload.

- Ensure that the phone application is connected to the shield (done previously). Go into "CMD line mode" and ensure that the 'end flag' is set to '\r\n'. The keyboard should come up ready for input.
- Download the following *LED* sketch (with settings above).
- Bring up the serial window.

Example 32 **LED sketch**

```
void setup()
{
  pinMode(13,OUTPUT);
  Serial.begin(9600);
  Serial1.begin(9600);
  waitForUser(9);
}
void loop()
{
  char fromBT;
  if (Serial1.available() > 0 ) // From BT
  {
    // Read from BT
    fromBT = Serial1.read();
    // Process data
    if(fromBT == '1')
      digitalWrite(13,HIGH);
    else if(fromBT == '0')
      digitalWrite(13,LOW);

    // Echo back char received from BT
    Serial.print(fromBT);  // To IDE
    Serial1.print(fromBT);  // To BT, echo
  }

  if(Serial.available() > 0) // From IDE
  {
    char fromIDE = Serial.read(); // Read IDE
    Serial.print(fromIDE); // To IDE, echo
    Serial1.print(fromIDE); // To BT
  }
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
  Serial.println("");
}
```

- Type and send characters on the phone. No data will be displayed on the IDE.
- Set the shield switch to "To Board" and type and send characters again.
- Data will appear in the IDE window. Those characters are then written back to the BT. Notice on the phone that the same characters transmitted are received.
- Type and send the character "1". The LED13 should turn on.
- Type and send the character "0". The LED13 should turn off.

The UART on both the Arduino* Uno and Arduino* 101 boards communicates on digital pins 0 (Rx) and 1 (Tx) as well.

25.4 Results

Compatible with Arduino* 101

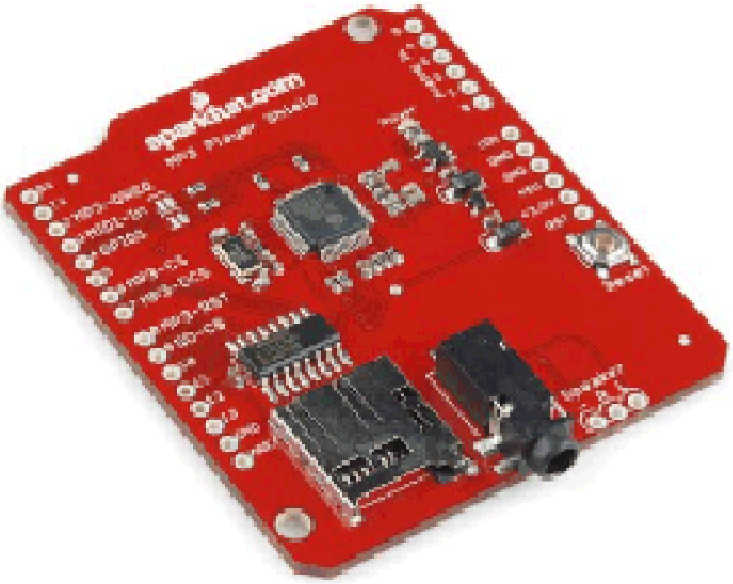
§

26 SparkFun* MP3 Player Shield

26.1 Use case

MP3 files can be pulled from a microSD card and played using only one shield, effectively turning any Arduino* Uno board into a fully functional standalone MP3 player! The MP3 shield uses the VLSI Solution* VS1053B MP3 audio decoder IC to decode audio files. The VS1053 is also capable of decoding Ogg Vorbis/MP3/AAC/WMA/MIDI audio and encoding IMA ADPCM and user-loadable Ogg Vorbis. The VS1053 receives its input bitstream through a serial input bus (SPI). After the stream has been decoded by the IC, the audio is sent out to both a 3.5 mm stereo headphone jack, as well as a 2-pin 0.1" pitch header.

Figure 67 **SparkFun* MP3 player shield**

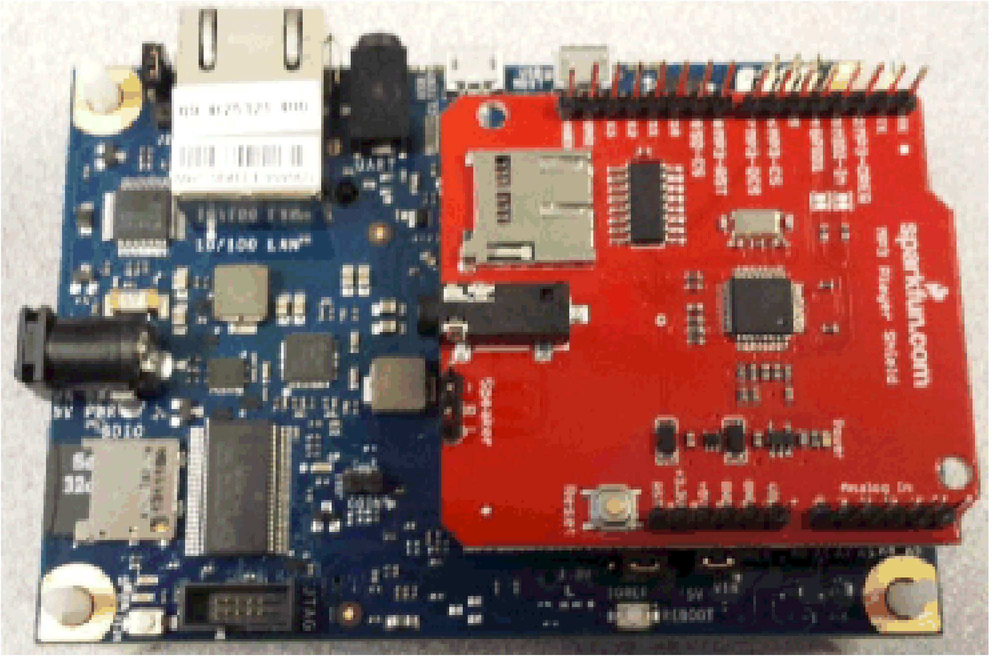


Key Info	Links
Order/product info	*https://www.sparkfun.com/products/10628*
Guide	*http://arduino.cc/en/Tutorial/GalileoSampleSequencer*
Library	The Sparkfun website has tutorials on this shield; however, it is Arduino* Uno-based and complex.
Sample files	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/MP3_Player_Files.zip* These sample files compile from the Sparkfun tutorial. They are called “track001.mp3” and “track002.mp3”.

26.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V. Should work at 3.3 V.
IOREF	Present but not used.
Use VIN as power source	No
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/MP3%20Shield-v13.pdf*

Figure 68 **SparkFun* MP3 player shield on an Intel® Galileo first generation board**



We modified the “Sample Sequencer” code to play an MP3 file. The code works as is; we just reduced the size by removing unused functions.

Format the SD card as FAT32 and copy both test files (cited above) to the root directory.

Insert the SD card, download the sample sketch. The sketch will wait for 5 seconds for the user to bring up the serial output (optional).

Example 33 **Sample sequencer sketch**

```
#include <SPI.h>
#include <SD.h>

#define CHUNK_SIZE 32
#define PAGE_SIZE 4096

//Create the variables to be used by SdFat Library
File track;

//This is the name of the file on the microSD card you would like to play
//Stick with normal 8.3 nomenclature. All lower-case works well.
//Note: you must name the tracks on the SD card with 001, 002, 003, etc.
//For example, the code is expecting to play 'track002.mp3', not track2.mp3.
char trackName[] = "track002.mp3";
int trackNumber = 1;

char errorMsg[100]; //This is a generic array used for sprintf of error messages

#define TRUE  0
#define FALSE  1

//MP3 Player Shield pin mapping. See the schematic
#define MP3_XCS 6 //Control Chip Select Pin (for accessing SPI Control/Status registers)
```

```
#define MP3_XDCS 7 //Data Chip Select / BSYNC Pin
#define MP3_DREQ 2 //Data Request Pin: Player asks for more data
#define MP3_RESET 8 //Reset is active low
//Remember you have to edit the Sd2PinMap.h of the sdfatlib library to correct control the SD
card.

//VS10xx SCI Registers
#define SCI_MODE 0x00
#define SCI_STATUS 0x01
#define SCI_BASS 0x02
#define SCI_CLOCKF 0x03
#define SCI_DECODE_TIME 0x04
#define SCI_AUDATA 0x05
#define SCI_WRAM 0x06
#define SCI_WRAMADDR 0x07
#define SCI_HDAT0 0x08
#define SCI_HDAT1 0x09
#define SCI_AIADDR 0x0A
#define SCI_VOL 0x0B
#define SCI_AICTRL0 0x0C
#define SCI_AICTRL1 0x0D
#define SCI_AICTRL2 0x0E
#define SCI_AICTRL3 0x0F

//Synth
#define notes_len_max 8
#define num_sounds 49
int pin_test=3;
int pin_save=9;
int pin_silent=10;
int pin_random=5;
int knob_choose_pin=A0;
int knob_nn_pin=A1;//number of notes knob

int sequence[notes_len_max];
int curr_sound_num;

int num_notes=notes_len_max;
float last_note_length=1;

//leds-74H595
#define latchPin A5

void setup() {
  pinMode(MP3_DREQ, INPUT);
  pinMode(MP3_XCS, OUTPUT);
  pinMode(MP3_XDCS, OUTPUT);
  pinMode(MP3_RESET, OUTPUT);

  pinMode(4, OUTPUT);
  digitalWrite(4, LOW);

  digitalWrite(MP3_XCS, HIGH); //Deselect Control
  digitalWrite(MP3_XDCS, HIGH); //Deselect Data
  digitalWrite(MP3_RESET, LOW); //Put VS1053 into hardware reset

  Serial.begin(9600); //Use serial for debugging
  delay(1000*5);      //Wait for user to open serial port

  //Serial.println("Type any character to start");
  //while (Serial.read() <= 0) {}
  Serial.println("MP3 Testing");

  if (!SD.begin(0)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("SD card initialized.");

  //From page 12 of datasheet, max SCI reads are CLKI/7. Input clock is 12.288MHz.
  //Internal clock multiplier is 1.0x after power up.
  //Therefore, max SPI speed is 1.75MHz. We will use 1MHz to be safe.
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16); //Set SPI bus speed to 1MHz (16MHz / 16 = 1MHz)
  SPI.transfer(0xFF); //Throw a dummy byte at the bus
  //Initialize VS1053 chip
  delay(10);
  digitalWrite(MP3_RESET, HIGH); //Bring up VS1053
  //delay(10); //We don't need this delay because any register changes will check for a high
DREQ

  //Mp3SetVolume(20, 20); //Set initial volume (20 = -10dB) LOUD
  Mp3SetVolume(40, 40); //Set initial volume (20 = -10dB) Manageable
  //Mp3SetVolume(80, 80); //Set initial volume (20 = -10dB) More quiet

  //Let's check the status of the VS1053
  int MP3Mode = Mp3ReadRegister(SCI_MODE);
  int MP3Status = Mp3ReadRegister(SCI_STATUS);
  int MP3Clock = Mp3ReadRegister(SCI_CLOCKF);

  Serial.print("SCI_Mode (0x4800) = 0x");
  Serial.println(MP3Mode, HEX);

  Serial.print("SCI_Status (0x48) = 0x");
  Serial.println(MP3Status, HEX);

  int vsVersion = (MP3Status >> 4) & 0x000F; //Mask out only the four version bits
  Serial.print("VS Version (VS1053 is 4) = ");
  Serial.println(vsVersion, DEC); //The 1053B should respond with 4. VS1001 = 0, VS1011 = 1,
VS1002 = 2, VS1003 = 3

  Serial.print("SCI_ClockF = 0x");
  Serial.println(MP3Clock, HEX);

  //Now that we have the VS1053 up and running, increase the internal clock multiplier and up
our SPI rate
  Mp3WriteRegister(SCI_CLOCKF, 0x60, 0x00); //Set multiplier to 3.0x

  //From page 12 of datasheet, max SCI reads are CLKI/7. Input clock is 12.288MHz.
  //Internal clock multiplier is now 3x.
  //Therefore, max SPI speed is 5MHz. 4MHz will be safe.
  SPI.setClockDivider(SPI_CLOCK_DIV4); //Set SPI bus speed to 4MHz (16MHz / 4 = 4MHz)

  MP3Clock = Mp3ReadRegister(SCI_CLOCKF);
  Serial.print("SCI_ClockF = 0x");
  Serial.println(MP3Clock, HEX);

  //MP3 IC setup complete

  //Synth
  pinMode(pin_test,INPUT);
  pinMode(pin_save,INPUT);
  pinMode(pin_silent,INPUT);
  pinMode(pin_random,INPUT);
  randSeq();

  //leds-74HC595
  pinMode(latchPin, OUTPUT);
  digitalWrite(latchPin,LOW);

}

void loop(){
  Serial.println("Attempting to play MP3. . .");
  playMP3("track001.mp3");
  while (1) {};
}

//PlayMP3 pulls 32 byte chunks from the SD card and throws them at the VS1053
//We monitor the DREQ (data request pin). If it goes low then we determine if
//we need new data or not. If yes, pull new from SD card. Then throw the data
//at the VS1053 until it is full.
void playMP3(char* fileName) {

  uint8_t *mp3DataBuffer;
  int offset = 0;
  uint32_t size;

  //Serial.println("Start MP3 decoding");

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File track = SD.open(fileName, FILE_READ);
  if (!track)
  {
    //Serial.print(fileName);
    //Serial.println(" not found");
    // don't do anything more:
    return;
  }

   size = track.size();

   mp3DataBuffer = (uint8_t *)malloc(size);
   if (!mp3DataBuffer)
   {
      //Serial.println("Failed to alloc mem for data buffer");
      return;
   }
  //Serial.println("Track open");

  offset = 0;
  while (offset < size)
  {
    int nbytes = size - offset;
    int ret;

    /* Read no more than one page at a time */
    if (nbytes > PAGE_SIZE)
      nbytes = PAGE_SIZE;

    ret = track.read(mp3DataBuffer+offset, nbytes);
    if (ret < 0)
    {
      //Serial.print("Failed to read file, error: ");
      //Serial.println(ret);
      return;
    }

    offset += ret;
  }

   //Serial.print("Read whole file, size is: ");
   //Serial.println(size);

  /* Start feeding data to the VS1053 */
  offset  = 0;
  digitalWrite(MP3_XDCS, LOW); //Select Data
  while(offset < size && !getCommand()) {
    //Once DREQ is released (high) we now feed 32 bytes of data to the VS1053 from our SD read
buffer
    while(!digitalRead(MP3_DREQ));

    SPI.transferBuffer(mp3DataBuffer + offset, NULL, (size - offset) > CHUNK_SIZE ? CHUNK_SIZE
: size - offset); // Send SPI bytes
    offset += CHUNK_SIZE;

    //getSynthInput();
  }
  digitalWrite(MP3_XDCS, HIGH); //Deselect Data

  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating transfer is complete

  digitalWrite(MP3_XDCS, HIGH); //Deselect Data

  track.close(); //Close out this track
  free(mp3DataBuffer);

  //sprintf(errorMsg, "Track %s done!", fileName);
  //Serial.println(errorMsg);
}

//Write to VS10xx register
//SCI: Data transfers are always 16bit. When a new SCI operation comes in
//DREQ goes low. We then have to wait for DREQ to go high again.
void Mp3WriteRegister(unsigned char addressbyte, unsigned char highbyte, unsigned char
lowbyte){
  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating IC is available
  digitalWrite(MP3_XCS, LOW); //Select control

  //SCI consists of instruction byte, address byte, and 16-bit data word.
  SPI.transfer(0x02); //Write instruction
  SPI.transfer(addressbyte);
  SPI.transfer(highbyte);
  SPI.transfer(lowbyte);
  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating command is complete
  digitalWrite(MP3_XCS, HIGH); //Deselect Control
}

//Read the 16-bit value of a VS10xx register
unsigned int Mp3ReadRegister (unsigned char addressbyte){
  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating IC is available
  digitalWrite(MP3_XCS, LOW); //Select control

  //SCI consists of instruction byte, address byte, and 16-bit data word.
  SPI.transfer(0x03);  //Read instruction
  SPI.transfer(addressbyte);

  char response1 = SPI.transfer(0xFF); //Read the first byte
  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating command is complete
  char response2 = SPI.transfer(0xFF); //Read the second byte
  while(!digitalRead(MP3_DREQ)) ; //Wait for DREQ to go high indicating command is complete

  digitalWrite(MP3_XCS, HIGH); //Deselect Control

  int resultvalue = response1 << 8;
  resultvalue |= response2;
  return resultvalue;
}

//Set VS10xx Volume Register

void Mp3SetVolume(unsigned char leftchannel, unsigned char rightchannel){
  Mp3WriteRegister(SCI_VOL, leftchannel, rightchannel);
}

char getCommand(){
  return false;//this function is disabled

  if(Serial.available()){
    byte b=Serial.read();
    if(b=='s')
      return true;
    else
      return false;
  }
}
void displayVal(int v){
  Serial.print(v);
  Serial.print(", ");
}
void randSeq(){
  for(int i=0;i<notes_len_max;i++){
    sequence[i]=(random()&0xff)/255.0*num_sounds;
  }
}
```

The small MP3 samples included in the Arduino library played fine. Longer (real song) MP3s sounded “chopped up”. It is not clear if this was due to the lack of performance of the SPI interface, buffering, sample rate, etc.

26.3 Results

Arduino* 101 not tested

SPI performance seems to be dependent on the SPI.tranfer() and SPI.transferBuffer() functions. Adjusting the arguments of these functions can make a difference. Increasing CHUNK_SIZE directly increases the transfer buffer. In the “Sample Sequencer” code, increasing the CHUNK_SIZE to 64 reduced “choppiness” significantly on larger MP3s; however, not completely correct. When playing a music file, imperfections were not obvious, although, when playing a constant 440 Hz tone, imperfections were noticeable.

.

27 Mayhew Labs* Mux Shield II

27.1 Use case

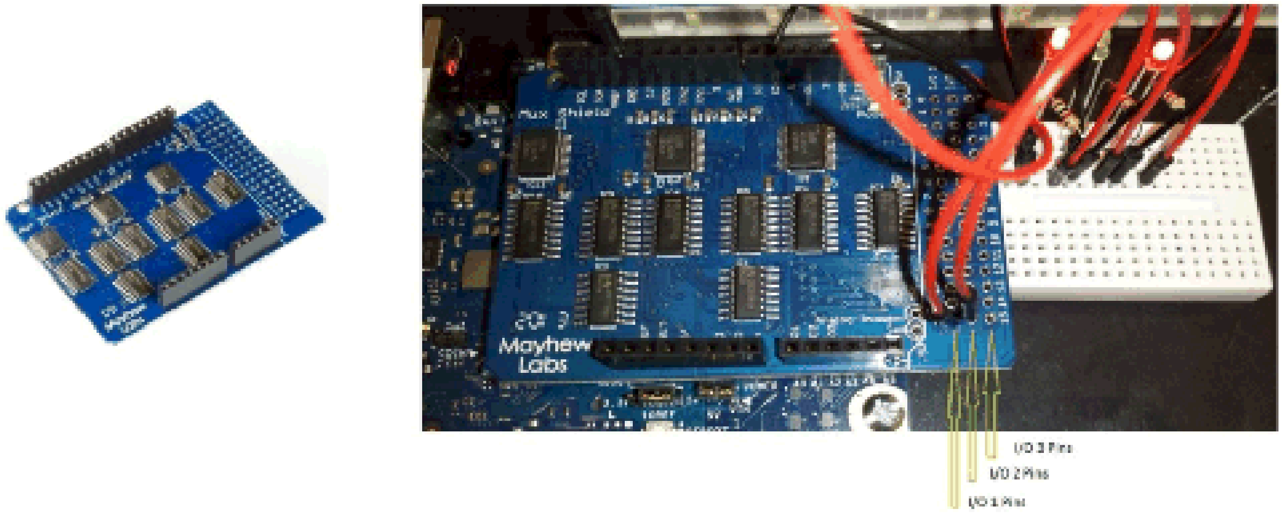
The Mux Shield II adds the capacity for up to 48 inputs or outputs. This shield uses three Texas Instruments* CD74HC4067 analog multiplexers that make it possible to have 48 analog/digital inputs or digital outputs in many combinations.

Key Info	Links
Order/product	*https://www.sparkfun.com/products/11723* *http://mayhewlabs.com/products/mux-shield-2* *http://www.robotmesh.com/sparkfun/mux-shield-ii*
Library	*http://mayhewlabs.com/code/MuxShield.zip* The exact same library that is also available from SparkFun*. No specific library date is available.
Schematics	*http://mayhewlabs.com/media/Mux_Shield_II_Rev0%20Schematic.pdf*
User guide	*http://mayhewlabs.com/media/Mux_Shield_II_User_Guide.pdf*

27.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V
IOREF	3.3 or 5 V (The mux digital outputs are at 5 V even if IOREF is set to 3.3 V. See also *Section 0.8 IOREF voltage.*))
Use VIN as power source	No
Intel® Galileo board firmware	1.0.4 (Production Release)
Intel® Edison board firmware	1.0.4
IDE	1.5.8-Intel.1.0.5
Shift registers	74HC595, *http://www.ti.com/lit/ds/symlink/sn74hc595.pdf*
Analog multiplexers	74HC4067, *http://www.ti.com/lit/ds/symlink/cd74hc4067.pdf*

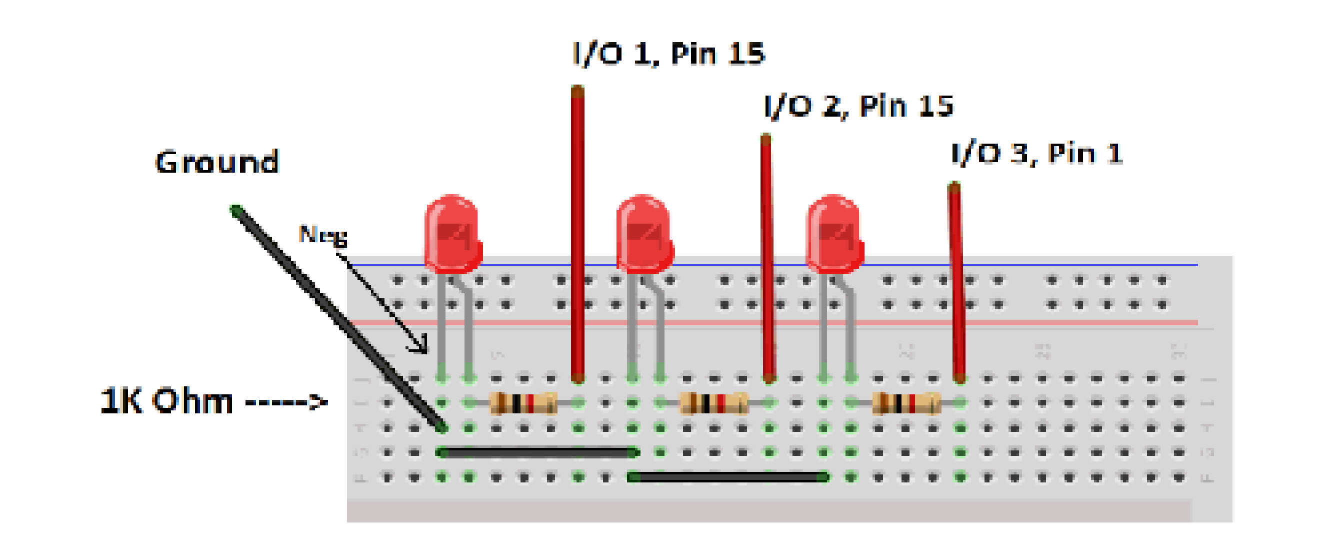
Figure 69 **Mux shield**



Pin Name	Function
GND	All negative LED leads are connected to ground, 5 V.
I/O 1, Pin 15 (on shield)	Connected to 1 kohm resistor, then connected to positive LED lead.
I/O 2, Pin 15 (on shield)	Connected to 1 kohm resistor, then connected to positive LED lead.
I/O 3, Pin 01 (on shield)	Connected to 1 kohm resistor, then connected to positive LED lead.

Insert the shield and wire up the LEDs as shown below on a prototype board. Attach the ground and I/O pins to specific pins on the shield. The LEDs will verify the assert signal.

Figure 70 **Mux shield LED wiring**



27.3 Companion library

27.4 Compile and upload

On the Arduino* 101 board, this shield worked without any modifications to the library or example for the Arduino* Uno board.

The library has four examples. This example used is the *MuxShieldDigitalOut* sketch included in the library.

The sketch defines each set of pins as digital. When the sketch runs, each pin in the I/O set (with delays between each pin) is asserted, and then proceeds to the next I/O set, etc. In the next entry into the loop function, the pins are de-asserted. The LED shall light up from left to right, and then turn off from left to right.

27.5 Results

Arduino* 101 compatible

§

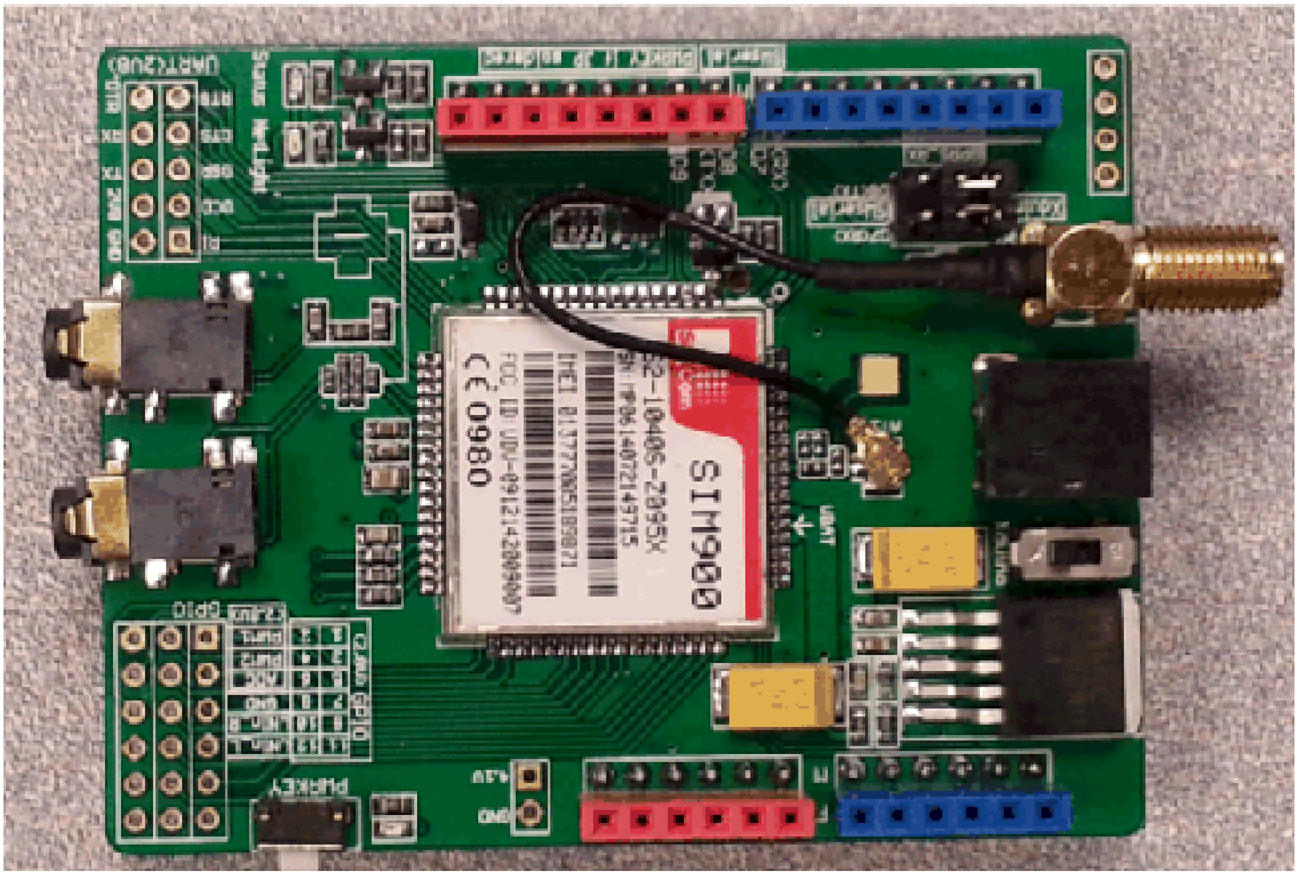
28 SIMCOM* Quadband Mobile Sim900 – Arduino* Pack

28.1 Use case

This GPRS shield is based on the SIM900 module from SIMCOM. The shield allows you to achieve SMS, MMS, GPRS, and audio via UART by sending AT commands. The shield has an interface for an external antenna. The shield has a jumper-selectable serial port so the Intel® Galileo boards are compatible with the hardware serial interface.

Key Info	Links
Product info	*http://store.cutedigi.com/sim900-gprs-gms-shield*
Library	No library. Use the serial communication interface to issue AT commands.
AT command set	*http://en.wikipedia.org/wiki/AT_command_set*

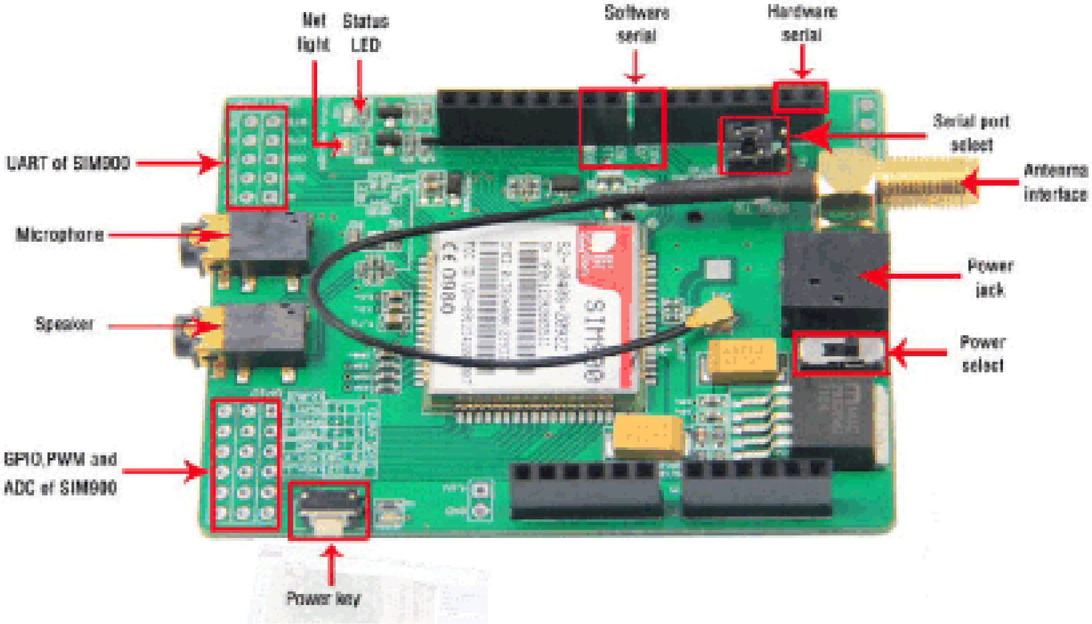
Figure 71 **SIMCOM* Quadband Mobile Sim900 - Arduino* Pack**



28.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	Yes
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Audio interface	2 connections (microphone and speaker).
SIM	Mini *SIM card* interface.
GSM module	SIM900.*http://www.simcom.ee/modules/gsm-gprs/sim900/*
Antenna	Antenna interface for external antenna. Antenna is provided with shield.
IDE	1.6.8

Figure 72 **SIMCOM* quadband mobile Sim900 layout**



Pin Name	Function
Rx of hardware serial port	Connected to D0
Tx of hardware serial port	Connected to D1
5 V	Connected to 5 V
GND	Connected to GND
D7	Only used if software serial is selected.
D8	Only used if software serial is selected.
D9	Used for software control of power-up/down of the SIM900.

28.3 Companion library

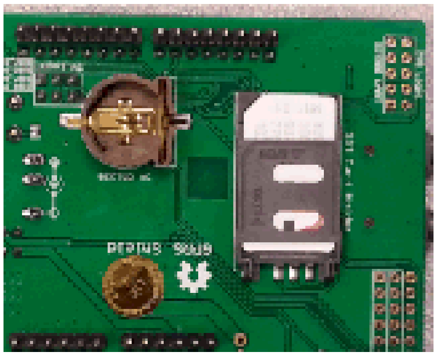
No library required.

28.4 Compile and upload

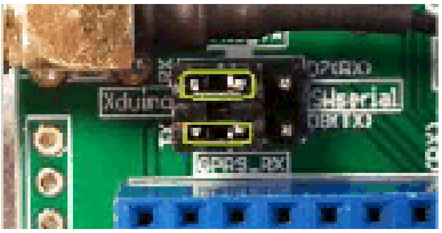
Do the following:

1. There is a SIM card holder on the back side of the card. Insert an unlocked SIM card. The SIM is a mini-SIM or 2FF size.
2. Attach the shield. There is no extra wiring necessary.

3. Set the Serial port select jumpers to the Hardware Serial position (Xduino).
4. Power up the Arduino* 101 and connect to computer with the USB.



5. Press the power key on the shield and hold for a couple of seconds. The net status light should begin blinking green once every 3 seconds if there is a network connection.



6. Update the following sketch to input a valid telephone number into the following lines of code. For this example, the call is in the United States using a 1 followed by a fictitious area code 555 and phone number 5555555.

```
char* gPhoneNumber = "15555555555";
```

7. Upload the following sketch that will enable sending text messages and making calls.
8. Open the serial terminal from the IDE and set rate to 9600.
9. From the serial terminal, input one of the following:
   - **a:** Answer a voice call. (The ring should appear in the serial terminal and the ring tone is played through the headset if connected.)
   - **d:** Dial a voice call. (Phone number already set in the sketch.)
   - **g:** Set up an IP session.
   - **r:** Display all received text messages.
   - **t:** Send a text message. (Text and phone number already set in the sketch.)

Example 34 **Issue AT commands over serial link sketch**

```
char  data[256];
char* gPhoneNumber = "15555555555"; // 15555555555
char* gTextLine1   = "G2>";
char* gTextLine2   = "Hello from Shield 29";

// Text Management
int   gSTART = 16;
int   gCOUNT = 2;

void setup()
{
  Serial.begin(9600);    // the GPRS baud rate
  Serial1.begin(9600);
  waitForUser(9);
  Serial.println("t=text, r=receive text, d=dial, a=answer call, h=hang up");
}
void loop()
{
    if (Serial.available())
       switch(Serial.read())
      {
       case 't':
          SendTextMessage();
          break;
       case 'r':
          ReceiveTextMessage();
          break;
       case 'd':
          DialVoiceCall();
          break;
       case 'a':
          AnswerVoiceCall();
          break;
       case 'g':
          GPRS();
          break;
       case 'h':
          HangupCall();
          break;
       case 'i':
          GetInformation();
          break;
   }
  if (Serial1.available())
    Serial.write(Serial1.read());
}
void AnswerVoiceCall()
{
  Serial1.println("ATA"); // A, Attempt to answer call
}
void HangupCall()
{
  Serial1.println("ATH1"); // H, hang up (1=hang up, 1=pickup)
  delay(100);
  Serial1.println();
}
void SendTextMessage()
{
  Serial1.print("AT+CMGF=1\r");    //Because we want to send the SMS in text mode
  delay(1000);
  Serial1.print("AT+CMGS=\"");
  Serial1.print(gPhoneNumber);
  Serial1.println("\"");
  delay(1000);

  Serial1.print(gTextLine1);
  Serial1.println(gTextLine2);
  delay(1000);

  Serial1.println((char)26);  //the ASCII code of the ctrl+z is 26 (0x1A)
  delay(1000);
  Serial1.println();
}
void ReceiveTextMessage()
{
  Serial1.println("AT+CMGF=1");    //Because we want to receive the SMS in text mode
  delay(1000);
  Serial1.println("AT+CPMS=\"SM\"");      // read first SMS
  delay(1000);
  Serial1.println("AT+CMGL=\"ALL\""); // show message
}
void DialVoiceCall()
{
  Serial1.print("ATD + ");//dial the number
  Serial1.print(gPhoneNumber);
  Serial1.println(";");
  delay(100);
  Serial1.println();
}
void ShowSerialData()
{
  while(Serial1.available()!=0)
      Serial.write(Serial1.read());
}
void GPRS()
{
  Serial1.println("AT+CPIN?");    // Is SIM ready to use?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+CGREG?");    // Is device registered?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+COPS?");    // Does SIM info match network?
  delay(1000);
  ShowSerialData();

  Serial.println("Check signal quality");
  Serial1.println("AT+CSQ");    // Check signal quality
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+cgatt=1");    // GPRS attach
  delay(1000);
  ShowSerialData();

  // define a PDP context with IP connection, ID is 1
  Serial1.println("AT+CGDCONT=1,\"IP\",\"fast.t-mobile.com\"");
  delay(1000);
  ShowSerialData();

  // list PDP contexts that are defined
  Serial1.println("at+cgdcont?");
  delay(3000);
  ShowSerialData();

  // setup the session using the appropriate PDP context
  Serial1.println("AT+CGACT=1,1");
  delay(1000);
  ShowSerialData();

  Serial.println("session is setup delay 5 seconds");
  delay(5000);

  // deactivate the PDP context
  Serial1.println("AT+CGACT=0,1");
  delay(1000);
  ShowSerialData();

  // detach from GPRS newtork
  Serial1.println("AT+CGATT=0");
  delay(1000);
  ShowSerialData();
}
void GetInformation()
{
  Serial1.println("ATI"); // Status Mfg, Model, Revison
  delay(1000);
  ShowSerialData();

  Serial1.println("ATI&V"); // Status
  delay(1000);
  ShowSerialData();
}
void TextMgtQuery()
{
  int j=1;
  Serial.println("...Reading");
  for(int i=gSTART; j<=gCOUNT; i++, j++)
  {
    if(i<1 || i>30) Serial.println("...ABORT");
    if(i<1 || i>30) return;

    Serial1.print("AT+CMGR="); // Read at a specific index
    Serial1.println(i);
    delay(1000);
    ShowSerialData();
  }
}
void TextMgtDelete()
{
  int j=1;
  Serial.println("...Deleting");
  for(int i=gSTART; j<=gCOUNT; i++, j++)
  {
    if(i<1 || i>30) Serial.println("...ABORT");
    if(i<1 || i>30) return;

    Serial1.print("AT+CMGD=");   // Delete at a specific index
    Serial1.println(i);
    delay(1000);
    ShowSerialData();
  }
```

```
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
Serial.println("");
} // waitForUser
```

28.5 Results

Arduino* 101 not tested

HTTP: Seems to work; however, a network error is received. If “AT+HTTPACTION=0” returns “+HTTPACTION:0,601,0” the AT response code 601 indicates a network error. If “AT+HTTPACTION=0” returns “+HTTPACTION:0,200,4” then HTTP GET is successful and it returns 4 bytes.

It is not determined if this network error could be a problem with the shield or the 3G service quality.

§

29 MCM* RS-232 Arduino* Shield

29.1 Use case

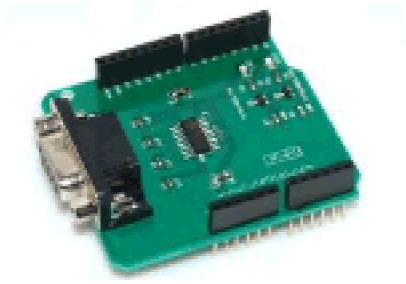
This shield is a standard serial communication port for industrial equipment. The purpose of this shield is to convert the UART to a RS-232 interface that is present on the shield. To talk to industrial equipment, this shield is added to connect the RS-232 port to a computer.

Key Info	Links
Order/product	*http://store.cutedigi.com/rs232-shield-for-arduino-v1/*
USB to serial cable	*http://www.newegg.com/Product/Product.aspx?gclid=CMKtvaeIkb8CFQqIfgodmVkA8g&Item=N82E16812107381&nm_mc=KNC-GoogleAdwords&cm_mmc=KNC-GoogleAdwords-_-pla-_-Serial+Cables-_-N82E16812107381&ef_id=UwmRrAAABAIRQ2K7:20140623223942:s* ***Note:*** This cable has LEDs for Tx and Rx. This gives a visual indication that data is flowing.
Library	None.

29.2 Hardware summary

Key Info	Description/Links (No wiring required)
Operating voltage	Designed for 5 V
IOREF	5 V only (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Intel® Galileo board firmware	1.0.4 (production release)
Intel® Edison board firmware	1.0.4
Schematics	*http://www.cutedigi.com/pub/Arduino/arduino_RS232.pdf*

Figure 73 **RS-232 Arduino** Shield**



This shield uses D0 and D1 on the Intel® Galileo/Arduino* Uno board with no jumpers to change the pin allocation. No wiring is required.

29.3 Companion library

No libraries required; however, there are a few additional steps to integrate the board with the PC as well as downloading a persistent sketch.

Note: A sketch cannot be downloaded to the board when the RS-232 shield is attached.

Since the shield cannot be attached while downloading a sketch, the sketch shall be persistent when the power is cycled.

This test will communicate from the board to the RS-232 Shield, through a connected 'Serial to USB' adapter to the PC. On the PC side, a driver (for the adapter) shall be installed such that the adapter is recognized (upon connection) and assigned a COM port. Communication to this adapter requires a serial communication program (CoolTerm, etc.) in addition to the serial connection from the IDE. The setting for this test is 9600 baud (8, N 1) for both COM ports.

29.4 Compile and upload

This sketch uses standard serial commands and standard calls to blink the LED. The LED blink (on Pin 13) is a visual conformation that the sketch is running on the Arduino* 101 board. The serial writes (in the sketch) are used to verify that the data transmitting through the RS-232 shield to the PC.

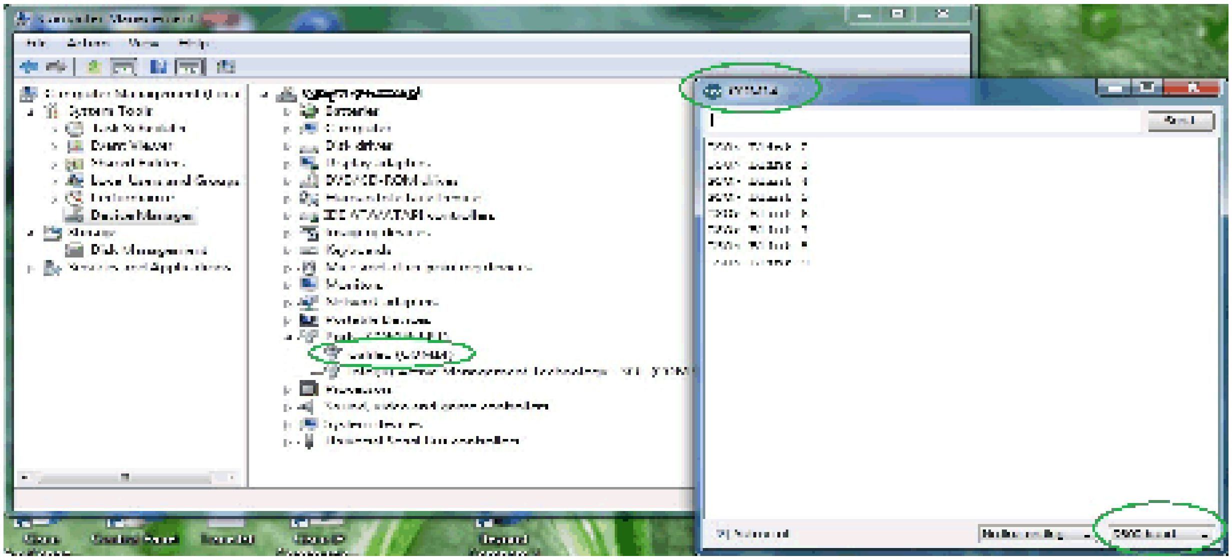
Example 35 **Example sketch**

```
int gCnt;                    // Loop iterations
int gLed = 13;               // Add led blink for visual verification (Pin 13)
void setup()
{
  Serial.begin(9600);
  Serial1.begin(9600); // Initialize serial port
  pinMode(gLed, OUTPUT);      // Initialize the digital pin as an output
  gCnt = 0;
}

void loop()
{
  if(Serial)
  {
    Serial.print("GSO> Blink ");
    Serial.println(gCnt);
  }
  if(Serial1)
  {
    Serial1.print("SO> Blink ");
    Serial1.println(gCnt);
  }

  digitalWrite(gLed, HIGH);  // Turn the LED on (HIGH is the voltage level)
  delay(1000*1);             // Wait in seconds

  digitalWrite(gLed, LOW);   // Turn the LED off (LOW is the voltage level)
  delay(1000*1);             // Wait in seconds
  gCnt++;                    // Counter
}
```

First, test the code above ***without the shield connected***. No additional wiring is required.

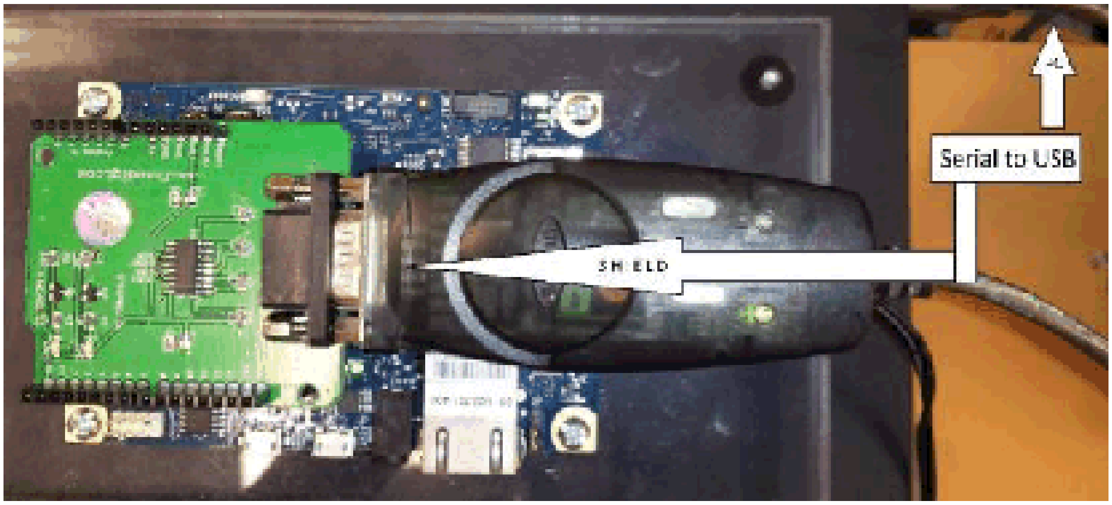
- Ensure that the shield is not connected to the board.
- Power up the board (wait the appropriate time).
- Connect the USB cable (used for sketch download) to the board.
- Deploy the sketch to the board. This operation will write the sketch to the SD card and will start up the sketch on the board.
- Look on the physical board and take note of the blinking LED.
- Start up the board’s IDE (do not attempt to download any programs) and start up the serial port.
- The serial terminal should display “Blink” with an iteration number.
- Power down the board.

Figure 74 **COM port test results – Intel® Galileo first generation board without the RS-232 Arduino* Shield**



The first test was successful on the board. The LED was blinking and the output was displayed on the serial port accessed Arduino*from the IDE. This test validated the functionality of the program.

Figure 75 **Connecting the RS-232 Arduino* Shield to an Intel® Galileo first generation board**



The second test uses the same program with the RS-232 shield attached. A ‘Serial to USB’ adapter will be attached from the RS-232 shield to the PC. The RS-232 shield will change the behavior of the same serial ports used in the sketch.

- Power down the board and remove the USB cable (used for sketch download).
- Plug in the USB side of the ‘Serial to USB’ adapter to the PC.
- Using Device Manager, verify that the adapter port has been assigned a COM port.

Note: If it is not there, install the proper driver.

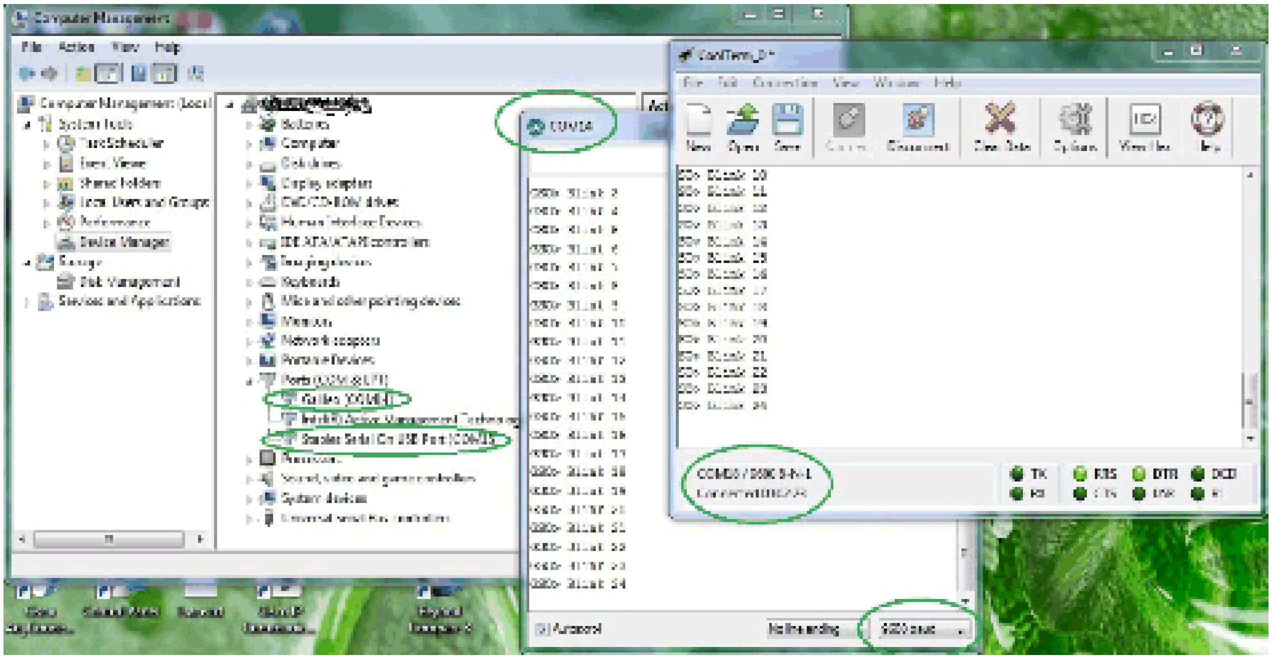
- Ensure that the board is powered down. (SD card shall be in the SD card slot.)
- Ensure the USB cable (used for sketch download) is not attached.

Note: The program downloaded in the first test is still present on the SD.

- Attach the RS-232 shield to the board.
- Attach the ‘Serial to USB’ adapter to the shield.
- Power up the board and wait (a full minute) until LED 13 is blinking. You should also see the Tx LED (on the shield) blinking.
- Connect the USB cable (used for sketch download) to the board. On the PC side, a COM port will be assigned to this connection.
- Start the IDE and bring up the serial port. Display for IDE Serial should appear.
- Start and configure the serial program for the USB/serial port on the PC.

If communication is working, then a blink count shall be present in the serial terminal for the RS-232 connection on the shield. LED13 shall also continue to blink on the board. The shield itself has Tx and Rx LEDs. The Tx LED should be blinking. The LEDs on the ‘Serial to USB’ adapter Tx LED should be blinking.

Figure 76 **COM port test results – Intel® Galileo first generation board with the RS-232 Arduino* Shield**



The second test showed a blinking LED. The serial port on the RS-232 shield is ‘Serial**1**’ . The standard serial on the IDE is ‘Serial’ . The result shows that data is being transmitted on both serial ports.

29.5 Results

Arduino* 101 compatible

§

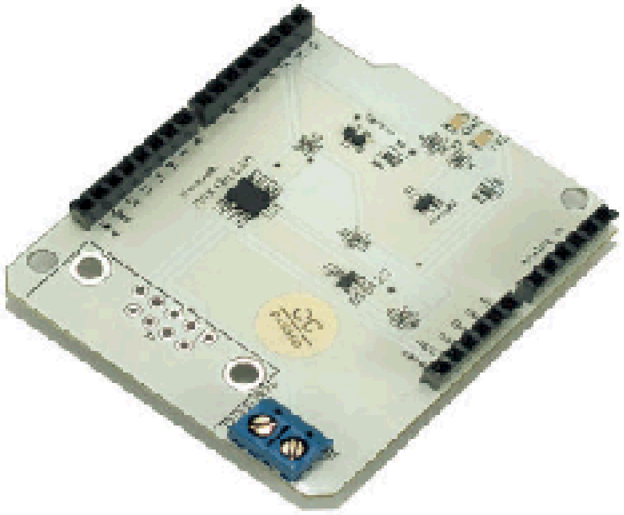
30 LinkSprite* RS-485 Shield

30.1 Use case

RS-485 is a standard communication port for field bus. The Arduino* Uno board only has a USB port and a TTL UART interface. In order to talk to a device that has an RS-485 bus, we can add an RS-485 port to the Arduino* Uno board using this RS-485 shield. Even though the RS-485 is sometimes thought of as an “archaic” protocol, it will allow up to 32 devices to communicate over the same data line over a cable length of up to 4000 ft., with a maximum data rate of 10 Mbps.

Key Info	Links
URL	*http://store.linksprite.com/rs485-shield-for-arduino-v2-1/* can't find link to version 1 shield so this link is for the version 2.1 shield
Library	Not required.
Guide	*http://linksprite.com/wiki/index.php5?title=RS485_Shield_for_Arduino*

Figure 77 **LinkSprite* RS-485 shield**



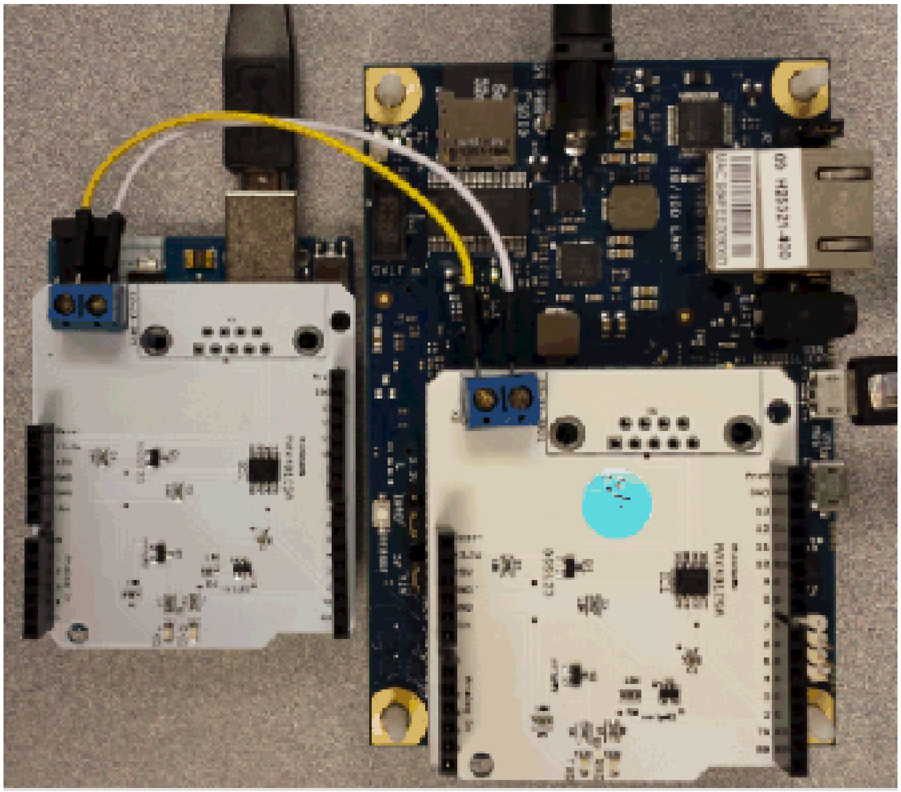
30.2 Hardware summary

Key Info	Description/Links
IOREF	No.
Use VIN as power source	No.
Operating voltage	3.3 to 5 V
Schematics	*https://s3.amazonaws.com/linksprite/Shields/RS485/RS485_schematics.pdf*

Connections used for testing:

- X2-1 Arduino* Uno board (transmitter) to X2-1 Arduino* 101(receiver)
- X2-2 Arduino* Uno board (transmitter) to X2-2 Arduino* 101(receiver)

Figure 78 **LinkSprite* RS-485 shield on an Intel® Galileo first generation board**



This RS-485 shield has the serial communication pins hardwired to D0 (Rx) and D1 (Tx). This means that the SoftwareSerial library does not need to be used on the Arduino* Uno board. This also means that the RS-485 shield and the USB port on the Arduino* Uno board are using the same TTL UART of the Atmel* Atmega328, and that you must remove the shield when you want to upload a sketch to Arduino* Uno board.

30.3 Compile and upload

The purpose of the following sketches is to test RS-485 communication between an RS-485 shield mounted on an Arduino* Uno board (acting as a transmitter) and an RS-485 shield mounted on a Arduino* 101 (acting as a receiver). The SoftwareSerial library can be used in the Arduino sketch but is not necessary and is commented out of the transmitter sketch below.

The sketch used on the Arduino* Uno device (transmitter) is as follows:

Example 36 **RS-485 transmitter sketch**

```
// T R A N S M I T T E R
void setup()
{
  //mySerial.begin (9600);
  Serial.begin(9600);
}

void loop()
{
  //mySerial.println ("Hello RS485!");
  Serial.println ("Hello RS485!");
  delay(1000);
}
```

The sketch used on the Arduino* 101 device (receiver) is as follows:

Example 37 **RS-485 receiver sketch**

```
// R E C E I V E R
void setup()
{
  Serial.begin(9600);
  Serial1.begin(9600);
}
void loop()
{
  while(Serial1.available() > 0)
  {
    char c = Serial1.read();
    Serial.write(c);
  }
}
```

30.4 Results

Arduino* 101 compatible

§

31 Seeed Studio* Relay Shield

31.1 Use case

This relay switch enables the Arduino* 101 or Arduino* Uno board to control the load of high current devices indirectly. The shield has four relays that can be wired as an NO (normally open) or as an NC (normally closed) connection for the high current devices.

Key Info	Links
Order/product	*http://www.SeeedStudio.com/depot/relay-shield-v20-p-1376.html?cPath=132_134*
Library	Not required.
Wiki link	*http://www.SeeedStudio.com/wiki/Relay_Shield_V2.0*

Figure 79 **Seeed Studio* Relay Shield**

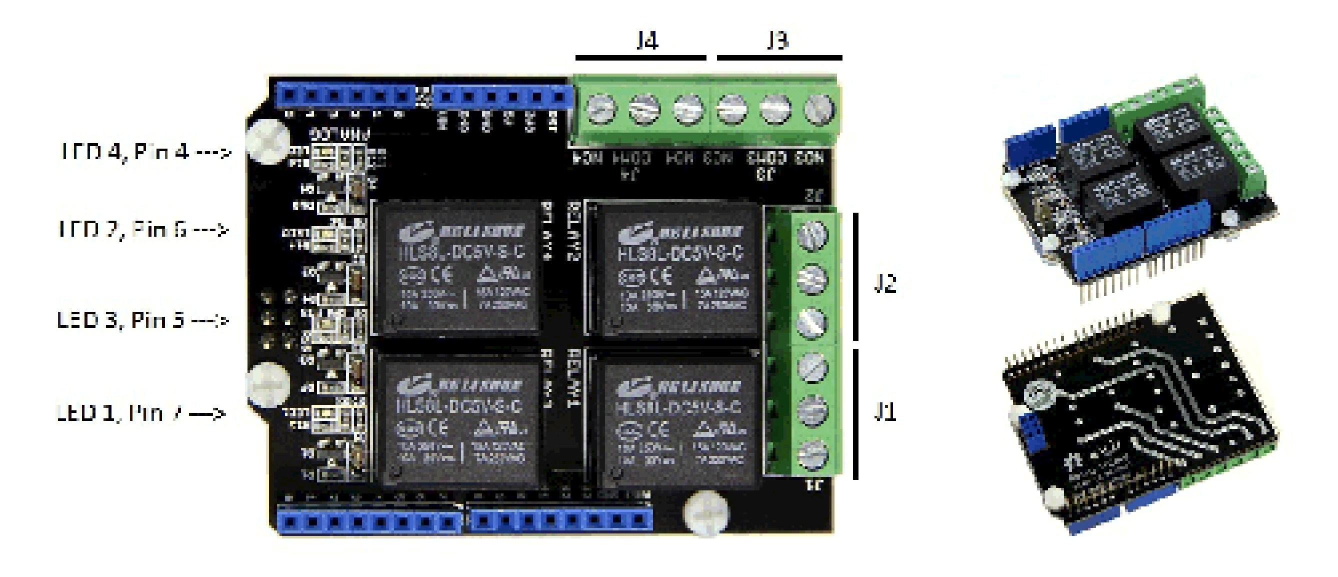
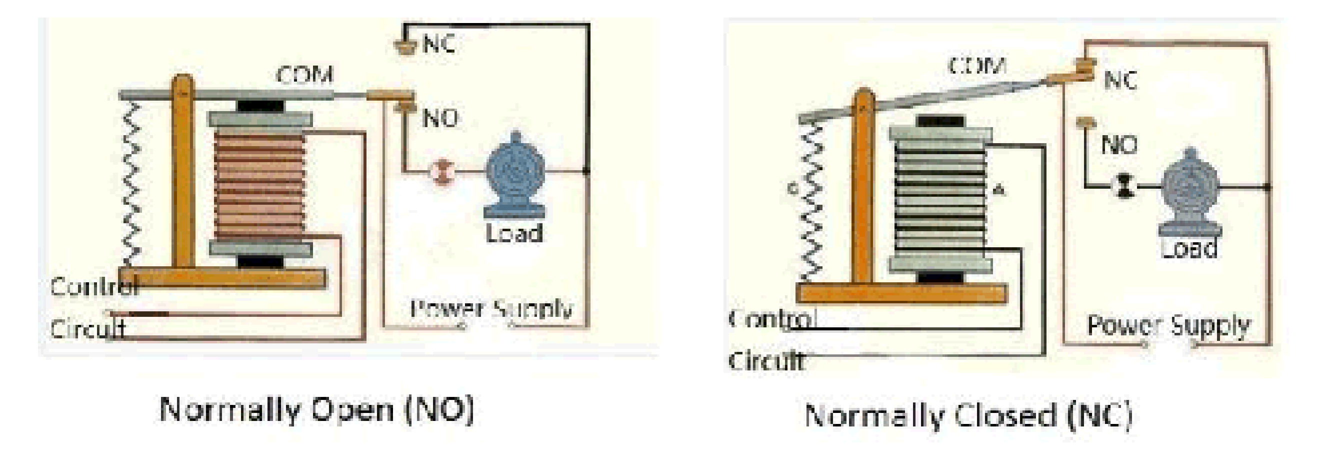


Figure 80 **Seeed Studio* relay shield NO and NC states**



31.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V.
IOREF	Present but not used.
Use VIN as power source	No.
Schematics	N/A.
Relay	HLS8L-DC5V-SC, *http://www.dipmicro.com/?datasheet=HLS8L.pdf*

No wiring is required. The sketch accesses the following digital pins:

Pin Name	Function (no wiring required)
D4	When asserted, references J4/COM4 channel interface (on the shield). Allows high power, up to 8 A and 30 V per channel.
D5	When asserted, references J3/COM3 channel interface.
D6	When asserted, references J2/COM2 channel interface.
D7	When asserted, references J1/COM1channel interface.

The disadvantage with this shield is that the relays are hard configured to a specific digital pin. The advantage is when a relay is asserted, a LED on the shield turns on, so testing of the shield can be done without any connections to the JN interfaces (NO/NC/COM) on the shield.

31.3 Compile and upload

Attach the shield to the board and download the sketch. When digital pin 5 is asserted, the relay will click and the LED light will be on. Use a multimeter to check the continuity between the COM and NO or NC terminals to see that the connection is being made on the proper terminal number.

Example 38 **Example sketch**

```
int MotorControl = 5;    // Arduino Pin to control the motor
// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 5 to be an output:
  pinMode(MotorControl, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop()  {
  digitalWrite(MotorControl,HIGH);// NO3 and COM3 Connected;
  delay(1000);
  digitalWrite(MotorControl,LOW);// NO3 and COM3 Disconnected;
  delay(1000);
}
```

31.4 Results

Arduino* 101 compatible

§

32 Seeed Studio* GPRS Shield

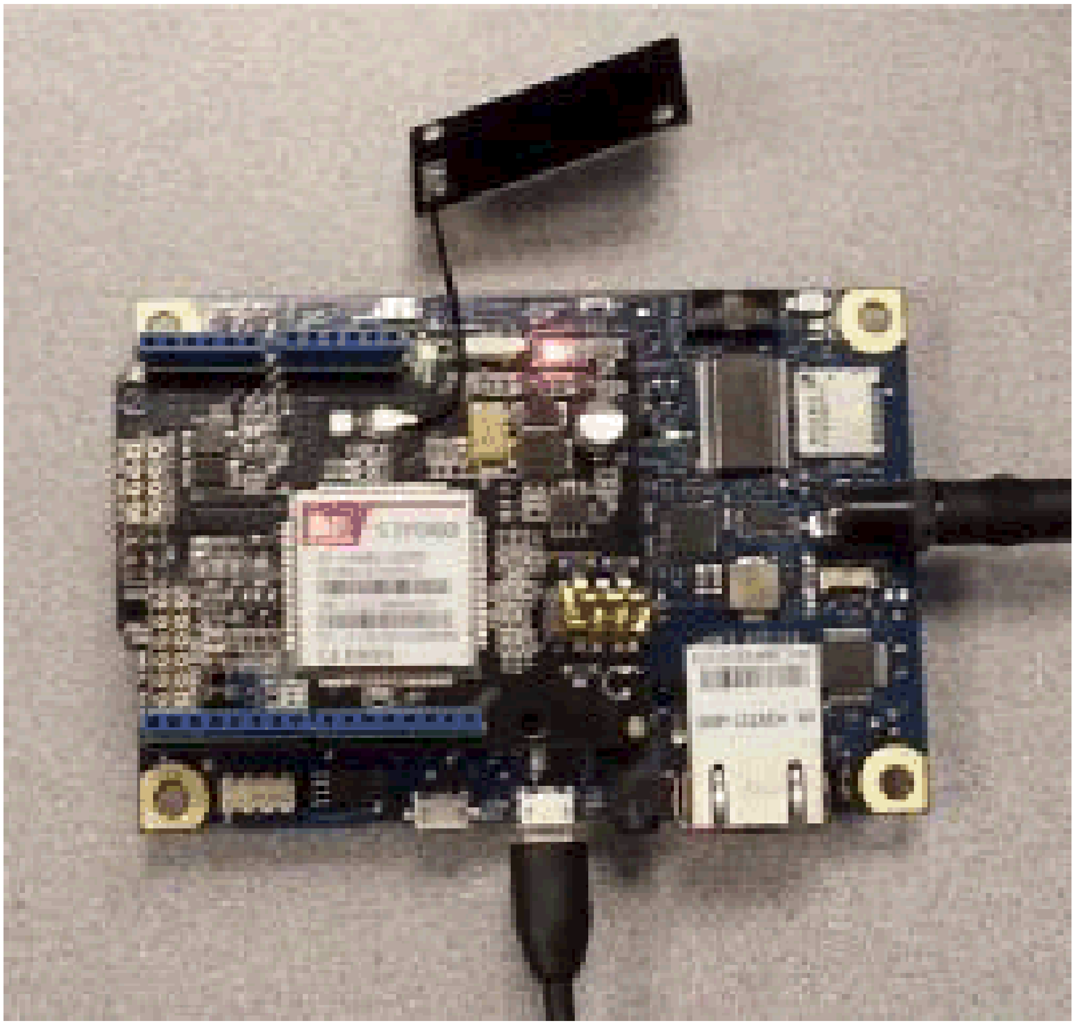
Note: This is on the "List of Supported Shields" (*Link*).

32.1 Use case

This shield makes it possible to hook the Arduino* 101/ Arduino* Uno board up to the GSM/GPRS cellular telephone network. It is possible to make and receive calls, or send and receive text messages using AT commands. The shield uses a SIMCOM* SIM900 quadband low power consumption GSM/GPRS module as well as a compact PCB antenna.

Key Info	Links
Product info	*http://www.SeeedStudio.com/wiki/GPRS_Shield_V2.0*
Guide	*http://www.geeetech.com/wiki/index.php/Arduino_GPRS_Shield*
Library	None.
AT commands	*http://en.wikipedia.org/wiki/AT_command_set*

Figure 81 **Seeed Studio* GPRS shield on an Intel® Galileo first generation board**



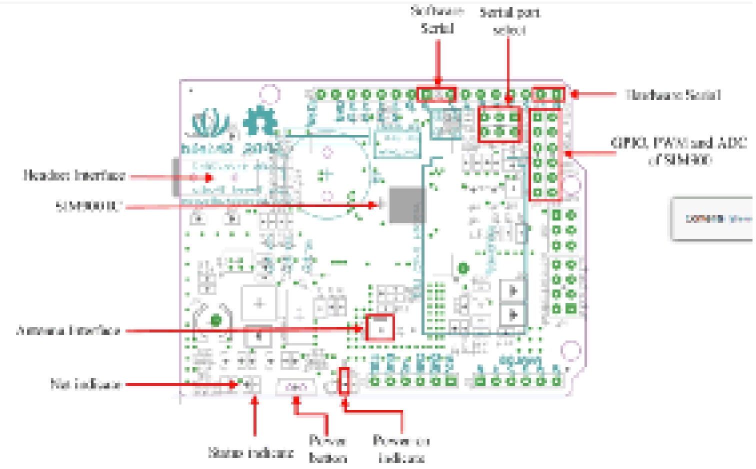
32.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	Yes
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Audio interface	2-in-1 TTRS headset
SIM	Mini SIM card interface.
GSM module	SIM900 *http://www.simcom.us/product_detail.php?cid=1&pid=37*
Antenna	Antenna interface for external antenna. PCB antenna is provided with the shield.
IDE	1.6.8

Pin Name	Function
Rx of hardware serial port	Connected to D0 (Input from board)
Tx of hardware serial port	Connected to D1 (Output from board)
Software power button for SIM900	Power SIM900 on or off – connected to D9.

LED	Status
Power-on indicator	Green if power on.
Status indicator for SIM900	Red if power on.
Net indicator (green)	64 ms on/800 ms off if network not found. 64 ms on/3000 ms off if network found. 64 ms on/300 ms off if GPRS communication down.

Figure 82 **Seeed Studio* GPRS shield layout**



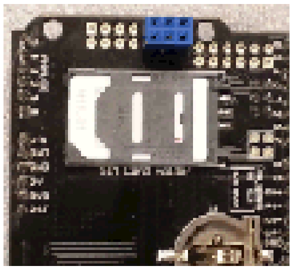
32.3 Companion library

No library required.

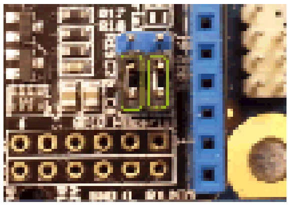
32.4 Compile and upload

Do the following:

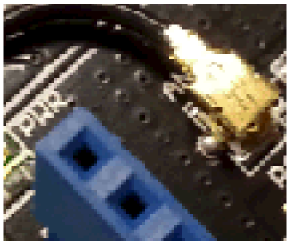
1. Insert an unlocked SIM card on the underside of the shield. The SIM is a mini-SIM (2FF size).
2. Attach the shield to the Arduino* 101 board. There is no extra wiring necessary.



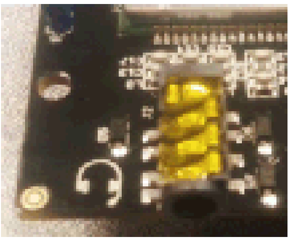
3. Set the serial port select jumpers to the *hardware serial* position. The default position is *software serial*.



4. Attach the PCB antenna that is supplied with the shield.

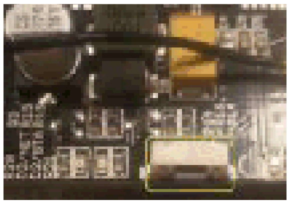


5. Connect a 2-in-1 TTRS headset to the headset connector on the shield.
6. Power up the board and connect it to a computer with the USB.



7. Press the shield power button for two seconds. You can skip this step if using a sketch that turns on the power, such as *Example 39*. The red status light should illuminate, and the green net status light should blink once every 3 seconds.
8. Update the following sketch to input a valid telephone number into the following lines of code. For this example, this is a US call using a 1 followed by a fictitious area code 555 and phone number 5555555.

```
char* gPhoneNumber = "15555555555"; // 1-555-555-5555
```



9. Upload the following sketch that will enable sending text messages and making voice calls.
10. Open the serial terminal from the IDE at 19200 baud.
11. From the serial terminal, input one of the following:
    - **a:** Answer a voice call. (The ring should appear in the serial terminal and the ring tone is played through the headset if connected.)
    - **d:** Dial a voice call. (Phone number already set in the sketch.)
    - **g:** Set up an IP session.
    - **r:** Display all received text messages.
    - **t:** Send a text message. (Text and phone number already set in the sketch.)

Example 39 **Issue AT commands over serial link sketch**

```
char  data[256];
char* gPhoneNumber = "15555555555"; // 15555555555
char* gTextLine1   = "G2>";
char* gTextLine2   = "Hello from Shield 33";

void setup()
{
  Serial.begin(19200);    // the GPRS baud rate
  Serial1.begin(19200);
  waitForUser(9);
  Serial.println("...Please power shield");
  Serial.println("...Wait for 'Call Ready'");
  Serial.println("t=text, r=receive text, d=dial, a=answer, h=hangup, g=display text");
}
void loop()
{
    if (Serial.available())
       switch(Serial.read())
      {
        case 't':
          SendTextMessage();
          break;
        case 'r':
          ReceiveTextMessage();
          break;
       case 'd':
          DialVoiceCall();
          break;
       case 'a':
          AnswerVoiceCall();
          break;
       case 'g':
          GPRS();
          break;
       case 'h':
          HangupVoiceCall();
          break;
   }
  if (Serial1.available())
    Serial.write(Serial1.read());
}
void AnswerVoiceCall()
{
  Serial1.println("ATA");
}
void HangupVoiceCall()
{
  Serial1.println("ATH1"); // H, hang up (1=hang up, 0=pickup)
  delay(100);
  Serial1.println();
}
void SendTextMessage()
{
  Serial1.print("AT+CMGF=1\r"); //We want to send the SMS in text mode
  delay(1000);

  Serial1.print("AT+CMGS=\"");
  Serial1.print(gPhoneNumber);
  Serial1.println("\"");
  delay(1000);

  Serial1.print(gTextLine1);
  Serial1.println(gTextLine2);
  delay(1000);

  Serial1.println((char)26);  //the ASCII code of the ctrl+z is 26 (0x1A)
  delay(1000);
  Serial1.println();
}
void ReceiveTextMessage()
{
  Serial1.println("AT+CMGF=1"); //We want to receive the SMS in text mode
  delay(1000);
  Serial1.println("AT+CPMS=\"SM\"");      // read first SMS
  delay(1000);
  Serial1.println("AT+CMGL=\"ALL\""); // show message
}
void DialVoiceCall()
{
  Serial1.print("ATD + ");//dial the number
  Serial1.print(gPhoneNumber);
  Serial1.println(";");
  delay(100);
  Serial1.println();
}
void ShowSerialData()
{
  while(Serial1.available()!=0)
      Serial.write(Serial1.read());
}
void GPRS()
{
  Serial1.println("AT+CPIN?");    // Is SIM ready to use?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+CGREG?");    // Is device registered?
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+COPS?");    // Does SIM info match network?
  delay(1000);
  ShowSerialData();

  Serial.println("Check signal quality");
  Serial1.println("AT+CSQ");    // Check signal quality
  delay(1000);
  ShowSerialData();

  Serial1.println("AT+cgatt=1");    // GPRS attach
  delay(1000);
  ShowSerialData();

  // define a PDP context with IP connection, ID is 1
  Serial1.println("AT+CGDCONT=1,\"IP\",\"fast.t-mobile.com\"");
  delay(1000);
  ShowSerialData();

  // list PDP contexts that are defined
  Serial1.println("at+cgdcont?");
  delay(3000);
  ShowSerialData();

  // setup the session using the appropriate PDP context
  Serial1.println("AT+CGACT=1,1");
  delay(1000);
  ShowSerialData();

  Serial.println("session is setup delay 5 seconds");
  delay(5000);

  // deactivate the PDP context
  Serial1.println("AT+CGACT=0,1");
  delay(1000);
  ShowSerialData();

  // detach from GPRS newtork
  Serial1.println("AT+CGATT=0");
  delay(1000);
  ShowSerialData();
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
  Serial.println("");
} // waitForUser
```

32.5 Results

Arduino* 101 not tested

32.6 Next steps

- Use better headset and microphone to test audio quality.
- Figure out how to clear text messages.

§

33 Seeed Studio* Solar Charger Shield v2

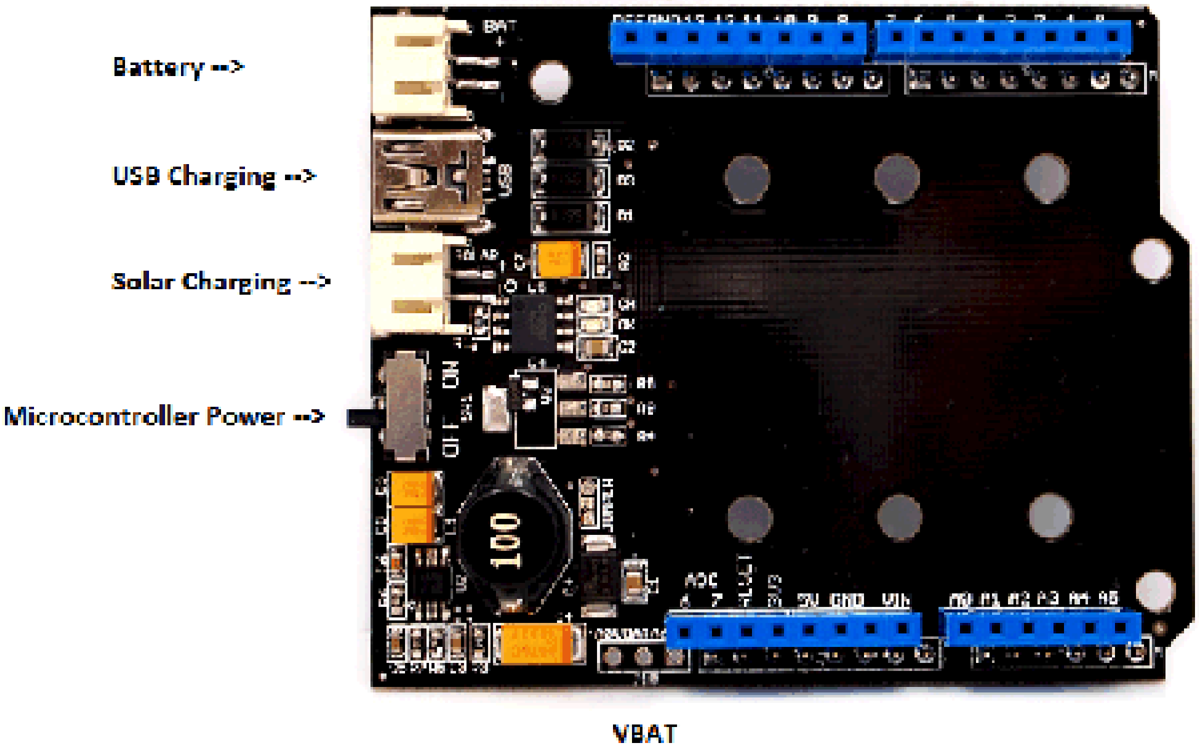
Note: This is on the “List of Supported Shields” (*Link*).

33.1 Use case

The solar charger is a stackable shield that enables adaptive battery power and can act as energy harvester for in-field charging. It is possible to use various batteries that have a voltage range of 2.7 to 4.2 V. It is possible to connect a Li-ion battery and solar panel to form an autonomous sensor unit. The maximum current provided by the board can get up to 700 mA. A USB connector is also useful to charge the battery. The shield features short-circuit protection, battery status indicator, and 5 V via USB port for powering small devices. The shield could be used in a wireless sensor unit application or for solar charging.

Key Info	Links
Product info	*http://www.SeeedStudio.com/depot/Solar-Charger-Shield-V2-p-914.html* *http://www.SeeedStudio.com/wiki/index.php?title=Solar_Charger_Shield_v2.0b*
Battery	*http://www.epictinker.com/Lithium-Ion-Polymer-LiPO-Battery-Pack-3A-p/pow105d1p.htm*
Solar panel	*http://www.SeeedStudio.com/depot/1W-Solar-Panel-80X100-p-633.html?cPath=1_118*
Guide	*http://www.SeeedStudio.com/wiki/index.php?title=Solar_Charger_Shield_v2.0b*
Library	None.

Figure 83 **Seeed Studio* solar charger shield v2**

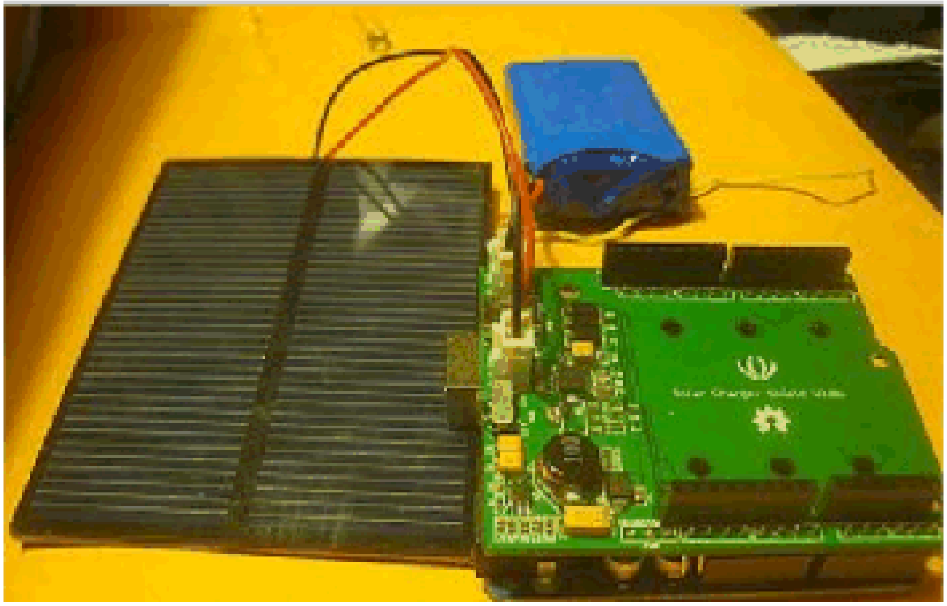


33.2 Hardware summary

Key Info	Description/Links
Operating voltage	2.7 to 5 V
IOREF	5 V only (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
VBAT	Pin to measure the output of the charging circuit of the battery.
Library	No library.

Pin Name	Function
BAT	Battery connection.
USB	Usb connection for charging.
SOLAR	Solar connection for charging.
SW1	On/Off Power switch to power on the microcontroller.
J10	A1/Bat/A6 Analog Input, Connect center pin to A0. ***Note:*** It is not clear what the first and third pins are.

Figure 84 **Seeed Studio* solar charger shield v2 with solar panel**



33.3 Companion library

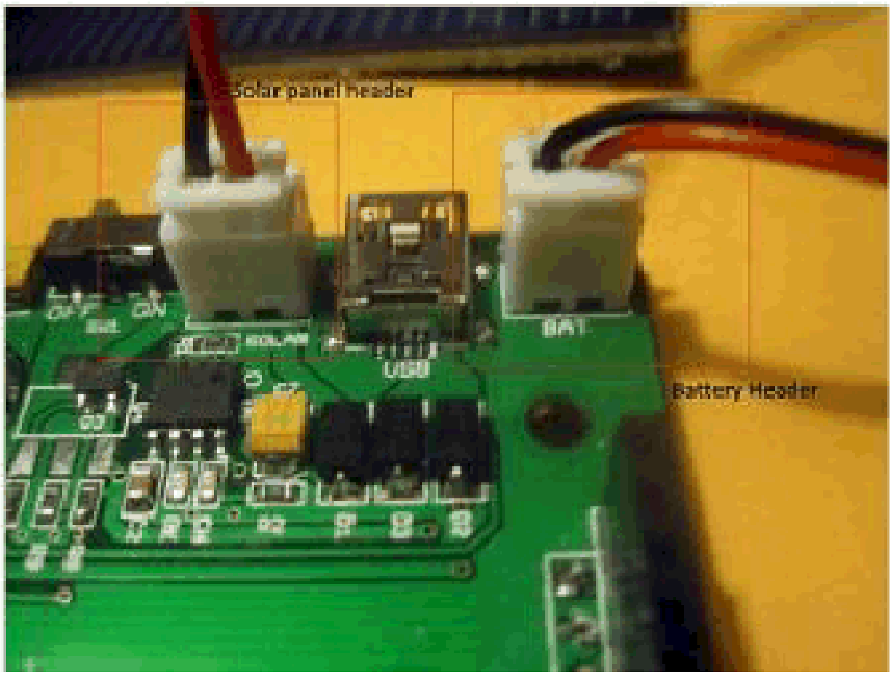
No library required.

33.4 Compile and upload

The solar shield does not need to be connected to the board to begin charging. We recommend charging the battery completely before connecting it to the microcontroller.

Connect the solar panel and the battery as shown in *Figure 85*.

Figure 85 **Seeed Studio* solar charger shield connections**

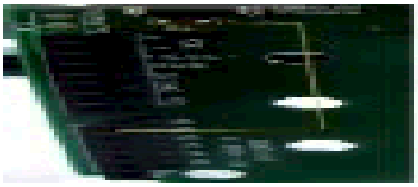


Place the solar panel facing either sunlight or filament bulbs so that it will begin charging the battery.

Verify that the red charging light is illuminated:

1. Unplug the battery header and note that the charging light should change to OK status since no current is flowing into the battery from the charging circuit. The green light should be glowing. It might take 5 to 7 hours to complete a full charging cycle. When the battery is fully charged, the green light will glow.
2. The battery can also be charged through the usb port. In this configuration, it is also not necessary for the shield to be attached to any microcontroller. When charging, the charging light will be red.
3. We will test the shield's capability of monitoring the battery. This will be done with an external power supply connected to the microcontroller. After the battery is charged, set the switch on the shield to 'OFF'. Remove the battery from the shield (along with solar or USB charging adapters).

4. The shield can be mounted on the Intel® Galileo first generation board. Connect VBAT pin (center pin) on the shield to pin A0 of the charger shield. Power up the Intel® Galileo board with the standard power supply.



Use the sketch in *Example 40* to measure the voltage of the battery.

Example 40 **Solar charger shield voltage measurement example**

```
/*
 Solar charger shield voltage measurement example. Connect VBAT pin to analog pin A0.

 The pin measures 2.0 V when not under direct exposure to sunlight and 5V when exposed to
sunlight.
 /*
 This example code is in the public domain.
 */

// These constants will not change.  They are used to give names
// to the pins used:
const int analogInPin = A0;  // Analog input pin that the VBAT pin is attached to
int   BatteryValue = 0;        // value read from the VBAT pin
float outputValue  = 0;       // variable for voltage calculation
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
void loop() {
  // read the analog in value:
  BatteryValue = analogRead(analogInPin);
  // Calculate the battery voltage value
  outputValue = (float(BatteryValue)*5)/1023*2;
 // print the results to the serial monitor:
  Serial.print("Analog value = " );
  Serial.print(BatteryValue);
  Serial.print("\t voltage = ");
  Serial.print(outputValue);
  Serial.println("V");

  // wait 10 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(1000*1); // Changed from 10 to 1000
}
```

The voltage measurement sketch prints two values: an analog value and an output value. Since the battery is not connected, the values are zero.

```
Sketch Output: When battery is not connected w/shield power ‘OFF’
Analog value = 0 voltage = 0.00V
Analog value = 0 voltage = 0.00V
Analog value = 0 voltage = 0.00V
Analog value = 0 voltage = 0.00V
Analog value = 0 voltage = 0.00V
```

Power down the microcontroller and attach the battery. Keep the shield switch on ‘OFF’. Download the sketch again (if needed). Observe that the readings are being obtained from the battery. It is not clear what the analog value represents, as it does not correlate to any readings taken from a voltage meter on the battery. However, it does come from reading the analog pin connected to the VBAT pin. The second value printed in the sketch is calculated using the previously printed value, but it is not clear what this value represents either.

```
Sketch Output: When battery is connected w/shield power ‘OFF’
Analog value = 392 voltage = 3.83V
Analog value = 392 voltage = 3.83V
Analog value = 392 voltage = 3.83V
Analog value = 391 voltage = 3.82V
Analog value = 391 voltage = 3.82V
```

Last, we can verify that the shield with a battery can power the microcontroller. Send the standard ‘Blink’ program and ensure that the sketch is persistent. This is necessary because the microcontroller will be powered from the battery and it is important not to plug in the USB cable (also used for power).

Set the switch on the shield to ‘OFF’. Remove the external power supply. Mount the shield to the board with a fully charged battery. Once the shield is attached, power on the board by setting the shield power to ‘ON’. When the board is fully booted, the LED 13 should be blinking.

33.5 Results

Arduino* 101 does not support a 5V power supply so it is not recommended to be powered by a 5V battery. The shield can be used for reading the voltage of a charge.

Charging batteries is not dependent on the board. The sketch that measures voltage compiles and uploads properly. In the example above, the sketch was run when the standard power supply was attached. It is possible to run the sketch with only battery power; however, this will require another means of communicating (Bluetooth, etc.) the data to the PC. This was not performed.

33.6 Next steps

- Tests to determine if voltage measurements from the battery are accurate.

§

34 MIFARE-One* RFID 13.56 MHz Keyfob

34.1 Use case

The MIFARE-One RFID keyfob (13.56 MHz) is widely used in electronic locks and customer identification as well as systems where a small and easy-to-carry tag is desired. The tag can be read by almost any 13.56 MHz RFID/NFC reader that can handle MIFARE cards. A passive tag, it is energized and activated by waves from an outside source.

Key Info	Links
Product Info	*http://www.SeeedStudio.com/depot/MifareOne-RFID-Tag-1356MHz-p-923.html*
Library	None.

34.2 Hardware summary

Key Info	Description/Links
IOREF	N/A.
Use VIN as power source	N/A.
Operating voltage	N/A.

34.3 Test

This RFID tag was tested while testing the Adafruit* NFC shield (#3). It was recognized as a MIFARE Classic card 1K with 4-byte UID, and it worked as expected during testing.

The MIFARE Classic 1K chip is created by NXP* specifically to be compatible with its hardware and not necessarily to adhere to NFC Forum protocols. The Nexus* 4, Nexus* 10, Samsung* Galaxy S4, and Nexus* 7 devices cannot write to or read anything that has been written to this tag. They can only read the UID (unique identifier). Other devices that do not use NFC hardware made by NXP may also have problems. The four devices mentioned above use Broadcom* NFC hardware that does adhere to the NFC Forum protocols.

34.4 Results

Arduino* 101 compatible

§

35 DFRobot* 2-Amp Motor Shield

Note: This is on the “List of Supported Shields” (*Link*).

35.1 Use case

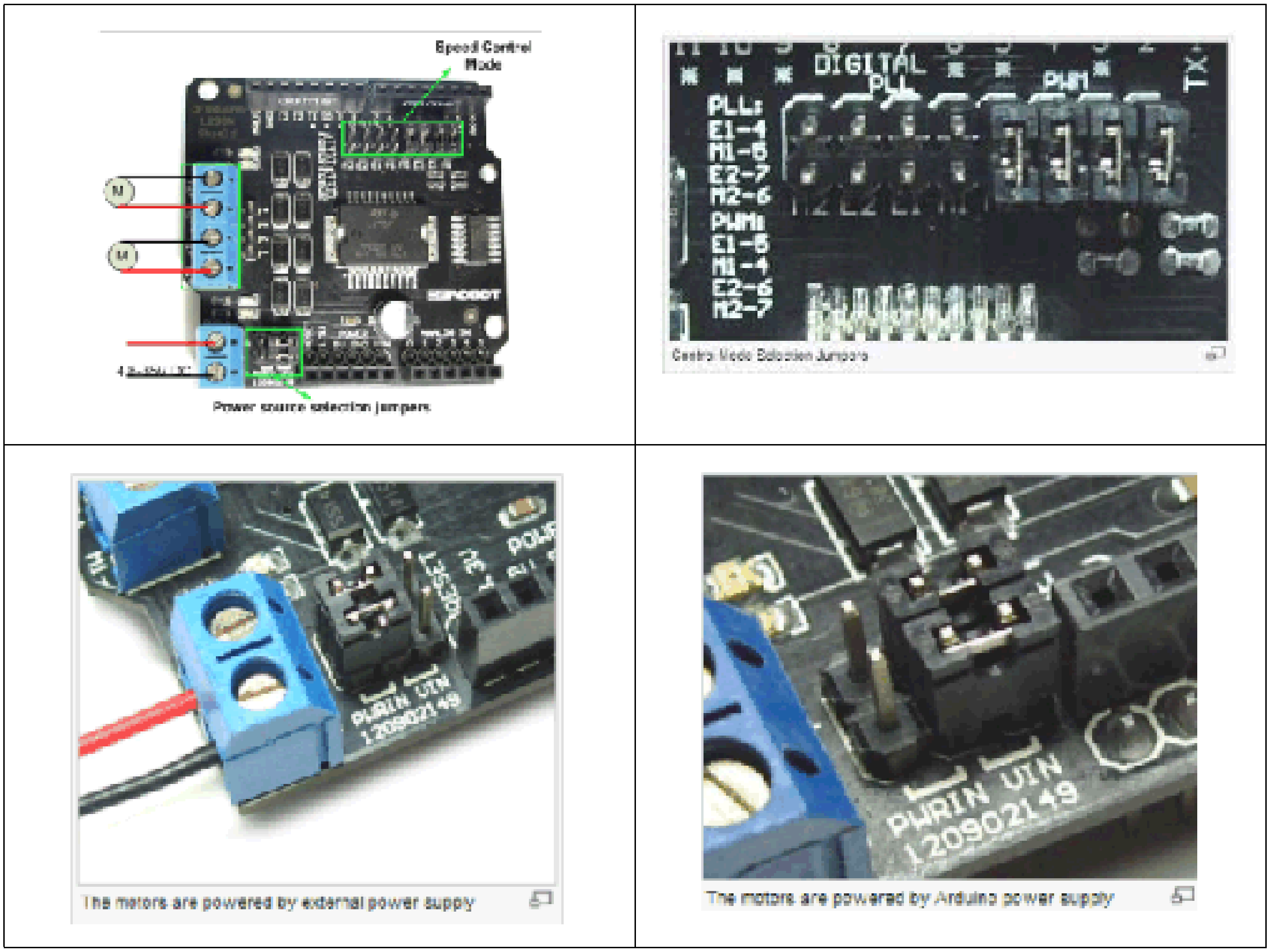
The DFRobot Arduino* Uno-compatible motor shield (2 A) uses the L298P chip which will drive two 5 to 12 VDC motors with maximum current of 2 A. The shield supports speed control by conventional PWM (pulse wave modulation) and PLL (phase locked loop) control modes. PLL mode should allow the motor to turn the same speed for variable loads.

Key Info	Links
Product info	*http://www.dfrobot.com/index.php?route=product/product&keyword=DRI0009&category_id=0&description=1&model=1&product_id=69#.U49LucRDuCl*
Library	None.
Guide	*http://www.dfrobot.com/wiki/index.php?title=Arduino_Motor_Shield_(L298N)_(SKU:DRI0009)*

35.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 to 12 V (logic control 5 V from the Intel® Galileo board).
Maximum current	2 A per channel.
IOREF	5 V, 3.3 V See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Motor controller	L298P *http://www.st.com/web/en/catalog/sense_power/FM142/CL851/SC1790/SS1555/PF63147*
IDE	1.6.8
Control mode	Pulse width modulation (PWM) on D5 and D6. Enable/disable function of the motor control is on D4 and D7. Phase locked loop (PLL) – set by jumpers on shield.
Motors	DC1, DC2.

Figure 86 **Connecting a 2-amp motor shield using an external power supply**



Control Mode Selection Jumpers

The motors are powered by external power supply

The motors are powered by Arduino power supply

35.3 Companion library

No library required.

35.4 Compile and upload

For Arduino* 101 - You need to disconnect the board before uploading the sketch

Example 41 **PWM speed control sketch**

```
//Arduino PWM Speed Control?
int E1 = 5; // M1, PWM
int M1 = 4; // M1, Enable/Disable function
int E2 = 6; // M2, PWM
int M2 = 7; // M2, Enable/Disable function

void setup()
{
//    setPwmSwizzler(3, 5, 10, 11);

    pinMode(M1, OUTPUT);
    pinMode(M2, OUTPUT);
}

void loop()
{
  int value=150;
  for(value = 0 ; value <= 255; value+=5)
  {
    digitalWrite(M1, HIGH);
    digitalWrite(M2, HIGH);
    analogWrite(E1, value);   //PWM Speed Control
    analogWrite(E2, value);   //PWM Speed Control
    delay(30);
  }
}
```

Example 42 **PLL speed control sketch**

```
//Arduino PLL Speed Control?
int E1 = 4;
int M1 = 5;
int E2 = 7;
int M2 = 6;

void setup()
{
    pinMode(M1, OUTPUT);
    pinMode(M2, OUTPUT);

   int value=150;
  for(value = 0 ; value <= 255; value+=5)
  {
    digitalWrite(M1,HIGH);
    digitalWrite(M2, HIGH);
    analogWrite(E1, value);   //PLL Speed Control
    analogWrite(E2, value);   //PLL Speed Control
    delay(30);
  }
}
void loop ()
{
}
```

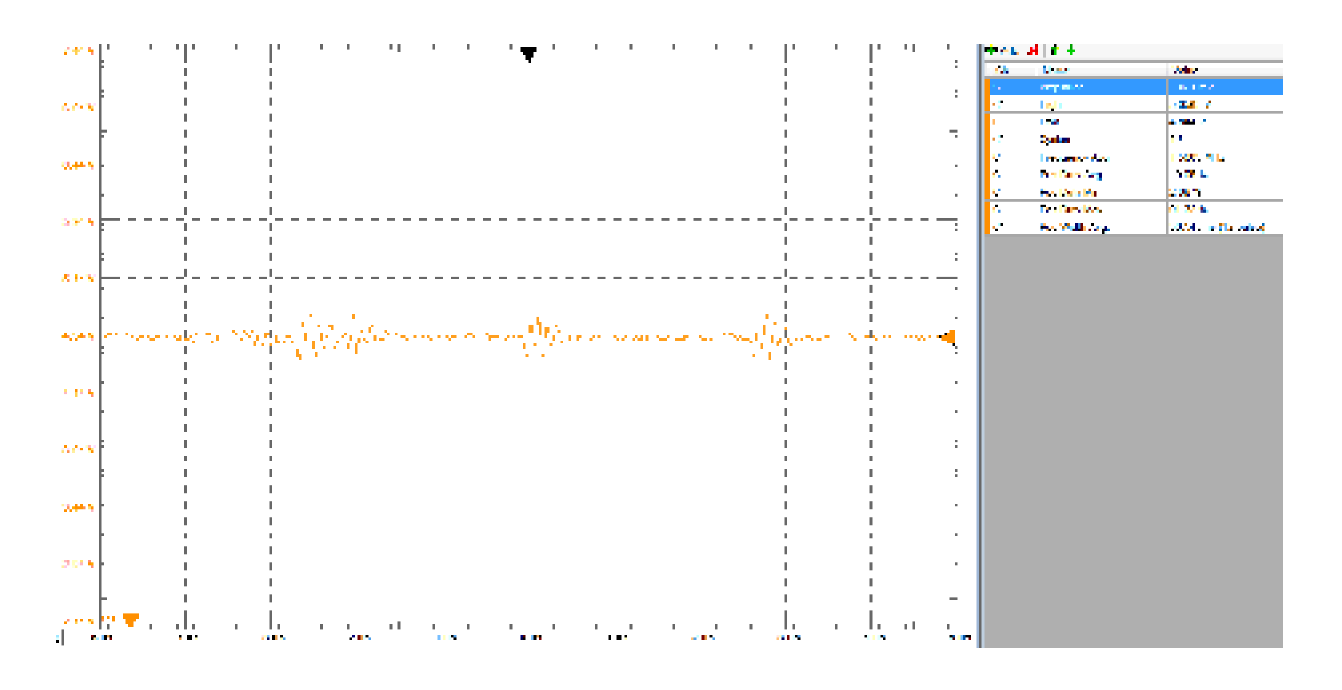


Figure 87 **Arduino* Uno board: PLL at 255 with a geared DC motor with no resistance applied**[1]

Notes:

1. Reading taken from pin D4.

35.5 Results

Arduino* 101 compatible for PWM and PLL modes

35.6 Next steps

- NA

§

36 Renbotics* Servoshield v2.0

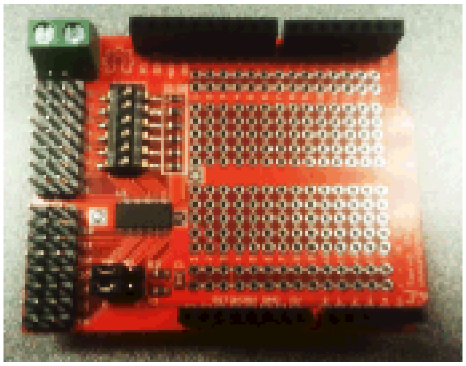
Note: This is on the “List of Supported Shields” (*Link*).

36.1 Use case

The Renbotics Servo Shield Rev 2 uses two pins to drive up to 16 servos per shield. The shield uses an NXP* PCA9685 over I^{2}C to provide 16 free-running servo outputs. The address of the board is selectable via a DIP switch, making it easy to stack up to 62 shields to control up to 992 servos, and 50 and 60 Hz modes are available. There is an easy to use API included for programming robotics, animatronics, and mechatronic art.

Key Info	Links
Product Info	*http://www.renbotics.com/servoshield2.php*
Library	*https://github.com/renbotics/Servo-Shield-2* up to date 3/31/2016
Guide	*https://github.com/renbotics/Servo-Shield-2/tree/master/Documentation*

Figure 88 **Renbotics* servoshield v2.0**



36.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V for shield, voltage for motor comes from an external source.
Maximum current	2 A per channel
Use VIN as power source	No
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Motor controller	NXP* PCA9685 *http://www.nxp.com/documents/data_sheet/PCA9685.pdf*
IDE	1.6.8
Pinout	Not specified, but there is a jumper selection to specify the I^{2}C address. The default address is 0x7F = 127d, therefore, the dip switch shall be set to all ‘OFF’. The ‘I2C SEL’ Jumpers are set to the ‘A’ side of the pins.
Servo tested	SV1

36.3 Companion library

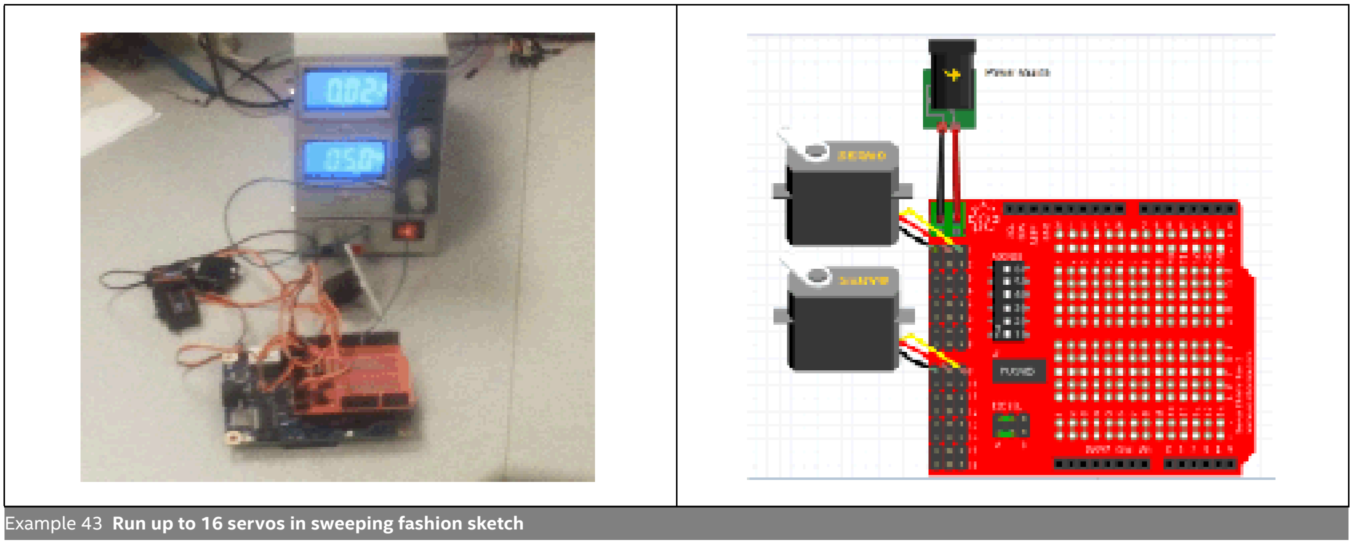
The library is called “ServoShield2”. The library works with no modifications.

36.4 Compile and upload

Servos are controlled by a pulse of variable width from the control wire. The pulse width is normally between 1 to 2 msec. The servo expects to see a pulse every 20 ms so the frequency is 50 Hz. This shield supports both 50Hz and 60Hz. The control of the servo shaft is by pulse width modulation, so the angle of movement is determined by the duration of the pulse. When a pulse is less than 1.5 ms, the servo rotates to a position some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms, the servo rotates counterclockwise.

Connect the shield to a 5 V power supply. The ground wire from the power supply should be connected to the side of the terminal nearest the edge of the board. Connect the servo motors to any row of pins with the ground side of the connector toward the edge of the board. It is possible to connect and run this sketch using 16 servo motors connected to the shield.

Figure 89 **Connecting the shield with an external power supply**

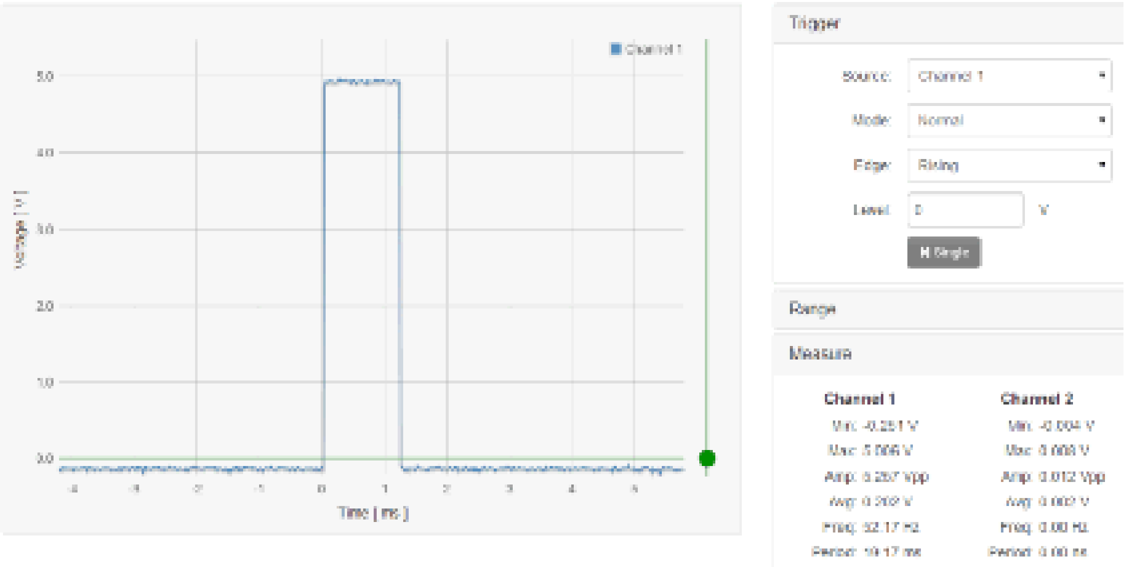


Example 43 **Run up to 16 servos in sweeping fashion sketch**

```
#include <Wire.h>
#include <ServoShield2.h>

ServoShield2 servos = ServoShield2(127);//Address of 127, using 50Hz mode

void setup() {
  Serial.begin(9600);
  Serial.println("Initializing...");
  servos.start();

  for (int servo = 0; servo < 16; servo++)//Initialize all 16 servos
  {
    servos.setbounds(servo, 1000, 2000);  // min and max pulse duration of the servo
    servos.setposition(servo, 1500);      //Set the initial position of the servo
  }
  Serial.println("Init Done");
}

void sweep()
{
  Serial.println("Sweeping");

  for(int pos = 1000; pos < 2000; pos += 20) //Move the servos from 0 to 180 degrees
  {
    for (int i = 0; i < 16; i++)
      servos.setposition(i, pos);
     delay(1);
  }
  for(int pos = 2000; pos >= 1000; pos -= 20) //Move the servos from 180 to 0 degrees
  {
    for (int i = 0; i < 16; i++)
      servos.setposition(i, pos);
    delay(1);
  }
}

void loop() {
  sweep();
}
```

Intel® Galileo first generation: The pulse frequency is 52 Hz while the pulse width is within the 1 to 2 ms range. The ranges tested were within valid ranges for servos.

Figure 90 **Intel® Galileo first generation board at 52 Hz**



Trigger

Source: Channel 1

Mode: Normal

Edge: Rising

Level: 0 V

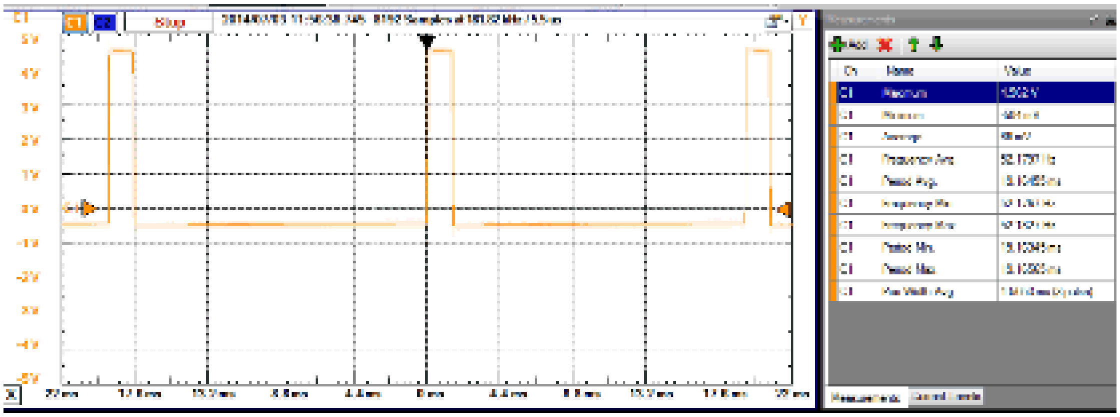
Range

Measure

Channel 1	Channel 2
Min: -0.251 V	Min: -0.004 V
Max: 5.006 V	Max: 0.008 V
Amp: 5.257 Vpp	Amp: 0.012 Vpp
Avg: 0.202 V	Avg: 0.002 V
Freq: 52.17 Hz	Freq: 0.00 Hz
Period: 19.17 ms	Period: 0.00 ns

Intel® Galileo second generation: Pulse width and frequency is within permissible ranges and servos are running smoothly.

Figure 91 **Intel® Galileo second generation board, frequency within permissible ranges**



Ch	Name	Value
C1	Maximum	4.502 V
C1	Minimum	[illegible]
C1	Average	98 mV
C1	Frequency Avg	52.1797 Hz
C1	Period Avg	19.16465 ms
C1	Frequency Min	52.1781 Hz
C1	Frequency Max	52.1821 Hz
C1	Period Min	19.16345 ms
C1	Period Max	19.16505 ms
C1	Pos Width Avg	[illegible]

36.5 Results

Arduino* 101 compatible. The Emax Motor only goes 90 degress.

§

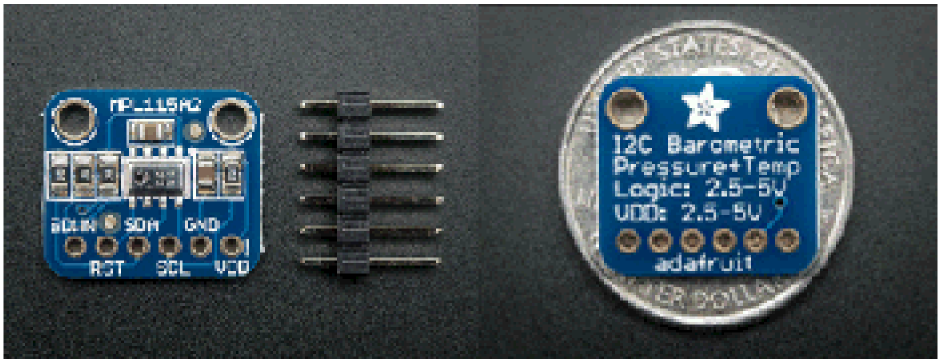
37 Adafruit* MPL115A2 Barometric Pressure/ Temperature I²C Sensor

37.1 Use case

Pressure and temperature sensor at 1.5 hPa (hectopascal) resolution with a range from 500 to 1150 hPa (up to 10 km altitude). It is great for basic barometric pressure sensing, but it is not recommended as a precision altimeter. There is no specification in the datasheet on the temperature accuracy.

Key Info	Links
Product info	*https://www.adafruit.com/products/992*
Library	*https://github.com/adafruit/Adafruit_MPL115A2* Name: Adafruit_MPL115A2

Figure 92 **Adafruit* barometric pressure/temperature I²C sensor**

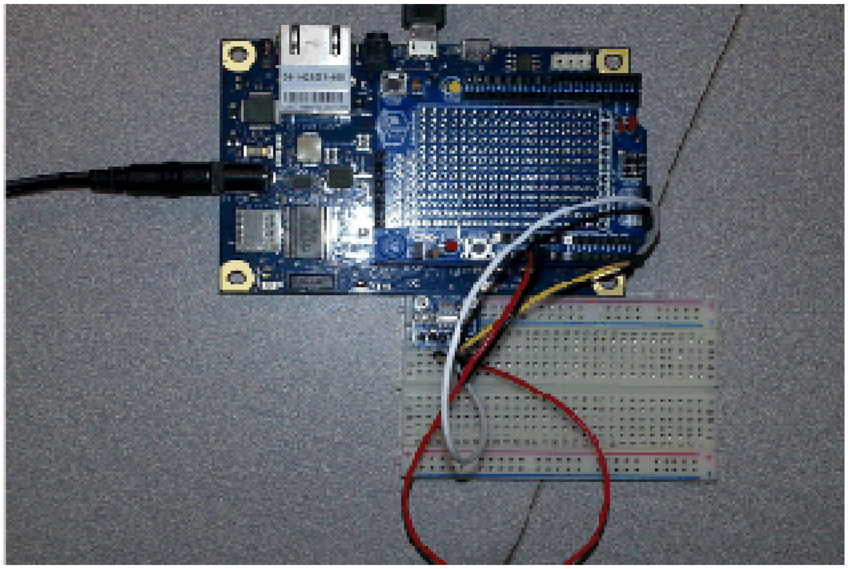


37.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 or 5 V
Use VIN as power source	No.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
IDE	1.5.8-Intel.1.0.5
Schematic	*http://www.adafruit.com/datasheets/MPL115A2.pdf*

Pin Name	Function
SDWN	Shutdown, not used in this example.
RST	Reset, not used for this example.
SDA	I²C, Serial Data Line - connect to A4.
SCL	I²C, Serial Clock Line - connect to A5.
Device address	I²C, Adafruit_MPL115A2.h defines the address as: #define MPL115A2_ADDRESS (0x60)
GND	Ground - connect to GND.
VDD	VDD power supply connection: range is 2.375 V to 5.5 V - connect to 5 V for this example.

Figure 93 **Adafruit* barometric pressure/temperature I²C sensor on an Intel® Galileo first generation board**



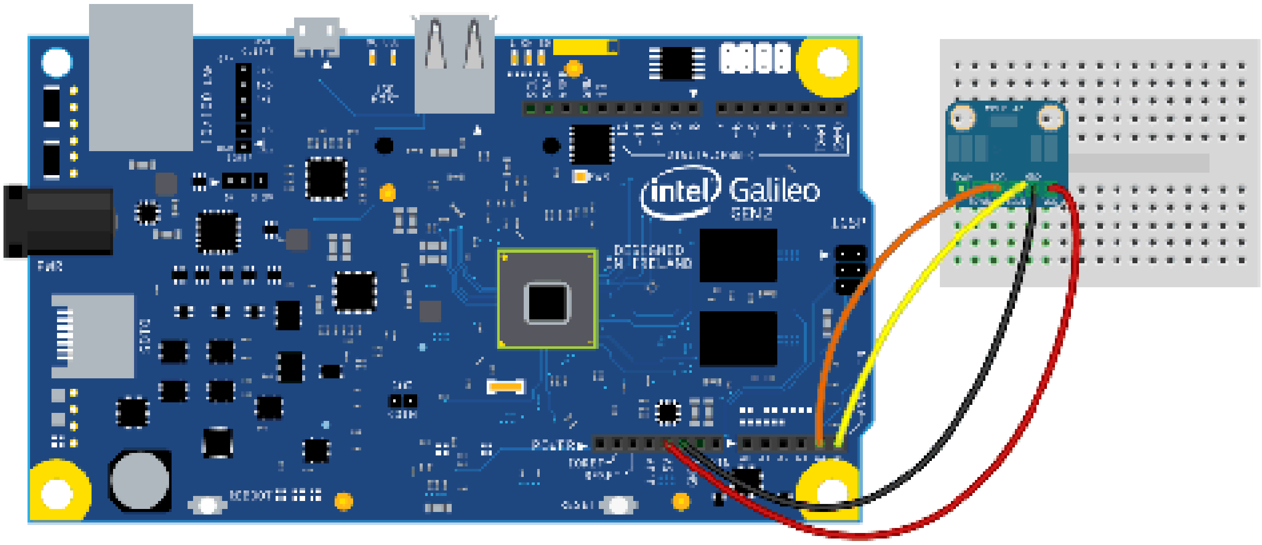
37.3 Companion library

The companion library compiles on both Arduino* Uno and Arduino* 101 boards. Two sketches are present in the library called 'getpressure' and 'getPT'.

37.4 Compile and upload

Connect the sensor as follows:

Figure 94 **Connecting an Adafruit* barometric pressure/temperature I²C sensor**



The setup will print the temperature and pressure in the serial console. Upload the following sketch and open a serial terminal for 9600 baud. The temperature and pressure values are displayed to the serial console.

Example 44 **getPT sketch**

```
#include <Wire.h>
#include <Adafruit_MPL115A2.h>
Adafruit_MPL115A2 mpl115a2;
void setup(void)
{
  Serial.begin(9600);
  Serial.println("Hello!");

  Serial.println("Getting barometric pressure ...");
  mpl115a2.begin();
}
void loop(void)
{
  float pressureKPA = 0, temperatureC = 0;
  mpl115a2.getPT(&pressureKPA,&temperatureC);
  Serial.print("Pressure (kPa): "); Serial.print(pressureKPA, 4); Serial.print(" kPa  ");
  Serial.print("Temp (*C): "); Serial.print(temperatureC, 1); Serial.println(" *C both
measured together");

  pressureKPA = mpl115a2.getPressure();
  Serial.print("Pressure (kPa): "); Serial.print(pressureKPA, 4); Serial.println(" kPa");

  temperatureC = mpl115a2.getTemperature();
  Serial.print("Temp (*C): "); Serial.print(temperatureC, 1); Serial.println(" *C");

  delay(1000);
}
```

37.5 Results

Arduino* 101 pass

§

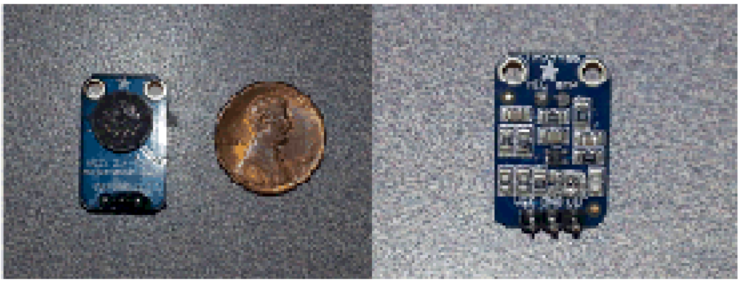
38 Adafruit* Electret Microphone Amplifier MAX4466

38.1 Use case

The microphone uses an op-amp for amplification. The output pin is not designed to drive anything more than the smallest in-ear headphones. Frequency response is from 20 Hz to 20 kHz. Adjustable gain is from 25x to 125x.

Key Info	Links
Product info	*http://www.adafruit.com/products/1063?gclid=CMXT_KeW2b0CFU1bfgodRnkA_g*
Library	No library.
Guide	Gain adjustment: *https://learn.adafruit.com/adafruit-microphone-amplifier-breakout/measuring-sound-levels*
Tone generator	*http://www.ringbell.co.uk/software/audio.htm* *http://onlinetonegenerator.com*

Figure 95 **Adafruit* MAX4466 electret microphone amplifier**



38.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 or 5 V
Use VIN as power source	No.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
IDE	1.5.8-Intel.1.0.5
MAX4466	Low-noise microphone amp datasheet. *http://www.adafruit.com/datasheets/MAX4465-MAX4469.pdf*
CMA-4544PFW	Electret capsule microphone datasheet. *http://www.adafruit.com/datasheets/CMA-4544PF-W.pdf*

Pin Name	Function
OUT	Connect to pin A0, audio out.
GND	Connect to GND pin.
VCC	Connect to 3.3 or 5 V. Power supply (2.4 to 5 V)

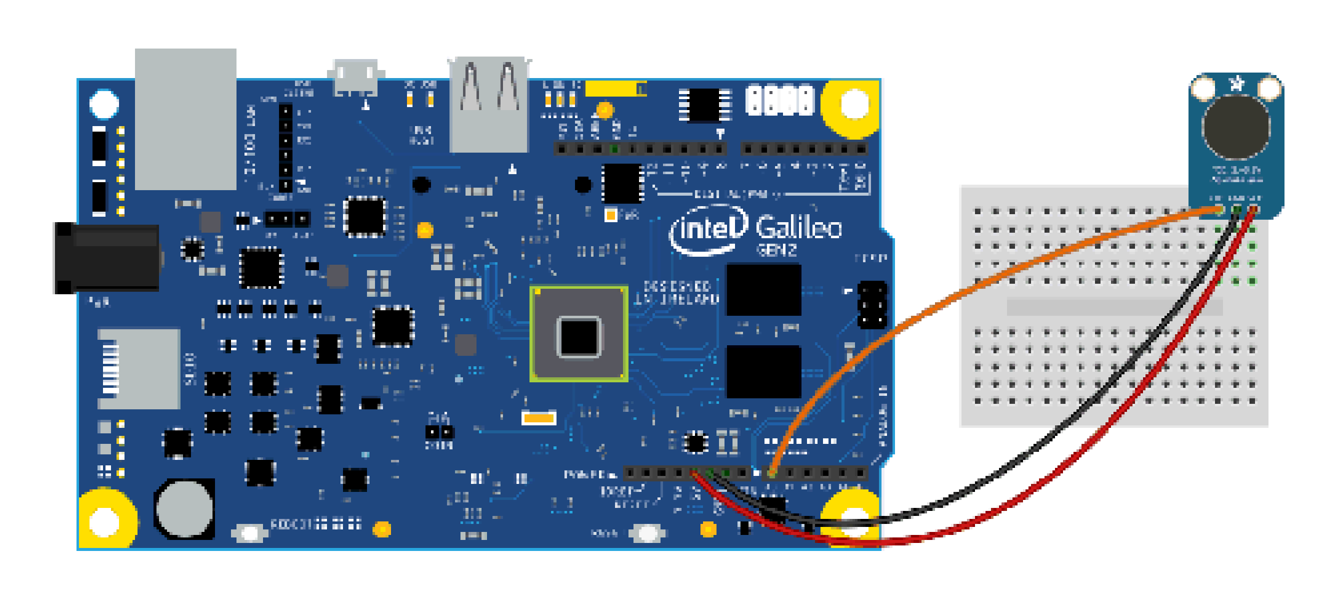
38.3 Companion library

No library.

38.4 Compile and test

Connect output of microphone to A0 (no adjustment to the gain was done), download the sketch below and bring up the serial console. The level of audio is measured and displayed on the serial console. Notice that the level changes according to voice or other tones input to microphone.

Figure 96 **Connecting an Adafruit* MAX4466 electret microphone amplifier . Shown here with an Intel® Galileo second generation board**

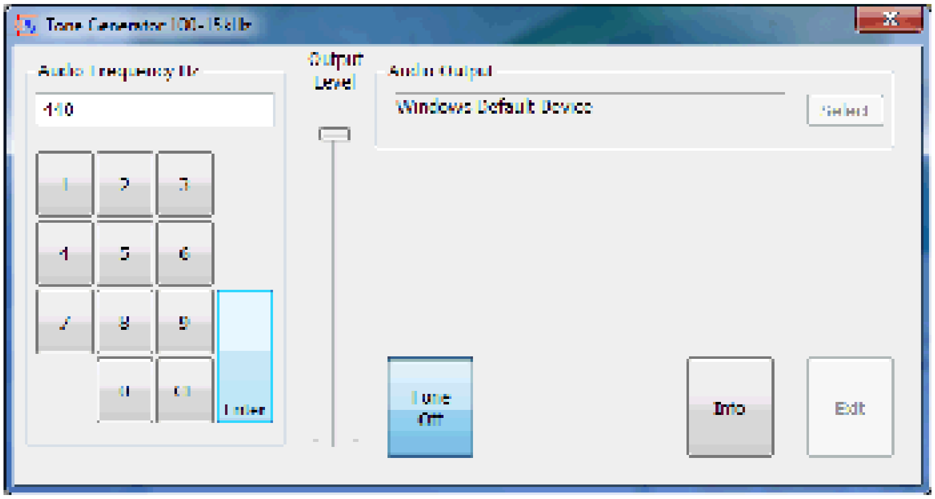


Example 45 **Sound level sketch**

```
/****************************************
Example Sound Level Sketch for the
Adafruit Microphone Amplifier
****************************************/
const int sampleWindow = 50; // Sample window width in mS (50 mS = 20Hz)
unsigned int sample;

void setup()
{
    Serial.begin(9600);
    Serial.println("Begin listening\n");
}

void loop()
{
    unsigned long startMillis= millis();  // Start of sample window
    unsigned int peakToPeak = 0;   // peak-to-peak level
    unsigned int signalMax = 0;
    unsigned int signalMin = 1024;

    // collect data for 50 mS
    while (millis() - startMillis < sampleWindow)
    {
        sample = analogRead(0);
        if (sample < 1024)  // toss out spurious readings
        {
            if (sample > signalMax)
            {
                signalMax = sample;  // save just the max levels
            }
            if (sample < signalMin)
            {
                signalMin = sample;  // save just the min levels
            }
        }
    }
    peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
    double volts = (peakToPeak * 3.3) / 1024;
    // convert to volts
    Serial.println(volts);
}
```

Next, analyze the signal with an oscilloscope by connecting OUT from microphone to the oscilloscope probe. Start the Tone Generator program and set the frequency as 440 Hz. Click the 'Enter' button to start the tone.

Figure 97 **Tone Generator window**



Tone Generator 100-15kHz

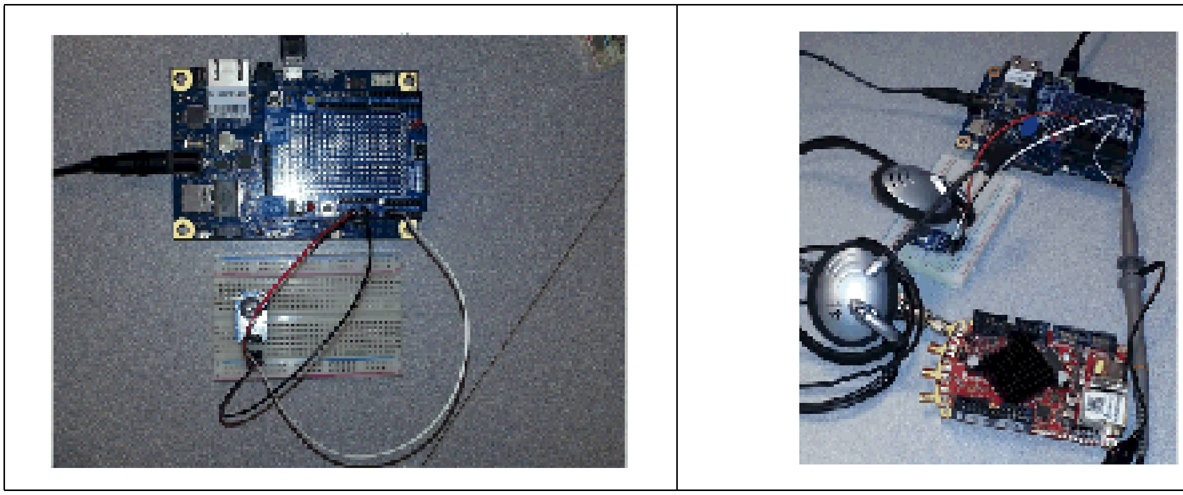
Audio Frequency Hz: 440

Audio Output: Windows Default Device

A 440 Hz tone is generated through the headphone and picked up by the microphone. The best method to prove the pickup is by capturing the tone with an oscilloscope.

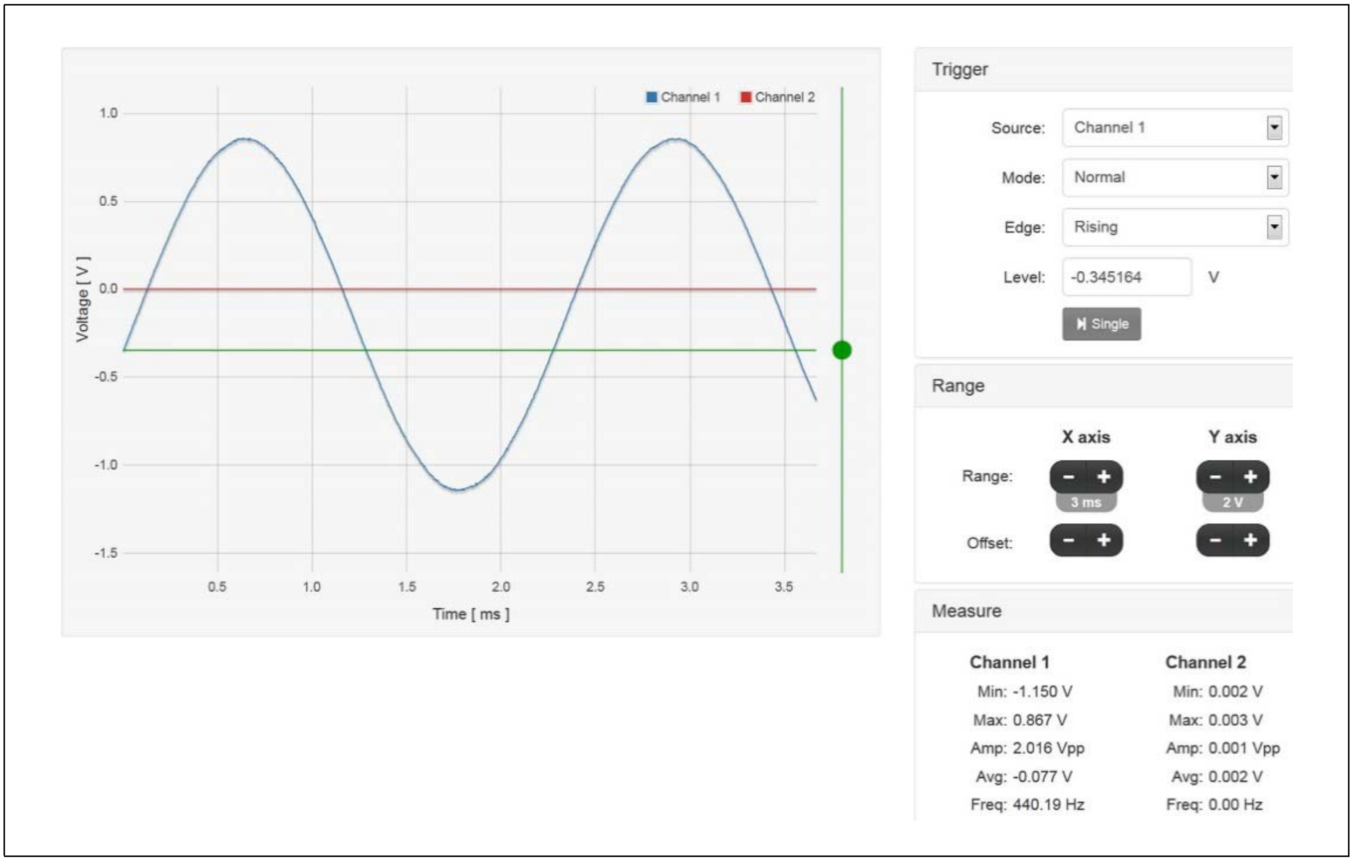
Connect OUT to A0 to see the levels change in the serial console (Figure 111).

Figure 98 **Watch the levels change in the serial console**



Connect OUT to an oscilloscope and play a signal through the headphones to the microphone (*Figure 99*).

Figure 99 **Oscilloscope signal**



Trigger

Source: Channel 1

Mode: Normal

Edge: Rising

Level: -0.345164 V

Range

	X axis	Y axis
Range:	3 ms	2 V
Offset:		

Measure

Channel 1	Channel 2
Min: -1.150 V	Min: 0.002 V
Max: 0.867 V	Max: 0.003 V
Amp: 2.016 Vpp	Amp: 0.001 Vpp
Avg: -0.077 V	Avg: 0.002 V
Freq: 440.19 Hz	Freq: 0.00 Hz

38.5 Results

Arduino* 101 compatible

§

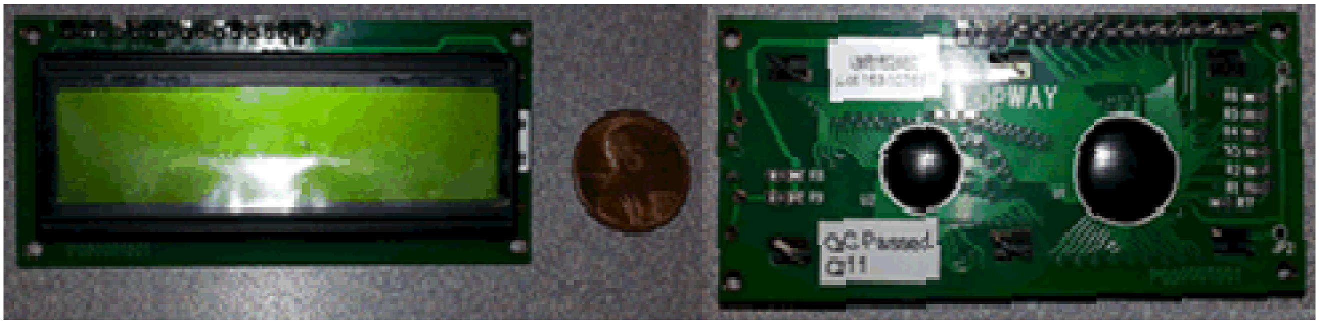
39 Topway* LCD LMB162ABC

39.1 Use case

This LCD is a 16 × 2 character (5 × 8 dot) panel. It supports all the LCD demos in the IDE release (Autoscroll, Blink, Custom Character, changing text direction, etc). Display colors are deep blue and yellow-green with a yellow-green backlight.

Key Info	Links
Product info	*https://www.seeedstudio.com/depot/datasheet/LMB162ABC-Manual-Rev0.2.pdf*
Library	No extra library; the LCD library that installs with the platform is used.

Figure 100 **Topway* LCD LMB162ABC**

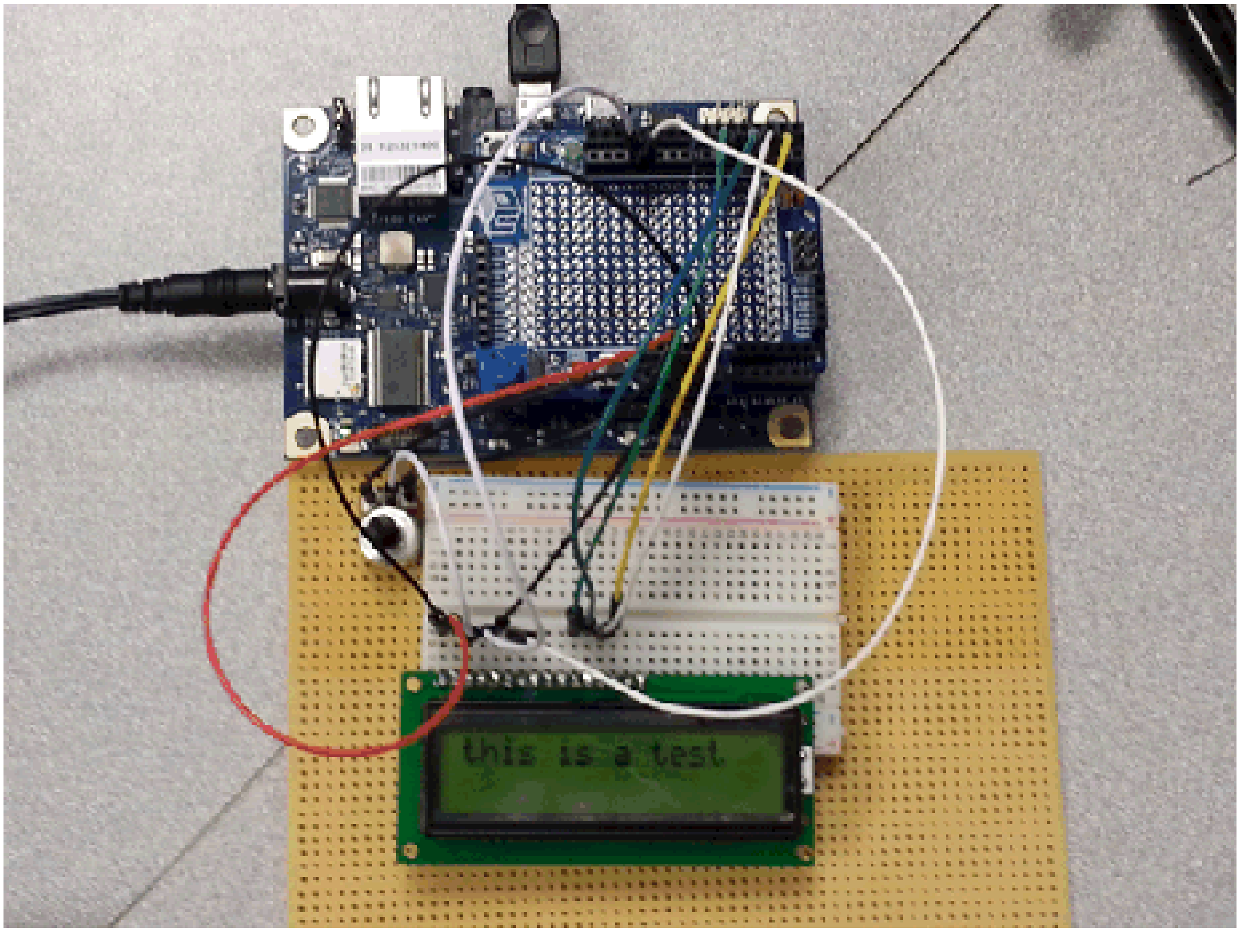


39.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	No.
Schematic	*http://www.SeeedStudio.com/depot/datasheet/LMB162ABC-Manual-Rev0.2.pdf*.

LCD Pin Name	Function/connection
VSS	Ground – connect to GND.
VDD	Power – connect to 5 V.
V0	LCD contrast reference supply – connect to middle pin of potentiometer wiper.
RS	Register select – connect to D12 – Low: transfer display data High: transfer instruction data.
R/W	Read/write control bus – connect to GND.
E	Data enable – connect to D11
07 through 10 not used 11 through 14 DB4 – DB7	Data bus – connect for 4 bit mode. In 4 bit mode, only DB4 through DB7 are used 11->D5, 12->D4, 13->D3 14->D2 for 4 bit mode In 8-bit mode, DB0 through DB7 are used.
BLA	Backlight positive supply – not using for this example.
BLK	Backlight negative supply – not using for this example.

Figure 101 **Topway* LCD LMB162ABC on an Intel® Galileo first generation board**



39.3 Companion library

No additional library required. This LCD uses the *LiquidCrystal* library included in the IDE.

39.4 Compile and upload

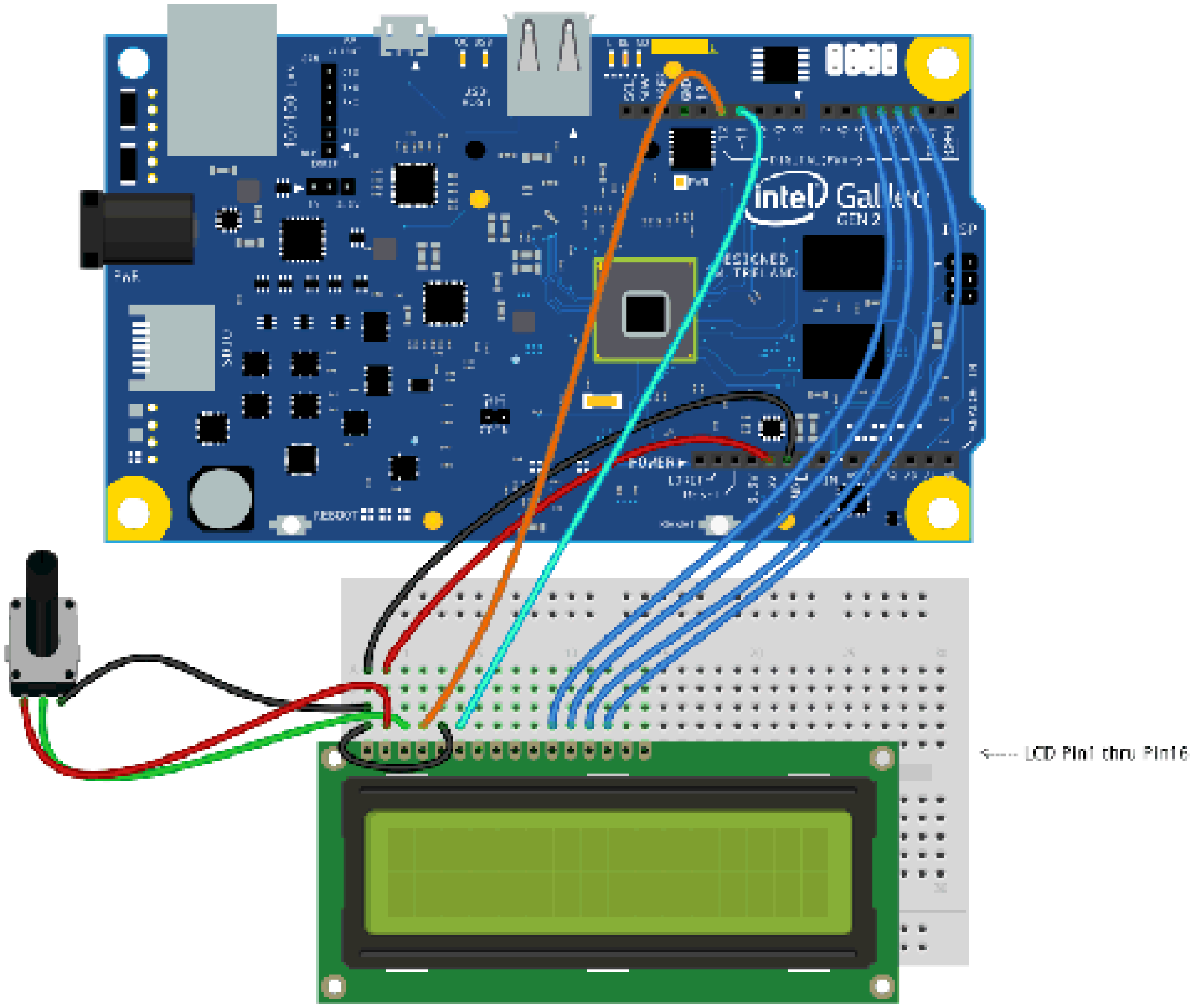
Connect the LCD to the board for 4-bit mode. This connection includes a potentiometer to control the contrast for the display.

Connections with potentiometer:

The pinout table describes the connection in the picture below. In addition to the pinout table in the hardware section, the following connections shall be made to control the contrast for the sketch:

- One end (not middle) of potentiometer to 5 V.
- Other end (not middle) of potentiometer to GND.

Figure 102 **Topway* LCD LMB162ABC connection, show here with an Intel® Galileo second generation board**



<---- LCD Pin1 thru Pin16

Run the demos that come with the IDE. Load the *HelloWorld* sketch. A counter and a text ‘Hello World’ are displayed.

39.5 Results

Arduino* 101 compatible.

§

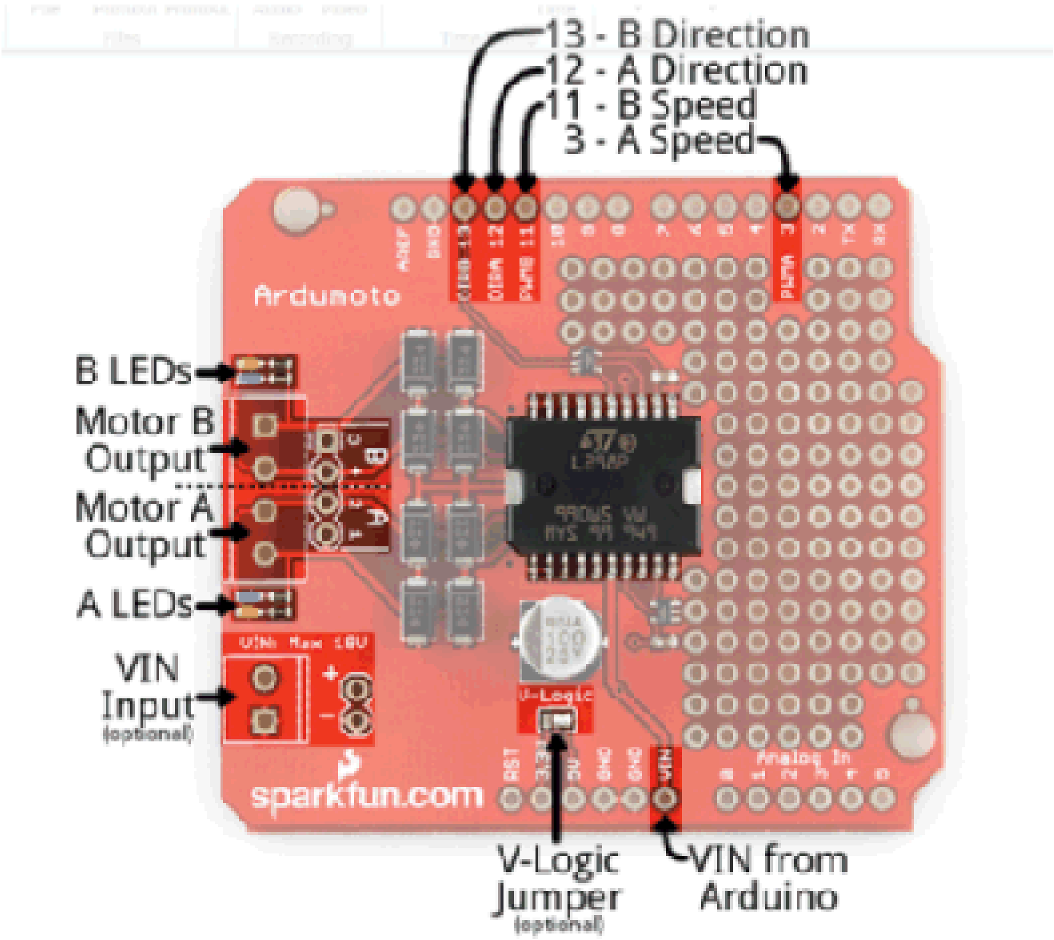
40 SparkFun* Ardumoto Motor-Driver Shield

40.1 Use case

The Ardumoto motor-driver shield is a motor shield that will control two DC motors. It is based on the L298 H-bridge, and it can drive up to 2 amperes per channel. It can be used to power an RC car or truck using a battery pack. The board is good for two-wheel drive vehicles.

Key Info	Links
Product info	*https://www.sparkfun.com/products/9815*.
Library	None.
Sample sketch	*http://dlnmh9ip6v2uc.cloudfront.net/downloads/Ardumoto/ardumoto_example.ino*.
Motors used	DC1, DC2

Figure 103 **SparkFun* Ardumoto motor-driver shield**



Motors tend to draw a lot of current so driving the motor from the Arduino Uno board's pins is not a good idea. The Ardumoto lets you control a higher level of current with a small signal. The board can spin motors clockwise or counter-clockwise and can use PWM (pulse width modulation) to control the speed.

40.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	Yes, it is possible to hook up an external source to the shield's VIN connection.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Motor driver	L298 H-bridge *https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf*.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematic	*https://www.sparkfun.com/datasheets/DevTools/Arduino/Ardumoto_v13.pdf*.

40.3 Compile and upload

Figure 104 **Connecting the shield without external power. Shown here with an Intel® Galileo first generation board**

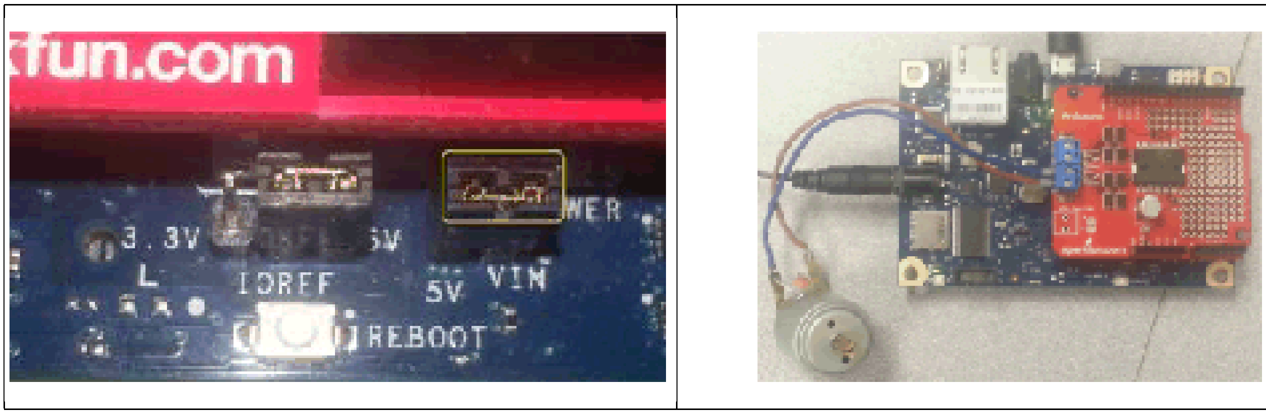


Figure 106 **Connecting the shield with external power. Shown here with an Intel® Galileo second generation board**

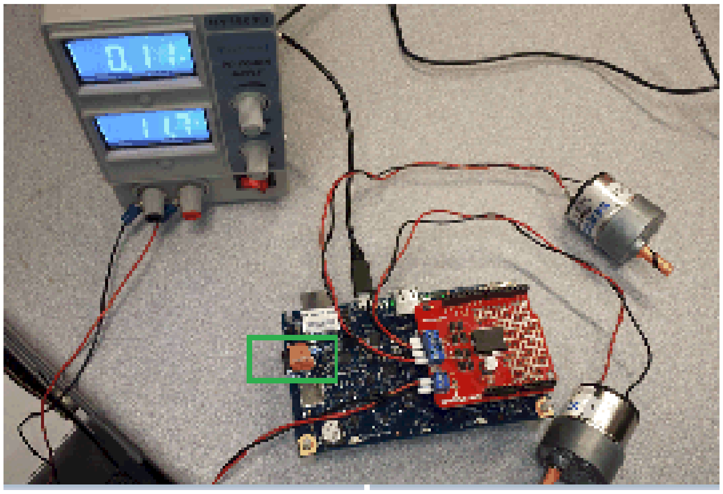


Figure 107 **Intel® Edison board PWM swizzle jumper settings to enable PWM on D03 and D11**

Example 46 modifies the program in the link to hold specific speeds for a longer length of time so that PWM measurements could be taken. Connect the DC motors to channel A and B, then set the vlogic jumper to 5 V.

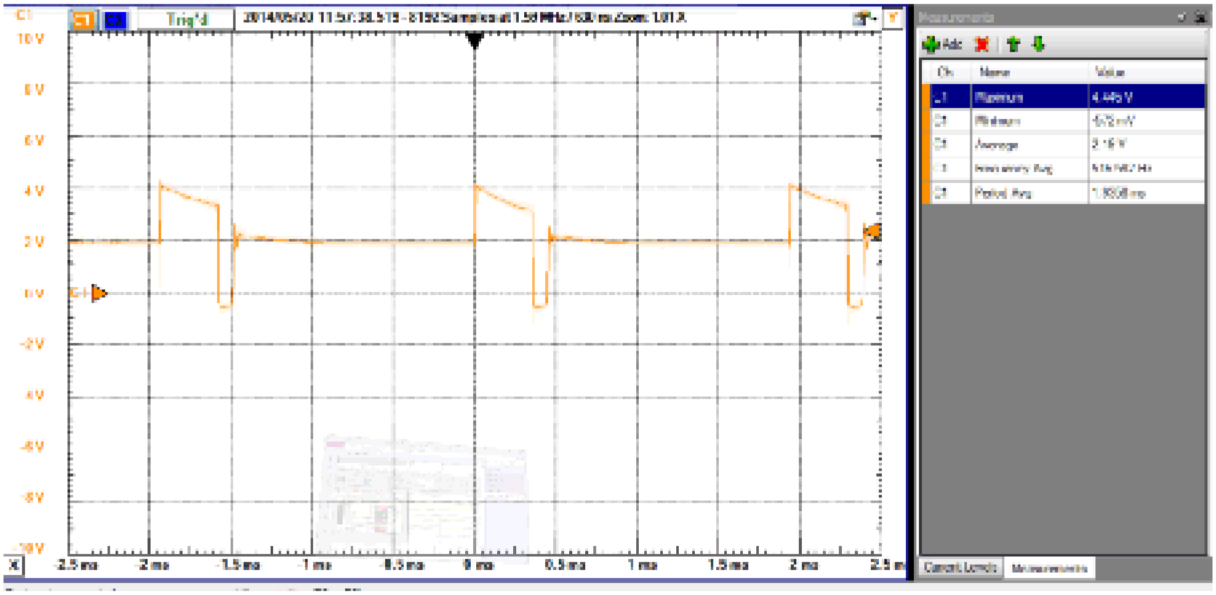
Example 46 **Modified PWM measurement sketch**

```
int pwm_a = 3;   //PWM control for motor outputs 1 and 2 is on digital pin 3
int pwm_b = 11;  //PWM control for motor outputs 3 and 4 is on digital pin 11
int dir_a = 12;  //direction control for motor outputs 1 and 2 is on digital pin 12
int dir_b = 13;  //direction control for motor outputs 3 and 4 is on digital pin 13
int val = 0;     //value for fade

void setup()
{
  // uncomment setPwmSwizzler() for Edison, leave commented for Galileo
  // setPwmSwizzler(3, 5, 10, 11);

  pinMode(pwm_a, OUTPUT);  //Set control pins to be outputs
  pinMode(pwm_b, OUTPUT);
  pinMode(dir_a, OUTPUT);
  pinMode(dir_b, OUTPUT);

  analogWrite(pwm_a, 100);  //set both motors to run at (100/255 = 39)% duty cycle (slow)
  analogWrite(pwm_b, 100);

}

void loop()
{
  forw();         //Set Motors to go forward Note : No pwm is defined with the for function,
so that fade in and out works
  slow();
  delay(1000*1);
  stopped();
  delay(1000);

  forw();
  fadein();       //fade in from 0-255
  delay(1000);
  forward();      //continue full speed forward
  delay(1000);
  fadeout();      //Fade out from 255-0
  delay(1000);    //Wait one second

  stopped();      // stop for 2 seconds
  delay(2000);


  back();         //Set motors to revers. Note : No pwm is defined with the back function, so
that fade in and out works
  fadein();       //fade in from 0-255
  delay(1000);
  backward();     //full speed backward
  delay(1000);
  fadeout();      //Fade out from 255-0
  delay(1000);
}

/* The forw and back functions are simply designating the direction the motors will turn once
they are fed a PWM signal.
If you only call the forw, or back functions, you will not see the motors turn. On a similar
note the fade in and out functions will only change PWM, so you need to consider
the direction you were last set to.
*/
```

```
void slow()
{
  analogWrite(pwm_a, 50);
  analogWrite(pwm_b, 50);
}

void forw() // no pwm defined
{
  digitalWrite(dir_a, HIGH);  //Reverse motor direction, 1 high, 2 low
  digitalWrite(dir_b, HIGH);  //Reverse motor direction, 3 low, 4 high
}

void back() // no pwm defined
{
  digitalWrite(dir_a, LOW);  //Set motor direction, 1 low, 2 high
  digitalWrite(dir_b, LOW);  //Set motor direction, 3 high, 4 low
}

void forward() //full speed forward
{
  digitalWrite(dir_a, HIGH);  //Reverse motor direction, 1 high, 2 low
  digitalWrite(dir_b, HIGH);  //Reverse motor direction, 3 low, 4 high
  analogWrite(pwm_a, 255);    //set both motors to run at (100/255 = 39)% duty cycle
  analogWrite(pwm_b, 255);
}

void backward() //full speed backward
{
  digitalWrite(dir_a, LOW);  //Set motor direction, 1 low, 2 high
  digitalWrite(dir_b, LOW);  //Set motor direction, 3 high, 4 low
  analogWrite(pwm_a, 255);   //set both motors to run at 100% duty cycle (fast)
  analogWrite(pwm_b, 255);
}

void stopped() //stop
{
  digitalWrite(dir_a, LOW); //Set motor direction, 1 low, 2 high
  digitalWrite(dir_b, LOW); //Set motor direction, 3 high, 4 low
  analogWrite(pwm_a, 0);    //set both motors to run at 100% duty cycle (fast)
  analogWrite(pwm_b, 0);
}

void fadein()
{
  // fade in from min to max in increments of 5 points:
  for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5)
  {
     // sets the value (range from 0 to 255):
    analogWrite(pwm_a, fadeValue);
    analogWrite(pwm_b, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}

void fadeout()
{
  // fade out from max to min in increments of 5 points:
  for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5)
  {
    // sets the value (range from 0 to 255):
    analogWrite(pwm_a, fadeValue);
    analogWrite(pwm_b, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}

void astop()                   //stop motor A
{
  analogWrite(pwm_a, 0);    //set both motors to run at 100% duty cycle (fast)
}
void bstop()                   //stop motor B
{
  analogWrite(pwm_b, 0);    //set both motors to run at 100% duty cycle (fast)
}
```

With a DC motor connected to channel A, the shield outputs this PWM when the speed is set to 50. The motor appears to turn over at a reasonable speed for 50/255. The plot below was obtained using the Intel Galileo power supply to provide power for a 5 V motor. This reading was taken at the output from the shield to the motor.

Figure 108 **Intel® Galileo first generation board PWM oscilloscope output with speed at 50**



2014/05/20 11:57:38.515 - 8192 Samples at 1.58 MHz / 630 ms Zoom: 1.01 X

Ch	Name	Value
C1	Maximum	4.445 V
C1	Minimum	-572 mV
C1	Average	2.16 V
C1	Frequency Avg	[illegible] Hz
C1	Period Avg	1.9[illegible] ms

40.4 Results

Arduino* 101 compatible.

The shield was tested to verify that it could run with two motors, change speeds, and change direction.

§

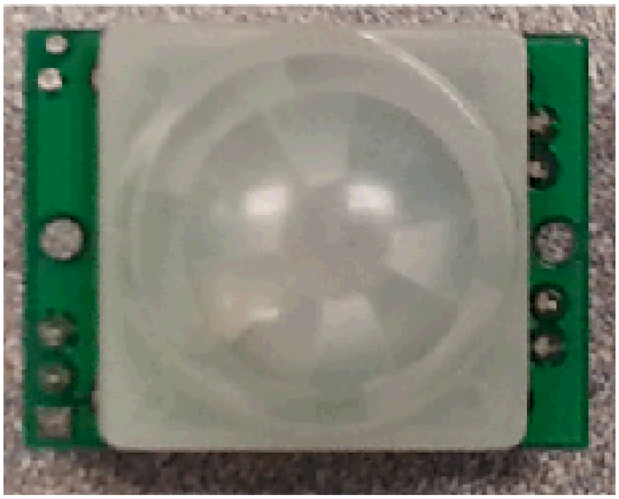
41 DFRobot* Digital Infrared Motion Sensor

41.1 Use case

This is a simple-to-use motion sensor. Power it up and wait 1 to 2 seconds for the sensor to get a snapshot of the still room. If anything moves after that period, the digital signal out pin will go high. There is a jumper to change the sensitivity between low and high.

Key Info	Links
URL	*http://www.dfrobot.com/wiki/index.php/Digital_Infrared_motion_sensor_(SKU:SEN0018)*.
Library	Not required.

Figure 109 **Digital infrared motion sensor**



41.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IOREF	No.
Use VIN as power source	No.
Operating voltage	5 V

Pin Name	Function
-	Connect to GND.
+	Connect to 5 V.
OUT	Connect to D2.

Figure 110 **Digital infrared motion sensor pins**

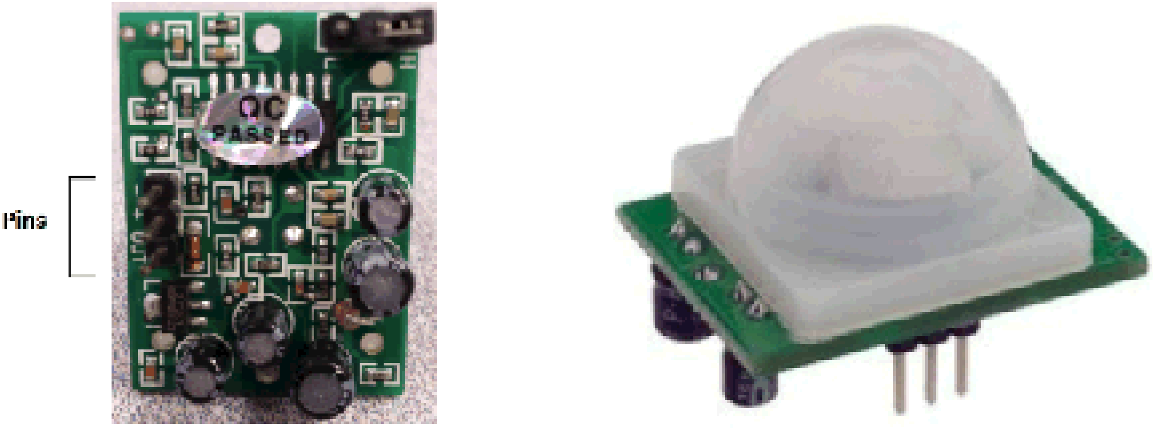
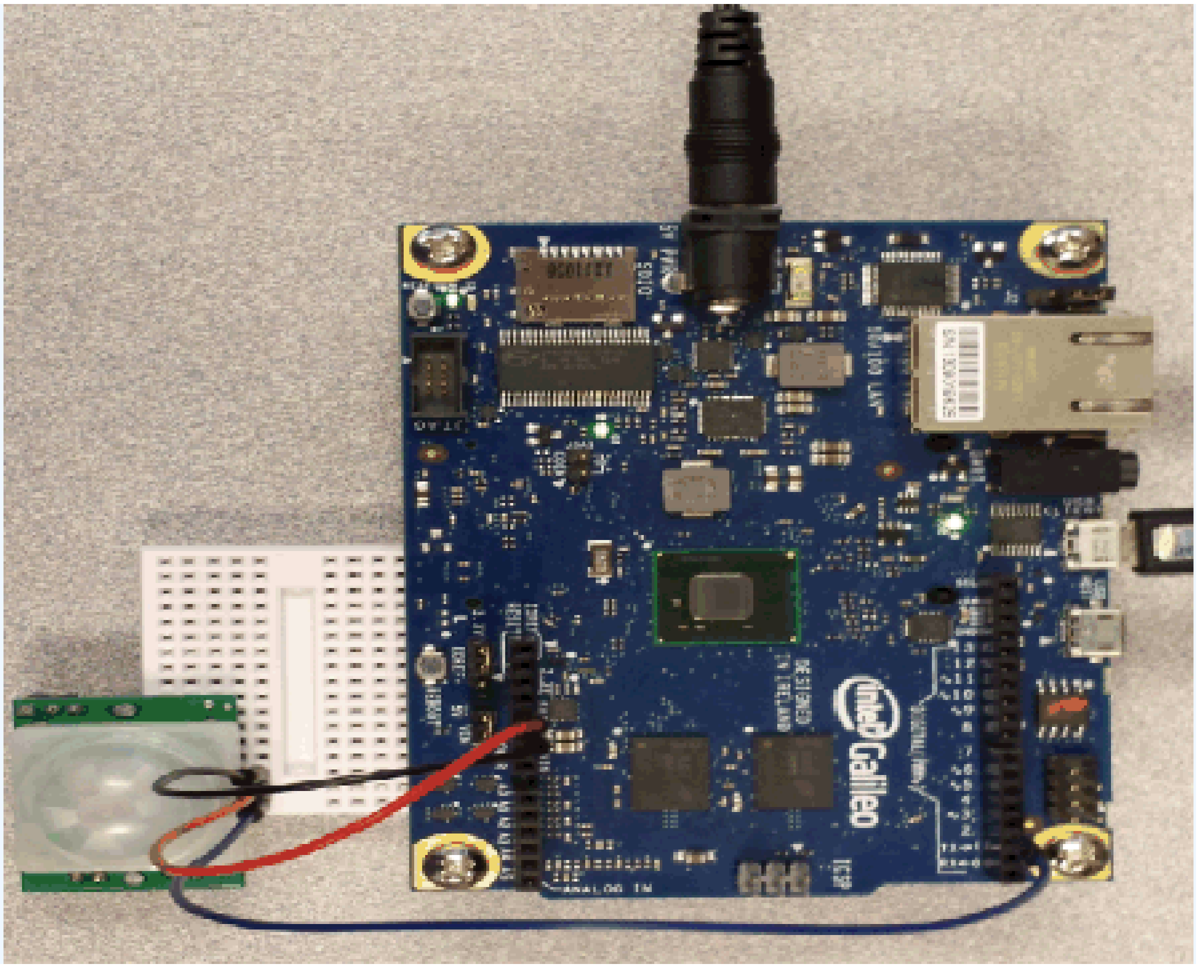


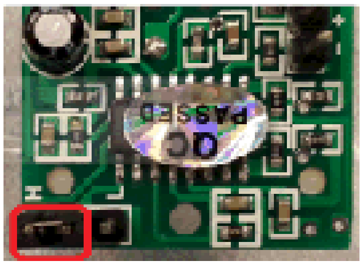
Figure 111 **Connecting a digital infrared motion sensor. Shown here with an Intel® Galileo first generation board**



41.3 Compile and upload

Set the jumper for high or low sensitivity, depending on the application. The board in *Figure 112* is jumpered for High (H) sensitivity.

Figure 112 **High sensitivity jumper**



Wire the sensor as displayed in *Figure 112* and download the example sketch. *Example 47* shows an example sketch.

Example 47 **Motion sensor for the Intel® Galileo board**

```
// Motion sensor for Galileo, example code
// Light LED (13) when motion is detected
const byte ledPin    = 13;   // LED pin
const byte motionPin =  2;   // motion detector output pin
void setup()
{
  Serial.begin(9600);
  delay(1000*3);
  // set the digital pin directions
  pinMode(ledPin, OUTPUT);
  pinMode(motionPin, INPUT);

  digitalWrite(ledPin, LOW);
  Serial.println("...setup");
}
void loop()
{       // Now watch for motion
  int senseMotion = digitalRead(motionPin);
  Serial.print("Read=");
  Serial.println(senseMotion);
  if (senseMotion == 1)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

41.4 Results

Arduino* 101 compatible.

§

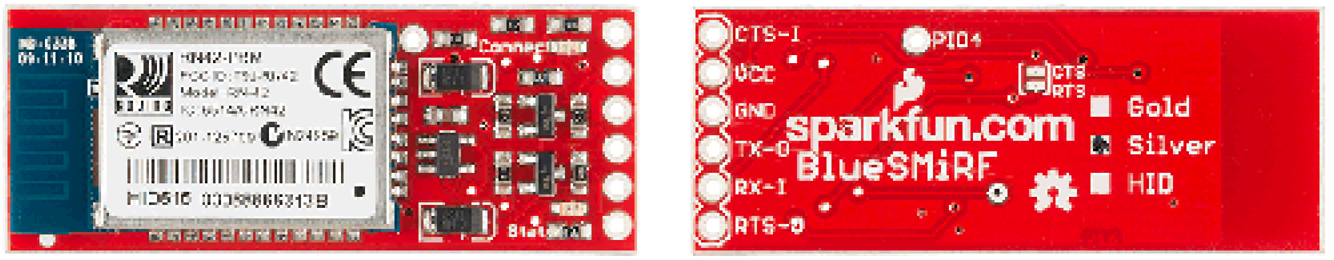
42 SparkFun* BlueSMiRF Silver – Bluetooth* Modem

42.1 Use case

This version of the popular BlueSMiRF uses the RN-42 module that has less range than the RN-41 module (used in the BlueSMiRF Gold). These modems work as a serial (Rx/Tx) pipe. Any serial stream from 2400 to 115,200 bps can be passed seamlessly from an Intel® Galileo board or Arduino* Uno board to your target. The remote unit can be powered from 3.3 to 6 V for easy battery attachment.

Key Info	Links
URL	*https://www.sparkfun.com/products/12577*.
Library	Not required.
Guide	*https://learn.sparkfun.com/tutorials/using-the-bluesmirf/hardware-overview*.

Figure 113 **SparkFun BlueSMiRF Silver Bluetooth modem**



42.2 Hardware summary

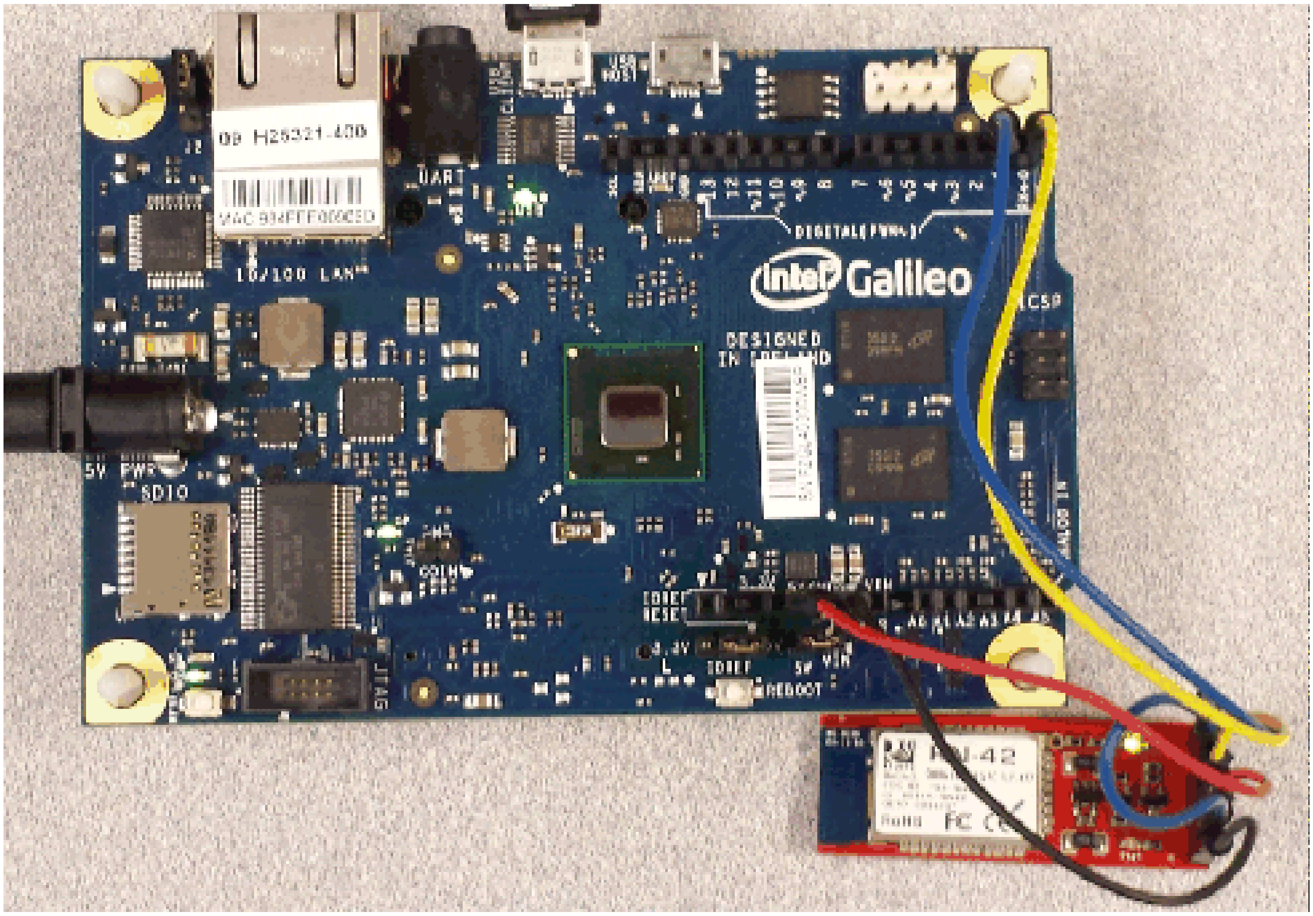
Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino 1.6.8 *https://www.arduino.cc/en/Main/Software*
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Operating voltage	3.3 to 6 V
Passcode	1234
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/Bluetooth/BlueSMiRF-Gold-ChipAnt-v1_rotat2.pdf*.

Pin Label	Pin Function	Input, Output, Power?	Description
RTS-O	Request to send	Output	RTS is used for hardware flow control in some serial interfaces. This output is not critical for simple serial communication.
Rx-I	Serial receive	Input	This pin receives serial data from another device. It should be connected to the Tx of the other device.
Tx-O	Serial transmit	Output	This pin sends serial data to another device. It should be connected to the Rx of the other device.
VCC	Voltage supply	Power In	This voltage supply signal is routed through a 3.3 V regulator, and then routed to the Bluetooth module. It should range from 3.3 to 6 V.
CTS-I	Clear to send	Input	CTS is another serial flow control signal. Like RTS, it is not required for most, simple serial interfaces.
GND	Ground	Power In	The 0 V reference voltage, common to any other device connected to the Bluetooth modem.

Connections used for testing:

- Tx-0 connected to the board's D0.
- Rx-1 connected to the board's D1.
- VCC connected to the board's 5 V.
- GND connected to the board's GND.
- RTS-0 and CTS-1 connected to each other.

Figure 114 **Connecting a BlueSMiRF Silver. Shown here with an Intel® Galileo first generation board**



By default, this Bluetooth module is in "Slave" mode. The second Bluetooth device is in "Master" mode. This typically would be a PC or smartphone.

42.3 Compile and load

The purpose of the test is to test the serial communication between the module and the board and test the communication with another Bluetooth* device. An Android* smartphone and an app called “Bluetooth SPP” were used as the second Bluetooth* device.

Example 48 shows the sketch used to test the Arduino* 101 board.

Example 48 **Bluetooth* modem sketch**

```
UARTClass*   bluetooth = &Serial1; // BT_Rx jumper to 1, BT_Tx jumper to 0

void setup()
{
  Serial.begin(9600);  // Begin the serial monitor at 9600bps
  bluetooth->begin(115200);  // The Bluetooth Mate defaults to 115200bps
  delay(2000);  // Short delay, wait for the Mate to send back CMD
  bluetooth->println("U,9600,N");  // Temp Change baudrate to 9600, no parity
  // 115200 can be too fast for NewSoftSerial to relay the data reliably
  bluetooth->begin(9600);  // Start bluetooth serial at 9600
  Serial.println("Setup-Done");
}

void loop()
{
  if(bluetooth->available())  // If the bluetooth sent any characters
  {
    // Send any characters the bluetooth prints to the serial monitor
    Serial.print((char)bluetooth->read());
  }
  if(Serial.available())  // If stuff was typed in the serial monitor
  {
    // Send any characters the Serial monitor prints to the bluetooth
    bluetooth->print((char)Serial.read());
  }
}
```

After uploading this sketch, open the IDE Serial Monitor to see the heartbeat and the communication messages. The RN-42 should be “advertising” itself and a pairing can be initiated between the phone and the device (pin code: 1234 by default).

Using “Bluetooth SPP” on the phone and the Serial Monitor you can send text messages between the phone and board.

It is also possible to enter “command mode” to issue commands and set up information to the device. The following link has information about this: *https://learn.sparkfun.com/tutorials/using-the-bluesmirf/example-code-using-command-mode*.

42.4 Results

Arduino* 101 pass using Serial1.

§

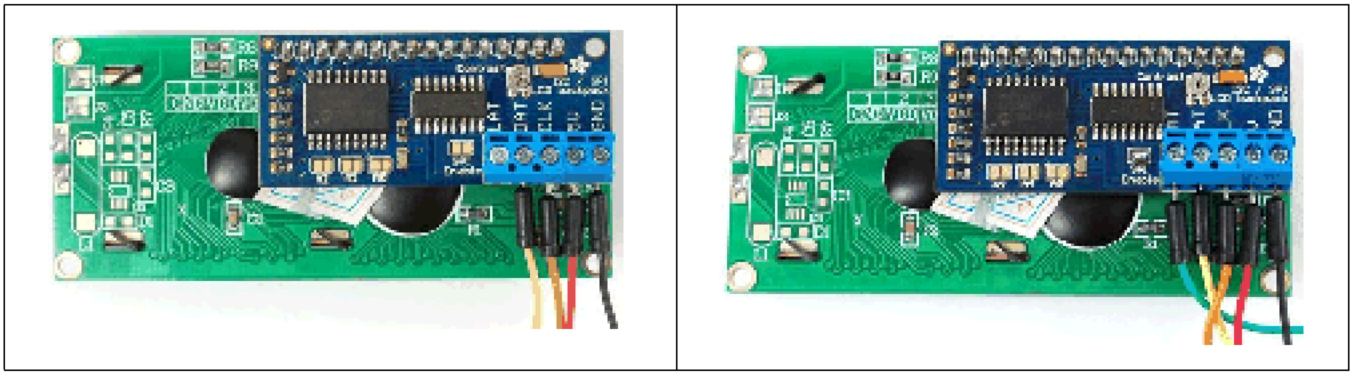
43 Adafruit* LCD Backpack with I^{2}C and SPI

43.1 Use case

This shield is a backpack (add-on circuit) that reduces the number of pins without a lot of expense. By using simple I^{2}C and SPI input/output expanders, it reduces the number of pins (only 2 pins are needed for I^{2}C) while still making it easy to interface with the LCD. For advanced users, this project can be used for general purpose I/O expansion. The MCP23008 has 8 I/O pins (7 are connected) with optional pullups, the SPI 74HC595 has 7 outputs.

Key Info	Links
Product info	*https://www.adafruit.com/products/292*.
Library	Library to replace the built-in Arduino* LiquidCrystal library that has no support for I^{2}C. *https://github.com/adafruit/LiquidCrystal*, dated 100513. For the Intel® Galileo board, the library requires a "shiftOut" function was obtained from: *https://github.com/WiringPi/WiringPi*, dated 032413. The function is in the .\*wiringPi*\wiringShift.c file.
Guide	*https://learn.adafruit.com/i2c-spi-lcd-backpack/overview*. *https://learn.adafruit.com/downloads/pdf/i2c-spi-lcd-backpack.pdf*.

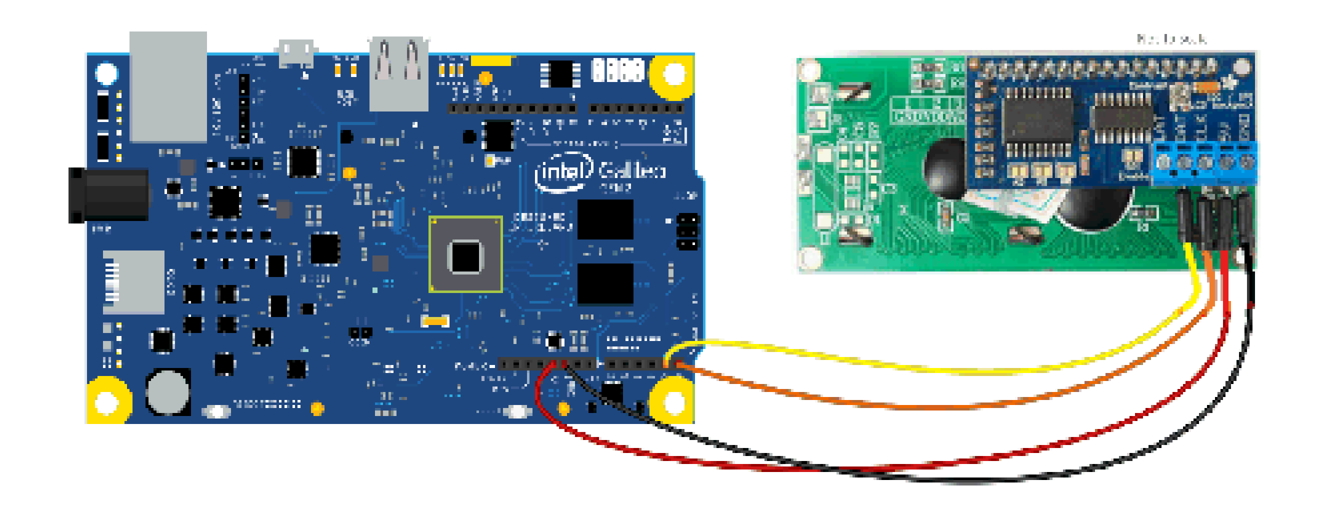
Figure 115 **Adafruit* LCD backpack w/I^{2}C and SPI**



43.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Arduino* 101 firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematic	*https://github.com/adafruit/i2c-SPI-LCD-backpack/blob/master/i2cspilcdbackpacksch.png*.

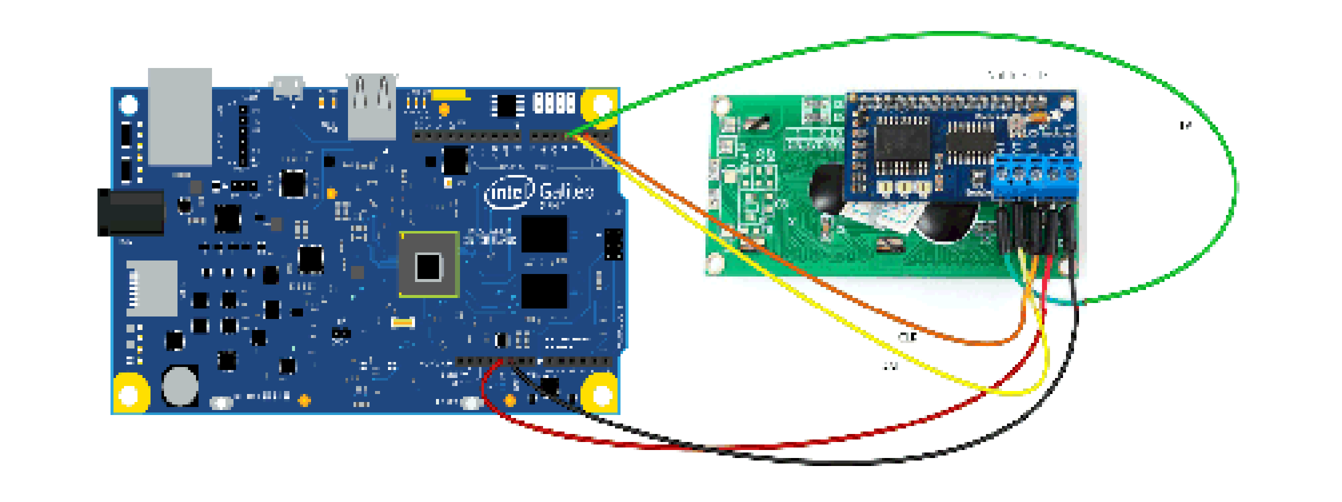
Figure 116 **I²C LCD backpack. Shown here with an Intel® Galileo second generation board**



Breakout Pin Name	I²C Function/Connection
GND	Ground – connect to GND.
5 V	Power – connect to 5 V.
CLK	Clock – connect to SCL or any analog/digital pin, used A5 in example – analog pin for I²C / digital pin for SPI.
DAT	Data – connect to SDA or any analog/digital pin, used A4 in example – analog pin for I²C / digital pin for SPI.
LAT	Latch – connect to any digital pin in SPI use.

I²C jumper	`LiquidCrystal lcd(0); // Pass the I2C Jumper`
	The LCD backpack has solder jumpers (A0 through A2, located on the back) that is used to change the I²C address of the device. By default, nothing is soldered, therefore, the I²C address defaults to 0x00.

Figure 117 **SPI LCD backpack on an Intel® Galileo second generation board**



Breakout Pin Name	SPI Function/Connection
GND	Ground – connect to GND.
5 V	Power – connect to 5 V.

Breakout Pin Name	SPI Function/Connection
CLK	Clock – connect to D2 for SPI.
DAT	Data – connect to D3 for SPI.
LAT	Latch – connect to D4 for SPI.

43.3 Companion library

43.4 The library is a replacement for the built-in LiquidCrystal library. Compile and upload

I^{2}C: Connect the LCD as described in the I^{2}C table. Use the following example sketch from the new LiquidCrystal library from Adafruit. The sketch came from the Adafruit library and is called ‘HelloWorld_i2c’.

Example 49 **HelloWorld_i2c sketch**

```
/*
 Demonstration sketch for Adafruit i2c/SPI LCD backpack using MCP23008 I2C expander
 ( http://www.ladyada.net/products/i2cspilcdbackpack/index.html )

 This sketch prints "Hello World!" to the LCD and shows the time.

  The circuit:
 * 5V to Arduino 5V pin
 * GND to Arduino GND pin
 * CLK to Analog #5
 * DAT to Analog #4
*/

// include the library code:
#include "Wire.h"
#include "LiquidCrystalAF.h"

// Connect via i2c, default address #0 (A0-A2 not jumpered)
LiquidCrystalAF lcd(0);

void setup() {
  // set up the LCD's number of rows and columns:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis()/1000);

  lcd.setBacklight(HIGH);
  delay(500);
  lcd.setBacklight(LOW);
  delay(500);
}
```

Example 50 **HelloWorld_SPI sketch**

```
// include the library code:
#include "Wire.h"
#include "LiquidCrystalAF.h"

// Connect via SPI. Data pin is #3, Clock is #2 and Latch is #4
LiquidCrystalAF lcd(3, 2, 4);
```

```
void setup() {

  // set up the LCD's number of rows and columns:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis()/1000);

  lcd.setBacklight(HIGH);
  delay(500);
  lcd.setBacklight(LOW);
  delay(500);
}
void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val)
{
  int8_t i;

  if (order == MSBFIRST)
    for (i = 7 ; i >= 0 ; --i)
    {
      digitalWrite (dPin, val & (1 << i));
      digitalWrite (cPin, HIGH);
      digitalWrite (cPin, LOW);
    }
  else
    for (i = 0 ; i < 8 ; ++i)
    {
      digitalWrite (dPin, val & (1 << i)) ;
      digitalWrite (cPin, HIGH);
      digitalWrite (cPin, LOW);
    }
}
```

43.5 Results

- Arduino* 101 cannot get I2C or SPI working, check for bad part.

44 DFRobot* SPI LCD ST7920, 12864ZW

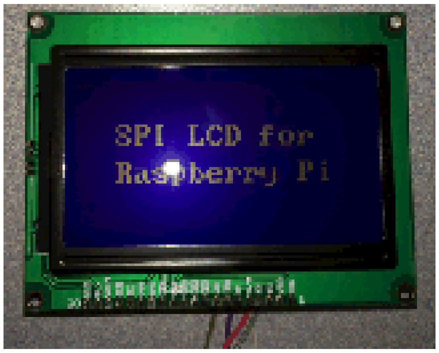
44.1 Use case

The ST7920 LCD controller/driver IC can display alphabets, numbers, Chinese fonts, and self-defined characters. It supports three kinds of bus interface: 8-bit, 4-bit, and serial. All functions, including display RAM, character generation ROM, LCD display drivers, and control circuits are all in a one-chip solution. With a minimum system configuration, a Chinese character display system can be easily achieved.

Key Info	Links
Product info	*http://www.dfrobot.com/index.php?route=product/product&filter_name=128x64 Graphic LCD&product_id=240*.
Library	None.
Guide	*http://www.cooking-hacks.com/documentation/tutorials/intel-galileo-tutorial-using-arduino-and-raspberry-pi-shields-modules-boards*.

ST7920 includes character ROM with 8192 16 × 16-dot Chinese fonts and 126 16 × 8-dot half-width alphanumerical fonts. Besides, it supports 64 × 256-dot graphic display area for graphic display (GDRAM). Mix-mode display with both character and graphic data is possible. ST7920 has built-in CGRAM and provides four sets of software-programmable 16 × 16 fonts.

Figure 118 **SPI LCD ST7920, 12864ZW**

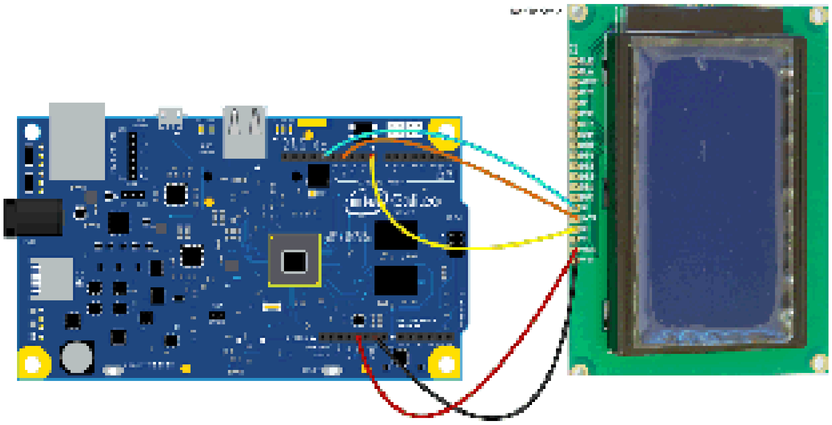


44.2 Hardware summary

Key Info	Description/Links
Operating voltage	2.7 to 5.5 V
IOREF	3.3 or 5 V(See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Controller	ST7920.
Schematic	In datasheet: *http://www.dfrobot.com/image/data/FIT0021/ST7920.pdf*.

Breakout pin	Function/connection
VSS	Ground – connect to GND.
VDD	Power – connect to 5 V.
RS, See wire notes	Chip select (CS) – connect to PIN 8.
R/W	Serial Data Input (MOSI) – connect to PIN 11.
E (SCLK) See Note.	Serial Clock (SCK) – connect to PIN 13.
Note: The OSC pin must have the shortest wiring pattern of all other pins. To prevent noise from other signal lines, it should also be enclosed by the largest GND pattern. Poor antinoise characteristics on the OSC line will result in malfunction, or adversely affect the clock's duty ratio.	

Figure 119 **Connecting an SPI LCD ST7920, 12864ZW to an Intel® Galileo second generation board**



44.3 Companion library

No additional library required. Uses libraries included in the IDE.

44.4 Compile and upload

The following sketch compiles and uploads without issue. Comment the SPI.setClockDivider(SPI_CLOCK_DIV64128) line.

Example 51 **SPI_LCD sketch**

```
/*
*  SPI_LCD example.
*
*  Copyright (C) 2014 Libelium Comunicaciones Distribuidas S.L.
*  http://www.libelium.com
*
*  This program is free software: you can redistribute it and/or modify
*  it under the terms of the GNU General Public License as published by
*  the Free Software Foundation, either version 3 of the License, or
*  (at your option) any later version.
*
*  This program is distributed in the hope that it will be useful,
*  but WITHOUT ANY WARRANTY; without even the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*  GNU General Public License for more details.
*
*  You should have received a copy of the GNU General Public License
*  along with this program.  If not, see http://www.gnu.org/licenses/
*
*  Version 0.1
*  Author: Luis Martin
*/

// include the SPI library:
#include <SPI.h>

int latchPin = 8;
unsigned char char1[]=" Cooking Hacks  ";
unsigned char char2[]="  SPI LCD for   ";
unsigned char char3[]="  Raspberry Pi  ";

void setup(){
        SPI.begin();
        SPI.setBitOrder(MSBFIRST);
        SPI.setDataMode(SPI_MODE0);
        SPI.setClockDivider(SPI_CLOCK_DIV128);

        initialise();
}

void loop(){
        displayString(0,0,char1,16);
        delay(2000);
        clear();
        displayString(1,0,char2,16);
        displayString(2,0,char3,16);
        delay(2000);
        clear();
}

void initialise(){
        pinMode(latchPin, OUTPUT);
        digitalWrite(latchPin, LOW);

        delayMicroseconds(80);

        writeCommand(0x30);
        writeCommand(0x0c);
        writeCommand(0x01);
        writeCommand(0x06);
}

void displayString(int X,int Y,unsigned char *ptr,int dat){
        int i;

        switch(X){
                case 0:  Y|=0x80;break;

                case 1:  Y|=0x90;break;

                case 2:  Y|=0x88;break;

                case 3:  Y|=0x98;break;

                default: break;
        }

        writeCommand(Y);

        for(i=0;i < dat;i++){
                writeData(ptr[i]);
        }

}

void writeCommand(int CMD){
        int H_data,L_data;
        H_data = CMD;
        H_data &= 0xf0;
        L_data = CMD;
        L_data &= 0x0f;
        L_data <<= 4;
        writeByte(0xf8);
        writeByte(H_data);
        writeByte(L_data);
}

void writeData(int CMD){
        int H_data,L_data;
        H_data = CMD;
        H_data &= 0xf0;
        L_data = CMD;
        L_data &= 0x0f;
        L_data <<= 4;
        writeByte(0xfa);
        writeByte(H_data);
        writeByte(L_data);
}

void writeByte(int dat){
        digitalWrite(latchPin, HIGH);
        delayMicroseconds(80);
        SPI.transfer(dat);
        digitalWrite(latchPin, LOW);
}

void clear(){
        writeCommand(0x30);
        writeCommand(0x01);
}
```

44.5 Results

Arduino* 101 not compatible. Saw it work on a couple of attempts. Need to make sure IOREF 3.3V is really supported because it still works on 5V cards.

Does not work on the Arduino* Uno board in its current configuration. Seen inconsistent behavior when the R/W (MISO) pin was not connected.

§

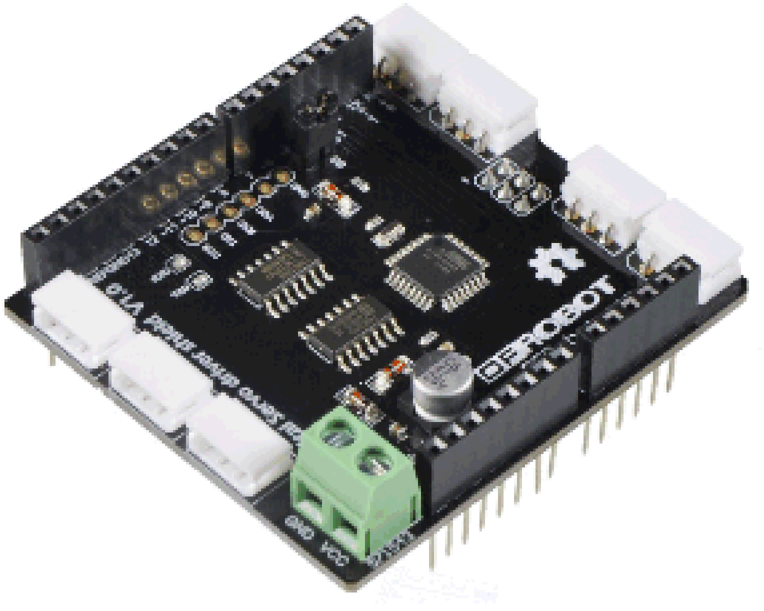
45 DFRobot* Digital Servo Driver Shield v1.0

45.1 Use case

This servo shield can be linked by serial bus, which can connect 200+ servos. Each unit can feedback its position, rotation velocity, torque, and current motor temperature. The motor can rotate all around and the velocity can be controlled. This feature enables it to work as a motor of wheeled robots or tracked robots. The transmit wire from the UART is connected to all of the AX-12 servos. The Arduino* 101 platform sketch can send a command over the wire and all of the servos will hear it. The message contains the destination servo ID so only one servo will process the message.

Key Info	Links
Order/product info	*http://www.dfrobot.com/image/data/SER0026/CDS55xx_Robot_Servo_Quick_Start_EN.pdf*.
Guide	*http://www.dfrobot.com/wiki/index.php/Digital_Servo_Shield_for_Arduino_SKU:DRI0027*
Library	*http://www.dfrobot.com/image/data/DRI0027/ServoCds55%20Library.zip* Downloaded 10/28/14.

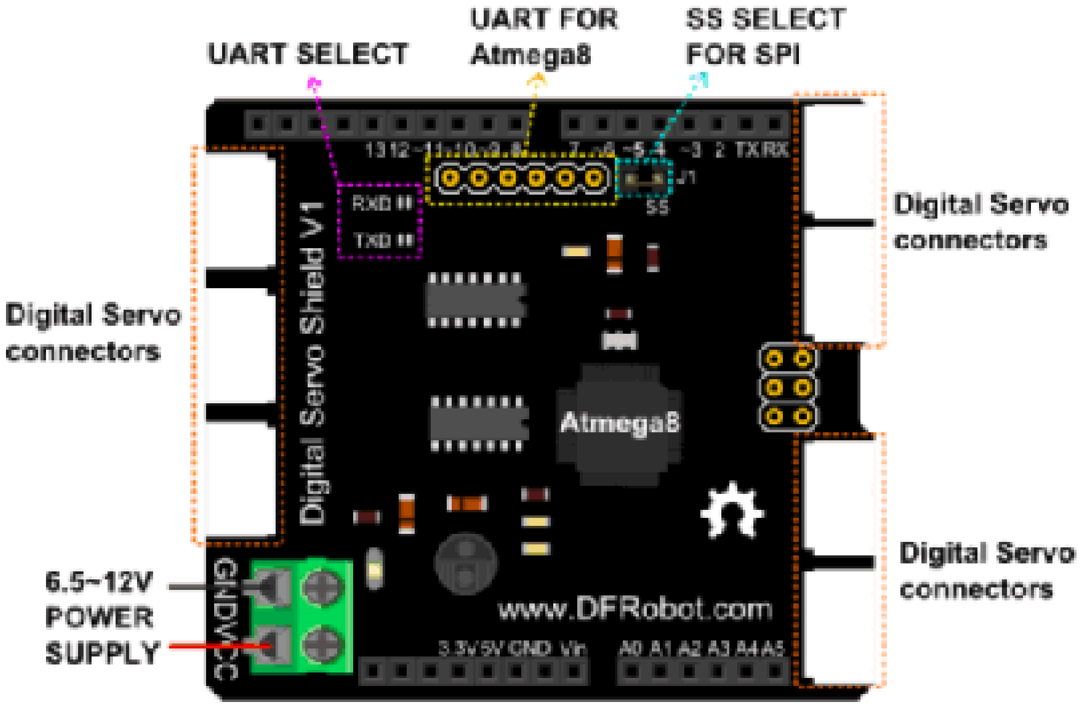
Figure 120 **DFRobot* Digital Servo Driver Shield v1.0**



45.2 Hardware summary

Key Info	Description/Links
Board voltage	5 V.
Motor voltage	6.5 to 12 V from external power source.
VIN as power source	No.
IOREF	3.3 or 5 V. (See *Section 0.8 IOREF voltage*.)
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
UART select	UART is already shorted by solder. When using UART for Atmel* ATmega8, remove solder.
SS select for SPI	Digital pin 10 is the default chip select. If using other digital pins for SS, remove the jumper cap and connect the SS header to another digital pin.
Motors	SV3

Figure 121 **DFRobot* Digital Servo Driver Shield layout**



Pin Name	Function
D10	SPI - Chip select (CS)
D13	SPI - Clocking
D11	SPI - MOSI
D12	SPI - MISO

45.3 Companion library

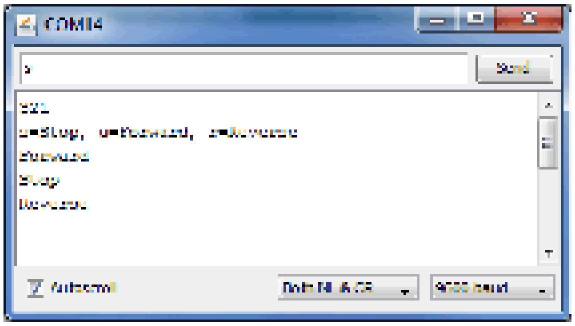
Library is called “CDS55Servo”. This library uses SPI for communication.

45.4 Compile and upload

Power up the Arduino* 101 (wait the appropriate time for the board to boot), then turn on the external power supply, connect the USB cable and upload the sketch.

This sketch will take command lines to run continuously clockwise, counter clockwise, and stop at a certain velocity. *Figure 122* shows the commands given through the Arduino IDE.

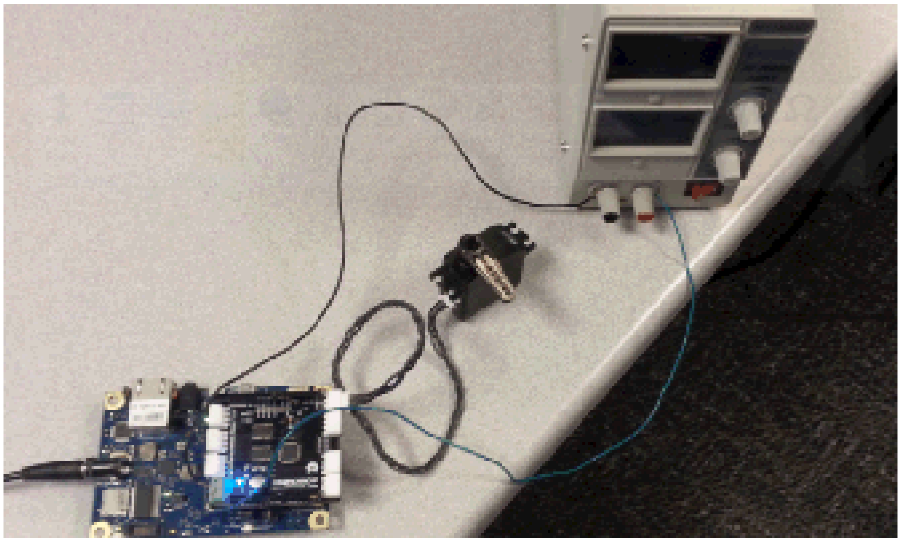
Figure 122 **Arduino** IDE sketch commands**



```
s
```

```
321
s=Stop, u=Forward, r=Reverse
Forward
Stop
Reverse
```

Figure 123 **DFRobot* digital servo driver with external PS. Shown here with an Intel® Galileo first generation board**



Example 52 **Modified servo sketch**

```
#include <SPI.h>
#include <ServoCds55.h>
//#include "pins_arduino.h"
ServoCds55 myservo;


void setup (void)
{
  Serial.begin (9600);
  waitForUser(3);
  digitalWrite(SS, HIGH);
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV8);
  Serial.println("s=Stop, u=Forward, r=Reverse");
} // setup
void loop() {
  if (Serial.available()) {
    char val = Serial.read();

    if (val != -1) {
      switch(val) {

        case 'p':
          myservo.WritePos(1, 0);
          //delay(1000);
          //myservo.WritePos(1, 300);
          //delay(1000);
          //myservo.WritePos(1,0);

          break;

        case 'v':
          myservo.setVelocity(50);
          break;

        case 'r':
          Serial.println("Reverse");
          myservo.rotate(1, -150);
          //delay(3000);
          break;
        case 'u':
          Serial.println("Forward");
          myservo.rotate(1, 150);
          break;

        case 's':
          Serial.println("Stop");
          //myservo.setVelocity(0);
          myservo.rotate(1, 0);
          break;
      }
    }
  }
} // loop
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);Serial.print(i);}
  Serial.println("");
Serial.println("");
} // waitForUser
```

45.5 Results

Arduino* 101 compatible.

§

46 ThingM* BlinkM RGB LED

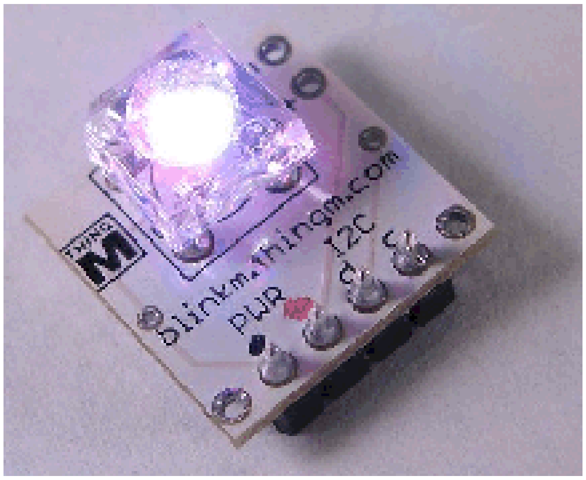
46.1 Use case

BlinkM is a “Smart LED”: a networkable and programmable full-color RGB LED for hobbyists, industrial designers, prototypers, and experimenters. It is designed to allow the easy addition of dynamic indicators, displays, and lighting to existing or new projects. BlinkM uses a high-quality, high-powered RGB LED and a small AVR microcontroller to allow a user to digitally control an RGB LED over a simple I^{2}C interface. Multiple BlinkMs can be strung together on an I^{2}C bus, allowing for some amazing light displays.

Key Info	Links
Order/product info	*http://thingm.com/products/blinkm*.
Guide	*http://www.cooking-hacks.com/documentation/tutorials/intel-galileo-tutorial-using-arduino-and-raspberry-pi-shields-modules-boards?utm_source=NewsletterCH&utm_medium=Email&utm_campaign=NCH-200514#i2c*.
Library	No additional libraries needed.
Datasheet	*http://thingm.com/fileadmin/thingm/downloads/BlinkM_datasheet.pdf*.

BlinkMs are also programmable. With ThingM’s sequencer software, it is possible to create a mix of colors with different time slices, upload that sequence to the BlinkM, and let it rip.

Figure 124 **BlinkM RGB LED**

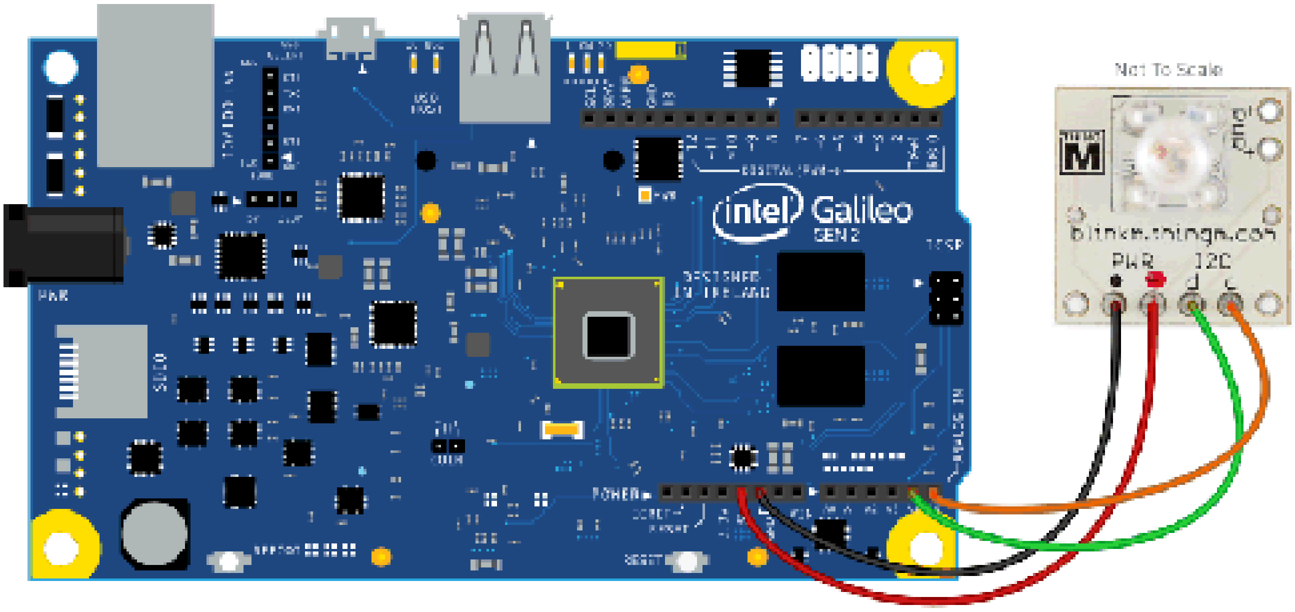


46.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V.
Use VIN as power source	No.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematics	In datasheet, page 45: *http://thingm.com/fileadmin/thingm/downloads/BlinkM_datasheet.pdf*.

Pin Name	Function
–	Ground – connect to GND.
+	Power – connect to 5 V.
c	Clock – connect to SCL.
d	Data – connect to SDA.

Figure 125 **Connecting a BlinkM. Shown here with an Intel® Galileo second generation board**



46.3 Companion library

No additional library required. Uses libraries included in the IDE.

46.4 Compile and upload

Upon plugging in power and ground, the LED should begin a default sequence.

This sketch will begin a red-blue blink and test fade transitions.

Example 53 **BlinkM sketch**

```
/*
*  I2C example for Intel Galileo.
*
*  Copyright (C) 2014 Libelium Comunicaciones Distribuidas S.L.
*  http://www.libelium.com
*
*  This program is free software: you can redistribute it and/or modify
*  it under the terms of the GNU General Public License as published by
*  the Free Software Foundation, either version 3 of the License, or
*  (at your option) any later version.
*
*  This program is distributed in the hope that it will be useful,
*  but WITHOUT ANY WARRANTY; without even the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*  GNU General Public License for more details.
*
*  You should have received a copy of the GNU General Public License
*  along with this program.  If not, see http://www.gnu.org/licenses/
*
*  Version 0.1
*  Author: Luis Martin
*/
```

```
#include "Wire.h"

void setup(){
        Wire.begin();
        Wire.beginTransmission(9);
        Wire.write('o'); //End the current Light script
        Wire.endTransmission();
}

void loop(){
        for (int i=0;i < 5;i++){
                Wire.beginTransmission(9);
                Wire.write('n'); //Change to color
                Wire.write(byte(0xff)); //Red component
                Wire.write(byte(0x00)); //Green component
                Wire.write(byte(0x00)); //Blue component
                Wire.endTransmission();

                delay(500);

                Wire.beginTransmission(9);
                Wire.write('n'); //Change to color
                Wire.write(byte(0x00)); //Red component
                Wire.write(byte(0x00)); //Green component
                Wire.write(byte(0xff)); //Blue component
                Wire.endTransmission();

                delay(500);
        }

        for (int i=0;i < 10;i++){
                Wire.beginTransmission(9);
                Wire.write('c'); //Fade to color
                Wire.write(byte(0xff)); //Red component
                Wire.write(byte(0x00)); //Green component
                Wire.write(byte(0x5a)); //Blue component
                Wire.endTransmission();

                delay(150);

                Wire.beginTransmission(9);
                Wire.write('c'); //Fade to color
                Wire.write(byte(0x55)); //Red component
                Wire.write(byte(0x20)); //Green component
                Wire.write(byte(0x5a)); //Blue component
                Wire.endTransmission();

                delay(150);
        }
}
```

46.5 Results

Arduino* 101 compatible.

§

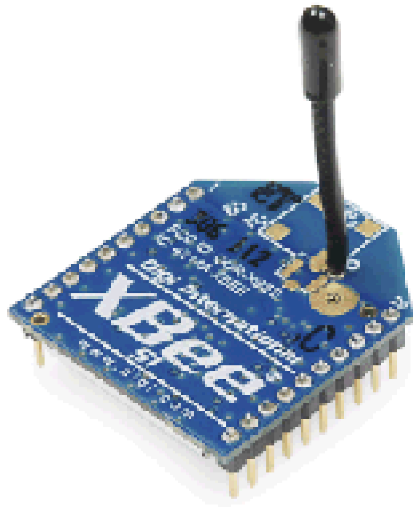
47 XBee* S1 Module w/Arduino* Wireless Shield

47.1 Use case

The XBee* modules are small radio devices implementing various protocols, all using the physical layer of the ZigBee* protocol. They are widely used in the Arduino* community because they are versatile and configurable with the software provided by Digi* (called X-CTU). These XBee* modules have a common footprint and plug into breakout board or a companion shield. This test used an Arduino wireless protoshield with a XBee* header socket (10-pin 2 mm socket).

Key Info	Links
URL	*http://arduino.cc/en/Main/ArduinoWirelessProtoShield*. *http://www.makershed.com/product_p/mkdg01.htm*.
Library	Not required.
Guide	*http://arduino.cc/en/Guide/ArduinoWirelessShield*.
Configure software	*http://www.digi.com/support/kbase/kbaseresultdetl?id=2125*. Run the X-CTU setup program as administrator. AT commands: *http://examples.digi.com/wp-content/uploads/2012/07/XBee_ZB_ZigBee_AT_Commands.pdf*.

Figure 126 **XBee S1 module**



This XBee module uses protocol 802.15.4, called the “Series 1” module. Right out of the box, the module can be queried (see configuration requirements below). Some tutorials claim that they work with no configuration; however, that was not the case for this test. Some tutorials claim that configuration of the module can be done within the X-CTU software, however, this test configuration was done by configuring the module through a serial port connection (through the X-CTU software).

The Arduino XBee protoshield where the XBee modules are mounted are not compatible with Arduino* 101 and the issue is being investigated. Symptoms are power cycling of the shield or not able to see the radios mounted.

47.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V.
Use VIN as power source	No.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Datasheet	*https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf*.
Schematics	None.

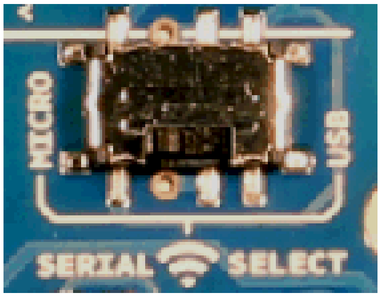
Pin Name	Function
D0	“Serial” Rx pin for Arduino* Uno, “Serial1” Rx pin for the Intel® Galileo board.
D1	“Serial” Tx pin for Arduino Uno, “Serial1” Tx pin for the Intel Galileo board.

The module communicates through the existing hardware serial ports. It was not clear, at this time, if other serial ports are available through the protoshield for debugging purposes.

47.3 Companion software

The wireless protoshield contains XBee header sockets to plug the module into. The board has a single switch labeled “MICRO” and “USB”. This switch is used to cross the Rx and Tx pin of the Hardware Serial on pins D0 and D1. The switch is designed to let the shield work with boards that use pins D0 and D1 (such as the Arduino* Uno) for programming without removing the shield. The switch also determines how the XBee’s serial communication connects to the serial communication between the microcontroller and USB-to-serial chip on the Arduino Uno board.

Figure 127 **XBee S1 module’s “MICRO/USB” switch**



When ‘Serial Select’ is in the ‘MICRO’ position, the DOUT pin of the wireless module is connected to the Rx pin of the microcontroller; and DIN is connected to Tx pin. The wireless module will then communicate with the microcontroller. Note that the Rx and Tx pins of the microcontroller are still connected to the Tx and Rx pins (respectively) of the USB-to-serial converter. Data sent from the microcontroller will then be transmitted to the computer (via USB) as well as being sent wirelessly by the wireless module. The microcontroller will not be programmable (via USB) in this mode. This switch is typically used for normal operation explained further below.

When ‘Serial Select’ is in the ‘USB’ position, the DOUT pin of the wireless module is connected to the Rx pin of the USB-to-serial converter, and DIN pin on the wireless module is connected to the Tx pin of the USB-to-serial converter. This means that the module can communicate directly with the computer. The microcontroller on the board is bypassed. The shield will be set into this mode so that the modules can be queried to determine their configuration.

Configuration:

At the time we conducted this test, the query was not functional on an Intel® Galileo board. This means that the Intel Galileo board (using this shield) cannot be used to configure the XBee module. For this test we used an Arduino* Uno board to configure the XBee module.

Query steps:

1. Insert XBee module into the header in the proper direction.
2. The switch “Serial Select” on the protoshield to the “USB” position.

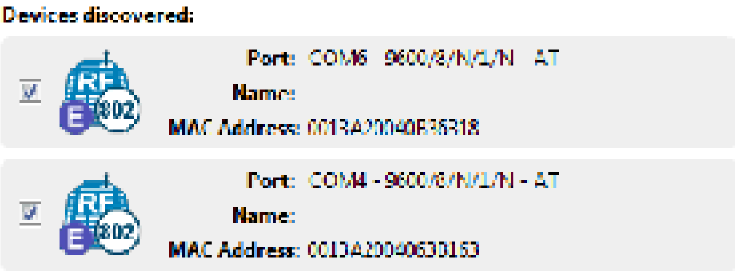
3. Download an ‘EmptySketch’ program to the Arduino Uno board. If any other program is present, the module will not query.

Sketch: EmptySketch

```
void setup() { }
void loop() { }
```

4. Select the ‘discovery’ option in the XCTU program. All known COM ports on the PC will be displayed.
5. Select the COM ports related to the connected Arduino shields. The X-CTU program will query the module through the selected COM ports. *Figure 128* shows that two XBee modules are discovered.

Figure 128 **Devices discovered output**



Devices discovered:

☑ Port: COM6 9600/8/N/1/N AT
Name:
MAC Address: 0013A20040B36318

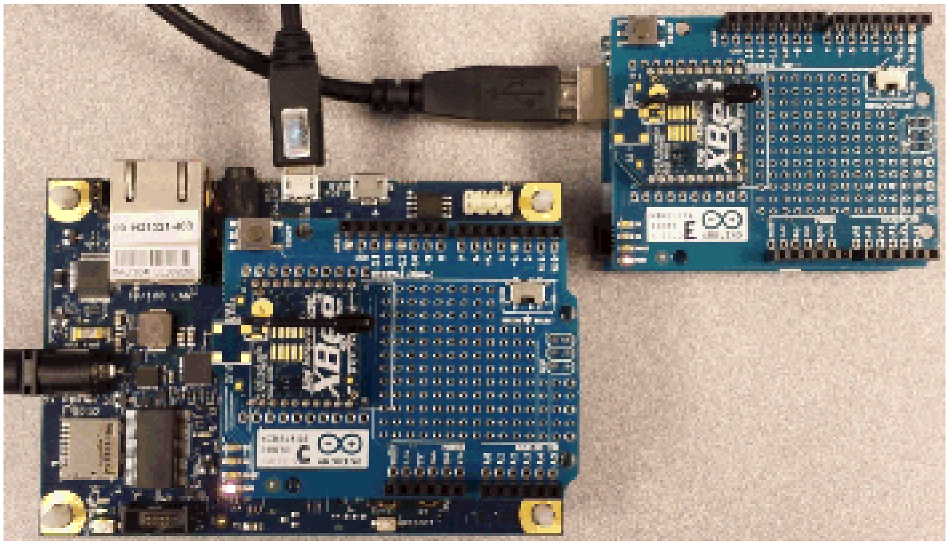
☑ Port: COM4 - 9600/8/N/1/N - AT
Name:
MAC Address: 0013A2004063B163

The two modules have different firmware versions that do not prevent them from communicating. Each module is assigned a different 16-bit address through the “ATMY” command. The destination address, “ATDL” for each address specified the 16-bit address for the other module. The value was saved using the “ATWR” command (not shown).

Table 7 **XBee* module setup**

Module setup 1	Module setup 2	Description
+++OK	+++OK	Command mode
ATCH	ATCH	
C	C	Same channel
ATID	ATID	
3332	3332	Same network ID
ATSH	ATSH	Serial number high
13A200	13A200	
ATSL	ATSL	Serial number low
4063B163	40B36318	High and low different
ATMY	ATMY	
0	1	Different net address
ATDH	ATDH	Dest high
0	0	
ATDL	ATDL	Dest low
1	0	DL will talk to MY
ATBD	ATBD	
3	3	
ATVR	ATVR	Firmware version
10E6	10EC	
ATHV	ATHV	
1744	1745	
ATCE	ATCE	Coordinator Enabled
EndDevice(0)	Coordinator	

Figure 129 **Connecting an XBee S1 module. Show here withan Intel® Galileo first generation board and an Arduino* Uno board**



47.4 Compile and upload

Two XBee* S1 modules were configured as described in the ‘Module Setup’ table. One board which is set as coordinator is downloaded a sender sketch and the other which is set as endpoint is downloaded with a receiver sketch.

To test, the end point module was placed on the Arduino* Uno board (with the switch set to USB) and an empty sketch was run. The controller module was placed on the Intel® Galileo board (with the switch set to micro) and the sketch above was run.

- Set “Serial Select” to “USB” to enable download.
- Download the ‘Sender’ sketch to the Intel Galileo board with the shield attached. For some unknown reason, when the shield is connected, LED13 is on by default. During the setup, LED13 is forced off to avoid confusion.
- Set “Serial Select” to “MICRO” on the sender only to start transmitting. The receiver should remain in USB to get the signals on Serial1.

Example 54 **Sender sketch**

```
// MODE: Sender
// A R D U I N O
HardwareSerial* gSerialShieldPtr = &Serial1;  // Rx pin 0, Tx pin 1: Arduino Uno (Serial)
// G A L I L E O

int gLed = 13;
void setup()
{
  gSerialShieldPtr->begin(9600);
  Serial.begin(9600);
  pinMode(gLed,OUTPUT); digitalWrite(gLed, LOW); // Force LED off
}
void loop()
{
  gSerialShieldPtr->println("H");//turn on the LED
  delay(1000);
  gSerialShieldPtr->println("L");//turn off the LED
  delay(1000);
}
```

- Set “Serial Select” to “USB” to enable download.
- Download the ‘Receiver’ sketch to the Arduino Uno board with the shield attached.

- Set “Serial Select” to “MICRO” to start receiving.

Normal operation:

There are no other pins involved in the shield. We tested the Arduino* 101 platform with one board sending a character to the other board through XBee*. The receiver then turned on LED-13 based on the received character.

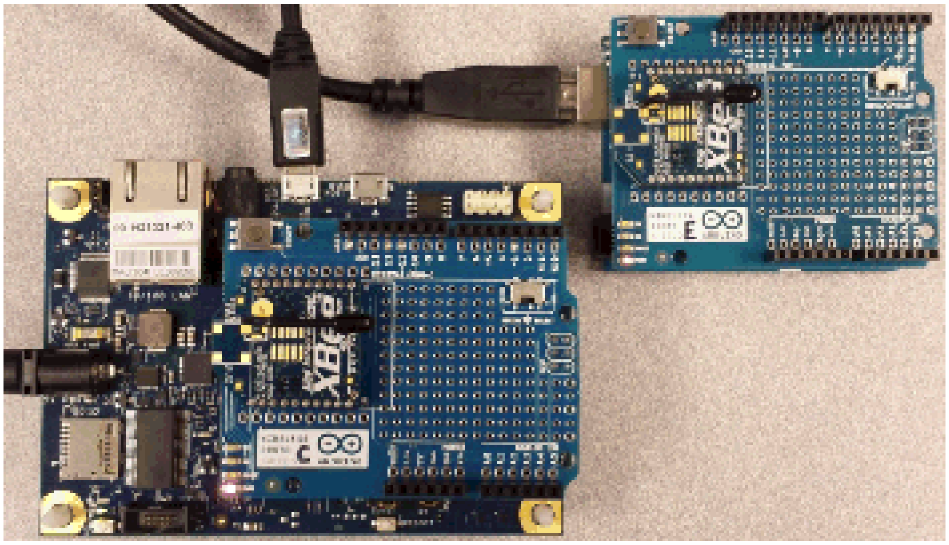


Figure 130 **Connecting an XBee S1 module. Shown here with an Intel® Galileo first generation board and an Arduino* Uno board**

47.5 Compile and upload

Two XBee* S1 modules were configured as described in the ‘Module Setup’ table. One board set as coordinator is downloaded a sender sketch and the other which is the endpoint is downloaded with a receiver sketch.

To test, the end point module was placed on the Arduino* Uno board (with the switch set to USB) and an empty sketch was run. The controller module was placed on the Intel® Galileo board (with the switch set to micro) and the sketch above was run.

- Set “Serial Select” to “USB” to enable download.
- Download the ‘Sender’ sketch to the Intel Galileo board with the shield attached. For some unknown reason, when the shield is connected, LED13 is on by default. During the setup, LED13 is forced off to avoid confusion.
- Set “Serial Select” to “MICRO” to start transmitting. Arduino* 101 should be set to USB to receive the signals.

Example 55 **Sender sketch**

```
// MODE: Sender
// A R D U I N O
HardwareSerial* gSerialShieldPtr = &Serial1;  // Rx pin 0, Tx pin 1: Arduino Uno (Serial)
// G A L I L E O

int gLed = 13;
void setup()
{
  gSerialShieldPtr->begin(9600);
  Serial.begin(9600);
  pinMode(gLed,OUTPUT); digitalWrite(gLed, LOW); // Force LED off
}
void loop()
```

```
{
  gSerialShieldPtr->println("H");//turn on the LED
  delay(1000);
  gSerialShieldPtr->println("L");//turn off the LED
  delay(1000);
}
}
```

- Set “Serial Select” to “USB” to enable download.
- Download the ‘Receiver’ sketch to the Arduino Uno board with the shield attached. The XBee module whould be set us endpoint
- Set “Serial Select” to “MICRO” to start receiving.

Example 56 **Receiver sketch**

```
// MODE: Receiver
// A R D U I N O
HardwareSerial* gSerialStdPtr    = &Serial;  // Rx pin 0, Tx pin 1: Arduino Uno (Serial)
HardwareSerial* gSerialShieldPtr = &Serial1;  // Rx pin 0, Tx pin 1: Arduino Uno (Serial)


char      gMsg     = ' ';  //contains the message from arduino sender
const int gLed     = 13;   //led at pin 13
boolean   gWaiting = true; //User feedback

void setup()
{
  gSerialStdPtr->begin(9600);
  gSerialShieldPtr->begin(9600);
  pinMode(gLed,OUTPUT); digitalWrite(gLed, LOW); // Force LED off
}
void loop()
{
  if(gWaiting == true)
  {
        gSerialStdPtr->println("Waiting");
        delay(1000);
  }
  while(gSerialShieldPtr->available() > 0)
  {
    gMsg=gSerialShieldPtr->read();
    if(gMsg=='H')
    {
      digitalWrite(gLed,HIGH);
    }
    if(gMsg=='L')
    {
      digitalWrite(gLed,LOW);
    }

    if(gWaiting == true)
    {
gSerialStdPtr->println("Led is blinking");
        gWaiting=false;
    }
  }
}
```

To validate on Arduino* 101, the LED on Pin 13 should blink and the receiver should display on the Serial Monitor that LED’s are blinking. You can also hook an FTDI on the receiver and see the H or L values being pushed via the XBee radio.

47.6 Results

NOT Intel® Curie™ compatible.

§

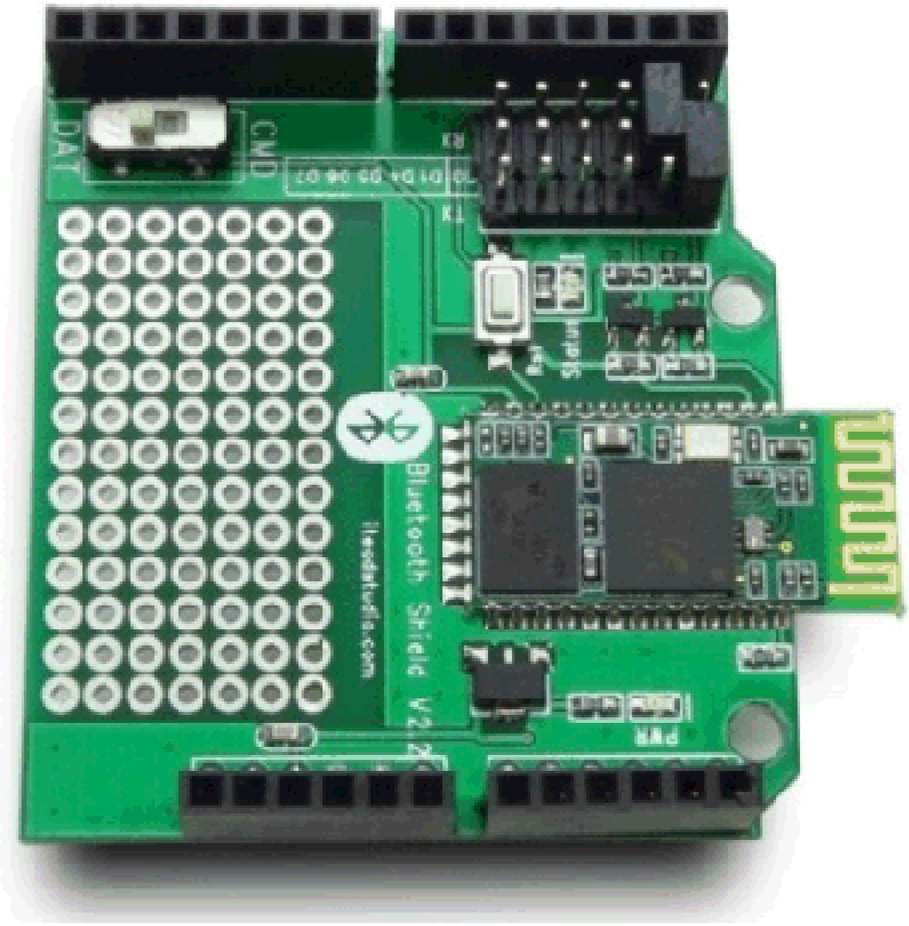
48 ITEAD* Bluetooth* Shield, Master/Slave

48.1 Use case

This Bluetooth shield integrates a HC-05 serial Bluetooth module. Bluetooth can be configured as a Master or Slave device. Jumpers are present to move the serial ports to hardware serial ports or software serial ports. This configurability is ideal for debugging purposes.

Key Info	Links
Order/product info	*http://imall.iteadstudio.com/im120417010.html*.
Guide	*http://shop.boxtec.ch/shield-v22-masterslave-stackable-bluetooth-shield-p-40422.html*.
Library	Not required.
AT configuration	*ftp://imall.iteadstudio.com/IM120417010_BT_Shield_v2.2/DS_IM120417010_BTShield.pdf*. *ftp://imall.iteadstudio.com/IM120417010_BT_Shield_v2.2/DS_BluetoothHC05.pdf*.
Phone software	*https://play.google.com/store/apps/details?id=mobi.dzs.android.BLE_SPP_PRO*. It may be possible to use the Bluetooth on the same PC as the IDE. However, there are cases where the Arduino** IDE will hang if Bluetooth is turned on (in cases where the BT is on the chipset). Alternative solution is to use a USB Bluetooth adapter. The best method is to use a device that independent from the IDE. In this case, we are using an Android* phone. This software is used to pair the device when it is configured as a slave.

Figure 131 **ITLEAD* Bluetooth* Shield (master)**



48.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 VDC.
Use VIN as power source	No.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Device type	BR/EDR Bluetooth.
Class of device	1f00.
AT firmware version	2.0-2010**0601.**
Serial parameter-DAT	9600,0,0.
Serial parameter-CMD	38400,8,N,1.
BT passcode	1234 (default).
HC-05 module	*ftp://imall.iteadstudio.com/IM120417010_BT_Shield_v2.2/DS_BluetoothHC05.pdf*.
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/Danger_Shield-v17.pdf*.

Wiring is required only if software serial is being used for Arduino* Uno boards.

<table>
<tr><th>Shield Pin Name</th><th colspan="2">Function (no wiring required)</th></tr>
<tr><td>Tx/Dn jumper</td><td colspan="2">Shield, Dn is selectable for Tx (D0, D1, D4 through D7).
Arduino* Uno, set to D7 (default)
Intel® Galileo, set to D1</td></tr>
<tr><td>Dn/Rx jumper</td><td colspan="2">Shield, Dn is selectable pins for Rx (D0, D1, D4 through D7).
Arduino Uno, set to D6 (default)
Intel Galileo, set to D0</td></tr>
<tr><td>CMD/DAT switch</td><td colspan="2">Set to 'CMD' to configure and query the state of the shield.
Set to 'DAT' for processing data being transmitted from the Rx/Tx</td></tr>
<tr><td rowspan="2">Status LED</td><td><b>Slave</b></td><td><b>Master</b></td></tr>
<tr><td>PAIRABLE, fast blink
PAIRED, steady blink
CONNECTED, blink-blink-pause</td><td>INITIALIZED,
INQUIRING, fast blink
PAIRED,</td></tr>
</table>

48.3 Companion library

No library is required; however, a PC serial port program is required to configure the shield. Configuration was not possible on an Intel® Galileo/Intel® Edison board. The possible reason is probably due to the multiplexing of the D0/D1 serial ports. Using an Arduino* Uno board, download an empty sketch to the master shield. This ensures that no communication is happening on the TX/RX pins.

Sketch: EmptySketch

void setup() { }
void loop() { }

Set the switch on the shield to "CMD" and startup the Cool Term program, and configure port to values in the *Hardware summary* section above. Enter the following commands to query default settings.

Example 57 **AT commands: Querying default settings**

```
AT+VERSION?
+VERSION:2.0-20100601
OK
AT+ORGL
OK
AT+UART?
+ADDR:2013:6:253622
OK
AT+UART=9600,0,0
OK
```

At this point, the shield can be tested as a slave. The shield seems to no longer talk as a slave; therefore, inconclusive. Previous attempts to set up were as follows.

Example 58 **AT commands: Querying default settings**

```
AT+VERSION?
+VERSION:2.0-20100601
OK
AT+NAME?
+NAME:itead
OK
AT+ADDR?
+ADDR:2013:6:253622
OK
AT+CLASS?
+CLASS:1f00
OK
AT+ROLE?
+ROLE:0
OK
```

The information gathered included the BT address (`“+ADDR:2013:6:253622” translate to “20:13:06:25:36:22”`), the BT name, and the role (`“+ROLE:0” translate to “Slave”`) that the BT shield is configured to.

Disconnect the serial program and set the switch on the shield to “DAT”. Using the phone software, query the Bluetooth devices. The BT address will display with the configured BT name. Pair and connect the device with the default passcode. This will verify that the shield connects as a slave.

Delete the pairing from the phone software completely. Set the switch on the shield to “CMD” and startup the Cool Term program, and configure port to values in the Hardware summary. Change the BT name and the BT role to a Master (`”1” translate to “Master”`) configuration.

Example 59 **AT commands: Configure the shield as a master**

Change the name and role	Verify the change
AT+NAME=ITeadMaster OK AT+ROLE=1 OK	AT+NAME? +NAME:ITeadMaster OK AT+ROLE? +ROLE:1 OK

The shield is now configured as a master.

48.4 Compile and upload

The first step is to connect to the BT master from a BT slave device. This approach was attempted using the _SparkFun* BlueSMiRF Silver – Bluetooth* Modem_, but querying for all BT devices did not display the master device with the expected BT name and BT address. Attempts to use the phone software to query BT devices did not display this BT device in master mode. Since the BT address was known, simply connecting to that directly failed.

A second step was to use _ITEAD* Bluetooth* Shield, Master/Slave_ (a similar brand). The first step was to inquire key information about the device using the Phone Software. The slave’s pairing was then removed from the phone. Keep the EmptySketch running on the slave until the master is connected.

```
Bluetooth* Slave Information
Device name     : ITeadSlave
Mac addr        : 20:13:12:05:01:75
Class of device : 1f00
Signal          : -61
Type            : Br/EDR Bluetooth
Bind state      : Nothing
```

We then connected to the master shield in command mode (defined previously). Attempts were made to connect the slave (through the master) as follows:

Example 60 **AT commands: The master connection to the slave**

```
AT+STATE?
+STATE:INITALIZED
AT+INIT
OK
AT+INQ
+INQ:2013:12:50175,1F00,7FFF
AT+STATE?
+STATE:INQUIRING
OK
AT+BIND?
+BIND:0:0:0
OK
AT+BIND=2013,12,50175
OK
AT+PAIR=2013,12,50175,20
OK
AT+STATE?
+STATE:PAIRED
OK
AT+LINK=2013,12,50175
OK
```

After the link command, the master was no longer accepting commands. It would have been beneficial to see the master state; however, it was not possible. Change the master shield toggle from “CMD” to “DAT”. Download the sample sketch to both the master and slave.

Example 61 **Master and slave sample sketch**

```
// This code demonstrates how to run the shield on Arduino and Galileo.
// A R D U I N O
HardwareSerial* gSerialStdPtr = &Serial1; // Arduino Uno, Tx(D1) Rx(D0)
HardwareSerial* gSerialTwoPtr = &Serial1; // Arduino Uno, Tx(D1) Rx(D0)
bool            gGalileo      = false;


void setup()
{
  gSerialStdPtr->begin(9600);
  gSerialTwoPtr->begin(9600);
  waitForUser(9);
} // setup
void loop()
{
  if (gSerialTwoPtr->available() > 0 )   // From BT
  {
    char fromBT = gSerialTwoPtr->read(); // Read BT
    if(gGalileo == true)
    {
      gSerialStdPtr->print(fromBT);  // To IDE
      gSerialTwoPtr->print(fromBT);  // To BT, echo
    }
    else
    {
      gSerialStdPtr->print(fromBT);  // To IDE, BT
    }
  }

  if(gSerialStdPtr->available() > 0)      // From IDE
  {
    char fromIDE = gSerialStdPtr->read(); // Read IDE
    if(gGalileo == true)
    {
      gSerialStdPtr->print(fromIDE); // To IDE, echo
      gSerialTwoPtr->print(fromIDE); // To BT
    }
    else
    {
      gSerialStdPtr->print(fromIDE); // To IDE, BT
    }
  }
} // loop
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
} // waitForUser
```

Bringing up the serial ports for both master and slave shields. Typing data in one serial window should transmit the data to the other serial window. In some cases, data was being transmitted, but it was garbled. The results are inconclusive and more debugging is required.

48.5 Results

Arduino* 101 inconclusive as a master. A Bluetooth sniffer cannot see it’s BTID being advertised. Duding configuration with AT commands the AT+STATUS? resulted in error and not sure if the ROLE and NAME settings stuck. Also cannot be programmed using Arduino 101 and still had to use Uno.

§

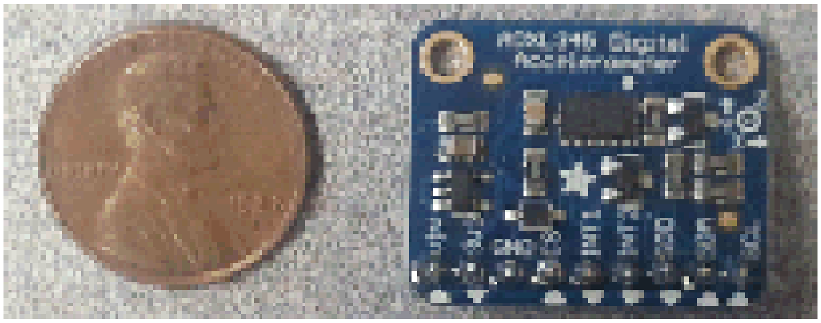
49 Adafruit* Digital Accelerometer

49.1 Use case

This is a low power, 3-axis MEMS accelerometer module with both I^{2}C and SPI interfaces. There are four sensitivity ranges from ±2G to ±16G. It supports output data rates ranging from 10 to 3200 Hz and is well suited for mobile devices. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Measurements of inclination of less than 1.0% are possible.

Key Info	Links
Product info	*https://www.adafruit.com/product/1231*.
Overview I^{2}C	*https://learn.adafruit.com/adxl345-digital-accelerometer*.
Library I^{2}C	*https://github.com/adafruit/Adafruit_Sensor*, dated 06/02/14. *https://github.com/adafruit/Adafruit_ADXL345*, dated 02/14/14.
Overview SPI	*https://www.sparkfun.com/tutorials/240*, 10/08/14.

Figure 132 **ADXL345 digital accelerometer**



49.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 to 5 V.
IOREF	3.3 to 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*

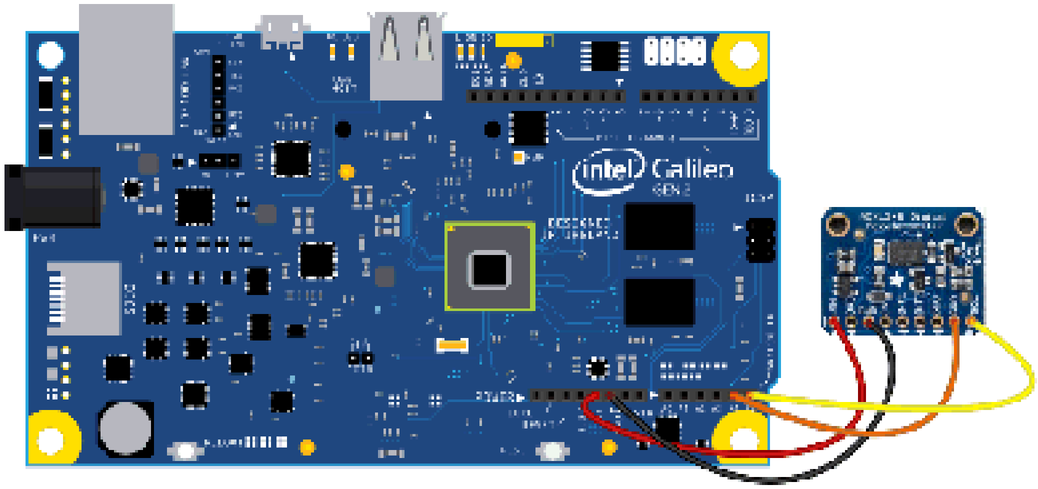
The breakout board has connection for I^{2}C and SPI.

I^{2}C configuration:

I^{2}C is a four-wire connection. See the diagram below that uses the SDA and SCL pins.

I^{2}C pin connections	
Breakout pin	**Function**
GND	Ground, connect to GND.
SDA	Data Line, connect to SDA or A4. Address in Adafruit_ADXL345_U.h file: `#define ADXL345_ADDRESS (0x53)`.
SCL	Clock Line, connect to SCL or A5.

Figure 133 **I^2C connections for the digital accelerometer to an Intel® Galileo second generation board**

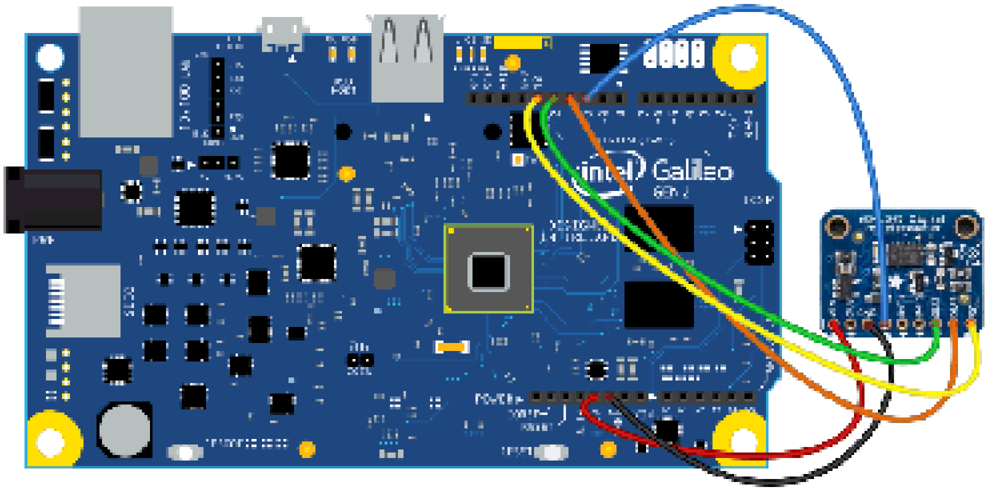


49.3 SPI configuration

SPI is a five-wire connection. At this time, the library that comes with the breakout board only supports I^2C. Sparkfun tutorial, see link above, was used to test for SPI.

SPI pin connections	
Pin Name	**Function**
GND	Ground, connect to GND.
CS	Chip Select/Slave Select, connect to D10.
SDO	MOSI, connect to D12.
SDA	MISO, connect to D11.
SCL	SCLK, connect to D13.

Figure 134 **SPI connections for the digital accelerometer to an Intel® Galileo second generation board**



49.4 Companion library

I^{2}C configuration:

SPI configuration:

The sketch, from the *Serial Peripheral Interface (SPI)*, compiles and download with no problems.

Example 62 **Sparkfun SPI sketch**

```
//Add the SPI library so we can communicate with the ADXL345 sensor
#include <SPI.h>

//Assign the Chip Select signal to pin 10.
int CS=10;

//This is a list of some of the registers available on the ADXL345.
//To learn more about these and the rest of the registers on the ADXL345, read the datasheet!
char POWER_CTL = 0x2D;//Power Control Register
char DATA_FORMAT = 0x31;
char DATAX0 = 0x32;//X-Axis Data 0
char DATAX1 = 0x33;//X-Axis Data 1
char DATAY0 = 0x34;//Y-Axis Data 0
char DATAY1 = 0x35;//Y-Axis Data 1
char DATAZ0 = 0x36;//Z-Axis Data 0
char DATAZ1 = 0x37;//Z-Axis Data 1

//This buffer will hold values read from the ADXL345 registers.
char values[10];
//These variables will be used to hold the x, y and z axis accelerometer values.
int x,y,z;

void setup(){
  //Initiate an SPI communication instance.
  SPI.begin();
  //Configure the SPI connection for the ADXL345.
  SPI.setDataMode(SPI_MODE3);
  //Create a serial connection to display the data on the terminal.
  Serial.begin(9600);

  //Set up the Chip Select pin to be an output from the Arduino.
  pinMode(CS, OUTPUT);
  //Before communication starts, the Chip Select pin needs to be set high.
  digitalWrite(CS, HIGH);

  //Put the ADXL345 into +/- 4G range by writing the value 0x01 to the DATA_FORMAT register.
  writeRegister(DATA_FORMAT, 0x01);
  //Put the ADXL345 into Measurement Mode by writing 0x08 to the POWER_CTL register.
  writeRegister(POWER_CTL, 0x08);  //Measurement mode
}

void loop(){
  //Reading 6 bytes of data starting at register DATAX0 will retrieve the x,y and z
acceleration values from the ADXL345.
  //The results of the read operation will get stored to the values[] buffer.
  readRegister(DATAX0, 6, values);

  //The ADXL345 gives 10-bit acceleration values, but they are stored as bytes (8-bits). To get
the full value, two bytes must be combined for each axis.
  //The X value is stored in values[0] and values[1].
  x = ((int)values[1]<<8)|(int)values[0];
  //The Y value is stored in values[2] and values[3].
  y = ((int)values[3]<<8)|(int)values[2];
  //The Z value is stored in values[4] and values[5].
  z = ((int)values[5]<<8)|(int)values[4];

  //Print the results to the terminal.
  Serial.print(x, DEC);
  Serial.print(',');
  Serial.print(y, DEC);
  Serial.print(',');
  Serial.println(z, DEC);
  delay(10);
}

//This function will write a value to a register on the ADXL345.
//Parameters:
//  char registerAddress - The register to write a value to
//  char value - The value to be written to the specified register.
void writeRegister(char registerAddress, char value){
  //Set Chip Select pin low to signal the beginning of an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the register address over SPI.
  SPI.transfer(registerAddress);
  //Transfer the desired register value over SPI.
  SPI.transfer(value);
  //Set the Chip Select pin high to signal the end of an SPI packet.
  digitalWrite(CS, HIGH);
}

//This function will read a certain number of registers starting from a specified address and
store their values in a buffer.
//Parameters:
//  char registerAddress - The register addresse to start the read sequence from.
//  int numBytes - The number of registers that should be read.
//  char * values - A pointer to a buffer where the results of the operation should be stored.
void readRegister(char registerAddress, int numBytes, char * values){
  //Since we're performing a read operation, the most significant bit of the register address
should be set.
  char address = 0x80 | registerAddress;
  //If we're doing a multi-byte read, bit 6 needs to be set as well.
  if(numBytes > 1)address = address | 0x40;

  //Set the Chip select pin low to start an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the starting register address that needs to be read.
  SPI.transfer(address);
  //Continue to read registers until we've read the number specified, storing the results to
the input buffer.
  for(int i=0; i<numBytes; i++){
    values[i] = SPI.transfer(0x00);
  }
  //Set the Chips Select pin high to end the SPI packet.
  digitalWrite(CS, HIGH);
}
```

49.5 Compile and upload

Running the I^{2}C sketch provides the output below.

The setup will print the x, y and z acceleration values in the serial terminal. Move the board in different planes to see the x, y, and z values change accordingly.

```
Sketch Output for ‘sensortest’
```

```
Accelerometer Test
------------------------------------
Sensor:       ADXL345
Driver Ver:   1
Unique ID:    12345
Max Value:    -156.91 m/s^2
Min Value:    156.91 m/s^2
Resolution:   0.04 m/s^2
------------------------------------

Data Rate:    100  Hz
Range:         +/- 16  g

X: -0.63  Y: -2.63  Z: 9.30  m/s^2
X: -0.59  Y: -2.55  Z: 9.34  m/s^2
X: -0.59  Y: -2.55  Z: 9.38  m/s^2
X: -0.59  Y: -2.55  Z: 9.38  m/s^2
X: -0.55  Y: -2.55  Z: 9.38  m/s^2
```

Running the SPI sketch provides the output below. Move the board in different planes to see the x, y, and z values change accordingly.

```
Sketch Output for SPI version
-10,-4,51
-10,-3,51
-15,-4,51
-10,-3,51
-10,-3,51
-15,-4,51
-10,-4,51
-15,-4,51
-15,-4,51
-10,-4,51
-15,-4,51
-10,-4,51
-10,-4,51
-16,-4,51
-10,-4,51
-10,-4,51
-10,-13,51
-15,-4,51
-10,-4,51
```

49.6 Results

Arduino* 101 compatible.

§

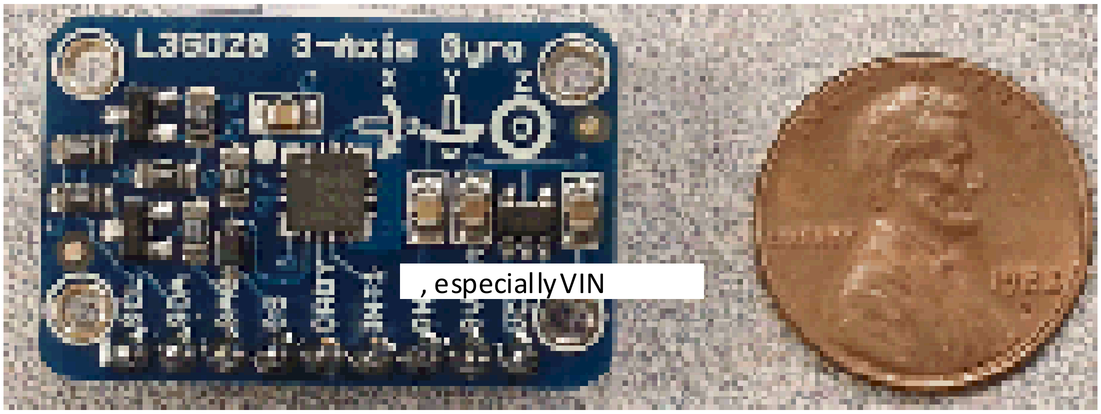
50 Adafruit* L3GD20 Gyro

50.1 Use case

The gyroscope sensor is an angular rate sensor that can sense twisting and turning motions. It is often paired with an accelerometer for 3D motion capture and inertial measurement in order to tell how an object is moving. The L3GD20 supports three full axes of sensing. The chip can be set to +- 250, 500 or 2000 degree/second scale. The breakout board supports I^{2}C and SPI interfaces. The library supports SPI on any 4 digital I/O pins.

Key Info	Links
Product info	*http://www.adafruit.com/products/1032*
Library	*https://github.com/adafruit/Adafruit_L3GD20* Date:5/11/15

Figure 135 **L3GD20 3-axis gyro**



50.2 Hardware summary

Key Info	Description/Links
Operating Voltage	3.3 or 5 V.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
L3GD20 Datasheet	Low-Noise Microphone Amp Datasheet. *http://www.adafruit.com/datasheets/L3GD20.pdf*.

Sketches can use the SPI or I^{2}C interface so some of the pins have a dual purpose in the descriptions.

Pin Name	Function
SCL	I^{2}C clock/SPI clock - Connect to SCL for I^{2}C or the Intel® Galileo pin used for SPI clock (pin 13).
SDA	I^{2}C data/SPI data input - Connect to SDA for I^{2}C or the Intel Galileo pin used for MOSI (pin 12).
SA0	I^{2}C least significant bit of device address/SPI MISO – not needed for I^{2}C or the pin used for MISO (pin 11)
CS	SPI chip select – Connect to pin for chip select (pin 10 default)
DRDY	Data ready/FIFO interrupt – not used in this example
INT1	Programmable interrupt – not used in this example
GND	Ground – connect to GND
Vout	Voltage output – not used in this example
VCC	Voltage source – Connect to 5 V

50.3 Companion library

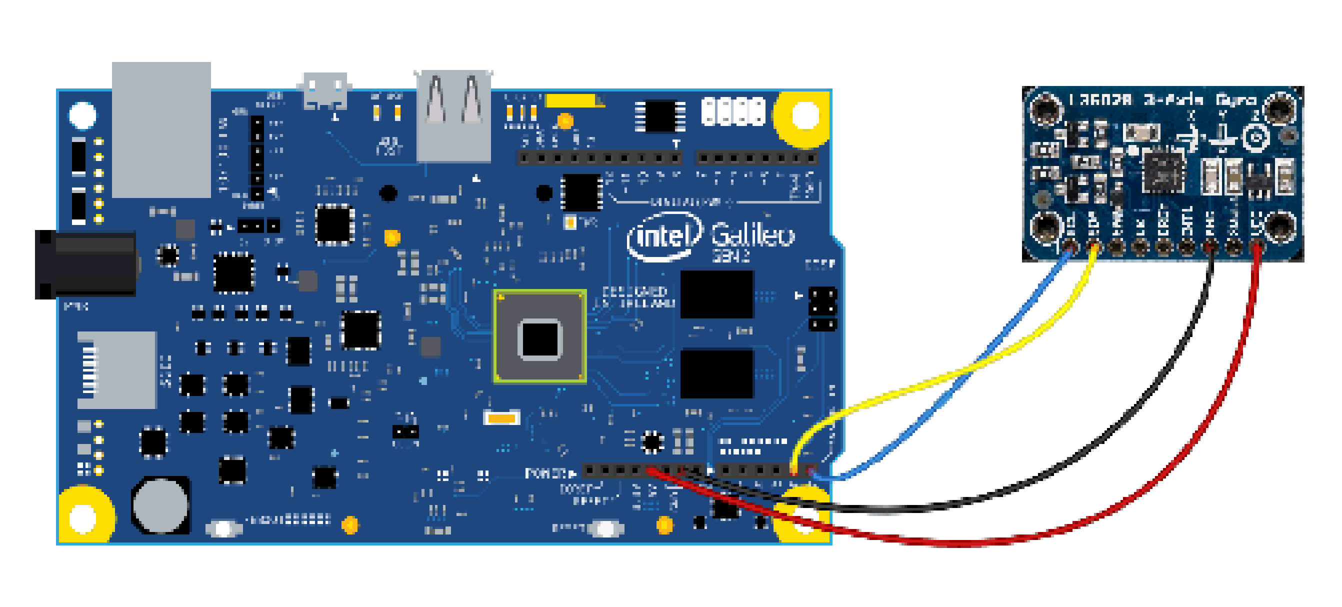
None

50.4 Compile and test

To connect the shield for I²C, make the following L3DG20 pin connections:

- SCL to SCL (or A5)
- SDA to SDA (or A4)
- VCC to 5 V
- GND to GND
- Uncomment the "#define USE_I2C"

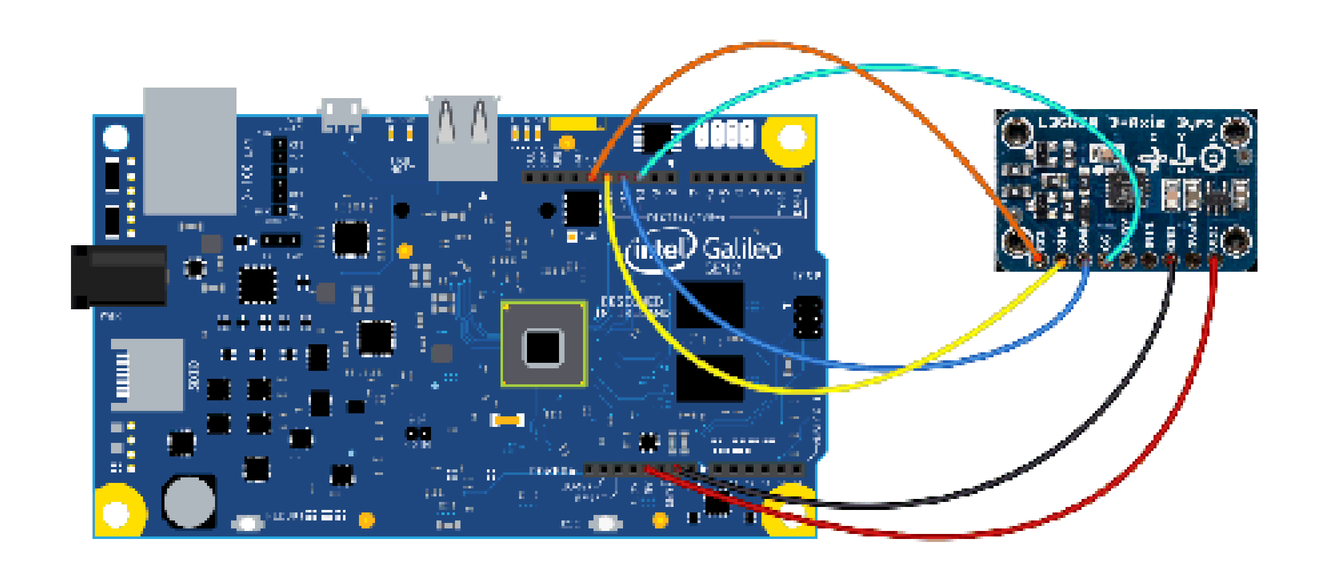
Figure 136 **Connecting an L3DG20 gyro using the I²C interface. Shown here with an Intel® Galileo second generation board**



To connect the shield for SPI, make the following L3DG20 pin connections:

- SCL to pin D13
- SDA(MOSI) to pin D12
- SA0 (MISO) to pin D11
- CS (chip select) to pin D10
- VCC to 5 V on the Intel® Galileo board
- GND to GND on the Intel Galileo board
- Comment out the "#define USE_I2C"

Figure 137 **Connecting an L3DG20 gyro using the SPI interface. Shown here with an Intel® Galileo second generation board**



The following sketch was taken from the library installation. Uncomment the define statement that can switch the sensor between I^{2}C and SPI. After uploading the sketch, rotate the sensor on different axes to see the data change for the X, Y, and Z axes.

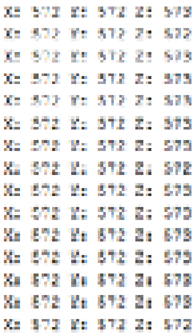
Example 63 **Adafruit_L3GD20_test sketch for SPI/I2C (installed with the library)**

```
/***************************************************
  This is an example for the Adafruit Triple-Axis Gyro sensor

  Designed specifically to work with the Adafruit L3GD20 Breakout
  ----> https://www.adafruit.com/products/1032

  These sensors use I2C or SPI to communicate, 2 pins (I2C)
  or 4 pins (SPI) are required to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Kevin "KTOWN" Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ****************************************************/

#include <Wire.h>
#include <Adafruit_L3GD20.h>

// Comment this next line to use SPI
#define USE_I2C

#ifdef USE_I2C
  // The default constructor uses I2C
  Adafruit_L3GD20 gyro;
#else
  // To use SPI, you have to define the pins
  #define GYRO_CS 10 // labeled CS
  #define GYRO_DO 11 // labeled SA0
  #define GYRO_DI 12  // labeled SDA
  #define GYRO_CLK 13 // labeled SCL
  Adafruit_L3GD20 gyro(GYRO_CS, GYRO_DO, GYRO_DI, GYRO_CLK);
```

```
#endif

void setup()
{
  Serial.begin(9600);

  // Try to initialise and warn if we couldn't detect the chip
   if (!gyro.begin(gyro.L3DS20_RANGE_250DPS))
  //if (!gyro.begin(gyro.L3DS20_RANGE_500DPS))
  //if (!gyro.begin(gyro.L3DS20_RANGE_2000DPS))
  {
    Serial.println("Oops ... unable to initialize the L3GD20. Check your wiring!");
    while (1);
  }
}

void loop()
{
  gyro.read();
  Serial.print("X: "); Serial.print((int)gyro.data.x);   Serial.print(" ");
  Serial.print("Y: "); Serial.print((int)gyro.data.y);   Serial.print(" ");
  Serial.print("Z: "); Serial.println((int)gyro.data.z); Serial.print(" ");
  delay(100);
}
```

The data is displayed to a serial terminal. The values of angular rate changes according to the pivoting axis. One issue see is that for SPI, some data values seem to be invalid on an Intel Galileo board.

Figure 138 **Serial terminal output**



```
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 572
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 572
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 573
X: 572 Y: 572 Z: 572
```

50.5 Results

Arduino* 101 compatible for I^{2}C/SPI.

§

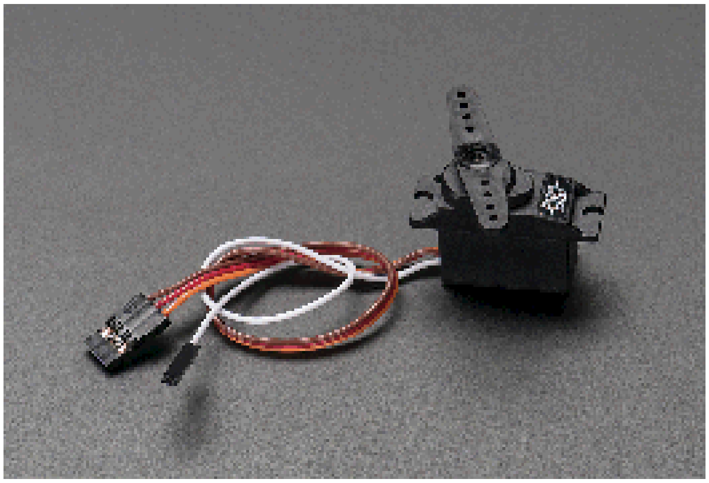
51 Datan* Analog Feedback Micro Servo - Metal Gear

51.1 Use case

This is a metal-geared “micro” sized hobby servo with the potentiometer wiper brought out to a fourth wire. This wire can be read with an analog input to get the servo’s position. This can be used to improve stability or allow recording of servo motion.

Key Info	Links
URL	*https://www.adafruit.com/products/1450*.
Library	No library needed.
Sketch	*https://github.com/adafruit/Feedback-Servo-Record-and-Play*.

Figure 139 **Analog feedback servo**

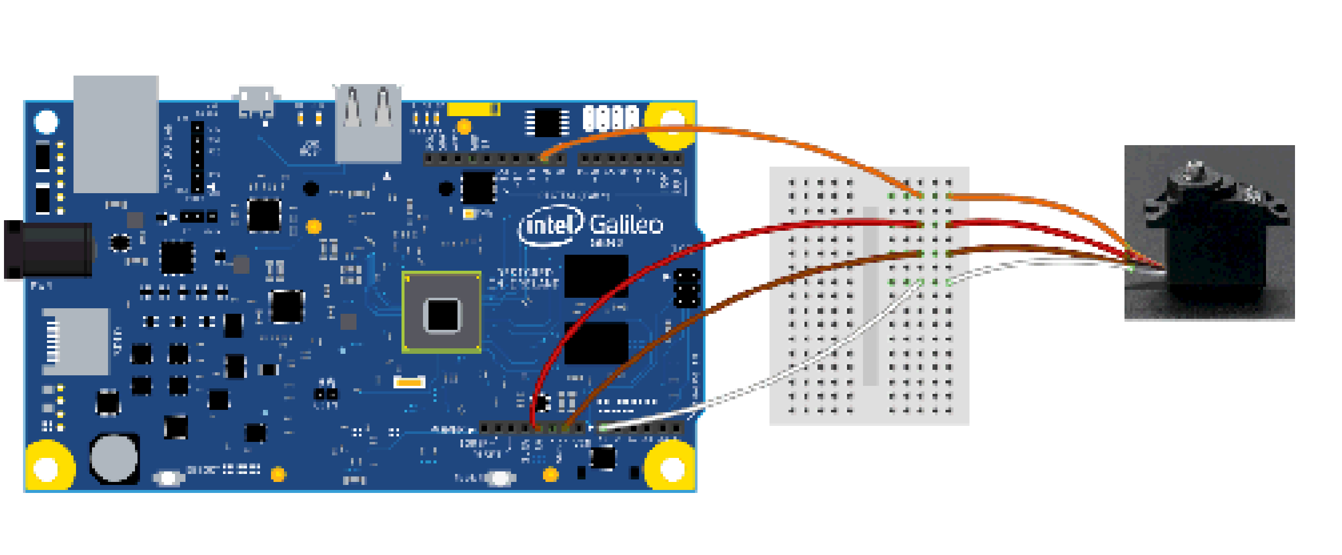


51.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Use VIN as power source	No.
Operating voltage	6 V.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Torque	25 oz-in/1.8*cm @ 6 V.
Speed	0.1 sec/60° @ 6 V.

Wire color	Function
Brown	Connect to GND.
Red	Connect to 5 V power supply.
Orange	Control wire – connect to D9.
White	Feedback wire – connect to A0.

Figure 140 **Connecting the analog feedback micro servo. Shown here with an Intel® Galileo second generation board**



51.3 Companion library

No library required.

51.4 Compile and upload

Connections:

- White wire (feedback) to A0
- Orange wire (servo) to D9
- Red to 5 V
- Brown to GND
- Arduino* 101 needs external power.

Run *Example 64* on the Intel® Galileo board and the servo should sweep through 180 degrees and show the position in the serial terminal.

Example 64 **Sketch to calibrate the feedback and sweep 180 degrees**

```
#include <Servo.h>

// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.


#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int servoPin = 9;
int feedbackPin = A0;

int pos = 0;    // variable to store the servo position

// calibration values
int minDegrees;
int maxDegrees;
int minFeedback;
int maxFeedback;

void setup()
{
  Serial.begin(9600);
  delay(2000);
  myservo.attach(servoPin);  // attaches the servo on pin 9 to the servo object

  calibrate(myservo, feedbackPin, 0, 180); // calibrate for the 20 - 160 degree range
}


void loop()
{
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    Serial.println(analogRead(feedbackPin));
    delay(100);                      // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    Serial.println(analogRead(feedbackPin));
    delay(100);                       // waits 15ms for the servo to reach the position
  }
}

/*
  This function establishes the feedback values for 2 positions of the servo.
  With this information, we can interpolate feedback values for intermediate positions
*/
void calibrate(Servo servo, int analogPin, int minPos, int maxPos)
{
  // Move to the minimum position and record the feedback value
  servo.write(minPos);
  minDegrees = minPos;
  delay(2000); // make sure it has time to get there and settle
  minFeedback = analogRead(analogPin);

  // Move to the maximum position and record the feedback value
  servo.write(maxPos);
  maxDegrees = maxPos;
  delay(2000); // make sure it has time to get there and settle
  maxFeedback = analogRead(analogPin);
}
```

51.5 Results

.

Arduino* 101 compatible. Needs Barreljack power

§

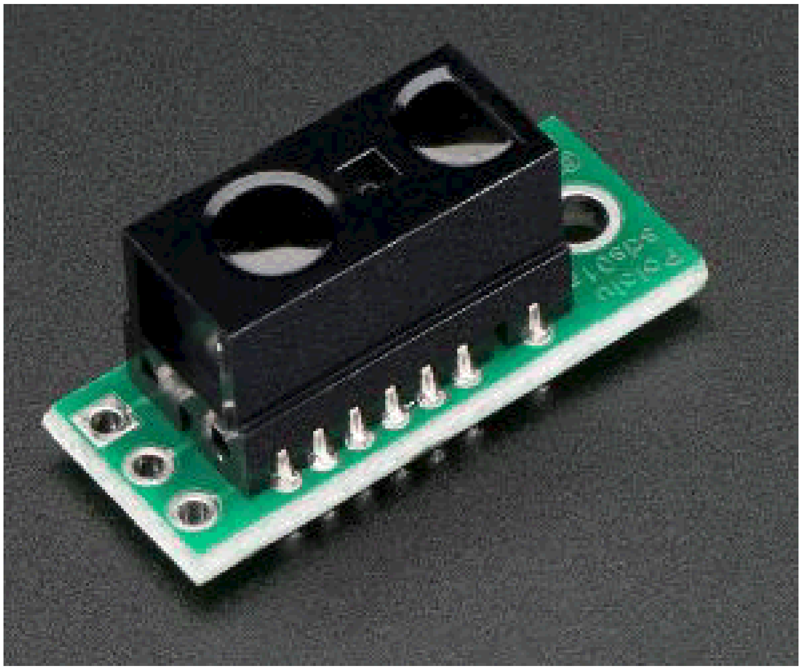
52 Sharp* Digital Distance Sensor w/Pololu* Carrier

52.1 Use case

This breakout board is a proximity sensor. When an object is within in a specific distance, the attached digital pin is signaled. This sensor only gives an indication when an object is nearby, not how far.

Key Info	Links
Order/product info	*https://www.adafruit.com/products/1927*.
Guide	*http://www.pololu.com/product/1134*.
Library	None.

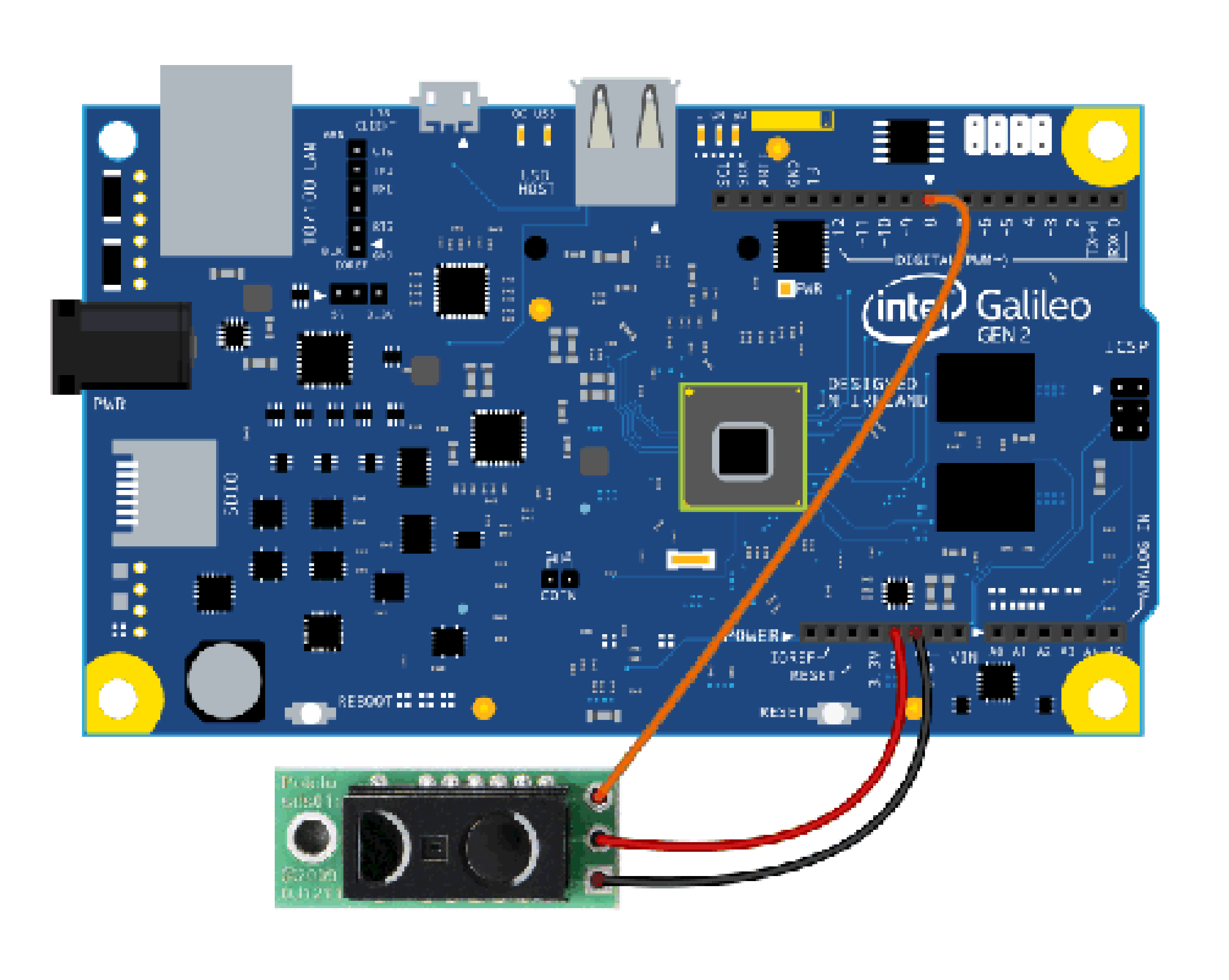
Figure 141 **Sharp* distance sensor**



52.2 Hardware summary

Key Info	Description/Links
Operating voltage	2.7 to 6.2 V.
Use VIN as power source	No.
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematics	None.
Sensor datasheet	*http://www.pololu.com/file/0J154/GP2Y0D810Z0F.pdf*.
Distance	0.8 to 3.9 in.

Figure 142 **Sharp* distance sensor. Shown here connected to an Intel® Galileo second generation board**



Breakout Pin	Function with Board Connections
GND	Ground, connect to GND.
OUT	Digital, connect to D8.

52.3 Companion library

No library required.

52.4 Compile and upload

Example 65 sets up the sensor to read from digital pin 8. The pin is then read continuously providing a HIGH or LOW feedback. Notice that the LED on the sensor will work without the digital pin connection; therefore, verification is done with the LED and/or the serial port.

Example 65 **Digital pin sketch**

```
int inputPin = 8;   // Digital Pin for Distance Sensor
int ledPin   = 13;  // Led
void setup()
{
  Serial.begin(9600);
  delay(1000*3);                   // For debugging
  pinMode(ledPin,   OUTPUT);
  pinMode(inputPin, INPUT);
  digitalWrite(ledPin, LOW);
  Serial.println("setup");
} // setup
void loop()
{
  if(digitalRead(inputPin) == LOW) // returns HIGH or LOW
  {
    digitalWrite(ledPin, HIGH);
    Serial.println("loop-LOW");
  }
  else
  {
    digitalWrite(ledPin, LOW);
    Serial.println("loop-HIGH");
  }
} // loop
```

When the sensor detects and object within the specified limits, the digital output will be LOW. The example sketch checks for a digital LOW and sets the LED to high and interacts with the configured led and the serial port.

52.5 Results

Arduino* 101 compatible.

§

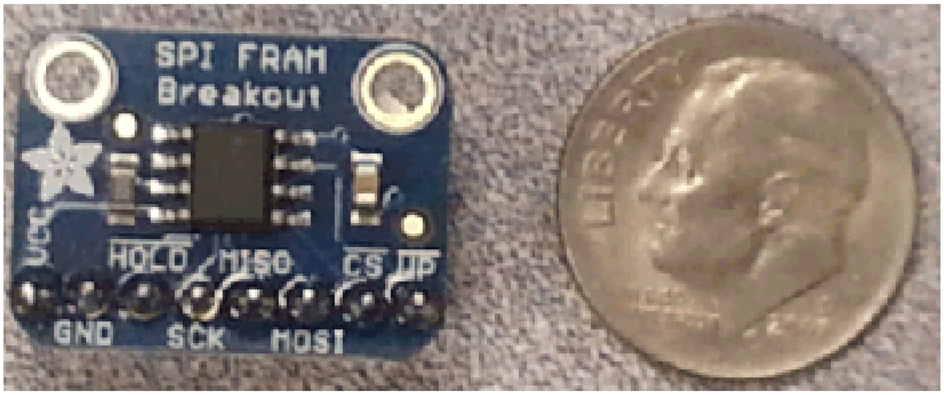
53 Adafruit* Nonvolatile FRAM Breakout SPI

53.1 Use case

FRAM (ferroelectric random-access memory) is similar to DRAM (dynamic random-access memory) except with a ferroelectric layer instead of a dielectric layer. This gives it stable handling for writing nonvolatile memory fast. With the SPI FRAM breakout board, FRAM storage can be added to a project for lower power usage and faster write performance. It's excellent for low-power or inconsistent-power data logging or data buffering where streaming fast and keeping data when there is no power are requirements. Unlike flash or EEPROM, there are no pages to worry about. Each byte can be read/written 10^12 times so there is no problem with wear leveling.

Key Info	Links
URL	*http://www.adafruit.com/product/1897*
Library	*https://github.com/adafruit/Adafruit_FRAM_SPI* *https://github.com/adafruit/Adafruit_FRAM_SPI/archive/master.zip* Date: 11/11/14

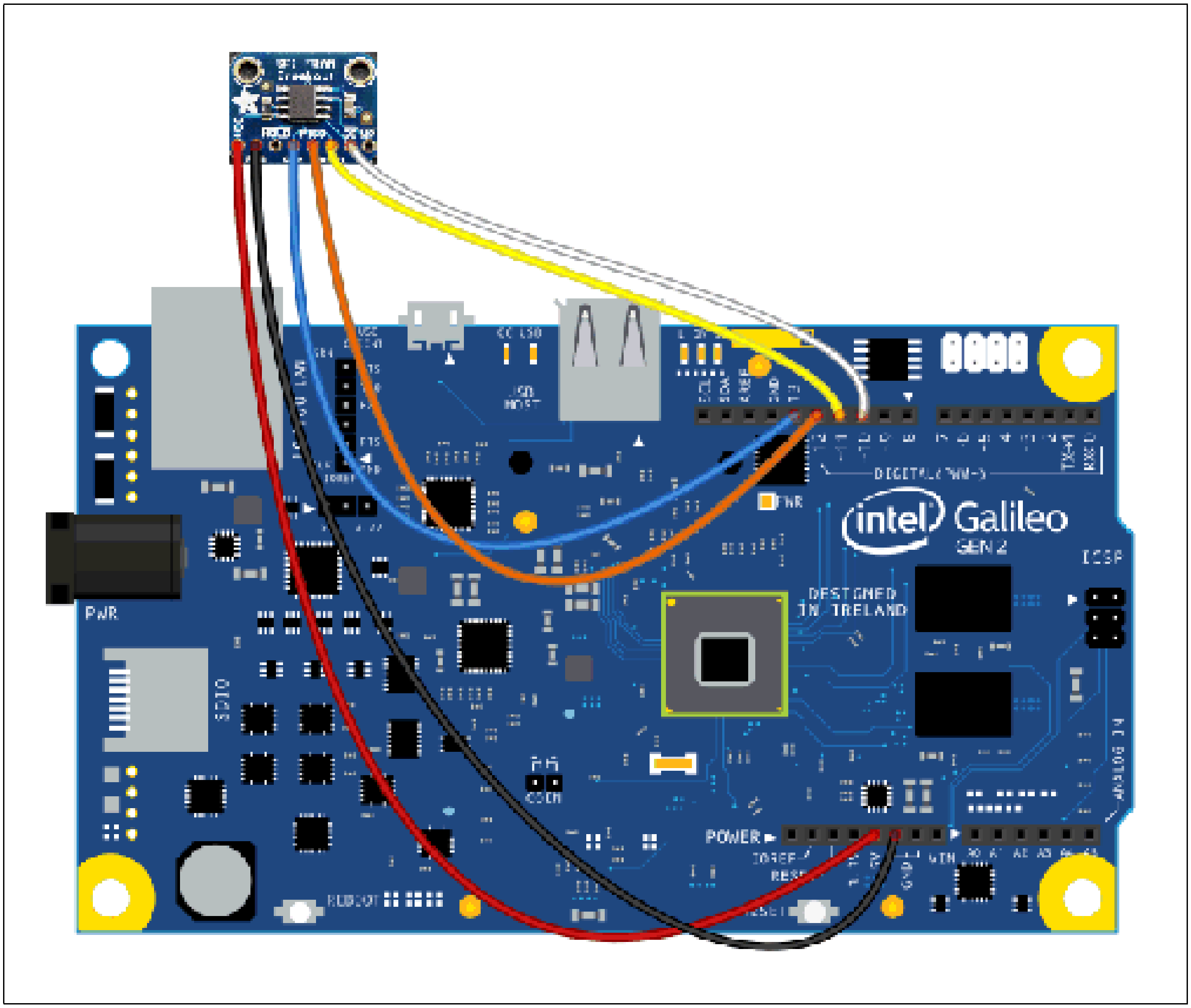
Figure 143 **SPI FRAM breakout**



53.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Use VIN as power source	No
Operating voltage	3.3 or 5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)

Figure 144 **Connecting the FRAM breakout. Shown here with an Intel® Galileo second generation board**



Pin	Function
VCC	Connect to 5 V.
GND	Connect to GND.
SCK	SPI Clock – Connect to D13
MISO	SPI MISO – Connect to D12
MOSI	SPI MOSI – Connect to D11
CS	SPI Chip Select – Connect to D10
UP	Not used in this test
HOLD	Not used in this test

53.3 Companion library

The library will compiles.

The MB85RS64V sketch that is installed with the library will compile and run.

53.4 Compile and upload

Install the library and make the modification from the previous section. Wire the Intel® Galileo board according to the pin diagram as in the diagram below. The chip select pin can be changed to another pin number, but it must be updated in the sketch (uint8_t FRAM_CS).

The following sketch reads/writes the number of restarts that have been generated on the FRAM. Following is a section to write specific values to all bytes and then a section to read the bytes for comparison. Any errors will be displayed in the serial terminal. Upload the sketch and open the serial terminal. The time it takes to write/read all 8192 bytes in milliseconds is displayed as well as the number of errors (which should be 0).

Example 66 **Read/write to FRAM**

```
#include <SPI.h>
#include "Adafruit_FRAM_SPI.h"

/* Example code for the Adafruit SPI FRAM breakout */
uint8_t FRAM_CS = 10;

//Adafruit_FRAM_SPI fram = Adafruit_FRAM_SPI(FRAM_CS);  // use hardware SPI

uint8_t FRAM_SCK= 13;
uint8_t FRAM_MISO = 12;
uint8_t FRAM_MOSI = 11;
//Or use software SPI, any pins!
Adafruit_FRAM_SPI fram = Adafruit_FRAM_SPI(FRAM_SCK, FRAM_MISO, FRAM_MOSI, FRAM_CS);

uint16_t          addr = 0;
uint16_t bytestocheck = 8192;
uint32_t starttime_ms;

void setup(void) {
  Serial.begin(9600);

  if (fram.begin()) {
    Serial.println("Found SPI FRAM");
  } else {
    Serial.println("No SPI FRAM found ... check your connections\r\n");
    while (1);
  }
  delay(2000);
  // Read the first byte
  uint8_t restarts = fram.read8(0x0);
  Serial.print("Restarted "); Serial.print(restarts); Serial.println(" times");

  starttime_ms = millis();

  // Test write ++
  for (uint16_t i = 0x0; i < bytestocheck; i++) {
    fram.writeEnable(true);
    fram.write8(i, i+restarts+1);
    fram.writeEnable(false);
  }

  // dump the entire 8K of memory!
  uint8_t value;
  uint8_t lastvalue;
  int errorcount = 0;
  for (uint16_t a = 0; a < bytestocheck; a++) {
    value = fram.read8(a);
    if (a > 0) {
      lastvalue++;
      if (value != lastvalue) {
        errorcount++;
        Serial.println("ERROR in value");
        Serial.print(value);
        Serial.print(" != ");
        Serial.println(lastvalue+1);
      }
    }
    else {
      lastvalue = value;
    }
  }
  Serial.println(" ");
  Serial.print("time to complete ");
  Serial.print(millis());
  Serial.println(" ms");

  Serial.print("Complete ");
  if (errorcount > 0) {
     Serial.print(errorcount);
     Serial.println(" mismatches - failure");
  }
  else {
     Serial.print(errorcount);
     Serial.println(" mismatches - success");
  }
}

void loop(void) {

}
```

53.5 Results

Arduino* 101 compatible.

§

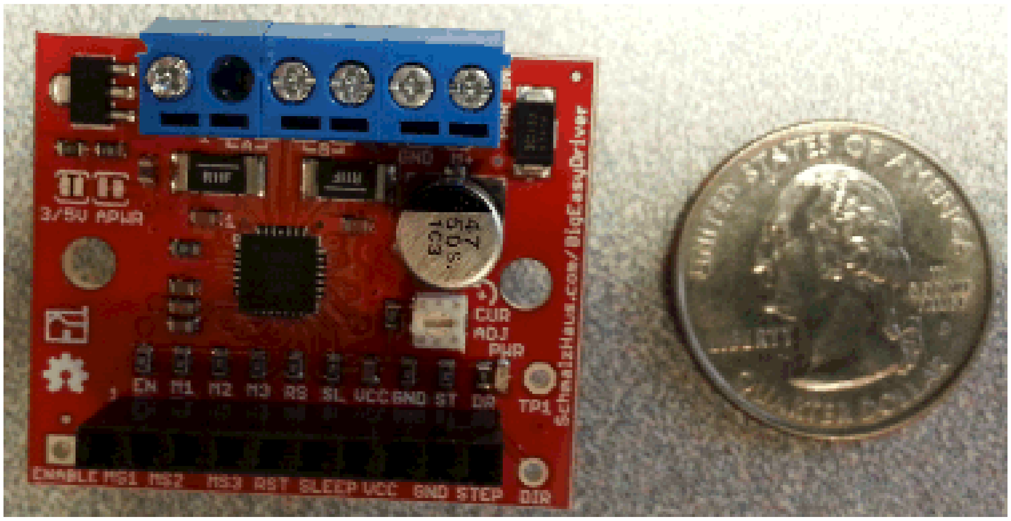
54 Sparkfun* Big Easy Driver V1.2

54.1 Use case

The Big Easy driver is a stepper motor driver board for bipolar stepper motors up to 1.4A/phase. It is based on the Allegro A4983 stepper driver chip. It is a chopper micro stepping driver which defaults to 16 step micro stepping mode. It can take a maximum motor drive voltage of around 35 V and includes onboard 5 V/3.3 V regulation, so only one supply is necessary.

Key Info	Links
Order/product info	*https://www.sparkfun.com/products/11876* *http://www.schmalzhaus.com/BigEasyDriver/*
Library	No library.
Sketches	*http://www.schmalzhaus.com/EasyDriver/Examples/EasyDriverExamples.html*

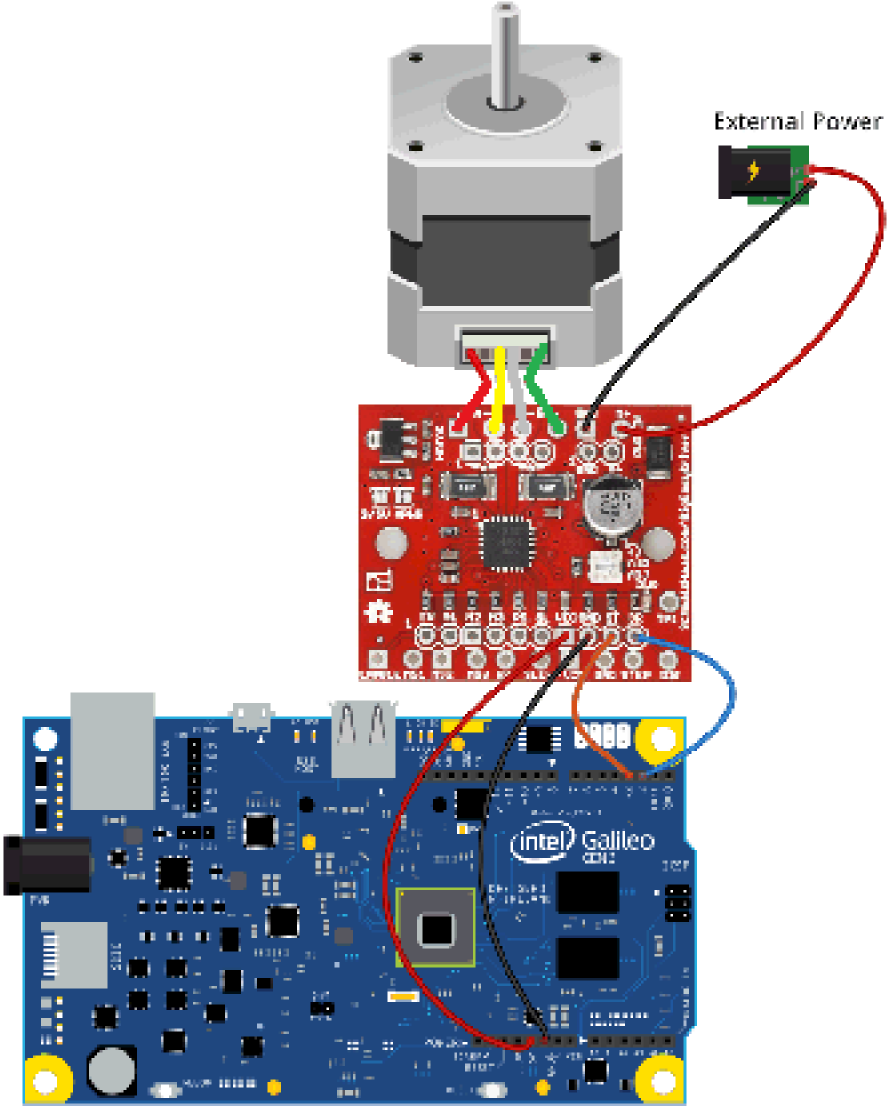
Figure 145 **Big Easy driver board v1.2**


GND M+
3/5V APWR
CUR ADJ
PWR
EN M1 M2 M3 RS SL VCC GND ST DR
TP1
SchmalzHaus.com/BigEasyDriver
ENABLE MS1 MS2 MS3 RST SLEEP VCC GND STEP DIR

54.2 Hardware summary

Key Info	Description/Links
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Use VIN as power source	No
Operating voltage	3.3 or 5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Schematic	*http://www.schmalzhaus.com/BigEasyDriver/v1_2/BigEasyDriver_v12_sch.pdf*
Motors	ST1

Figure 146 **Big Easy driver (parts are not to scale) connection diagram. Shown here with an Intel® Galileo second generation board**



Breakout pin	Function
DR	Direction – connect to D2
ST	Stepper – connect to D3
GND	Connect to GND
VCC	Connect to 5 V ***Note:*** On some production boards, the potentiometer is known to be backwards.
SL	Sleep – not connected for this test
RS	Reset – not connected for this test
M3	1/16 microstep – not connected for this test
M2	1/16 microstep – not connected for this test
M1	1/16 microstep – not connected for this test
EN	Enable – not connected for this test

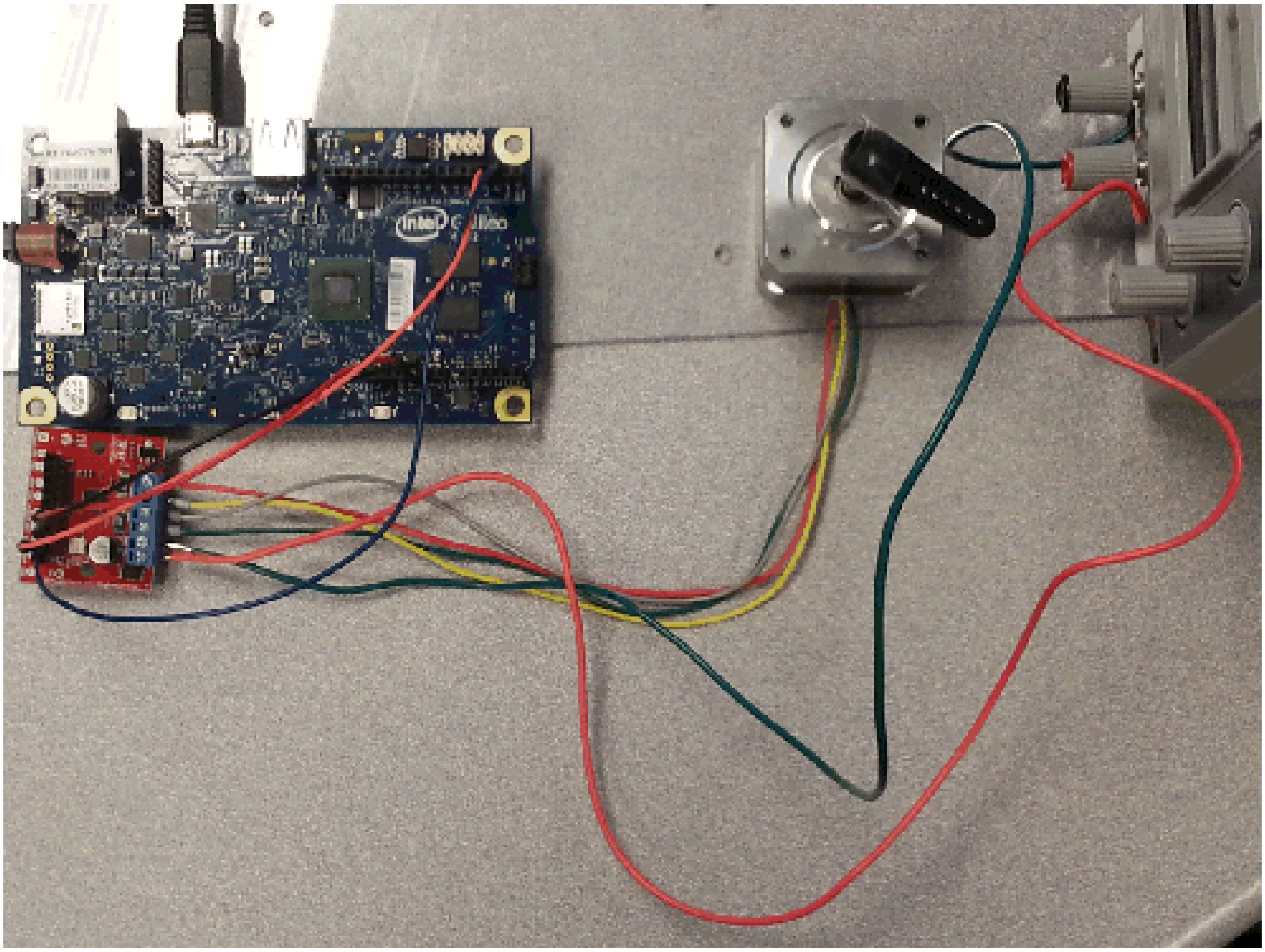
54.3 Companion library

No library.

54.4 Compile and upload

The physical connections are show as follows:

Figure 147 **Connecting the Big Easy driver using a NEMA017 servo to an Intel® Galileo second generation board**



Example 67 runs the stepper motor forward and backward. Make the connections from above and connect to a DC power supply using the required voltage for the stepper motor and reduce the current level.

Example 67 **Move motor forward and backward sketch**

```
int Distance = 0;  // Record the number of steps we've taken
#define dirpin 2
#define stpin 3

void setup() {
  pinMode(dirpin, OUTPUT);
  pinMode(stpin, OUTPUT);
  digitalWrite(dirpin, LOW);
  digitalWrite(stpin, LOW);
}

void loop() {
  digitalWrite(stpin, HIGH);
  delayMicroseconds(100);
  digitalWrite(stpin, LOW);
  delayMicroseconds(100);
  Distance = Distance + 1;   // record this step

  // Check to see if we are at the end of our move
  if (Distance == 3600)
  {
    // We are! Reverse direction (invert DIR signal)
    if (digitalRead(dirpin) == LOW)
    {
      digitalWrite(dirpin, HIGH);
    }
    else
    {
      digitalWrite(dirpin, LOW);
    }
    // Reset our distance back to zero since we're
    // starting a new move
    Distance = 0;
    // Now pause for half a second
    delay(500);
  }
}
```

54.5 Results

Arduino* 101 compatible

§

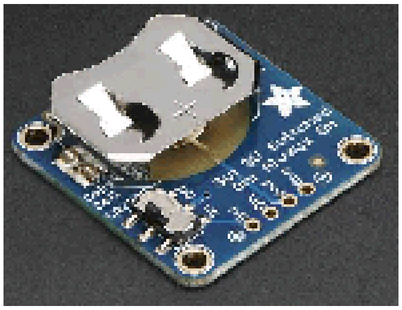
55 Adafruit* Coin Cell Battery

55.1 Use case

The breakout board uses a coin cell battery holder, soldered to the board, to provide power to the Intel® Galileo internal clock. The board has an on/off switch to enable the battery power. There is an option to bypass the switch.

Key Info	Links
Order/product info	*https://www.adafruit.com/products/1871*
Library	None

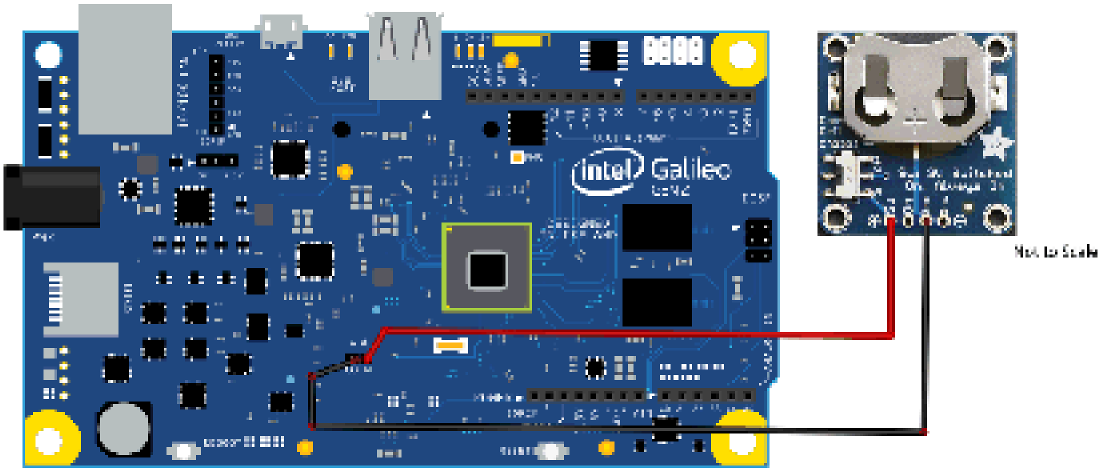
Figure 148 **Coin cell battery breakout board**



55.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 V
Use VIN as power source	No
Battery	CR2032 ***Note:*** The board has the wrong battery type imprinted on the breakout. It cites CR2302, which does not exist.
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*

Figure 149 **Coin cell battery on an Intel® Galileo second generation board**



Breakout pin	Function
SW	Power, connect to 'Coin' positive pin. Only one power is necessary.
ON	Power, Not Used
GND	Ground, connect to GND.
GND	Ground, Not used

The wiring diagram is connected to the on/off switch. Ensure that the switch is turned on so that the clock is provided power.

55.3 Companion library

No library required.

55.4 Compile and upload

Example 68 makes a Linux call to obtain the current date and time. The default time is set for January 2001. *Example 68* will display the current time in the IDE serial port.

Example 68 **Obtain current date and time in Linux**

```
// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial;  // Galileo, /dev/ttyGSO, Tx pin

void setup()
{
  gSerialStdPtr->begin(9600);
  waitForUser(5);                    // Give user time to open serial terminal

  gSerialStdPtr->println("Current Times");
}
void loop()
{
  gSerialStdPtr->println("SW Clock");
  system("date > /dev/ttyGS0");      // Current Time
  gSerialStdPtr->println("HW Clock");
  system("hwclock > /dev/ttyGS0");      // Current Time
  delay(1000*3);
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
}
```

Example 69 will make the Linux calls to set the software time, and a second call to sync the software clock and the hardware clock. The main loop will simply display the current date and time. The set date and time is July 4, 2014 at 8AM.

Example 69 **Set date and time in Linux**

```
// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial;  // Galileo, /dev/ttyGSO, Tx pin

void setup()
{
  gSerialStdPtr->begin(9600);
  waitForUser(5);                    // Give user time to open serial terminal

  gSerialStdPtr->println("SW Clock");
  system("date > /dev/ttyGS0");      // Current Time
  delay(1000*2);
  gSerialStdPtr->println("HW Clock");
  system("hwclock > /dev/ttyGS0");      // Current Time
  delay(1000*2);

  gSerialStdPtr->println("Set Software Clock");
  system("date 070408002014 > /dev/ttyGSO");       // Software Clock
  delay(1000*2);

  gSerialStdPtr->println("Set Hardware Clock");
  system("hwclock --systohc --utc > /dev/ttyGSO"); // Hardware Clock
  delay(1000*2);
}
void loop()
{
  system("date > /dev/ttyGS0");      // Current Time
  system("hwclock > /dev/ttyGS0");      // Current Time
  delay(1000*5);
}
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
}
```

Download the first sketch again to prevent the clock being set when the board powers up.

Turn the physical power off on the board. Remember to disconnect the USB cable also to ensure the board receives no power.

Restart the board and load the first sketch (if required). The clock should retain its time.

55.5 Results

Arduino* 101 is not supported.

§

56 Sparkfun* Line Sensor Breakout

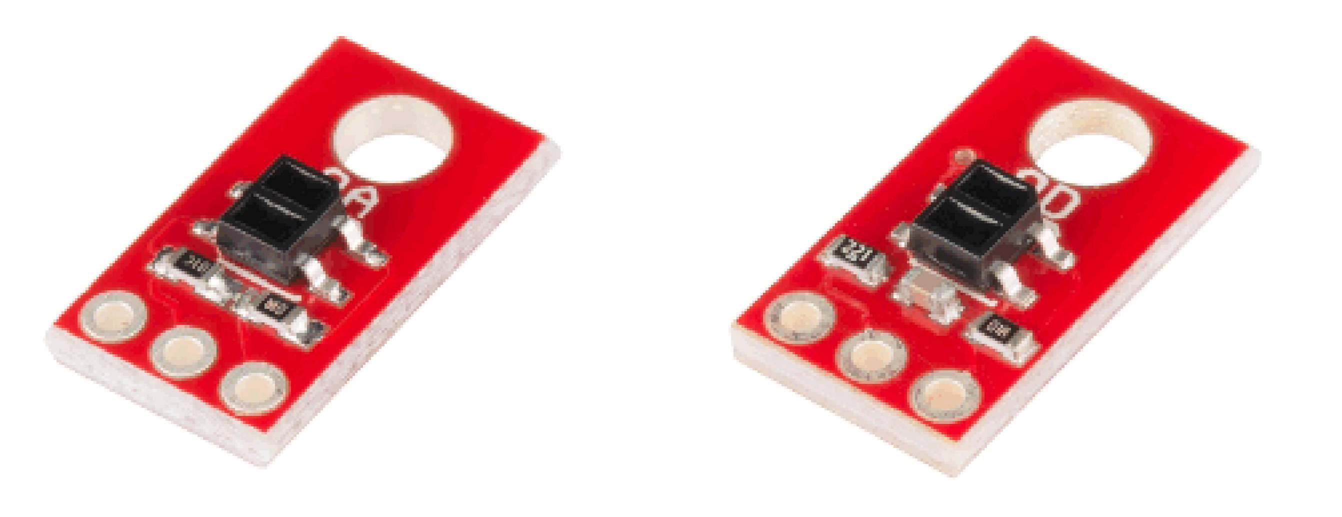
56.1 Use case

This breakout board's reflectance sensor is comprised of two parts (an IR emitting LED and an IR sensitive phototransistor). When applying power to VCC and GND, the IR LED inside the sensor will illuminate. A 100 ohm resistor is onboard and placed in series with the LED to limit current. The output of the phototransistor is tied to a 10 nF capacitor. The faster that capacitor discharges, the more reflective the surface is.

The sensor is available in Analog and Digital. These sensors are widely used in line following robots. White surfaces reflect more light than black, so, when directed towards a white surface, the capacitor will discharge faster than it would when pointed towards a black surface.

Key Info	**Links**
Order/product info	*https://www.sparkfun.com/products/9453 (Analog)* *https://www.sparkfun.com/products/9454 (Digital)*
Guide	*http://bildr.org/2011/06/qre1113-arduino/*
Library	None.

Figure 150 **Sparkfun* Line Sensor Breakout (analog left/digital right)**



56.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Infrared/QRE1113%20Line%20Sensor%20Breakout%20-%20Analog.pdf (Analog)* *http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Proximity/QRE1113-Digital-Breakout-v11.pdf* (Digital)
Datasheet	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Proximity/QRE1113.pdf* (Analog) *https://www.sparkfun.com/datasheets/Robotics/QR_QRE1113.GR.pdf* (Digital)

Breakout pin	Function
VCC	Power, connect to 5 V
OUT	Analog: Signal, connect to A0 Digital: Signal, connect to D2
GND	Ground, connect to GND

56.3 Companion library

None.

56.4 Compile and upload

The method of reading the values from the analog and digital version is different; therefore, the guide provided two examples.

The analog version displays integer values. Any value over 1000 means nothing was reflected (Black). The lower the value means more reflection (White). Many factors play into the return values (lighting, line width, line darkness, surface, distance, etc.); therefore, it is based on the usage scenario. In this test case, a white paper with black electrical tape that was ¾in width.

Example 70 **Code for the QRE1113 analog board**

```
//Code for the QRE1113 Analog board
//Outputs via the serial terminal - Lower numbers mean more reflected
int QRE1113_Pin = 0; //connected to analog 0
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  int QRE_Value = analogRead(QRE1113_Pin);
  Serial.println(QRE_Value);
}
```

The digital version displays integer values. If a value greater than 3000, means that nothing was reflected. The lower the value means more reflection. Many factors play into the return values (lighting, line width, line darkness, surface distance etc.); therefore, it is based on the usage scenario.

Example 71 **Code for the QRE1113 digital board**

```
//Code for the QRE1113 Digital board
//Outputs via the serial terminal - Lower numbers mean more reflected
//3000 or more means nothing was reflected.
int QRE1113_Pin = 2; //connected to digital 2
void setup()
{
  Serial.begin(9600);
}
void loop()
{

  int QRE_Value = readQD();
  Serial.println(QRE_Value);

}
int readQD()
{
  //Returns value from the QRE1113
  //Lower numbers mean more reflective
  //More than 3000 means nothing was reflected.
  pinMode( QRE1113_Pin, OUTPUT );
  digitalWrite( QRE1113_Pin, HIGH );
  delayMicroseconds(10);
  pinMode( QRE1113_Pin, INPUT );

  long time = micros();

  //time how long the input is HIGH, but quit after 3ms as nothing happens after that
  while (digitalRead(QRE1113_Pin) == HIGH && micros() - time < 3000);
  int diff = micros() - time;

  return diff;
}
```

56.5 Results

Arduino* 101 compatible

§

57 Parallax* Ping* Ultrasonic Distance Sensor

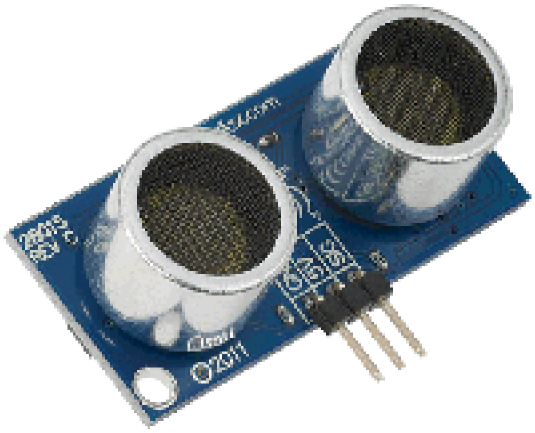
57.1 Use case

The PING™ ultrasonic sensor provides an easy method of distance measurement. This sensor is perfect for any number of applications that require measurements between moving or stationary objects.

A single I/O pin is used to trigger an ultrasonic burst (well above human hearing) and then "listen" for the echo return pulse. The sensor measures the time required for the echo return, and returns this value to the microcontroller as a variable-width pulse via the same I/O pin.

Key Info	Links
Order/product info	*http://www.parallax.com/product/28015*
Guide	*http://www.SeeedStudio.com/wiki/E-ink_Display_Shield*
Example code	*http://www.arduino.cc/en/Tutorial/Ping*
Library	None.

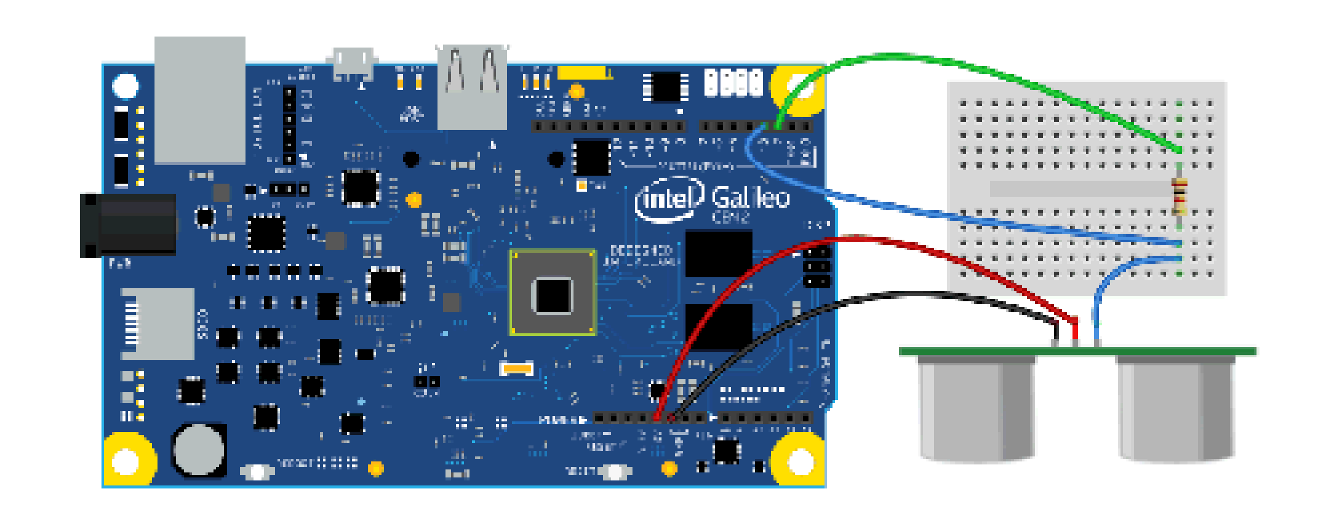
Figure 151 **Ping* ultrasonic distance sensor (28015, REV A)**



57.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/Danger_Shield-v17.pdf*
Datasheet	*http://www.jameco.com/Jameco/Products/ProdDS/282861.pdf*
Distance	0.8 inches to 10 feet

Figure 152 **Distance sensor. . Shown here with an Intel® Galileo second generation board**



Sensor Pin Name	Function and board connection
GND	Ground, connect to GND
5 V	Power, connect to 5 V
SIG	Signal, connect to board, then to D3 Signal, connect to 1 kohm resistor D3-onboard then to D2 (see above for visual configuration)

57.3 Companion library

No companion library required.

57.4 Compile and upload

Example 72 **Sample sketch for Intel® Galileo first generation, Intel® Galileo second generation, and Intel® Edison platforms**

```
/* Ping))) Sensor

   This sketch reads a PING))) ultrasonic rangefinder and returns the
   distance to the closest object in range. To do this, it sends a pulse
   to the sensor to initiate a reading, then listens for a pulse
   to return.  The length of the returning pulse is proportional to
   the distance of the object from the sensor.

   The circuit:
        * +V connection of the PING))) attached to +5V
        * GND connection of the PING))) attached to ground
        * SIG connection of the PING))) attached to digital pin 2

   http://www.arduino.cc/en/Tutorial/Ping

   created 3 Nov 2008
   by David A. Mellis
   modified 30 Aug 2011
   by Tom Igoe

   This example code is in the public domain.

 */
// this constant won't change.  It's the pin number
// of the sensor's output:
//#define GALILEO // comment this line out for Arduino101
#ifdef GALILEO
const int pingPin = 2; // Using FAST_IO
const int pongPin = 3; // Using FAST_IO
#else
const int pingPin = 2;
#endif

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
#ifdef GALILEO
  pinMode(pingPin, OUTPUT_FAST);
  pinMode(pongPin, INPUT_FAST);
#endif
}

void loop()
{
  // establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  long duration, inches, cm;

  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
#ifdef GALILEO
  // API's for Galileo Firmware 1.0.2
  //fastGpioDigitalWrite(GPIO_FAST_IO2, LOW);
  //delayMicroseconds(2);
  //fastGpioDigitalWrite(GPIO_FAST_IO2, HIGH);
  //delayMicroseconds(10);
  //fastGpioDigitalWrite(GPIO_FAST_IO2, LOW);

  // API's for Galileo Firmware 1.0.3
  fastDigitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  fastDigitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  fastDigitalWrite(pingPin, LOW);
#else
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);
#endif

  // The same pin is used to read the signal from the PING))): a HIGH
  // pulse whose duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
#ifdef GALILEO
  duration = pulseIn(pongPin, HIGH);
#else
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);
#endif

  // convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}

long microsecondsToInches(long microseconds)
{
  // According to Parallax's datasheet for the PING))), there are
  // 73.746 microseconds per inch (i.e. sound travels at 1130 feet per
  // second).  This gives the distance travelled by the ping, outbound
  // and return, so we divide by 2 to get the distance of the obstacle.
  // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

57.5 Results

Arduino* 101 compatible

§

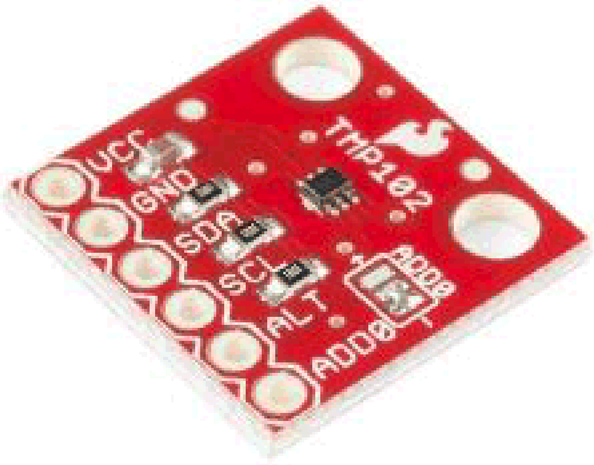
58 Sparkfun* Digital Temperature Breakout

58.1 Use case

This breakout board utilizes the small TMP102 digital temperature sensor. This sensor requires low current and has no onboard voltage regulator. Filtering capacitors and pull-up resistors are supplied on the board. Communication with the sensor uses the I^{2}C interface.

Key Info	Links
Order/product info	*https://www.sparkfun.com/products/11931*
Guide	*http://bildr.org/2011/01/tmp102-arduino/*
Library	None.

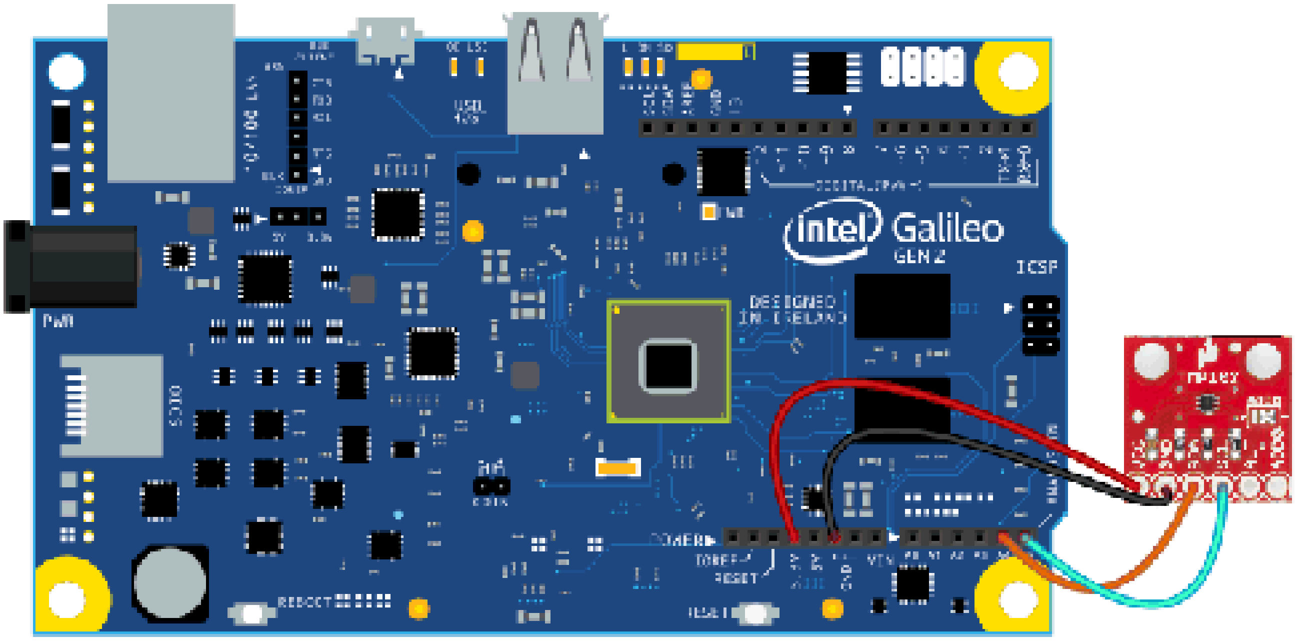
Figure 153 **Sparkfun* TMP102 digital temperature sensor**



58.2 Hardware summary

Key Info	Description/Links
Operating voltage	3.3 V
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*
Schematics	*http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/Digital%20Temperature%20Sensor%20Breakout%20-%20TMP102.pdf*
Datasheet	*https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf*

Figure 154 **TMP102 breakout on an Intel® Galileo second generation board**



Breakout pin	Function
VCC	Power, connect to 5 V
GND	Ground, connect to GND
SDA	Data Line, connect to SDA or A4. If solder is present on 'ADDR' int tmp102Address = 0x48;
SCL	Clock Line, connect to SCL or A5
ALT	Alert, not used
ADD0	Address is configurable when the solder is removed. Note: The I²C address is changeable if the solder is completely removed from the 'ADDR' solder pads. In this configuration: 'ADD0' to GND uses I²C Address 0x48 'ADD0' to 3.3 V uses I²C Address 0x49

58.3 Companion library

Not required.

58.4 Compile and upload

The tutorial guide references a previous version of the sensor; however, the same sketch works with this version. The guide has a slightly different wiring scheme where the newer version eliminates a single wire.

Example 73 **Guide example sketch**

```
//Arduino 1.0+ Only
//Arduino 1.0+ Only

//////////////////////////////////////////////////////////////////
//©2011 bildr
//Released under the MIT License - Please reuse change and share
//Simple code for the TMP102, simply prints temperature via serial
//////////////////////////////////////////////////////////////////

#include <Wire.h>
int tmp102Address = 0x48; // default

void setup(){
  Serial.begin(9600);
  Wire.begin();
} // setup

void loop(){

  float celsius = getTemperature();
  Serial.print("Celsius: ");
  Serial.println(celsius);

  float fahrenheit = (1.8 * celsius) + 32;
  Serial.print("Fahrenheit: ");
  Serial.println(fahrenheit);

  delay(200); //just here to slow down the output. You can remove this
} // loop

float getTemperature(){
  int   nBytes  = Wire.requestFrom(tmp102Address, 2);
  float celsius = 0.0;
  if(nBytes == 2) // Was request honered?
  {
    byte MSB = Wire.read();
    byte LSB = Wire.read();
    //it's a 12bit int, using two's compliment for negative
    int TemperatureSum = ((MSB << 8) | LSB) >> 4;
    celsius = TemperatureSum * 0.0625;
  }
  else
  {
    Serial.println("No temp data received");
  }
  return celsius;
} // getTemperature
```

58.5 Results

Arduino* 101 compatible

§

59 XBee* Wi-Fi Module w/Arduino* Wireless Proto Shield

59.1 Use case

The XBee Wi-Fi module with wire antenna provides simple serial to IEEE 802.11 connectivity. By bridging the low-power/low-cost requirements of wireless device networking with the proven infrastructure of 802.11, the XBee Wi-Fi creates new wireless opportunities for energy management, process and factory automation, wireless sensor networks, intelligent asset management and more. The module gives developers IP-to-device and device-to-cloud capability. In a nutshell, this Wi-Fi module is serial over Wi-Fi.

Key Info	Links
Order/product info	*https://www.sparkfun.com/products/12571*
Guide	*https://learn.sparkfun.com/tutorials/xbee-wifi-hookup-guide*
Library	None.
Router used	Linksys, WRT54G

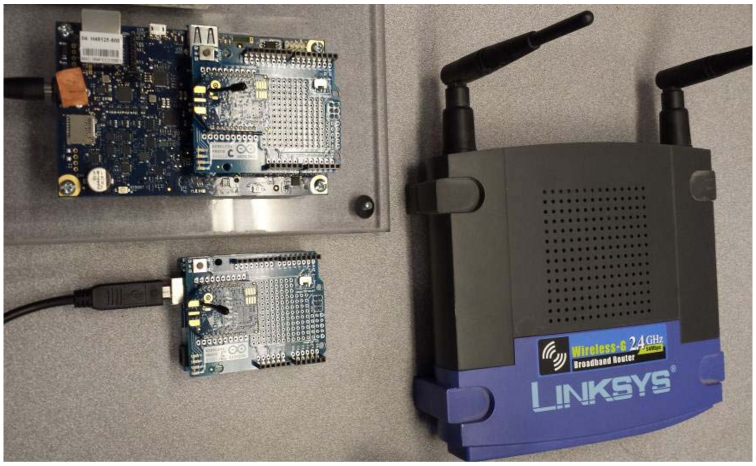
The Arduino* XBee protoshield where the XBee modules are mounted are not compatible with Arduino* 101 and the issue is being investigated. Symptoms are power cycling of the shield or not able to see the radios mounted.

Figure 155 **XBee* Wi-Fi module**

59.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Use VIN as power source	No
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	Arduino* 1.6.8 *https://www.arduino.cc/en/Main/Software*

Figure 156 **XBee* Wi-Fi module shield, shown with an Arduino* Uno board and an Intel® Galileo second generation board**



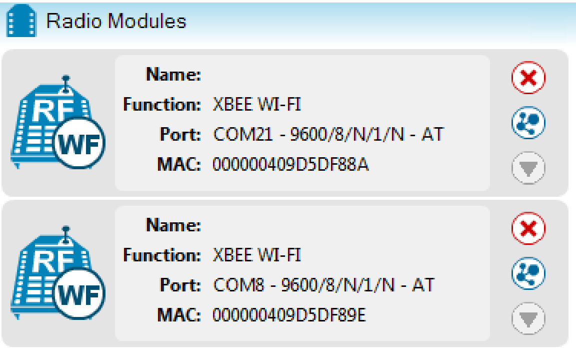
1. Attach module to shield and attach shield to Arduino* Uno board. This module cannot be queried or configured on an Intel® Galileo/Intel® Edison. Use the Arduino Uno board to configure the module to the router.
2. Set protoshield switch to "USB". If not set to 'USB', the configuration software will not detect it and a sketch cannot be downloaded.
3. Download an empty sketch.

Sketch: EmptySketch

```
void setup() { }
void loop() { }
```

4. Bring up the XCTU application and "Discover radio modules connected to your machine". If setup properly, a detected module will be displayed as follows as shown below. In this case, two modules are being configured on two Arduino* Uno boards.

Figure 157 **Radio Modules dialog from Windows**


Radio Modules

Name:
Function: XBEE WI-FI
Port: COM21 - 9600/8/N/1/N - AT
MAC: 000000409D5DF88A

Name:
Function: XBEE WI-FI
Port: COM8 - 9600/8/N/1/N - AT
MAC: 000000409D5DF89E

5. Select “Add selected devices” to add module to the left screen.
6. If not already completed, set the following:
   a. Select “Scan for Access points in the vicinity”
   b. Select SSID of the router. Enter password if necessary.
   c. Select “Connect”, enable ‘Save the SSID configuration in the module’
   d. Once the module is connected, it will be supplied an IP address. Note this address. You will be using this address as the destination address on the other module
   e. Once connected, select the device which will display the Radio Configuration.

Table 8 shows the key network/serial information.

- The ‘MY module IP address’ is the IP address assigned by the router. Both modules are given a unique IP address.
- The ‘DL destination IP address’ is the IP address the module will communicate to. The two modules are cross referenced. This information needs to be edited.
- When the serial portion of the sketch is set up, the baud rate needs to be set. The appropriate baud rate is set shown here.

Table 8 **Key network/serial information**

Configuration settings	Module 1	Module 2
Addressing / NS DNS address	192.168.1.1	192.168.1.1
Addressing / DL destination IP address	192.168.1.**102**	192.168.1.**100**
Addressing / GW IP address of gateway	192.168.1.1	192.168.1.1
Addressing / MK IP address Mask	255.255.255.0	255.255.255.0
Addressing / MY module IP address	192.168.1.**100**	192.168.1.**102**
Serial interfacing / BD baud rate	9600 [3]	9600 [3]
Serial interfacing / NB parity	No parity [0]	No parity[0]
Serial interfacing / SP stop bit	One stop bit [0]	One stop bit [0]
Device Option / DO	4	4
Infrastructure Mode CE	STA	STA
IP Protocol	UDP	UDP
Device Option / DO	4	4
Infrastructure Mode CE	STA	STA
IP Protocol	UDP	UDP

7. The Device Option should be set to 4. The other options are for cloud connectivity. Also set the Infrastructure mode and the IP Protocol if they are set differently. Save the module settings.
8. You can now attach the shield to 2 Arduino 101 boards.

59.3 Companion library

No library is used. The module will communicate through the serial port.

59.4 Compile and upload

The modules are setup as a sender and receiver. One module will send data and the other will receive data. Download the sketch when the switch is set to ‘USB’. Once downloading is complete, set the switch to ‘MICRO’. No data is transmitted through the module until the ‘MICRO’ switch is set.

Example 74 **Sender**

```
// Sender
// A R D U I N O   1 0 1
HardwareSerial* gSerialStdPtr = &Serial;
HardwareSerial* gSerialOnePtr = &Serial1;

char* gPromptPtr = "TX> ";
int   gCtr       = 0;
int   gTXcnt     = 0;

void setup()
{
  gSerialStdPtr->begin(9600);
  gSerialOnePtr->begin(9600);
  waitForUser(5);
  gSerialStdPtr->println("Setup-Done");
  gCtr   = 0;
  gTXcnt = 0;
} // setup

void loop()
{
   if((gCtr % 10) == 0)
   {
     gTXcnt++;
     gSerialOnePtr->print(gPromptPtr); // Write to BT & IDE
     gSerialOnePtr->println(gTXcnt);
   }

   if(gCtr >= 99)
     gCtr = 0;
   else
     gCtr = gCtr + 1;
   gSerialStdPtr->print(gPromptPtr);
   gSerialStdPtr->print(gCtr);
   gSerialStdPtr->print(",");
   gSerialStdPtr->println(gTXcnt);
   delay(1000*2);
} // loop
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
} // waitForUser
```

The sender is sending at a slower rate to prevent flooding of data to the receiver. The receiver is processing data at a faster rate.

Example 75 **Receiver**

```
// Receiver
// A R D U I N O   1 0 1
HardwareSerial* gSerialStdPtr = &Serial;
HardwareSerial* gSerialOnePtr = &Serial1;


char* gPromptPtr = "RX> ";
int   gCtr = 0;
int   gBTcnt  = 0;
void setup()
{
  gSerialStdPtr->begin(9600);
  gSerialOnePtr->begin(9600);
  waitForUser(5);
  gSerialStdPtr->println("Setup-Done");
  gBTcnt = 0;
  gCtr   = 0;
} // setup

void loop()
```

```
{
   char rcvData;
   if(gSerialOnePtr->available()>0)   // BT data?
   {
     rcvData = gSerialOnePtr->read(); // Read data
     if(rcvData == '\n')
       gSerialStdPtr->print("<NL>");
     else if(rcvData == '\r')
       gSerialStdPtr->print("<CR>");
     else
     {
       gSerialStdPtr->print(rcvData); // Write to IDE
       gSerialStdPtr->print("   ");
     }
     gBTcnt++;
   }
   else
   {
     gSerialStdPtr->print("    ");
   } // if

   gCtr = gCtr + 1;
   gSerialStdPtr->print(gPromptPtr);
   gSerialStdPtr->print(gCtr);
   gSerialStdPtr->print(",");
   gSerialStdPtr->println(gBTcnt);
   delay(500);
   if(gCtr >= 1000)
   {
     gCtr    = 0;
     gBTcnt  = 0;
   } // if
} // loop
void waitForUser(unsigned int aSec)
{
  // Give user time to bring up the serial port
  for(int i=aSec; i>0; i--){delay(1000*1);gSerialStdPtr->print(i);}
  gSerialStdPtr->println("");
} // waitForUser
```

There is an issue where communication lags very slowly. The loop count on the receiver hits 100 before the first set of data is received. After that, the data seems to be transmitting consistently.

Sometimes the module becomes overwhelmed if constantly downloading the sketch; therefore, the module needs to be powered off to resume functionality.

If the IP address expires there will be an issue if the DHCP router gives the address to another device. Using XCTU look at the program IP address. Then disconnect and ping from your PC those addresses while connected to the router. If they do not ping then you can re-use. Otherwise you need to rescan and connect to get new IP assignments.

59.5 Results

Intel® Curie™ module compatible.

§

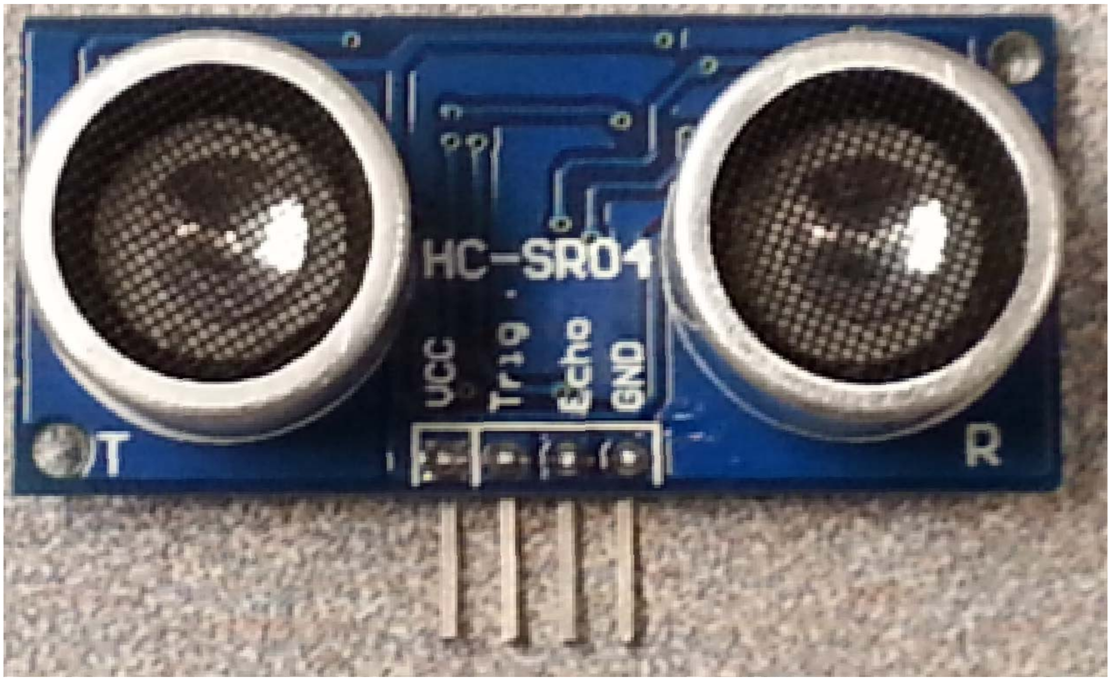
60 Ultrasonic Ranging Module HC-SR04

60.1 Use case

The HC-SR04 provides 2 to 400 cm noncontact measurement function with a ranging accuracy that can reach 3 mm. The module includes ultrasonic transmitters, receiver, and a control circuit. The module sends eight 40 kHz signals and tests whether there is a pulse signal back. The test distance is equal to the high level time times the velocity of sound divided by 2. There are four pins and the module uses two digital pins for trigger and echo.

Key Info	Links
Order/product info	*http://www.sainsmart.com/ultrasonic-ranging-detector-mod-hc-sr04-distance-sensor.html*
Guide	*http://www.micropik.com/PDF/HCSR04.pdf*
Example code	*http://fritzing.org/media/fritzing-repo/projects/h/hc-sr04-project/code/hc_sr04.ino*
Library	None.

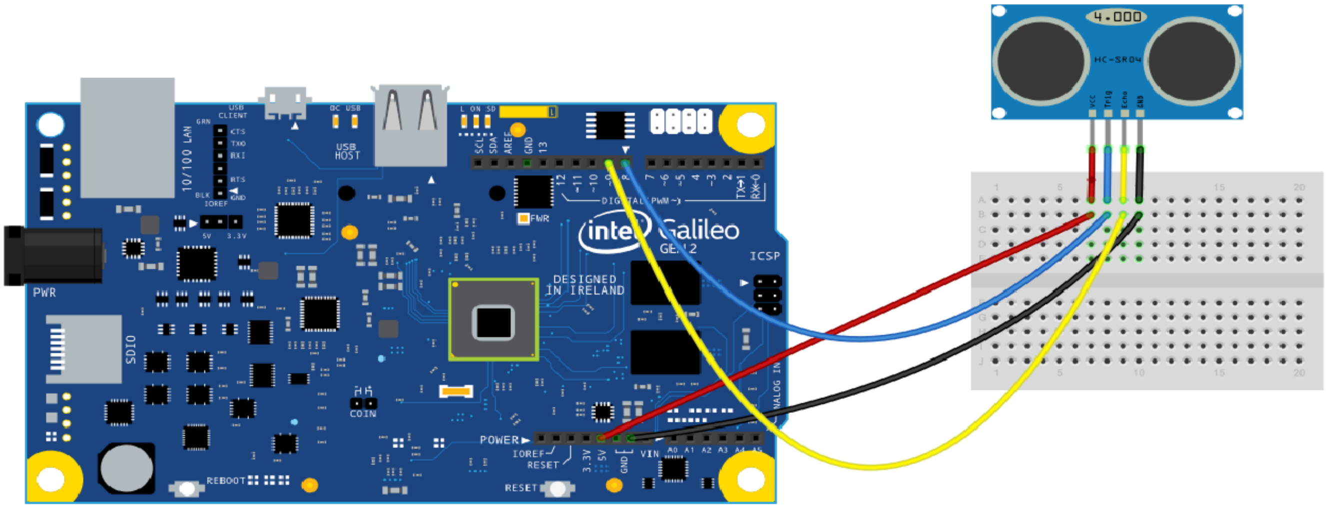
Figure 158 **Ultrasonic Ranging Module HC-SR04**



60.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Working current	15 mA
Use VIN as power source	No
IOREF	3.3 or 5 V (Intel® Galileo first generation boards don't work with either. See *Section 0.8 IOREF voltage*.)
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
IDE	1.6.8
Distance	2 cm to 4 meters

Figure 159 **Intel® Galileo second generation board connection for distance sensor**



Sensor Pin Name	Function and board connection
GND	Ground, connect to GND
5 V	Power, connect to 5 V
Trig	Trigger pulse – input – connected to D8
Echo	Echo pulse – output – connected to D9

60.3 Companion library

No companion library required.

60.4 Compile and upload

Example 76 is unchanged from the example code link above. Upload the sketch and open a serial terminal. Put an object in front of the sensor and move it to different distances the distance should be displayed in the terminal.

Example 76 **Distance sensor sample sketch from fritzing.org**

```
/*
HC-SR04 for Arduino

Original project from http://www.swanrobotics.com

This project demonstrates the HC-SR
The distance presented in the code is in mm, but you can uncomment the line for distance in
inches.
The schematics for this project can be found on http://www.swanrobotics.com

This example code is in the public domain.
*/

const int TriggerPin = 8;      //Trig pin
const int EchoPin = 9;         //Echo pin
long Duration = 0;

void setup(){
  pinMode(TriggerPin,OUTPUT);  // Trigger is an output pin
  pinMode(EchoPin,INPUT);      // Echo is an input pin
  Serial.begin(9600);          // Serial Output
}

void loop(){
  digitalWrite(TriggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(TriggerPin, HIGH);          // Trigger pin to HIGH
  delayMicroseconds(10);                   // 10us high
  digitalWrite(TriggerPin, LOW);           // Trigger pin to HIGH

  Duration = pulseIn(EchoPin,HIGH);        // Waits for the echo pin to get high
                                           // returns the Duration in microseconds
  long Distance_mm = Distance(Duration);   // Use function to calculate the distance

  Serial.print("Distance = ");             // Output to serial
  Serial.print(Distance_mm);
  Serial.println(" mm");

  delay(1000);                             // Wait to do next measurement
}

long Distance(long time)
{
    // Calculates the Distance in mm
    // ((time)*(Speed of sound))/ toward and backward of object) * 10

    long DistanceCalc;                      // Calculation variable
    DistanceCalc = ((time /2.9) / 2);     // Actual calculation in mm
    //DistanceCalc = time / 74 / 2;         // Actual calculation in inches
    return DistanceCalc;                    // return calculated value
}
```

60.5 Results

Arduino* 101 Compatible

§

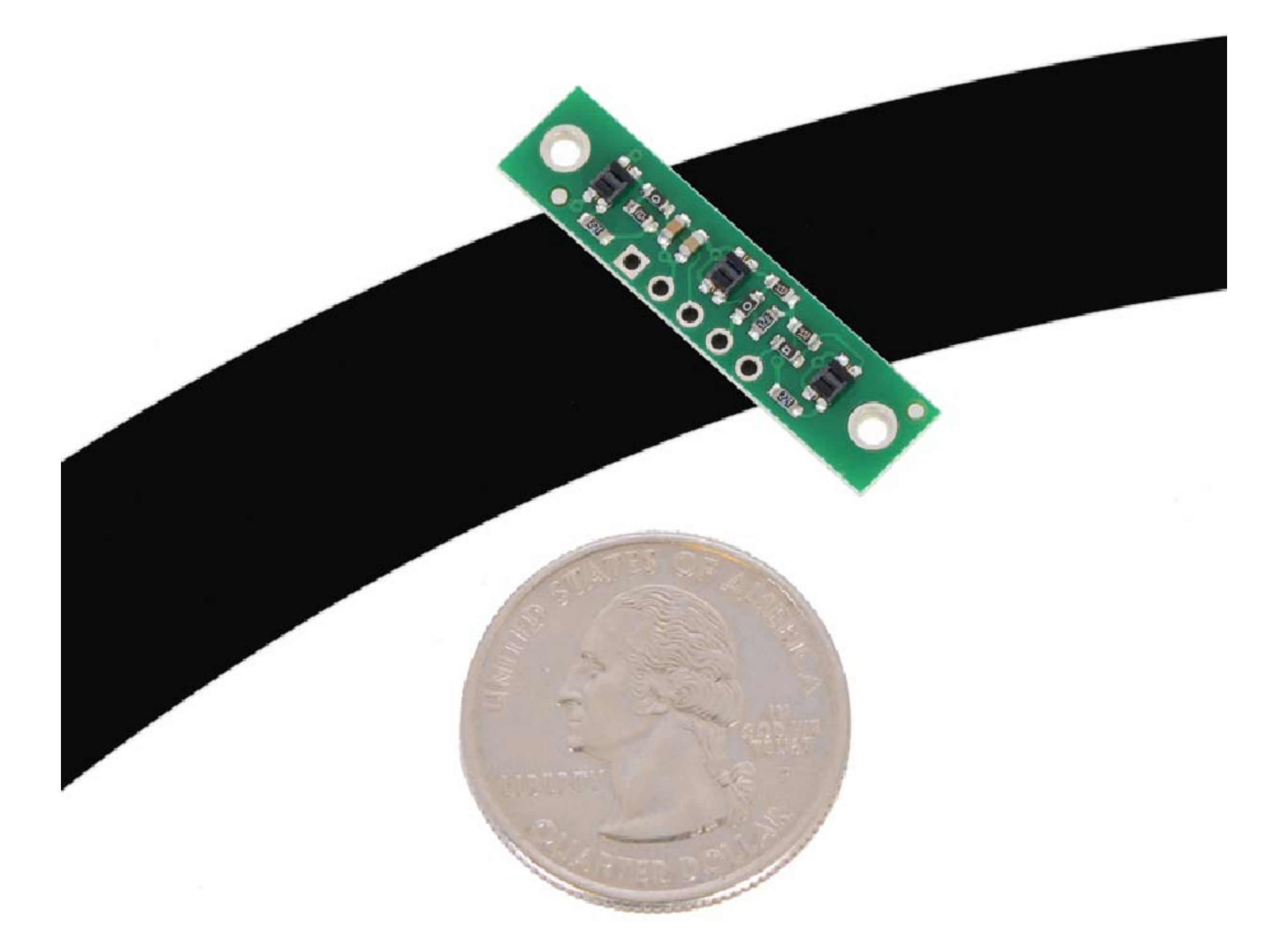
61 Pololu* QTR-3A Reflective Sensor Array

61.1 Use case

The compact module packs three IR LED/phototransistor pairs onto a 1.25 × 0.3 in. board. The sensors are mounted on a 0.375 pitch, making this array a great minimal sensing solution for a line following robot. Each sensor provides a separate analog voltage output. The QTR-3A reflective sensor array is intended as a line sensor, but it can be used as a general purpose proximity or reflective sensor.

Key Info	Links
Order/product info	*http://www.pololu.com/product/2456*
Guide	*http://www.pololu.com/docs/0J19*
Library	*https://github.com/pololu/qtr-sensors-arduino* Date: December 3, 2015

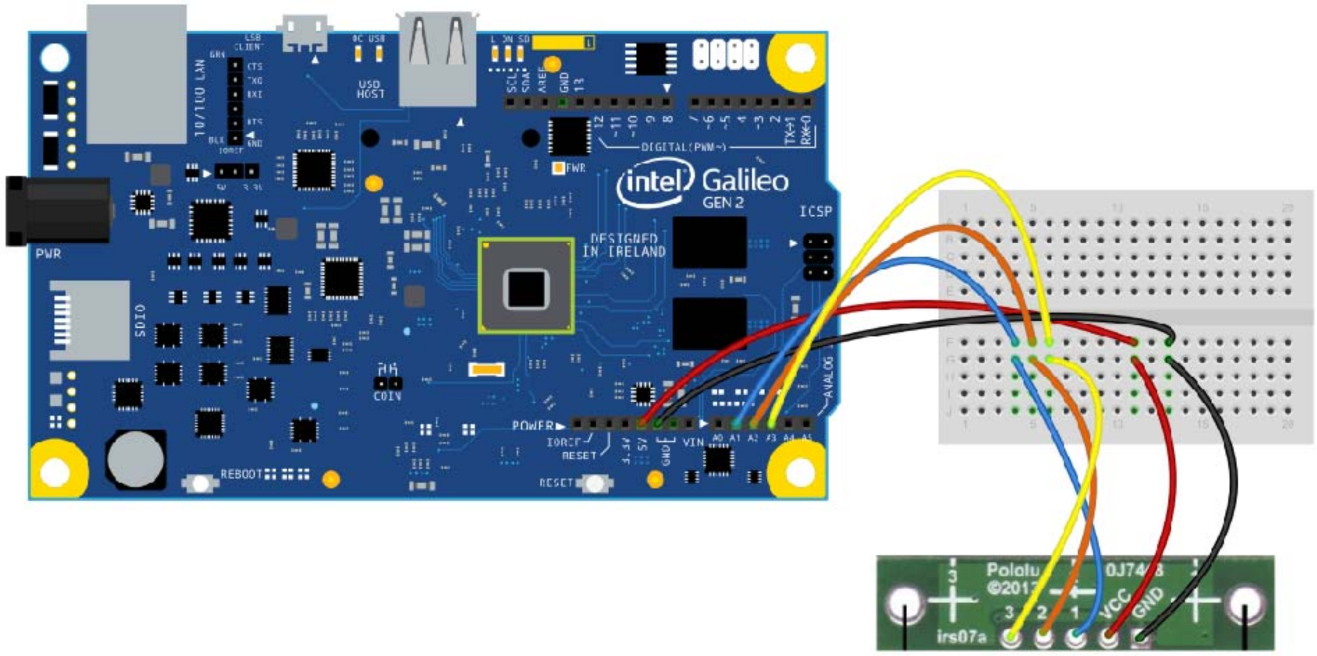
Figure 160 **QTR-3A reflective sensor on a ¾" line**



61.2 Hardware summary

Key Info	Description/Links
Operating voltage	5 V
Working current	50 mA
VIN as power source	No
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
IDE	1.6.8
Optimal sensing distance	0.125 in. (3 mm)
Maximum recommended sensing distance	0.25 in. (6 mm)

Figure 161 **Intel® Galileo second generation board connections for reflective sensor**



Sensor Pin Name	Function and Board Connection
GND	Ground, connect to GND
VCC	Power, connect to 5 V
1	Analog output for sensor 1, connect to any available analog pin
2	Analog output for sensor 2, connect to any available analog pin
3	Analog output for sensor 3, connect to any available analog pin

61.3 Companion library

The library installed is called *QTRSensors*. This library handles analog and digital sensors. This part is analog. The library handles returning raw data as well as data that is used with its own calibration function.

61.4 Compile and upload

Use the wiring diagram above and upload this sketch. Open a serial terminal and move sensor across a 3/4 in. black line on white paper. The higher value sensors should be the ones above the black line.

```
Reflective sensor sample sketch using raw sensor values
#include <QTRSensors.h>

// The main loop of the example reads the raw sensor values (uncalibrated).
// You can test this by taping a piece of 3/4" black electrical tape to a piece of white
// paper and sliding the sensor across it.  It prints the sensor values to the serial
// monitor as numbers from 0 (maximum reflectance) to 1023 (minimum reflectance).


#define NUM_SENSORS             3  // number of sensors used
#define NUM_SAMPLES_PER_SENSOR  4  // average 4 analog samples per sensor reading
#define EMITTER_PIN             QTR_NO_EMITTER_PIN  // emitter is controlled by digital pin 2

// sensors 1 through 3 are connected to analog inputs 1 through 3, respectively
QTRSensorsAnalog qtra((unsigned char[]) {1, 2, 3},
  NUM_SENSORS, NUM_SAMPLES_PER_SENSOR, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];


void setup()
{
  delay(500);
  Serial.begin(9600); // set the data rate in bits per second for serial data transmission
  delay(1000);
}

void loop()
{
  // read raw sensor values
  qtra.read(sensorValues);

  // print the sensor values as numbers from 0 to 1023, where 0 means maximum reflectance and
  // 1023 means minimum reflectance
  for (unsigned char i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(sensorValues[i]);
    Serial.print('\t'); // tab to format the raw data into columns in the Serial monitor
  }
  Serial.println();

  delay(250);
}
```

61.5 Results

Arduino* 101 compatible

§

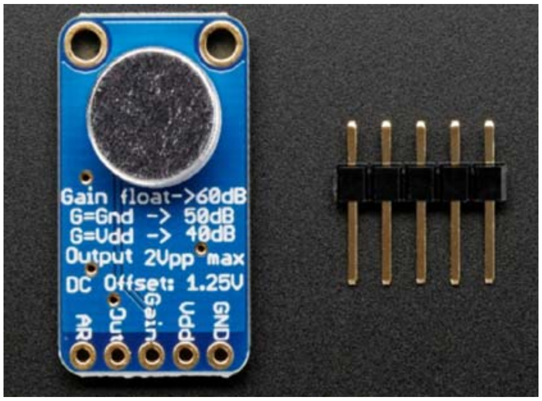
62 Adafruit* MAX9814 AGC Electret Microphone Amplifier

62.1 Use case

This microphone amplifier module has built-in AGC (automatic gain control). The AGC in the amplifier causes nearby “loud” sounds to be quieted so they don’t clip the amplifier and even quiet far away sounds will be amplified. This amplifier is ideal for recording in an audio setting where levels change and gain doesn’t need to be tweaked all the time. The output from the amp is about 2 Vp-p on a 1.25 VDC bias. The output can be piped into a line input using a 1 μF blocking capacitor in series.

Key Info	Links
Order/product info	*https://www.adafruit.com/products/1713*
Order/product info/guide	*https://learn.adafruit.com/adafruit-agc-electret-microphone-amplifier-max9814/overview*
Tone generator	*http://www.ringbell.co.uk/software/audio.htm* *http://onlinetonegenerator.com/*

Figure 162 **Adafruit* MAX9814 AGC electret microphone amplifier**



62.2 Hardware summary

Key Info	Description/Links
Operating voltage	2.7 to 5.5 V
IOREF	3.3 or 5 V (See *Section 0.8 IOREF voltage*.)
Working current	3 mA
VIN as power source	No
Output	2 Vpp on 1.25 V bias
IDE	1.6.8
Schematics	*https://learn.adafruit.com/assets/14290*
Datasheet	*http://www.adafruit.com/datasheets/MAX9814.pdf*

The electrolytic capacitor can be anywhere from 1 to 100 μF with positive lead on the side of the microphone’s out pin.

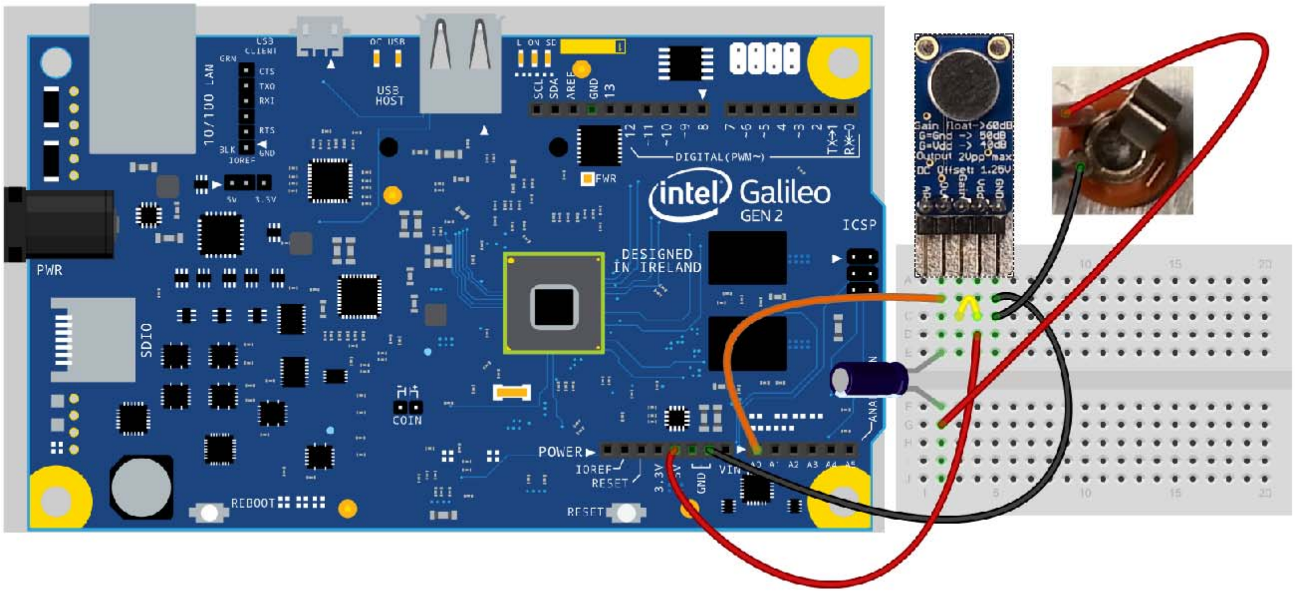
Breakout Pin Name	Connection and Function
GND	Ground, connect to GND
VDD	Power, connect to 3.3 or 5 V
Gain	Gain, connect to 3.3 or 5 V
Out	Output, connect to 1 through 100 microfarad capacitor – if capacitor is polarized, connect OUT to positive side.
AR	Attack/response ratio. Can connect to 5 V or GND, but default no wire works for most purposes.

62.3 Companion library

No library.

62.4 Compile and upload

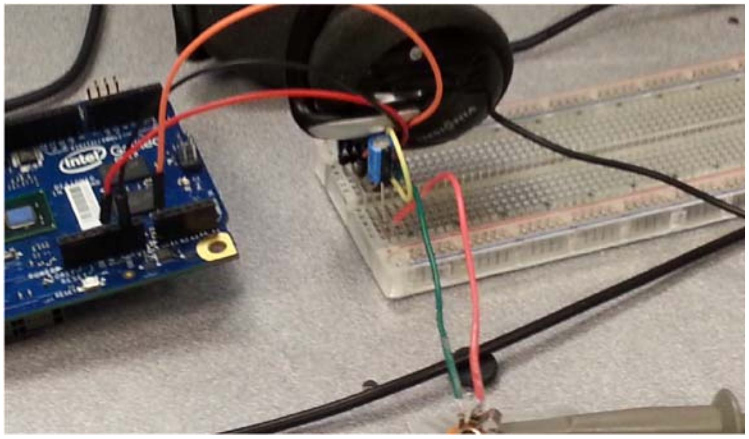
Figure 163 **Connecting the microphone. . Shown here with an Intel® Galileo second generation board**



Do the following:

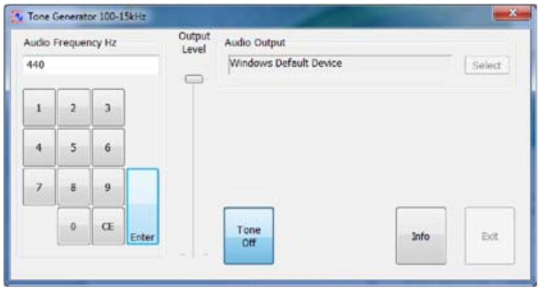
1. Connect the microphone to the Arduino* 101 using a breadboard as in the diagram above. Use an electrolytic capacitor from 1 to 100 µF. Connect the positive lead (longer lead) to the OUT pin.
2. Connect a 1/8 inch audio jack. Connect the sleeve to ground. Connect the tip to the negative side of the capacitor (side of capacitor with stripe). If the audio jack is a stereo jack then connect the left and right pins together with the negative side of the capacitor.
3. Connect a head phone to the audio jack.
4. Power up the board, connect USB and upload the sketch below.
5. Open a serial terminal from the IDE.
6. Test the sound quality by having somebody speak into the microphone or by using some other technique.
7. The level of audio is measured and displayed on the serial console. Notice that the level changes according to voice or other tones input to microphone.
8. Next, analyze the signal with an oscilloscope by connecting an oscilloscope probe to the OUT pin from microphone or the A0 pin. Connect a headphone to the PC audio output jack. Position the headphone so that as much output as possible is received by the microphone.

Figure 164 **Connecting for oscilloscope measurement. . Shown here with an Intel® Galileo first generation board**



9. Set the PC volume so it is not muted.
10. Start the Tone Generator program and set the frequency as 440 Hz (or some other audible frequency). Click the *Enter* button to start the tone generation. A 440 Hz tone is generated through the headphone and picked up by the microphone.

Figure 165 **Tone generator window**



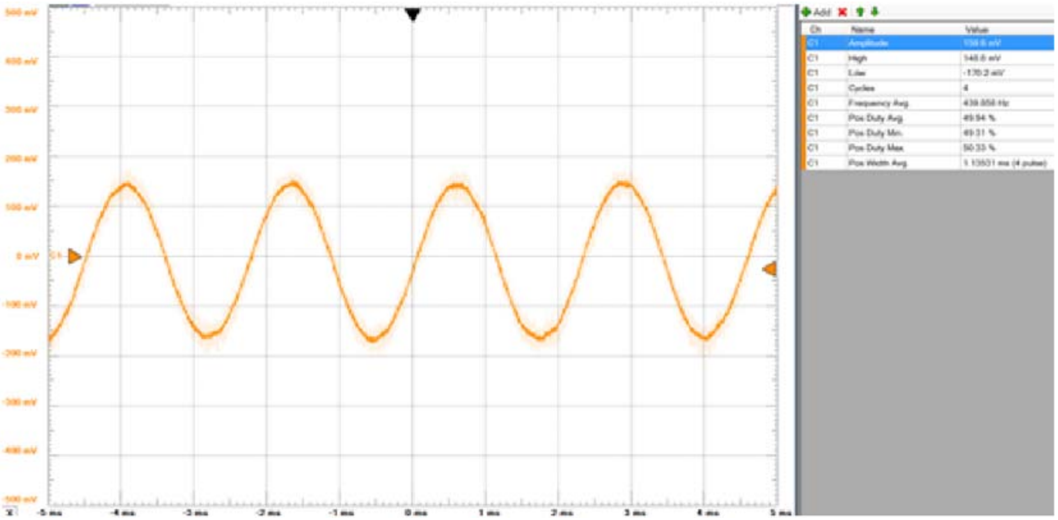
Tone Generator 100-15kHz

Audio Frequency Hz: 440

Audio Output: Windows Default Device

11. Set the oscilloscope to capture the 440 Hz tone. The oscilloscope voltages should approximate the values found in the serial terminal.

Figure 166 **Oscilloscope output captured by microphone from a headphone playing a 440 Hz tone**



Ch	Name	Value
C1	Amplitude	158.6 mV
C1	High	148.6 mV
C1	Low	-170.2 mV
C1	Cycles	4
C1	Frequency Avg	438.858 Hz
C1	Pos Duty Avg	49.94 %
C1	Pos Duty Min	49.31 %
C1	Pos Duty Max	50.33 %
C1	Pos Width Avg	1.13531 ms (4 pulses)

Example 77 **Sound level measurement sketch**

```
/****************************************
Example Sound Level Sketch for the
Adafruit Microphone Amplifier
****************************************/
const int sampleWindow = 50; // Sample window width in mS (50 mS = 20Hz)
unsigned int sample;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   unsigned long startMillis= millis();  // Start of sample window
   unsigned int peakToPeak = 0;   // peak-to-peak level
   unsigned int signalMax = 0;
   unsigned int signalMin = 1024;

   // collect data for 50 mS
   while (millis() - startMillis < sampleWindow)
   {
      sample = analogRead(0);
      if (sample < 1024)  // toss out spurious readings
      {
         if (sample > signalMax)
         {
            signalMax = sample;  // save just the max levels
         }
         else if (sample < signalMin)
         {
            signalMin = sample;  // save just the min levels
         }
      }
   }
   peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
   double volts = (peakToPeak * 3.3) / 1024;  // convert to volts

   Serial.println(volts);
}
```

62.5 Results

Arduino* 101 compatible

§

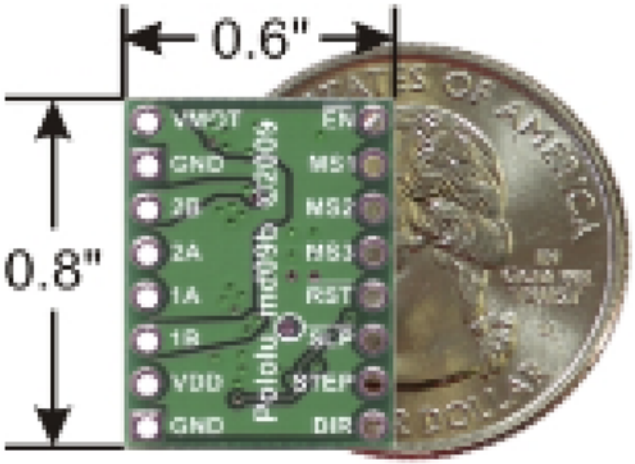
63 Pololu* A4983 Stepper Motor Driver Carrier

63.1 Use case

This stepper motor carrier is a breakout board for Allegro's A4983 microstepping bipolar stepper motor driver. The driver features adjustable current limiting and five different microstep resolutions. It operates from 8 to 35 V and can deliver up to two amps per coil. The board has overtemperature thermal shutdown, under voltage lockout and crossover current protection. Four, six and eight wire stepper motors can be driven. The *FabScan* 3D Scanner* has 4 separate pinouts for this carrier to run stepper motors.

Key Info	Links
Order/product info	*http://www.pololu.com/product/1182* *http://www.pololu.com/product/2128*
FAQs	*http://www.pololu.com/product/1201/faqs*
Library	No library.

Figure 167 **Pololu* A4983 stepper motor driver carrier**

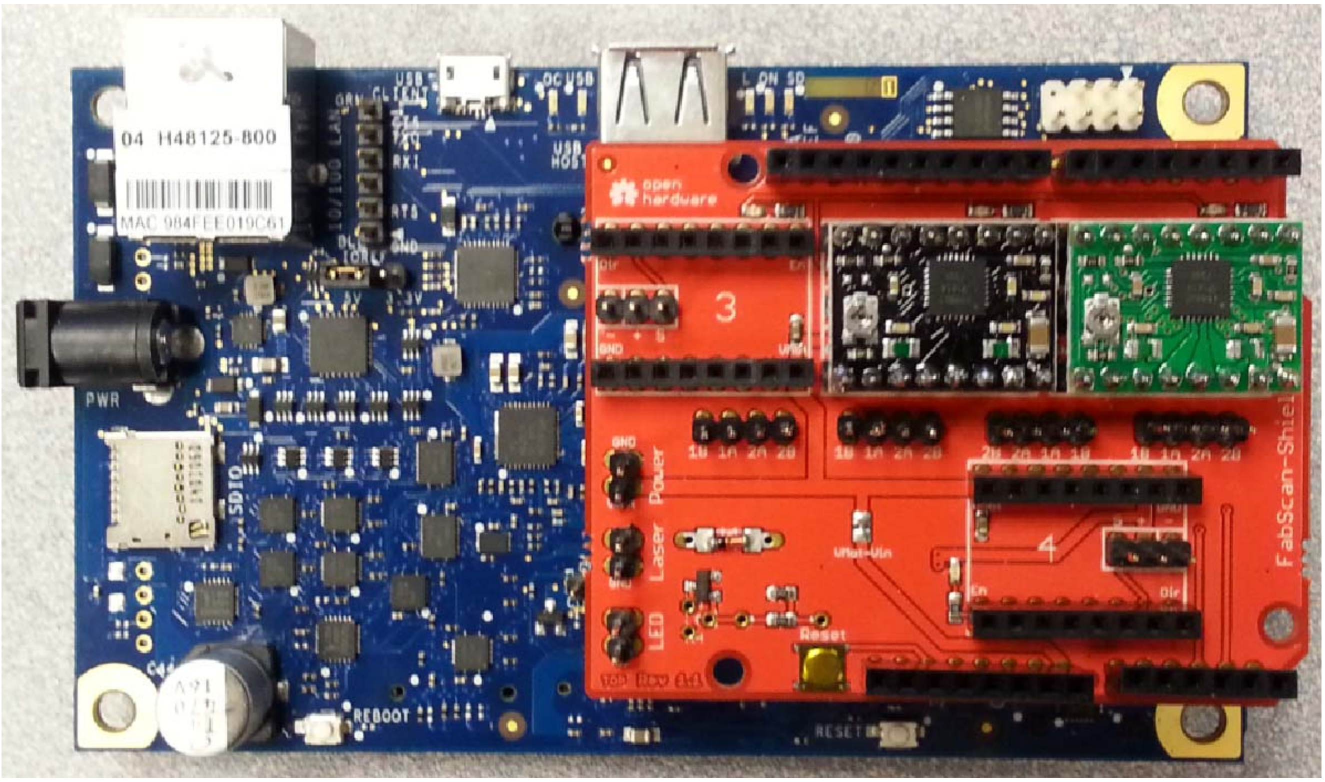


63.2 Hardware summary

Key Info	Description/Links
Operating voltage	Logic supply 3 to 5.5 V, Motor 8 to 35 V
Working current	2A per coil
VIN as power source	No
Datasheet	*http://www.pololu.com/file/download/a4988_DMOS_microstepping_driver_with_translator.pdf?file_id=0J450*
Motor	ST1

Sensor Pin Name	Intel® Galileo Connection and Function
GND	Ground, connect to GND
VCC	Power, connect to 5 V
EN	Enable for each stepper motor port for motors 0 – 3 connected to pins 2, 5, 11 and 14 respectively. Each carrier attached uses one of these depending on the port connected.
STEP	Stepper pin for each stepper motor port for motors 0 – 3 connected to pins 3, 6, 12 and 15. Each carrier attached uses one of these depending on the port connected.
DIR	Direction pin for each stepper motor port for motors 0 – 3 connected to pins 3, 6, 12 and 15. Each carrier attached uses one of these depending on the port connected.

Figure 168 **Two stepper motor carriers on FabScan* driver. Shown here with an Intel® Galileo second generation board**

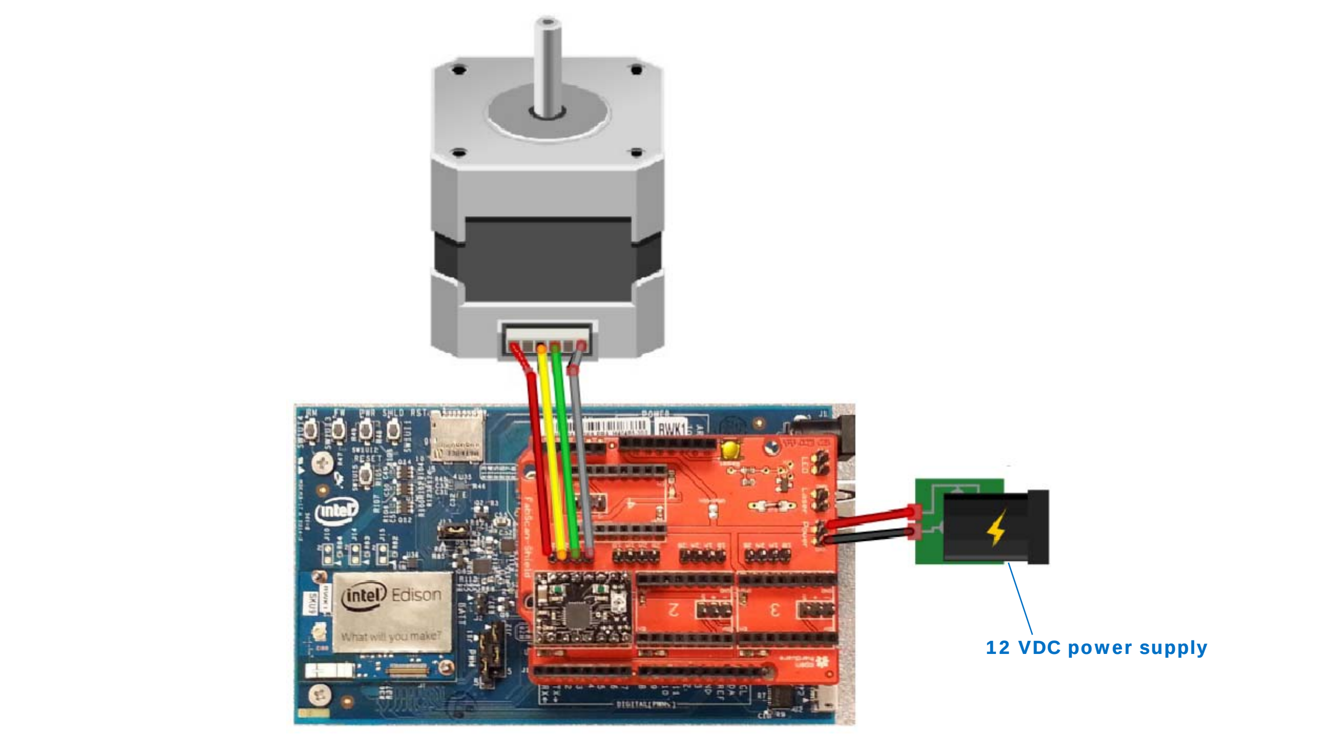


63.3 Companion library

No library.

63.4 Compile and upload

Figure 169 **Connecting the stepper carrier to the FabScan* shield. Shown here with an Intel® Edison board**



- Connect the *FabScan* 3D Scanner* shield to the board

Connect the stepper driver carrier breakout board to the FabScan stepper slot 1. The orientation of the pins can be determined by corresponding text on the corner pins.

Connect a unipolar stepper motor (see *Table 4*) or a similar stepper motor. You may connect up to four motors, but for the illustrations here only one motor is connected. The sketch is written to handle up to four motors.

Wire	Pin
Red wire	pin 2B
Yellow wire	pin 2A
Green wire	pin 1A
Gray wire	pin 1B

- The wire pairs for each coil should be together, but the polarity between the wires of each pair is not important. In this case the pairs are Red/Yellow and Green/Gray.
- For Arduino* 101 there is no need to connect the power supply for the board as the board will be powered up by the external source connect to the shield.
- If using a regulated power supply, turn the current and voltage controls to 0 V before powering on. Connect a 12 V power supply or battery to the two power pins. Black wire in this case is ground and the red wire is positive.
- If using a regulated power supply turn up the voltage and current alternating in small steps until 12 V is reached.
- Open the IDE and upload the sketch from below. The sketch is attempting to run 4 stepper motors at once so add motors as needed to the other stepper ports.
- Open a serial terminal from the IDE in order to see comments on what the sketch is attempting to perform.
- The motor should perform the following motions
- One revolution clockwise with no microstepping (fast)
- One revolution counter-clockwise with no microstepping (fast)
- One revolution clockwise with 16x microstepping (slow)

- One revolution counter-clockwise with 16x microstepping (slow)
- Repeat the above steps until exiting
- Check the motor occasionally for excessive heat.

Example 78 **Run stepper motors with driver carrier attached to fabscan shield**

```
#define STEPS_PER_REVOLUTION 200  // this is specific to the motor with microstepping disabled

#define MS_PIN    19  // pin used to turn microstepping on and off

//Stepper 1 as labeled on Shield, Turntable
#define ENABLE_PIN_0  2
#define STEP_PIN_0    3
#define DIR_PIN_0     4

//Stepper 2, Laser Stepper
#define ENABLE_PIN_1  5
#define STEP_PIN_1    6
#define DIR_PIN_1     7

//Stepper 3, currently unused
#define ENABLE_PIN_2  11
#define STEP_PIN_2    12
#define DIR_PIN_2     13

//Stepper 4, currently unused
#define ENABLE_PIN_3  14
#define STEP_PIN_3    15
#define DIR_PIN_3     16

void setup()
{
  Serial.begin(9600);
  delay(2000);

 pinMode(MS_PIN, OUTPUT);

  pinMode(ENABLE_PIN_0, OUTPUT);
  pinMode(DIR_PIN_0, OUTPUT);
  pinMode(STEP_PIN_0, OUTPUT);

  pinMode(ENABLE_PIN_1, OUTPUT);
  pinMode(DIR_PIN_1, OUTPUT);
  pinMode(STEP_PIN_1, OUTPUT);

  pinMode(ENABLE_PIN_2, OUTPUT);
  pinMode(DIR_PIN_2, OUTPUT);
  pinMode(STEP_PIN_2, OUTPUT);

  pinMode(ENABLE_PIN_3, OUTPUT);
  pinMode(DIR_PIN_3, OUTPUT);
  pinMode(STEP_PIN_3, OUTPUT);

  //enable HIGH to turn off, LOW to turn on
  digitalWrite(ENABLE_PIN_0, LOW);
  digitalWrite(ENABLE_PIN_1, LOW);
  digitalWrite(ENABLE_PIN_2, LOW);
  digitalWrite(ENABLE_PIN_3, LOW);

}

//loop just steps all four steppers. Attached motors should turn!
```

```
void loop()
{
  spin360(false, true);  // 360 revolution no microstepping, clockwise
  spin360(false, false); // no microstepping, counter-clockwise
  spin360(true, true);   // 16 microstepping, clockwise
  spin360(true, false);  // 16 microstepping, counter-clockwise
}

// function to take all steppers 360 degrees, options for using microstepping and direction
void spin360(boolean microstepping, boolean clockwise)
{
  int steps;

  if (microstepping) {
    Serial.print("Microstepping, ");
    digitalWrite(MS_PIN, HIGH);  // HIGH for 16 microstepping
    steps = STEPS_PER_REVOLUTION*16;
  }
  else {
    Serial.print("No Microstepping, ");
    digitalWrite(MS_PIN, LOW);  // LOW for no microstepping
    steps = STEPS_PER_REVOLUTION;
  }
  if (clockwise) {
    Serial.println("Clockwise");
    digitalWrite(DIR_PIN_0, LOW);
    digitalWrite(DIR_PIN_1, LOW);
    digitalWrite(DIR_PIN_2, LOW);
    digitalWrite(DIR_PIN_3, LOW);
  }
  else {
    Serial.println("Counter-Clockwise");
    digitalWrite(DIR_PIN_0, HIGH);
    digitalWrite(DIR_PIN_1, HIGH);
    digitalWrite(DIR_PIN_2, HIGH);
    digitalWrite(DIR_PIN_3, HIGH);
  }

  for (int step_count = 0; step_count < steps; step_count++) {
    digitalWrite(STEP_PIN_0, HIGH);
    digitalWrite(STEP_PIN_1, HIGH);
    digitalWrite(STEP_PIN_2, HIGH);
    digitalWrite(STEP_PIN_3, HIGH);
    delay(1);
    digitalWrite(STEP_PIN_0, LOW);
    digitalWrite(STEP_PIN_1, LOW);
    digitalWrite(STEP_PIN_2, LOW);
    digitalWrite(STEP_PIN_3, LOW);
  }
}
```

63.5 Results

Arduino* 101 Compatible

63.6 Next steps

§

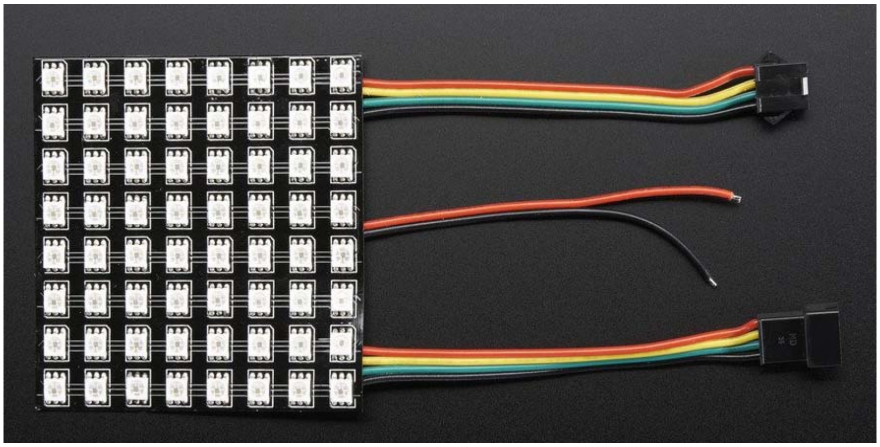
64 Adafruit* DotStar 8X8 Matrix

64.1 Use case

This is a test of using Adafruit* DotStar matrix on the Arduino* 101.

Key Info	Links
URL	*https://www.adafruit.com/prducts/2734*
Library	Install using library manager Adadafruit DotStar 1.0.1 - *https://github.com/adafruit/Adafruit_DotStar*
Guide	*https://learn.adafruit.com/adafruit-dotstar-leds/overview*

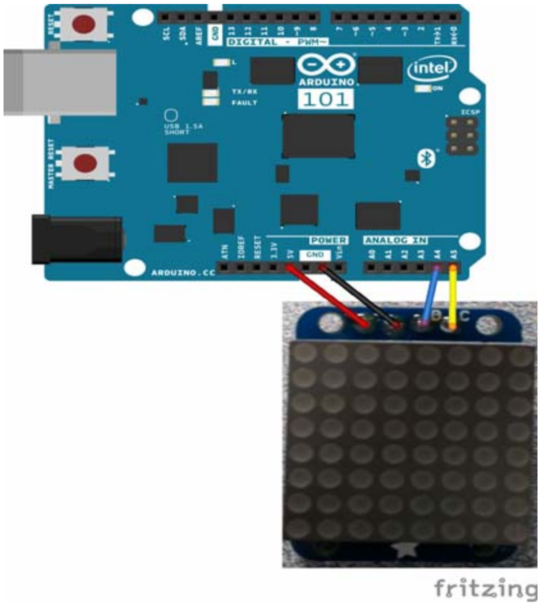
Figure 170 **Arduino* Wi-Fi shield**



64.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V
IOREF	Used, the shield operates at 5 V. (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	Yes
Power	5 V
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Datasheet	*https://cdn-shop.adafruit.com/product-files/2734/2734+data.pdf*

Adafruit recommends using a capacitor. Also an external power supply. However if you power the Arduino* 101 using an external power, you should be able to run the examples here with no worries. If you are going to run all 64 pixels full white bright you will draw 3.84 amps and you have to connect the matrix to a 5V power with 4 amps



64.3 Companion library

Adafruit DotStar

64.4 Compile and upload

```
#include <Adafruit_DotStar.h>
// Because conditional #includes don't work w/Arduino sketches...
#include <SPI.h>         // COMMENT OUT THIS LINE FOR GEMMA OR TRINKET
//#include <avr/power.h> // ENABLE THIS LINE FOR GEMMA OR TRINKET

#define NUMPIXELS 30 // Number of LEDs in strip

// Here's how to control the LEDs from any two pins:
#define DATAPIN    4
#define CLOCKPIN   5
Adafruit_DotStar strip = Adafruit_DotStar(
  NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR_BRG);
// The last parameter is optional -- this is the color data order of the
// DotStar strip, which has changed over time in different production runs.
// Your code just uses R,G,B colors, the library then reassigns as needed.
// Default is DOTSTAR_BRG, so change this if you have an earlier strip.

// Hardware SPI is a little faster, but must be wired to specific pins
// (Arduino Uno = pin 11 for data, 13 for clock, other boards are different).
//Adafruit_DotStar strip = Adafruit_DotStar(NUMPIXELS, DOTSTAR_BRG);

void setup() {

#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000L)
  clock_prescale_set(clock_div_1); // Enable 16 MHz on Trinket
#endif

  strip.begin(); // Initialize pins for output
  strip.show();  // Turn all LEDs off ASAP
}
```

```
// Runs 10 LEDs at a time along strip, cycling through red, green and blue.
// This requires about 200 mA for all the 'on' pixels + 1 mA per 'off' pixel.

int      head  = 0, tail = -10; // Index of first 'on' and 'off' pixels
uint32_t color = 0xFF0000;      // 'On' color (starts red)

void loop() {

  strip.setPixelColor(head, color); // 'On' pixel at head
  strip.setPixelColor(tail, 0);     // 'Off' pixel at tail
  strip.show();                     // Refresh strip
  delay(20);                        // Pause 20 milliseconds (~50 FPS)

  if(++head >= NUMPIXELS) {         // Increment head index.  Off end of strip?
    head = 0;                       //  Yes, reset head index to start
    if((color >>= 8) == 0)          //  Next color (R->G->B) ... past blue now?
      color = 0xFF0000;             //   Yes, reset to red
  }
  if(++tail >= NUMPIXELS) tail = 0; // Increment, reset tail index
}
```

64.5 Results

This shield works with no problems on the Arduino* 101board.

Arduino* 101 compatible

65 Adafruit* NeoPixel 60 LED Reel

65.1 Use case

This is a test of using Adafruit NeoPixel LED reel on the Arduino* 101 board.

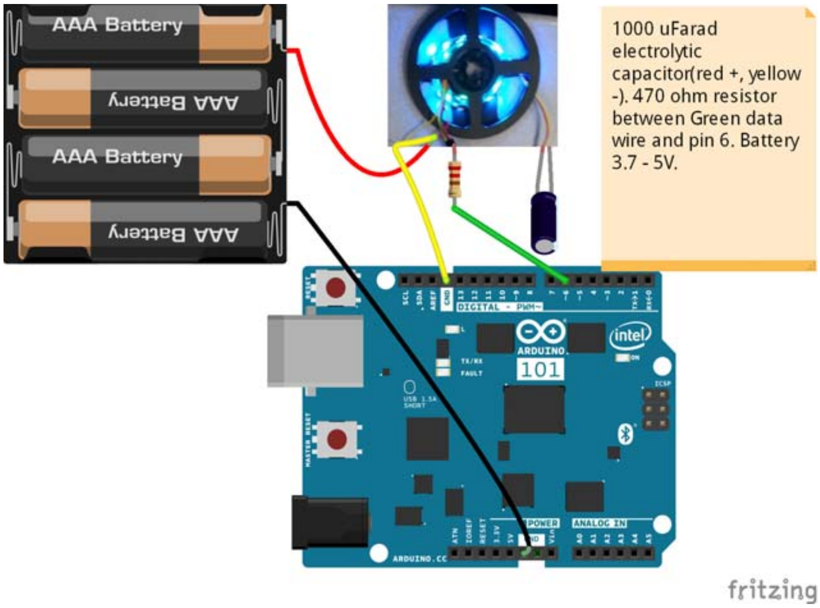
Key Info	Links
URL	*https://www.adafruit.com/products/2734*
Library	Use Library Manger to install. tested with Version 1.0.5 or download *https://github.com/adafruit/Adafruit_NeoPixel*
Guide	*https://learn.adafruit.com/adafruit-neopixel-uberguide/overview*

Figure 171 **Adafruit NeoPixel 60 LED Reel**

65.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V
IOREF	Used, the shield operates at 5 V. (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	Yes
Power	5 V
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Datasheet	*http://datasheet.octopart.com/1260-Adafruit-Industries-datasheet-26352316.pdf*

In this test we used an external power as well as the following parts 1000 uFarad electrolytic capacitor(red +, yellow -). 470 ohm resistor between the green data wire and pin 6. Battery 3.7 - 5V. You can also use a 5V external power supply that can supply 3.6 amps in the event that you power all 60 lights at full brightness. Please refer to the power guide. For this testing we were able to adequately power using battery pack consisting of 4 AA batteries.



1000 uFarad electrolytic capacitor(red +, yellow -). 470 ohm resistor between Green data wire and pin 6. Battery 3.7 - 5V.

65.3 Companion library

Adafruit NeoPixel version 1.0.5

65.4 Compile and upload

```
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif

#define PIN 6

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);

// IMPORTANT: To reduce NeoPixel burnout risk, add 1000 uF capacitor across
// pixel power leads, add 300 - 500 Ohm resistor on first pixel's data input
// and minimize distance between Arduino and first pixel.  Avoid connecting
// on a live circuit...if you must, connect GND first.

void setup() {
  // This is for Trinket 5V 16MHz, you can remove these three lines if you are not using a
Trinket
  #if defined (__AVR_ATtiny85__)
    if (F_CPU == 16000000) clock_prescale_set(clock_div_1);
  #endif
  // End of trinket special code
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  // Send a theater pixel chase in...
  theaterChase(strip.Color(127, 127, 127), 50); // White
  theaterChase(strip.Color(127, 0, 0), 50); // Red
  theaterChase(strip.Color(0, 0, 127), 50); // Blue

  rainbow(20);
  rainbowCycle(20);
  theaterChaseRainbow(50);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
}

void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Theatre-style crawling lights.
void theaterChase(uint32_t c, uint8_t wait) {
  for (int j=0; j<10; j++) {  //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();
      delay(wait);

      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}

//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait) {
  for (int j=0; j < 256; j++) {     // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

65.5 Results

This shield works with no problems on the Arduino* 101 board.

Arduino* 101 compatible

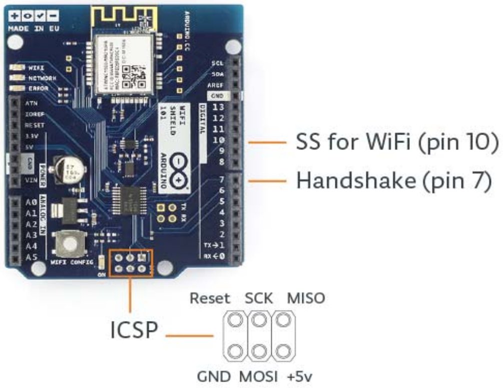
66 Arduino* Wi-Fi 101 Shield

66.1 Use case

This is a test of using Wifi 101 shield on Arduino* 101. The Wifi 101 shield will not work for boards with built-in wifi. Unlike the previous Arduino WiFi shield it does not have a SD card. Arduino WiFi Shield 101 is a powerful IoT shield with crypto-authentication, developed with ATMEL, that connects your Arduino Uno or Intel Galileo board to the internet wirelessly. It supprts 802.11 b/g/n and can support speeds up to 72 Mbps

Key Info	Links
URL	*https://www.arduino.cc/en/Main/ArduinoWiFiShield101*
Library	None. A Wi-Fi 101 Version 0.8.0 library that will need to be installed using the Library manager
Guide	*https://www.arduino.cc/en/Guide/ArduinoWiFiShield101*

Figure 172 **Arduino* Wi-Fi shield**



66.2 Hardware summary

Key Info	Description/Links
Operating voltage	Designed for 5 V and 3.3V
IOREF	Used, the shield can operate either at 3.3 and 5 V. (See *Section 0.8 IOREF voltage*.)
Use VIN as power source	No.
Power	5 V
Arduino* 101 board firmware	Production: Bootloader: ATP1LAKBLR-1541C5640 x86: ATP1LAK000-1541C5635
Schematics	*https://www.arduino.cc/en/uploads/Main/ArduinoWiFiShield101.zip*
Shield firmware	The MicroUSB is used for updating the Atmel SmartConnect-WINC1500
Microcontroller for 802.11 b/g/n	Atmel SmartConnect-WINC1500 *http://www.atmel.com/images/atmel-42376-smartconnect-winc1500-mr210pa_datasheet.pdf*
® Crypto Engine	ATECC508A *http://www.atmel.com/devices/ATECC508A.aspx*

The AVR32UC3 and the HDG204 both work with 3.3 V that is generated by the onboard voltage regulator.

The pins involved in communication with the controller board are protected by level shifters.

The handshake signal on pin digital 7 tells to the controller board when the shield is ready to communicate. It is not implemented with hardware interrupts but rather by polling the shield pin. The latency on the Intel® Galileo board GPIO could affect the performance of the other code in the sketch.

Pin Name	Function (no wires required)
D5	Reset Pin
D7	Handshake pin. Tells when the shield is ready for communication.
D10	Slave select for Wi-Fi.
ICSP header	SPI interface through 6-pin ICSP header. SS (slave connect)
Jumper	Leave unconnected. This is used for shield firmware update.

66.3 Companion library

You need to download the Wifi101 Shield library using library manager. The highest working version is 0.8.0 at the time of this report was written.

66.4 Compile and upload

```
#include <SPI.h>
#include <WiFi101.h>

void setup() {
  // initialize serial and wait for the port to open:
  Serial.begin(9600);
  while(!Serial) ;

  // attempt to connect using WEP encryption:
  Serial.println("Initializing Wifi...");
  printMacAddress();

  // scan for existing networks:
  Serial.println("Scanning available networks...");
  listNetworks();
}

void loop() {
  delay(10000);
  // scan for existing networks:
  Serial.println("Scanning available networks...");
  listNetworks();
}

void printMacAddress() {
  // the MAC address of your Wifi shield
  byte mac[6];

  // print your MAC address:
  WiFi.macAddress(mac);
  Serial.print("MAC: ");
  Serial.print(mac[5],HEX);
  Serial.print(":");
  Serial.print(mac[4],HEX);
  Serial.print(":");
  Serial.print(mac[3],HEX);
  Serial.print(":");
  Serial.print(mac[2],HEX);
  Serial.print(":");
  Serial.print(mac[1],HEX);
  Serial.print(":");
  Serial.println(mac[0],HEX);
}

void listNetworks() {
  // scan for nearby networks:
  Serial.println("** Scan Networks **");
  byte numSsid = WiFi.scanNetworks();

  // print the list of networks seen:
  Serial.print("number of available networks:");
  Serial.println(numSsid);

  // print the network number and name for each network found:
  for (int thisNet = 0; thisNet<numSsid; thisNet++) {
    Serial.print(thisNet);
    Serial.print(") ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm");
    Serial.print("\tEncryption: ");
    Serial.println(WiFi.encryptionType(thisNet));
  }
}
```

66.5 Results

This shield works with no problems on the Arduino* 101 board.

Arduino* 101 compatible