

Intel® Galileo Board, Intel® Galileo Gen 2 Board, and Intel® Edison Board

Shield Testing Report

October 2014

Revision 002



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.

* Other brands and names may be claimed as the property of others.

Copyright © 2014 Intel Corporation. All rights reserved.



Contents

Revision History	16
Testing Overview.....	17
Galileo 1/Galileo Gen 2/Edison SD card library.....	17
Wi-Fi library on Intel® Galileo, Intel® Galileo Gen 2, Intel® Edison.....	17
Inter-Integrated Circuit (I2C) on Intel® Galileo, Intel® Galileo Gen 2, and Intel® Edison	18
Serial Peripheral Interface (SPI) on Intel® Galileo, Intel® Galileo Gen 2, and Intel® Edison.....	18
Galileo 1 jumpers.....	18
I ² C jumper.....	19
IOREF jumper.....	19
VIN jumper	20
Test hardware.....	20
PWM (pulse width modulation).....	21
Edison only	21
Test motors	22
SIM card for GSM/GPRS shields.....	23
Galileo serial ports.....	24
Galileo/Edison onboard clock.....	24
Galileo/Edison firmware	25
1 Arduino® Motor Shield R3	26
Use case	26
Hardware summary.....	27
Companion library.....	28
Compile and upload	28
Results.....	31
Next steps.....	31
2 XBee® S2 Module w/Arduino® Wireless Proto Shield	32
Use case	32
Hardware summary.....	33
Compile and upload	35
Results.....	37
Next steps.....	37
3 Adafruit® RFID/NFC Shield.....	38
Use case	38
Hardware summary.....	39
Companion library.....	40
Compile and upload	40
Results.....	41
4 Adafruit® Ultimate GPS Logger Shield	42
Use case	42
Hardware summary.....	42
Compile and upload	44
Companion library.....	45
Results.....	45
Next steps.....	45



5	SeeedStudio® Bluetooth® Shield v1.1.....	46
	Use case	46
	Hardware summary.....	46
	Companion library.....	49
	Compile and upload	49
	Results.....	50
	Next steps.....	50
6	Cooking Hacks® eHealth Shield v2.0	51
	Use case	51
	Hardware summary.....	52
	Companion library.....	53
	Compile and upload	54
	Results.....	54
	Next steps.....	54
7	SeeedStudio® E-Ink Shield	55
	Use case	55
	Hardware summary.....	55
	Companion library.....	56
	Compile and upload	57
	Results.....	57
8	SeeedStudio® NFC Shield V1.0	58
	Use case	58
	Hardware summary.....	58
	Companion library.....	59
	Compile and upload	62
	Results.....	62
	Next steps.....	62
9	EMAX® ES08A Analog Servo	63
	Use case	63
	Hardware summary.....	63
	Companion library.....	64
	Compile and upload	64
	Results.....	67
	Next steps.....	67
10	FreeIMU*	68
	Use case	68
	Hardware summary.....	68
	Companion library.....	69
	Results.....	69
	Next steps.....	69
11	Adafruit® Motor Shield v2.0.....	70
	Use case	70
	Hardware summary.....	70
	Compile and upload	71
	Results.....	75
	Next steps.....	75
12	Arduino® GSM Shield	76
	Use case	76



Hardware summary.....	77
Companion library.....	78
Compile and upload	78
Results.....	82
Next steps.....	82
13 Arduino* TFT Screen.....	83
Use case	83
Hardware summary.....	83
Companion library.....	85
Compile and upload	85
Results.....	87
Next steps.....	87
14 FabScan* 3D Scanner.....	88
Use case	88
Hardware summary.....	88
Companion library.....	88
Compile and upload	88
Results.....	92
Next steps.....	92
15 RedBearLabs* Nordic BLE Shield	93
Use case	93
Hardware summary.....	94
Companion library.....	95
Compile and upload	97
Results.....	97
Next steps.....	97
16 Adafruit* I2C RGB LCD Shield Kit with 16 × 2 Display.....	98
Use case	98
Hardware summary.....	98
Companion library.....	99
Compile and upload	100
Results.....	101
Next steps.....	101
17 SparkFun* Serial-Enabled LCD Shield	102
Use case	102
Hardware summary.....	102
Companion library.....	103
Compile and upload	103
Results.....	106
Next steps.....	106
18 SeeedStudio* CAN Bus Shield	107
Use case	107
Hardware summary.....	108
Companion library.....	109
Compile and upload	109
Results.....	110
Next steps.....	110
19 Arduino* Wi-Fi Shield	111



Use case	111
Hardware summary.....	111
Companion library.....	112
Compile and upload	112
Results.....	112
Next steps.....	112
20 Gravitech* 7-Segment Shield	113
Use case	113
Hardware summary.....	114
Compile and upload	114
Results.....	120
Next steps.....	120
21 Adafruit* Data Logging Shield for Arduino*	121
Use case	121
Hardware summary.....	121
Companion library.....	122
Compile and upload	124
Results.....	124
Next steps.....	124
22 LinkSprite* 3G + GPS Shield for Arduino*	125
Use case	125
Hardware summary.....	126
Companion library.....	126
Compile and upload	126
Results.....	130
Next steps.....	130
23 LinkSprite* ATWIN Quad-band GPRS/GSM Shield for Arduino*	131
Use case	131
Hardware summary.....	132
Companion library.....	133
Compile and upload	133
Results.....	136
Next steps.....	136
24 SparkFun* Danger Shield.....	137
Use case	137
Hardware summary.....	138
Compile and upload	138
Results.....	143
Next steps.....	143
25 ITEAD* Bluetooth* Shield (Slave).....	144
Use case	144
Hardware summary.....	145
Compile and upload	145
Results.....	147
Next steps.....	147
26 SparkFun* MP3 Player Shield	148
Use case	148
Hardware summary.....	148



Results.....	155
Next steps.....	155
27 Mayhew Labs* Mux Shield	156
Use case	156
Hardware summary.....	156
Companion library.....	158
Compile and upload	159
Results.....	159
Next steps.....	159
28 LinkSprite* Quad-band GPRS/GSM Shield	160
Use case	160
Hardware summary.....	160
Companion library.....	161
Compile and upload	161
Results.....	163
Next steps.....	163
29 SIMCOM* Quad-band Mobile Sim900 – Arduino* Pack.....	164
Use case	164
Hardware summary.....	165
Companion library.....	165
Compile and upload	166
Results.....	169
Next steps.....	169
30 MCM* RS-232 Arduino* Shield.....	170
Use case	170
Hardware summary.....	170
Companion library.....	171
Compile and upload	171
Results.....	174
Next steps.....	174
31 LinkSprite* RS-485 Shield	175
Use case	175
Hardware summary.....	175
Compile and upload	177
Results.....	177
Next steps.....	177
32 SeeedStudio* Relay Shield	178
Use case	178
Hardware summary.....	179
Compile and upload	179
Results.....	179
Next steps.....	180
33 SeeedStudio* GPRS Shield	181
Use case	181
Hardware summary.....	181
Companion library.....	182
Compile and upload	183
Results.....	186



Next steps.....	186
34 SeeedStudio* Solar Charger Shield v2.....	187
Use case	187
Hardware summary.....	188
Companion library.....	188
Compile and upload	189
Results.....	191
Next steps.....	191
35 MIFARE-One* RFID Tag 13.56 MHz (Keyfob)	192
Use case	192
Hardware summary.....	192
Test.....	192
Results.....	192
36 DFRobot* 2-Amp Motor Shield.....	193
Use case	193
Hardware summary.....	193
Companion library.....	194
Compile and upload	194
Results.....	198
Next steps.....	198
37 Renbotics* Servoshield v2.0.....	199
Use case	199
Hardware summary.....	199
Companion library.....	199
Compile and upload	200
Results.....	202
Next steps.....	202
38 Adafruit* MPL115A2 Barometric Pressure/ Temperature I2C Sensor.....	203
Use case	203
Hardware summary.....	203
Companion library.....	204
Compile and upload	204
Results.....	205
Next steps.....	205
39 Adafruit* Electret Microphone Amplifier MAX4466.....	206
Use case	206
Hardware summary.....	206
Companion library.....	207
Compile and test	207
Results.....	209
Next steps.....	209
40 Topway* LCD LMB162ABC.....	210
Use case	210
Hardware summary.....	210
Companion library.....	211
Compile and upload	211
Results.....	212
Next steps.....	212



41	SparkFun* Ardumoto Motor-Driver Shield	213
	Use case	213
	Hardware summary.....	214
	Compile and upload	214
	Results.....	219
	Next steps.....	219
42	DFRobot* Digital Infrared Motion Sensor.....	220
	Use case	220
	Hardware summary.....	220
	Compile and upload	222
	Results.....	222
	Next steps.....	222
43	SparkFun* BlueSMiRF Silver – Bluetooth* Modem.....	223
	Use case	223
	Hardware summary.....	223
	Compile and Load	225
	Results.....	226
	Next steps.....	226
44	Adafruit* LCD Backpack w/I2C, SPI.....	227
	Use case	227
	Hardware summary.....	227
	Companion library.....	229
	Compile and upload	230
	Results.....	232
45	DFRobot* SPI LCD ST7920, 12864ZW.....	233
	Use case	233
	Hardware summary.....	233
	Companion library.....	234
	Compile and upload	234
	Results.....	237
	Next steps.....	237
46	DFRobot* Digital Servo Driver Shield, v1.0.....	238
	Use case	238
	Hardware summary.....	238
	Companion library.....	239
	Compile and upload	239
	Results.....	241
	Next steps.....	241
47	ThingM* BlinkM RGB LED	242
	Use case	242
	Hardware summary.....	242
	Companion library.....	242
	Compile and upload	242
	Results.....	244
48	XBee* S1 Module w/Arduino* Wireless Shield.....	245
	Use case	245
	Hardware summary.....	246
	Companion Software	246



Compile and upload	248
Results.....	250
Next steps.....	250
49 ITEAD* Bluetooth* Shield, Master/Slave	251
Use case	251
Hardware summary.....	251
Companion library.....	252
Compile and upload	254
Results.....	256
Next steps.....	256
50 Adafruit* Digital Accelerometer.....	257
Use case	257
Hardware summary.....	257
SPI configuration.....	258
Companion library.....	259
Compile and upload	261
Results.....	262
Next steps.....	262
51 Adafruit* L3GD20 Gyro	263
Use case	263
Hardware summary.....	263
Companion library.....	264
Compile & Test.....	264
Results.....	266
Next steps.....	266
52 Datan* Analog Feedback Micro Servo - Metal Gear.....	267
Use case	267
Hardware summary.....	267
Companion library.....	268
Compile and upload	268
Results.....	270
Next steps.....	270
53 Sharp* Digital Distance Sensor w/Pololu* Carrier	271
Use case	271
Hardware summary.....	271
Companion library.....	272
Compile and upload	273
Results.....	273
Next steps.....	273
54 Adafruit* Nonvolatile FRAM Breakout SPI.....	274
Use case	274
Hardware summary.....	274
Companion library.....	275
Compile and upload	276
Results.....	277
Next steps.....	278
55 Sparkfun* Big Easy Driver v1.2	279
Use case	279



Hardware summary.....	279
Companion library.....	281
Compile and upload	281
Results.....	282
Next steps.....	282
56 Adafruit* Coin Cell Battery.....	283
Use case	283
Hardware summary.....	283
Companion library.....	284
Compile and upload	284
Results.....	285
Next steps.....	285
57 Sparkfun* Line Sensor Breakout	286
Use case	286
Hardware summary.....	286
Companion library.....	288
Compile and upload	288
Results.....	289
Next steps.....	289
58 Parallax* Ping* Ultrasonic Distance Sensor	290
Use case	290
Hardware summary.....	290
Companion library.....	291
Compile and upload	292
Results.....	294
Next steps.....	294
59 Sparkfun* Digital Temperature Breakout.....	295
Use case	295
Hardware summary.....	295
Companion library.....	296
Compile and upload	296
Results.....	297
Next steps.....	297
60 XBee Wi-Fi Module w/Arduino Wireless Proto Shield.....	298
Use case	298
Hardware summary.....	298
Companion library.....	300
Compile and upload	300
Results.....	303
Next steps.....	303
61 Ultrasonic Ranging Module HC-SR04.....	304
Use case	304
Hardware summary.....	304
Companion library.....	305
Compile and upload	305
Results.....	306
Next steps.....	306
62 Pololu QTR-3A Reflective Sensor Array	307



Use case	307
Hardware summary.....	307
Companion library.....	308
Compile and upload	308
Results.....	309
Next steps.....	309

Figures

Figure 1 Galileo 1 jumpers.....	18
Figure 2 IOREF jumper and multimeter on Galileo 1	19
Figure 3 VIN jumper removed on Galileo 1	20
Figure 4 Servo PWM oscilloscope output for Galileo Gen 2	21
Figure 5 Setting the correct PWM swizzler jumpers – Edison only	21
Figure 6 Edison PWM swizzle jumper settings for pins 3, 5, 10 and 11.....	22
Figure 7 Arduino motor shield	26
Figure 8 Edison PWM Swizzle Jumper Settings used for Arduino Motor Shield R3.....	28
Figure 9 Galileo Gen 2: PWM with a DC motor using a 6 V external power supply	30
Figure 10 Galileo Gen 2: Shield output of the motor using a 6 V external power supply	31
Figure 11 XBee module	32
Figure 12 Radio modules.....	34
Figure 13 Serial switch set to micro.....	34
Figure 14 C and E endpoints on XBee shield w/Galileo 1 and Arduino	35
Figure 15 Adafruit RFID/NFC shield.....	38
Figure 16 Adafruit RFID/NFC shield on Galileo 1	39
Figure 17 MiFare* 1K card	40
Figure 18 Adafruit* Ultimate GPS logger shield.....	42
Figure 19 Adafruit Ultimate GPS logger shield schematic	43
Figure 20 Adafruit Ultimate GPS logger shield Tx/Rx connections	44
Figure 21 SeeedStudio Bluetooth shield v1.1	46
Figure 22 SeeedStudio Bluetooth shield connectors.....	47
Figure 23 Cooking Hacks eHealth shield v2.0.....	51
Figure 24 Cooking Hacks eHealth shield connectors	52
Figure 25 Glucometer connector and graphic LCD connector.....	52
Figure 26 Cooking Hacks eHealth shield sensors	53
Figure 27 Temperature connector output.....	54
Figure 28 SeeedStudio E-ink shield	55
Figure 29 SeeedStudio E-ink shield on Galileo 1	56
Figure 30 SeeedStudio NFC shield on Galileo 1	59
Figure 31 EMAX ES08A Analog Servo	63
Figure 32 EMAX ES08A Analog Servo on Galileo 1	64
Figure 33 Connecting an Analog Servo EMAX ES08A.....	65
Figure 34 Galileo 1 pulse generation	66
Figure 35 Galileo 2 pulse generation	66
Figure 36 Edison pulse generation	67
Figure 37 FreeIMU.....	68
Figure 38 Adafruit Motor Shield v2.0.....	70
Figure 39 Remove Galileo 1 VIN jumper if using an external power supply.....	71
Figure 40 Pulse width while connected to DC motor speed set at 50.....	73
Figure 41 Pulse width while connected with DC motor speed set at 255	73
Figure 42 Connections for a stepper motor test on Galileo 1	74



Figure 43	Arduino GSM shield.....	76
Figure 44	Speaker and microphone connectors	76
Figure 45	Modem and Tx/Rx connectors	77
Figure 46	Wiring the Arduino GSM shield.....	78
Figure 47	Unlocked SIM card inserted	79
Figure 48	Arduino TFT screen pins	83
Figure 49	Connecting an Arduino TFT screen to Galileo 2	84
Figure 50	FabScan 3D scanner	88
Figure 51	FabScan 3D scanner on Galileo 2	91
Figure 52	Galileo 2: Pulse width and frequency graph	92
Figure 53	RedBearLabs Nordic BLE shield	93
Figure 54	RedBearLabs Nordic BLE shield layout.....	95
Figure 55	Adafruit* I2C RGB LCD shield kit with 16 × 2 display	98
Figure 56	Adafruit* I2C RGB LCD shield kit with 16x2 display on Galileo 2.....	99
Figure 57	Galileo 1 – SPI (J2) jumper	100
Figure 58	SparkFun serial-enabled LCD shield	102
Figure 59	Connecting a SparkFun* serial-enabled LCD shield	103
Figure 60	SeeedStudio CAN bus shield	107
Figure 61	SeeedStudio CAN bus shield layout.....	108
Figure 62	SeeedStudio CAN bus shield on Galileo 1 and Arduino.....	109
Figure 63	Arduino Wi-Fi shield	111
Figure 64	Gravitech* 7-segment shield.....	113
Figure 65	Adafruit data logging shield for Arduino.....	121
Figure 66	3G + GPS shield for Arduino	125
Figure 67	3G + GPS shield for Arduino on Galileo 1	126
Figure 68	3G + GPS shield jumpers	127
Figure 69	LinkSprite ATWIN quad-band GPRS/GSM shield for Arduino	131
Figure 70	LinkSprite ATWIN quadband GPRS/GSM shield layout.....	132
Figure 71	Danger Shield.....	137
Figure 72	Danger shield on a Galileo 1	138
Figure 73	ITEAD* Bluetooth* shield (Slave)	144
Figure 74	ITEAD* Bluetooth* shield on Galileo 2	145
Figure 75	SparkFun MP3 player shield	148
Figure 76	SparkFun MP3 player shield on Galileo 1	149
Figure 77	Mux shield.....	157
Figure 78	Mux shield LED wiring	157
Figure 79	Quad-band GPRS/GSM shield	160
Figure 80	Quad-band Mobile Sim900 - Arduino pack.....	164
Figure 81	Quadband Mobile Sim900 layout.....	165
Figure 82	RS-232 Arduino shield.....	170
Figure 83	Galileo COM port test results – without the RS-232 Arduino shield.....	172
Figure 84	Connecting the RS-232 Arduino shield on Galileo 1	173
Figure 85	Galileo 1 COM port test results – with the RS-232 Arduino shield.....	174
Figure 86	LinkSprite RS-485 shield	175
Figure 87	LinkSprite RS-485 shield on a Galileo 1	176
Figure 88	SeeedStudio relay shield	178
Figure 89	SeeedStudio relay shield NO and NC states.....	178
Figure 90	SeeedStudio GPRS shield on Galileo 1	181
Figure 91	SeeedStudio GPRS shield layout	182
Figure 92	Seeed Studio* Solar Charger Shield v2.....	187
Figure 93	Seeed Studio* Solar charger shield v2 with Solar Panel.....	188
Figure 94	Seeed Studio solar charger shield connections.....	189



Figure 95	MIFARE-One RFID tag 13.56 MHz (keyfob)	192
Figure 96	Connecting a 2-amp motor shield using an external power supply	194
Figure 97	Galileo 1: PWM mode at 100 (out of 255) speed with a DC motor using external 6 V for power. Reading taken from pin D5.....	196
Figure 98	Edison: PWM mode at 100 (out of 255) speed with a DC motor using external 6 V for power. Reading taken from pin D5.....	196
Figure 99	Edison: PLL mode at 100 (out of 255) speed with a DC motor using VIN, motor doesn't turn. Reading taken from pin 4.....	197
Figure 100	Arduino: PLL mode at 255 (out of 255) speed with a geared DC motor with no resistance applied. Reading taken from pin 4.....	197
Figure 101	Renbotics Servoshield v2.0	199
Figure 102	Connecting the Renbotics Servoshield on Galileo 1.....	200
Figure 103	Galileo 1: Frequency 52 Hz	201
Figure 104	Galileo 2: Frequency within permissible ranges	202
Figure 105	MPL115A2 barometric pressure/temperature I2C sensor	203
Figure 106	MPL115A2 barometric pressure/temperature I2C sensor on Galileo 1	204
Figure 107	Connecting an MPL115A2 barometric pressure/temperature I2C sensor	204
Figure 108	MAX4466 electret microphone amplifier	206
Figure 109	Connecting an MAX4466 electret microphone amplifier.....	207
Figure 110	Tone Generator display	208
Figure 111	Watch the levels change in the serial console.....	209
Figure 112	Oscilloscope signal.....	209
Figure 113	Topway LCD LMB162ABC	210
Figure 114	Topway* LCD LMB162ABC on Galileo 1	211
Figure 115	Galileo 2 Topway* LCD LMB162ABC Connection	212
Figure 116	SparkFun* Ardumoto motor-driver shield	213
Figure 117	Remove the VIN jumper on the Galileo 1 board	214
Figure 118	Edison PWM Swizzle Jumper Settings used for Ardumoto.....	215
Figure 119	Connecting the Ardumoto motor-driver shield on Galileo 1	215
Figure 120	Connecting the Ardumoto motor-driver shield on Galileo Gen 2	215
Figure 121	First generation Galileo oscilloscope output.....	219
Figure 122	Digital infrared motion sensor	220
Figure 123	Digital infrared motion sensor pins	221
Figure 124	Connecting a digital infrared motion sensor on Galileo 1.....	221
Figure 125	High sensitivity jumper	222
Figure 126	SparkFun BlueSMiRF Silver Bluetooth modem.....	223
Figure 127	Connecting BlueSMiRF Silver to Galileo 1	224
Figure 128	Adafruit* LCD backpack w/I2C and SPI	227
Figure 129	I2C LCD Backpack on Galileo 2	228
Figure 130	SPI LCD Backpack on Galileo2	228
Figure 131	SPI LCD ST7920, 12864ZW	233
Figure 132	Connecting an SPI LCD ST7920, 12864ZW on Galileo 2	234
Figure 133	DFRobot digital servo driver shield, v1.0	238
Figure 134	DFRobot digital servo driver shield layout.....	239
Figure 135	Connecting the DFRobot digital servo driver shield	240
Figure 136	Galileo 1 - VIN jumper	240
Figure 137	XBee S1 module	245
Figure 138	XBee S1 module's "MICRO/USB" switch.....	246
Figure 139	Devices discovered output	247
Figure 140	Connecting an XBee S1 module on Galileo 1 and Arduino	248
Figure 141	ITLEAD* Bluetooth* shield (Master).....	251
Figure 142	ADXL345 Digital Accelerometer.....	257



Figure 143	I2C Connection for the Digital Accelerometer to Galileo 2	258
Figure 144	SPI Connection for the Digital Accelerometer to Galileo 2	258
Figure 145	L3GD20 3-Axis Gyro.....	263
Figure 146	Connecting an L3DG20 Gyro using the I2C interface.....	264
Figure 147	Serial Terminal Output	266
Figure 148	Analog Feedback Servo.....	267
Figure 149	Connecting the Analog Feedback Micro Servo to Galileo 2	268
Figure 150	Sharp* Distance Sensor.....	271
Figure 151	Sharp* Distance Sensor on Galileo 2	272
Figure 152	SPI FRAM Breakout.....	274
Figure 153	Connecting the FRAM Breakout to Galileo 2	275
Figure 154	Big Easy Driver Board v1.2	279
Figure 155	Big Easy Driver (parts are not to scale) connection diagram to Galileo 2	280
Figure 156	Connecting the Big Easy Driver using a NEMA017 Servo to Galileo 2	281
Figure 157	Coin Cell Battery Breakout Board.....	283
Figure 158	Coin Cell battery on a Galileo 2	283
Figure 159	Line Sensor Breakout Analog (left)/Digital (right)	286
Figure 160	Analog Line Sensor on a Galileo 2	287
Figure 161	Digital Line Sensor on a Galileo 2	287
Figure 162	Ping* Ultrasonic Distance Sensor (28015, REV A).....	290
Figure 163	Galileo 2 connection for distance sensor.....	291
Figure 164	Sparkfun* TMP102 Digital Temperature Sensor.....	295
Figure 165	TMP102 breakout on a Galileo 2	296
Figure 166	XBee Wi-Fi Module	298
Figure 167	XBee* Wi-Fi Module shield with Arduino* and Galileo 2	299
Figure 168	Name of shield layout, connections, "on a Galileo", etc.....	299
Figure 169	Ultrasonic Ranging Module HC-SR04.....	304
Figure 170	Galileo 2 connection for distance sensor.....	305
Figure 171	QTR-3A Reflective Sensor on a ¾" line	307
Figure 172	Galileo 2 connection for reflective sensor.....	308

Tables

Table 1	DC motors	22
Table 2	Servo motors	23
Table 3	Stepper motors	23
Table 4	Sketch serial ports	24



Revision History

Revision	Description	Date
001	Initial public release of document.	July 2014
002	Updated to include Intel® Edison platform. Also reformatted using latest Intel template.	October 2014

§



Testing Overview

This document reports results and known issues for shields and sensors executed on an Intel® Galileo development board. Tests were performed to evaluate the hardware and library compatibility on the x86 architecture.

Each test was performed on an Arduino* Uno R3 Board to be used as a comparison before executing it on the first generation Intel® Galileo development board (G1 in this document) production and Intel® Galileo Gen 2 preproduction board (G2 in this document). Both Galileo and Galileo Gen 2 use the same firmware binaries.

Another set of tests was performed on [Intel® Edison using the Arduino Breakout Kit](#). The Arduino breakout board implements the Arduino shield interface and the shields/breakout sensors tested previously was tested on this kit.

We used prerelease firmware versions, not publicly available, in the testing process. However, production releases of the firmware (v1.0.3, released August 2014; and v1.0.4, released October 2014) are available.

The structure of the testing is:

- Examine the documentation related to the shield and the companion library.
- Explanation of the shield's functionality.
- Hardware specification of the shield and checked compatibility with the Galileo before attempting to connect it.
- Examine the library general structure and check if AVR specific code is present.
- Test on the Arduino UNO R3.
- Compile against x86 architecture and attempt to fix any errors.
- Upload and functional test.
- Attempt to resolve problems and report the best results obtained for key features of the shield.

Galileo 1/Galileo Gen 2/Edison SD card library

At this time, there is no support for SD cards that are present on shields or breakout boards. Galileo has an SD card present on the microcontroller board that is accessible through the SPI interface. The interface is called through the SDClass library. The chip select (CS) is passed when the library is initialized.

```
SD.begin(10);
```

On Galileo, the IDE has replaced the entire SD library (SD.h) with respect to Arduino. Deep in the SD library, the CS is hardcoded to 10. If the "begin" function passes another value, that value is ignored, therefore, it is not possible to change the CS value for the SD Card. This was validated using an oscilloscope.

We did not explore using a completely different SD library.

Wi-Fi library on Intel® Galileo, Intel® Galileo Gen 2, Intel® Edison

On Galileo, the IDE has replaced the entire Wi-Fi library with respect to Arduino. The example in the IDE, by default, utilizes the Ethernet port on the Galileo board. At this time, for Galileo there is no support for Wi-Fi that exists on a shield using the IDE default Wi-Fi Library. For G1/G2 an alternative the Ethernet port is to use an mPCIe* Wireless card for. Another alternative for G1/G2 is to use the [XBee Wifi Module](#) which is simply a Serial over WiFi at the sketch level. Edison has WiFi Capability, however, that has not yet been investigated.

Inter-Integrated Circuit (I2C) on Intel® Galileo, Intel® Galileo Gen 2, and Intel® Edison

The Galileo supports I2C for Arduino using the Wire Library. I2C is used on the Galileo board to write to the EEPROM, change the PWM settings of a pin, etc. An Arduino sketch communicates through I2C through the A4/A5 Analog pins. I2C can also communicate through the 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) on Intel® Galileo, Intel® Galileo Gen 2, and Intel® Edison

The Galileo SD card is implemented in SPI using the Chip Select (CS)/Slave Select (SS) value of 10, therefore, SPI shields may need to change the default CS. If the CS is changed, the initialization of CS to 10 must remain.

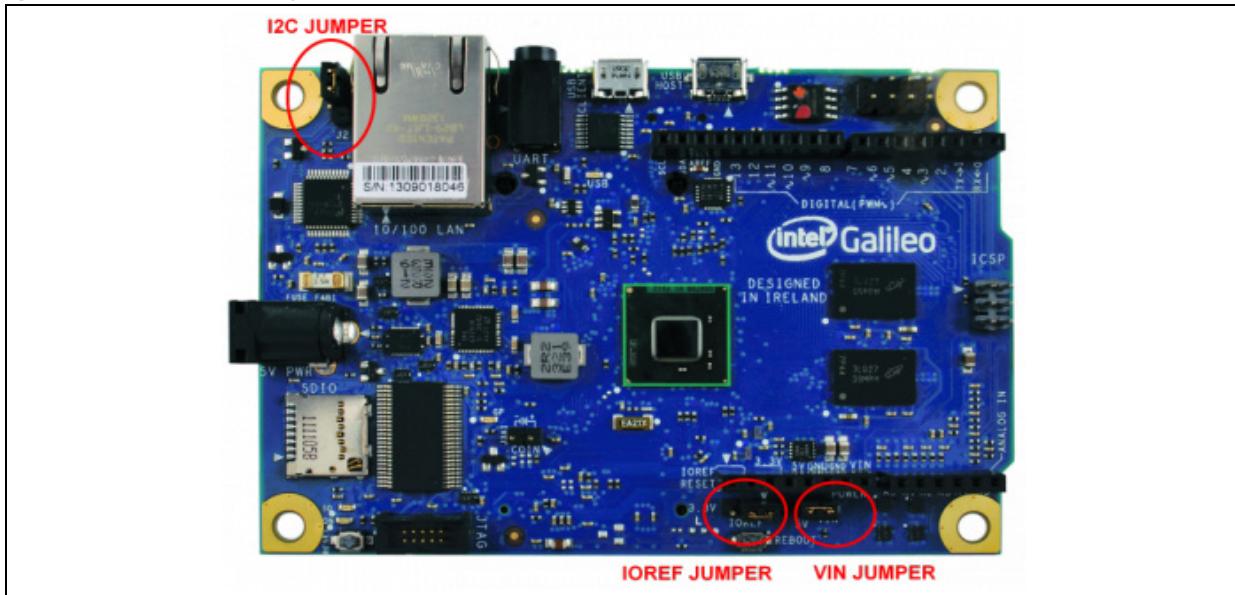
In the Galileo 1 and Galileo 2 release notes, it states that SPI that uses LSB-format mode is not supported ([Issue 55631](#)). Due to the limitation in the silicon, SPI LSP-format is not expected to be supported. Some shields ([E-Ink Shield](#), [NFC Shield](#)) in this test were determined to use the LSB-format, and therefore, were shown not to function. By default, the Galileo board uses MSB-format in the library. Before purchasing a shield, download the library and verify that it does not attempt to put the shield/breakout board in LSB-format. If possible, look at the datasheet for the raw components and verify that it supports MSB-format.

In the [Intel Galileo Board User Guide](#), it states in the “Description of Key Components” table that SPI will act only as a master; therefore, Galileo cannot be an SPI slave to another SPI master. There is one shield, the [CAN Bus Shield](#), that only worked when Galileo was the master. There is an exception using the USB Client connection for the Galileo to be a SPI slave. This was not investigated.

Galileo 1 jumpers

These jumpers are explained on the Arduino site (<http://arduino.cc/en/ArduinoCertified/IntelGalileo>), however, it is beneficial to relate jumpers to these specifically tested shields.

Figure 1 Galileo 1 jumpers





I²C jumper

On Galileo 1, the I²C Jumper is located beside the Ethernet port (shown connected across pin 1 and pin 2). This jumper allows you to change the I²C address of the expansion pins I/O and of the EEPROM memory, functionality both managed via the I²C-bus in slave mode.

This jumper does not exist on Galileo Gen 2 and Edison, so changing the I²C address for expansion pins I/O and of the EEPROM memory is not possible. Before purchasing, verify that the shield/breakout board default I²C address can be changed.

Galileo 1		
I ² C device	Pin1 and Pin 2 (default)	Pin 2 and Pin 3
Expansion pins I/O	0100001 (0x21) 33d	0100000 (0x20) 32d
EEPROM memory	1010001 (0x51) 81d	1010000 (0x50) 80d

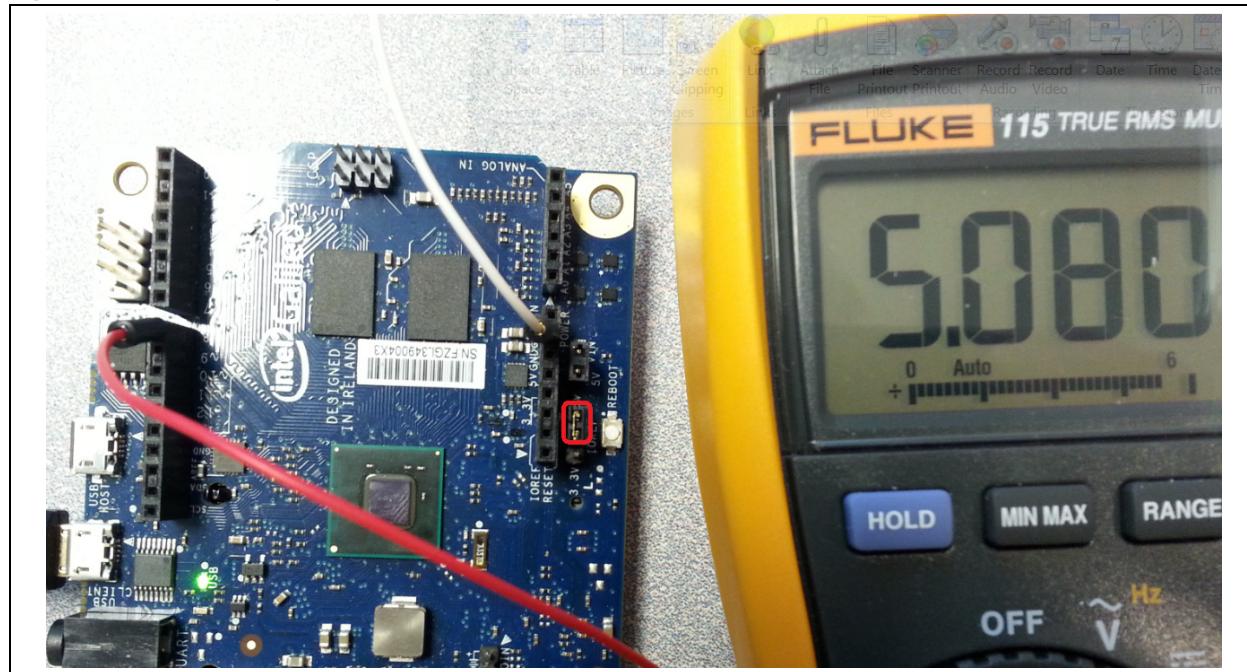
On Galileo 1, some tested shields (*Adafruit LCD Backpack*, etc.) conflicted with the default jumper settings; therefore, changing the Galileo I²C pin settings to the nondefault position enabled the shield to function. Another means of handling I²C address conflicts is to change the I²C address on the hardware (if the shield/breakout board allows it) and then matching that change in software.

IOREF jumper

This jumper is provided to support for 3.3V and 5V shields. Setting this jumper to 5V results in 5V to IO pin (3.3V for 3.3V shields).

Here is an example of setting the IOREF jumper, on Galileo 1, to the 5V position and running a sketch that sets the digital pins high. Notice the ~5V reading on the multimeter.

Figure 2 IOREF jumper and multimeter on Galileo 1



VIN jumper

On Galileo Gen 2 and Edison, this jumper does not exist.

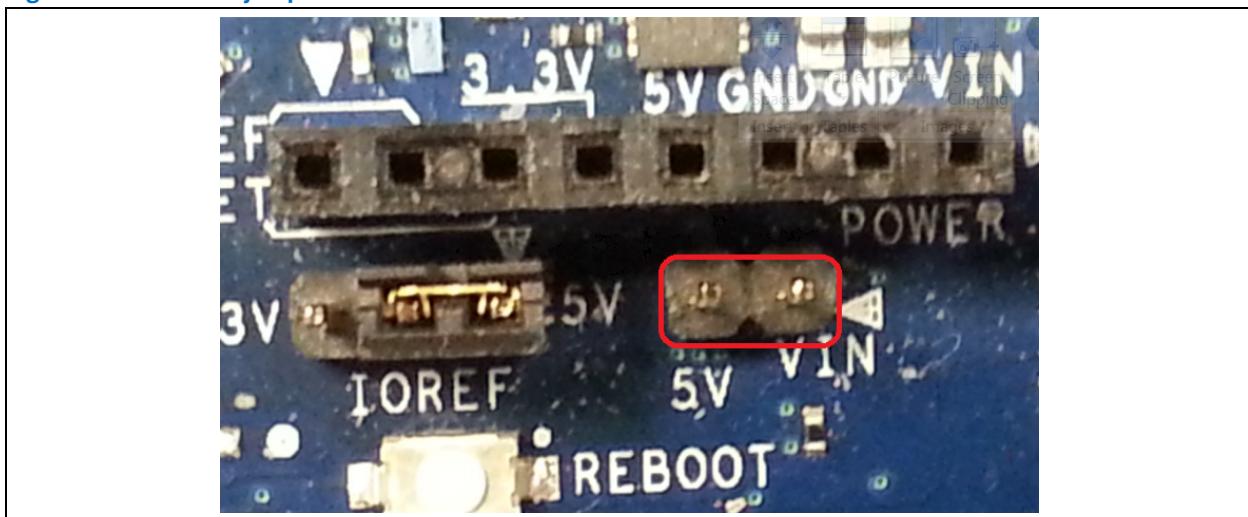
The VIN jumper can be used to supply 5V to shields from the regulated power supply connected to the Galileo 1 power jack. If a shield needs more than 5V, the VIN jumper should be removed from the Galileo to break the connection between the on-board 5V supply and the VIN connection on the board header.

Gen1 Only: Removing the VIN jumper is critical in situations where external power supplies (in excess of 5V) are being used to drive shields with Galileo Gen1. Most motor shields require that this jumper be removed before powering motors with voltage requirements greater than 5V. In some cases the external power supply will also feed power to the Galileo board.

Warning: The Galileo board can be damaged and rendered unusable if you attempt to power motors with an external power supply with this jumper in place.

Various shields (#1, #11, #36, #37, #41, and #46) were tested with the VIN jumper removed and powered by an external power supply. By default, this jumper is present. Refer to the [Galileo Shield List](#) document in the "About the VIN Jumper" Section.

Figure 3 VIN jumper removed on Galileo 1



Test hardware

Some signals were verified using an oscilloscope. DC power supply was used to produce external power to test power related jumpers on the Galileo board and/or on the shield.

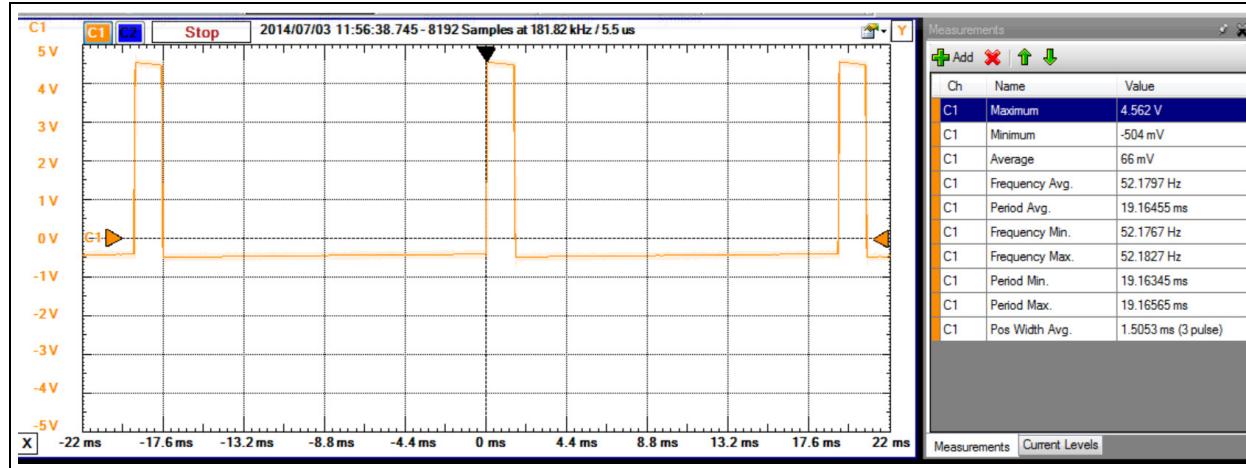
Key info	Description / links
Variable DC Power Supply	http://www.amazon.com/gp/product/B000CSQK5E?ref_=oh_details_o00_s00_i00&redirect=true&pssc=1&pldnSite=1 Primarily used for motor testing. This generated external power for specific shields.
Oscilloscope	http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,842,1018&Prod=ANALOG-DISCOVERY&CFID=4899125&CFTOKEN=d6665606d1992697-11280C15-5056-0201-02608083BE4E9864 Analog Discovery USB Oscilloscope. Many of the oscilloscope screen shots were taken with this software as well as verification of pin activity.



PWM (pulse width modulation)

DC and Servo motors most often use PWM for speed control. On Galileo Gen1 and Gen2 the supported PWM channels are 3, 5, 6, 1, 7 and 4. On Edison there are 6 available pins for PWM, but only 4 can be set for PWM at a time. The 4 pins used for PWM will need a jumper setting as well as a function call in the sketch. Following is the screen shot of a servo motor pulse width. The frequency for PWM is normally set to either 50Hz or 60Hz. For servo motors the range of the positive duty cycle of the pulse is between 1 ms and 2 ms. The servo motor should stop when the pulse width is at 1.5 ms, rotate one direction if between 1 ms and less than 1.5 ms and rotate the other direction when the pulse width is between 1.5 ms and 2 ms.

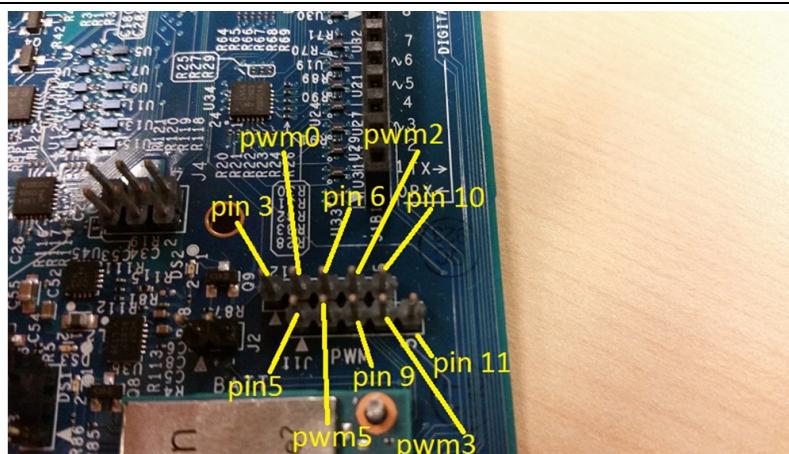
Figure 4 Servo PWM oscilloscope output for Galileo Gen 2



Edison only

Some motor shields use certain pins for PWM without the ability to change. By default, Edison is set to use pins 3, 5, 6, and 9 for PWM. It is possible to use pins 10 and 11 by using a different jumper setting and by adding a function call to the sketch using PWM. Each of the 4 pins to use for PWM must be connected to pwm0, pwm2, pwm3, or pwm5.

Figure 5 Setting the correct PWM swizzler jumpers – Edison only

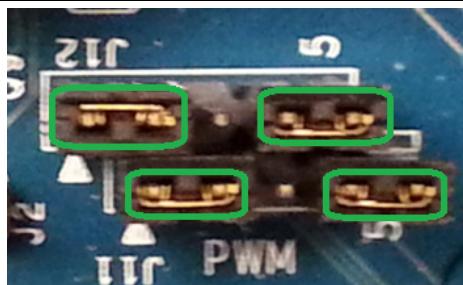


Four jumpers are included to connect 4 pairs of pins. Only 4 pairs can be used at once (4 PWM pins). Below is a photo of the default setting. This setting enables pins 3, 5, 6 and 9 for PWM.

Figure 6 Default PWM swizzler jumper setting – Edison only


If changing from the default configuration of the swizzler jumpers then the sketch using PWM would require an extra function call which could be used in the setup function. Here is an example of the swizzler jumper settings for using PWM on pins 3, 5, 10 and 11. The following function call is added to the top of the setup function in order to use these pins for PWM:

```
setPwmSwizzler(3, 5, 10, 11);
```

Figure 6 Edison PWM swizzle jumper settings for pins 3, 5, 10 and 11


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 1 DC motors

#	Motor	Voltage	Gear ratio	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

Inexpensive servomotors are more problematic for Galileo 1. These motors seemed to get unusually hot. This is typically caused by the PWM setting ([Issue 55603](#)) to gain finer granularity.

**Table 2 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 3 Stepper motors

#	Motor type	Voltage	Stepping	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.

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 U.S. \$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.





Galileo serial ports

Software serial ports are not supported on Galileo 1, Galileo 2, and Edison boards. It is not expected to be supported in the future. The Intel boards do support Hardware Serial that is referenced simply as serial ports. The serial ports are typically used from the IDE for debugging; therefore, it can be confusing if a shield or breakout board utilizes the serial port through digital pins.

The primary serial port on Galileo 1, Galileo 2, and Edison is a single wire, however, in the sketch they are in fact two serial ports that are automatically multiplexed when a sketch is downloading. However, in the sketch itself, the multiplex serial port is referenced as two objects called 'Serial' and 'Serial1'. The 'Serial' object is the serial port on the IDE. 'Serial1' is the serial port connected to D0/D1 for transmit and receive of data on those digital pins.

For Galileo 1, 'Serial2' is the audio connector typically used for a login shell. To use it in a sketch may require some remapping since it is tied to a login process (this has not been investigated).

For Galileo 2, 'Serial2' is mapped into D2/D3 for transmit and receive. The 'Serial2' object can be used as a second serial port accessible through a sketch. In G2, there exist another serial port next to the Ethernet port that can be used as the serial console (previously used by G1 and the audio connector).

For Edison, additional serial ports have not been investigated.

Table 4 Sketch serial ports

Board	# Ports	Serial1	Serial2	Software serial
Galileo 1	1	D0/D1		Not Supported
Galileo 2	2	D0/D1	D2/D3	Not Supported
Edison	1	D0/D1		Not Supported

Galileo/Edison onboard clock

Galileo 1, Galileo 2, and Edison boards do have an on-board clock. The clock can be set through the serial console or through a sketch. However, when the board is powered off, the clock loses its time.

There exist two pins on the board(s) where a battery can be connected to keep minimal power to the board to keep the clock's time. A [breakout board](#) with a battery was tested to demonstrate this functionality. The key step to this procedure is that both the software and hardware clock needs to be set to function correctly.



Galileo/Edison firmware

There is a file on the Linux side of Galileo 1 and Galileo 2 that reports the firmware version. It is possible to make Linux calls from a sketch to display the installed firmware version on the board.

Query Galileo Firmware Sketch

```
void setup()
{
}
void loop()
{
    system("cat /sys/firmware/board_data/flash_version > /dev/ttys0");
    delay(1000*3);
}
```

A few past versions are as follows:

Hex value	Version	Release type
0x000300	1.0.3	Offical Release
0x000200	1.0.2	Offical Release
0x000105	1.0.1	Offical Release

A file also exists on Edison, however, the location is different and the store value is formatted differently.

Query Edison Firmware Sketch

```
void setup()
{
}
void loop()
{
    system("cat /etc/version > /dev/ttys0");
    delay(1000*3);
}
```

The string display is a full date string such as "Edison-weekly_build_70_2014-09-11_19-53-17".

Value	Version	Release type
09/11/14	1.0.3	Offical Release

§

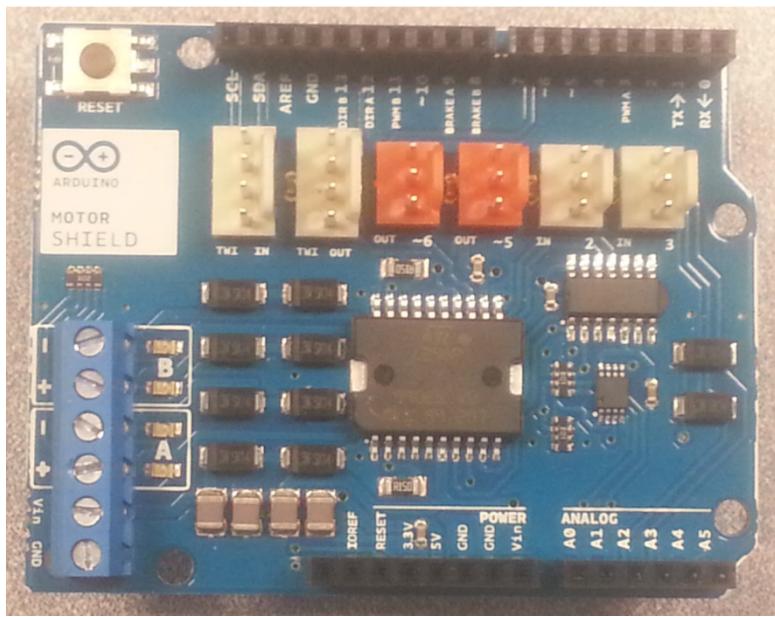
1 Arduino* Motor Shield R3

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 Galileo pins each to drive or sense the motor. 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 7 Arduino motor shield



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	Yes
Use VIN	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
Galileo board firmware	1.0.3 (Production Release)
Edison board firmware	1.0.3 (WW37)
Current sensing	1.65 V/A
Motors	DC1, DC2

Jumpers:

- **Galileo 1 only:** If using more than 5V cut the "Vin Connect" jumper on the back side of the shield and connect the input voltage to Vin on the screw terminal. Another option is to remove the VIN jumper from Galileo. If supplying more than 5V from an external power supply with the "Vin Connect" jumper still soldered and the Galileo VIN jumper installed, damage can occur to the Galileo.
- **Galileo Gen 2/Edison only:** If using an external power supply and using the Galileo 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 jumpers on the shield:



TinkerKit* pins:

- 2 TinkerKit connectors for two analog inputs (in white), connected to A2 and A3.
- 2 TinkerKit connectors for two analog outputs (in orange, in the middle), connected to PWM outputs on pins D5 and D6.
- 2 TinkerKit connectors for the TWI interface (in white, with 4 pins), one for input and the other for output.



One motor runs at a time. To run motor A, change variables to the following:



Pin name	Function for motor A
Direction	Set direction of motor - connected to D12
PWM for Speed	Pulse width modulated speed for motor – connected to D3
Brake	Brake for motor – connected to D9
Current Sensing	Current sensing for motor – connected to A0

To run motor B, change variables to the following:

Pin name	Function for motor B
Direction	Set direction of motor - connected to D13
PWM for Speed	Pulse Width Modulated speed for motor – connected to D11
Brake	Brake for motor – connected to D8
Current Sensing	Current Sensing for motor – connected to A1

The Galileo can provide up to 5 V to the shields. The amount of current that can be provided to the shield depends on the power supply feature. To control motors that require a power supply higher than 5 V remove the VIN jumper on the Galileo or cut the jumper on the motor shield. Providing a voltage >5V could damage the Galileo. The TPS652510 DC/DC converter supports up to 16 V to VIN.

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.

Companion library

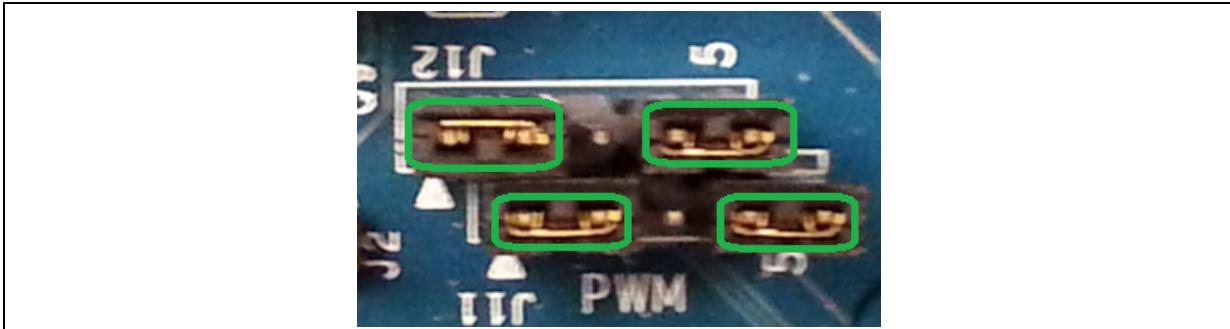
Not required.

Compile and upload

Caution:Galileo 1 only: Remove the VIN jumper on the Galileo board or damage to the board will occur. Read the section

Edison only: The pins used for PWM are 3 and 11 for motor terminals A and B. By default Edison can use pins 3, 5, 6 and 9. See the [PWM section](#) to set the Edison PWM swizzle jumpers so that pins 3 and 11 can be used for PWM. The setPwmSwizzler(3, 5, 10, 11) function will also need to be called in the sketch when changing from the default PWM pins.

Figure 8 Edison PWM Swizzle Jumper Settings used for Arduino Motor Shield R3



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

Example Sketch



```

Example Sketch

int DIR_A      = 12;
int PWM_A      = 3;
int BRAKE_A   = 9;
int SNS_A      = A0;
int SEC        = 5;

void setup() {
    // uncomment this function for Edison, leave commented for Galileo
    // 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

    // Open Serial communication
    Serial.begin(9600);
    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

    analogWrite(PWM_A, 255); // Set the speed of the motor, 255 is the
maximum value

    delay(1000*SEC); // hold the motor at full speed for 5
seconds
    Serial.print("current consumption at full speed: ");
    Serial.println(analogRead(SNS_A));

    // 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
    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

    analogWrite(PWM_A, 255); // Set the speed of the motor
}

```

Example Sketch

```

delay(1000*SEC);
Serial.print("current consumption backward: ");
Serial.println(analogRead(SNS_A));

// 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

Serial.print("current brake: ");
Serial.println(analogRead(SNS_A));
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 9 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 a Galileo Gen2.

Figure 9 **Galileo Gen 2: PWM with a DC motor using a 6 V external power supply**

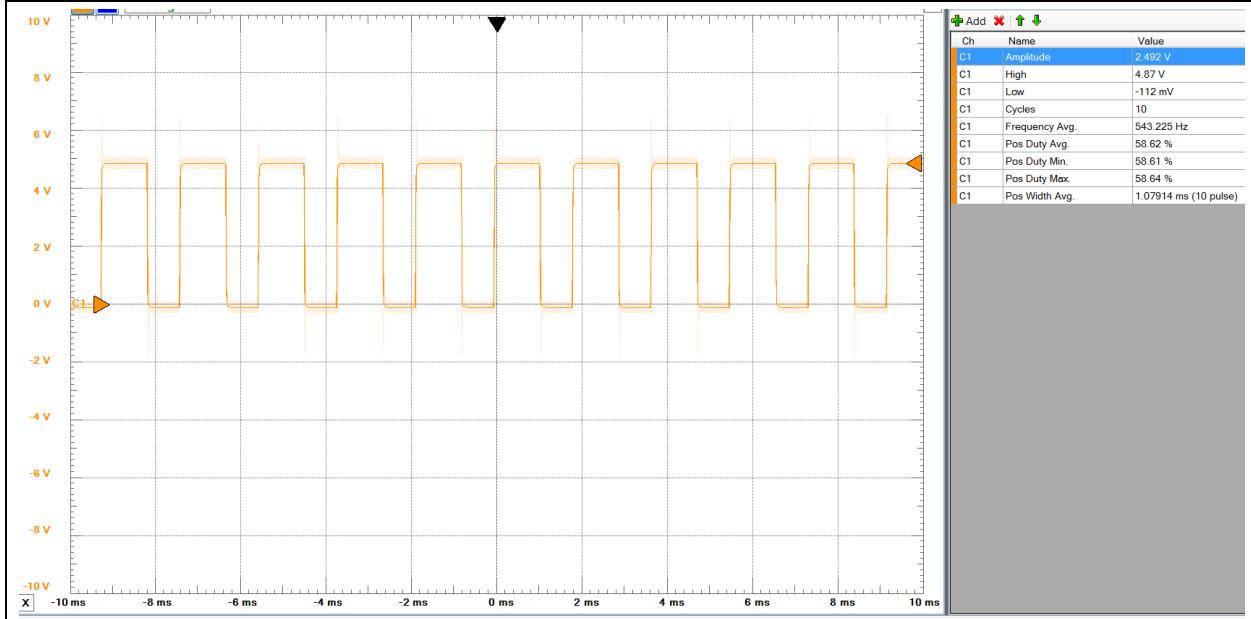
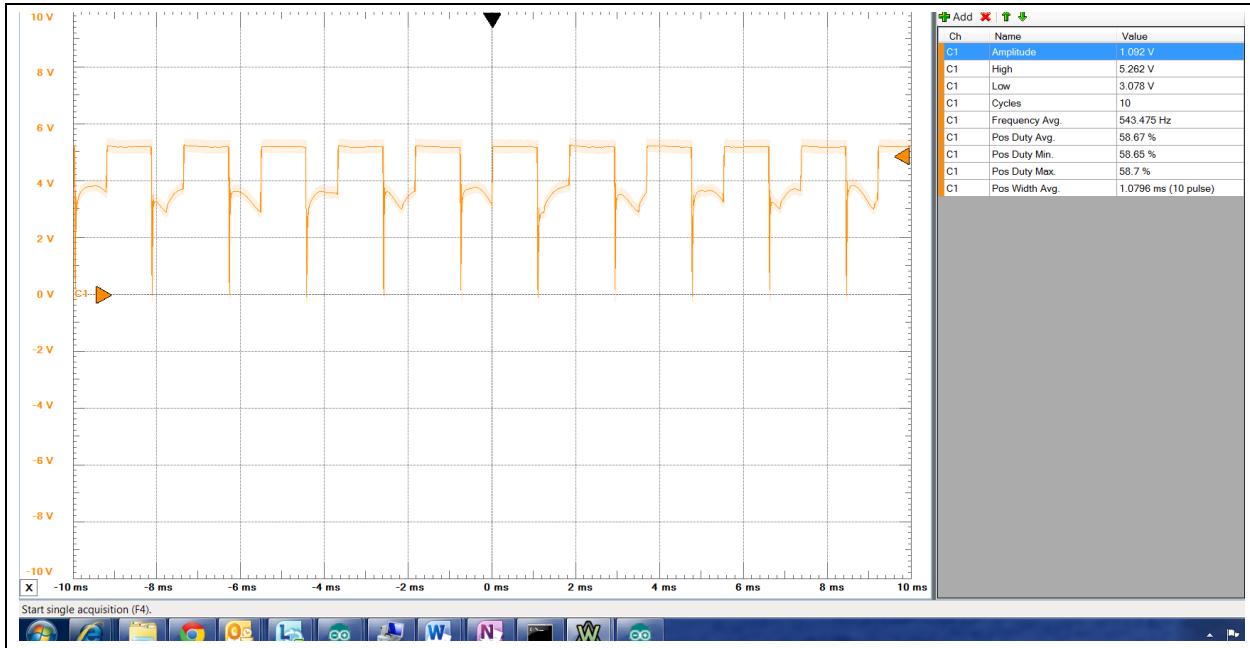


Figure 10 shows a reading from the shield output to the motor with the speed set at 150 (out of 255). The motor is using 6 V from external power supply. This is on a Galileo Gen2.

Figure 10 Galileo Gen 2: Shield output of the motor using a 6 V external power supply

Results

G1/G2/Edison Compatible.

Next steps

- Test with stepper motors.

§

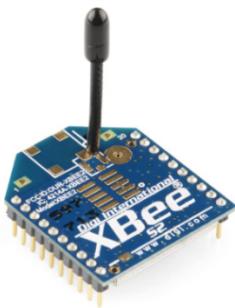


2 XBee® S2 Module w/Arduino® Wireless Proto Shield

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 users' 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 http://www.makershed.com/product_p/mkdg02.htm
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 http://tutorial.cytron.com.my/2012/03/08/xbee-series-2-point-to-point-communication/ The instructions on this site are for version 5.2.8.6 of the X-CTU. The latest version of the software (6.1.0) has an improved interface but is sometimes hard to correlate with instructions for the older version.

The XBee Series 2 modules have two main modes of operation. The most 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.

To create a complex mesh networks (API mode) this requires updating the XBee firmware. To reduce complexity, use a library to operate in API mode.



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (Build WW37)
IOREF	Yes
Use VIN as power source	No
Source	from 5V 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.

Note: It took MANY tries to get the firmware changed. Both the new and old XCTU applications were tried. They would lose communication with the board somewhere in the middle of the process and ask for it to be reset; however, resetting the board did not solve the problem. For both the Controller and End Device, they suddenly worked one time.

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) 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 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.

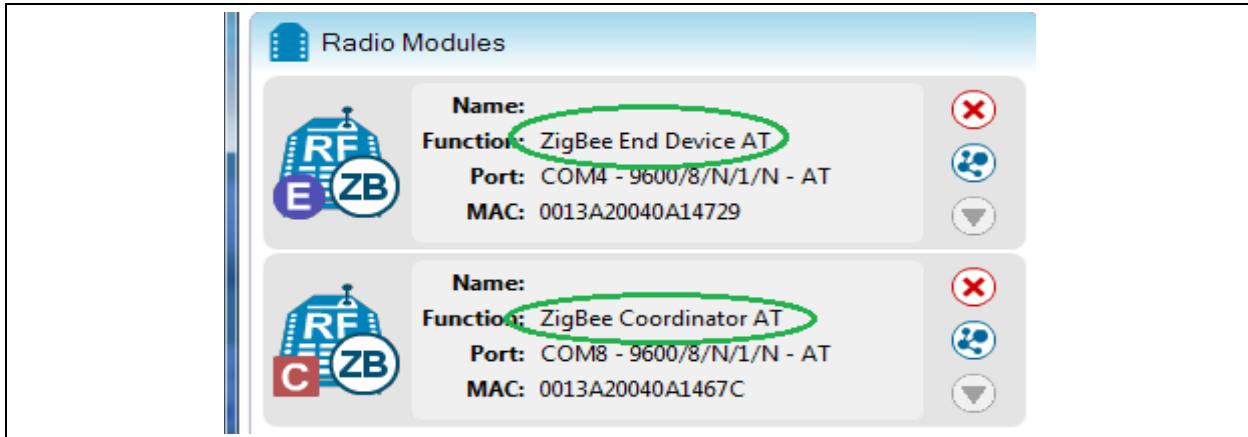
When 'Serial Select' is in the 'USB' position, the DOUT pin of the wireless module is connected to the Rx pin of the USB-to-serial converter, and DIN pin on the wireless module is connected to the Tx pin of the USB-to-serial converter. This means that the module can communicate directly with the computer. 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

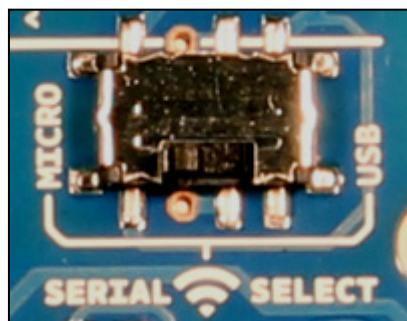


At the time we conducted this test, the query was not functional on a Galileo. Selecting all ports options to query took about 45 minutes, with no success. This means that the Galileo board cannot be used to program the XBee module. The Galileo UART usage is always slightly different from Arduino because the UART on pins 0 and 1 is a different serial port than the one which communicates with the IDE serial monitor. The UART on the Arduino communicates on digital pins 0 (Rx) and 1 (Tx) as well as with the computer via USB.

Normal operation

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

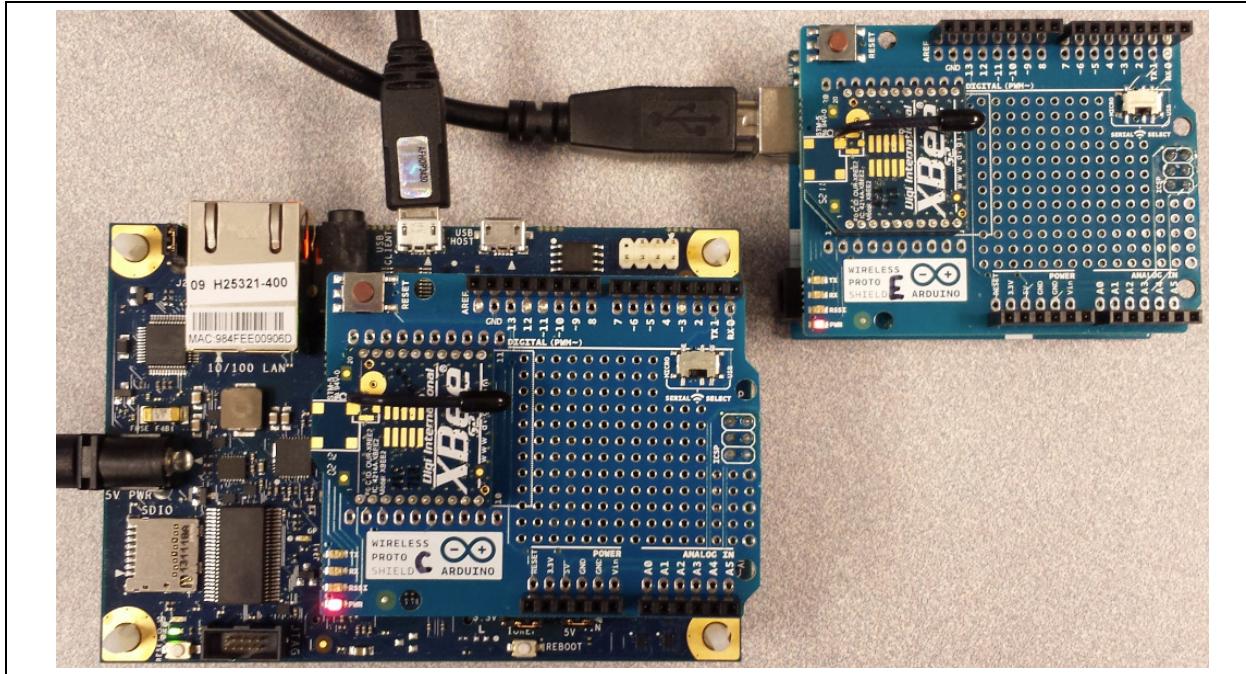
Figure 13 Serial switch set to micro



There are no other pins involved in the shield. Galileo was tested with the XBee shield as the 'Coordinator' and the 'End Point' and vice versa. Notice the "C" and "E" written on the shield.



Figure 14 C and E endpoints on XBee shield w/Galileo 1 and Arduino



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 and the Controller Module was placed on the Galileo. Two example sketches are based on the configuration of Controller or Endpoint.

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

```
Sketch: Receiver
// R E C E I V E R
// This code demonstrates how to run the shield on Arduino and Galileo.
// 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; // Galileo, /dev/ttyGS0, Tx pin
// TTYUARTClass* gSerialTwoPtr = &Serial1; // Galileo, /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("");
}
```

- Set switch to “USB”
- Download the “Sender” sketch to the Galileo. This sketch sends a message repetitively over the Serial (XBee) port. It will be received on the Controller.
- Set switch to “MICRO” and open serial terminal.

Sketch: Sender

```
// S E N D E R
// This code demonstrates how to run the shield on Arduino and Galileo.
// 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; // Galileo, /dev/ttyGSO, Tx pin
TTYUARTClass*   gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO, Rx pin
bool            gGalileo      = true;

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.

The End Point shield was placed on the Edison, the serial port(s) in the sketch was changed over to Galileo settings, and the sketch was downloaded. The Edison board received data transmitted from the Controller.

These shields are compatible without using a library.

Results

G1/G2/Edison Compatible.

Next steps

- Test with the second serial port on Galileo 2. It is not clear if this is possible.
- Determine if the entire XBee library works on the Galileo method.
- Test in API mode and investigate the firmware upgrade process.
- See additional related steps in [XBee S1](#).

§

3 Adafruit* RFID/NFC Shield

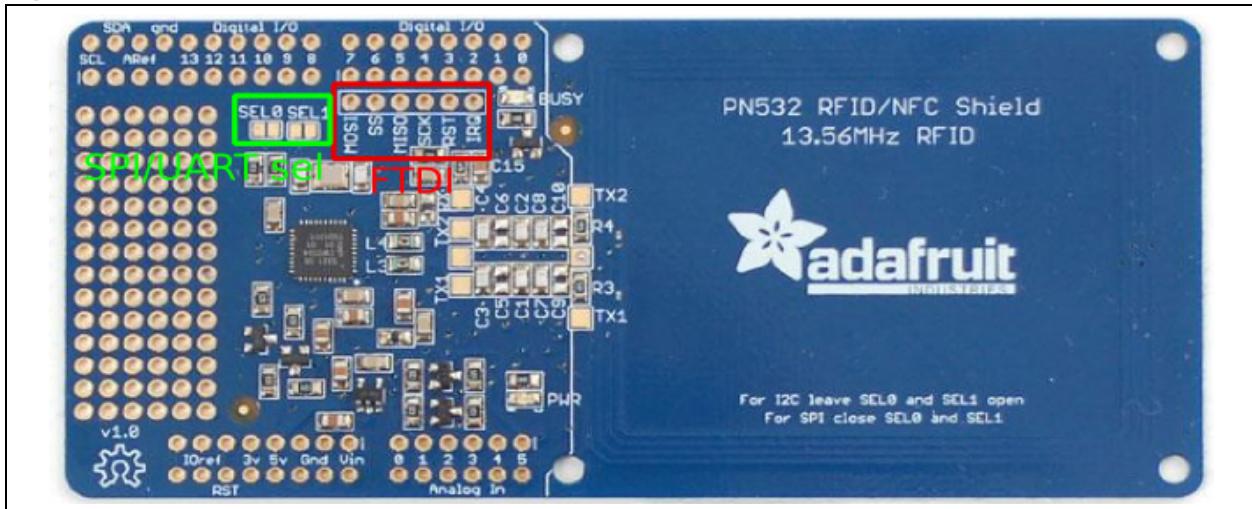
Use case

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_NFCShield_I2C date 10/21/13
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 I2C bus.

Figure 15 Adafruit RFID/NFC shield



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Operating Voltage	Designed for 5V.
IOREF	No (present but not used).
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 I²C by default. I²C 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). I²C 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. The IO2 (D2) pin on the Galileo is a high-speed GPIO so it should be able to handle the IRQ pin.

Figure 16 Adafruit RFID/NFC shield on Galileo 1

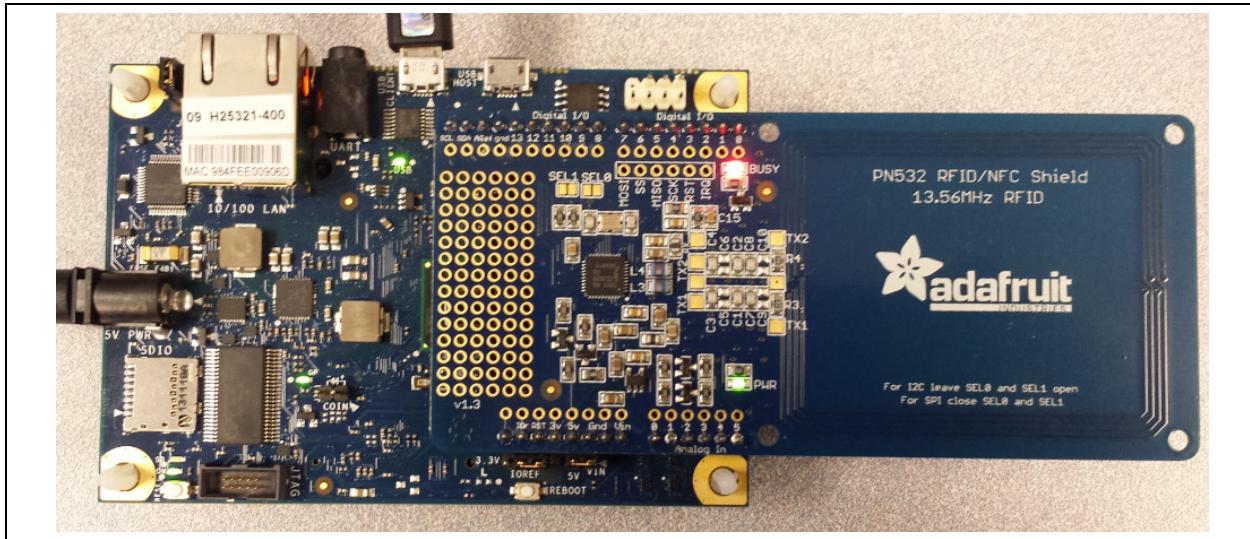




Figure 17

MiFare* 1K card



Companion library

The library is "Adafruit_NFCShield_I2C" that comes with seven example sketches. The examples run with no problems on the Arduino. For Galileo, the only modification required to compile example sketches was in the "Adafruit_NFCShield_I2C.cpp" file where the wire interface was changed; however, notice that the IDE serial baud rate may need to be set to 9600 bps in the sketch. The following code is an example change:

Adafruit_NFCShield_I2C.cpp	
Before	After
#include <Wire.h> #ifdef __AVR__ #define WIRE Wire #else // Arduino Due #define WIRE Wire1 #endif #include "Adafruit_NFCShield_I2C.h"	#include <Wire.h> #ifdef __AVR__ #define WIRE Wire #else // Arduino Due #define WIRE Wire #endif #include "Adafruit_NFCShield_I2C.h"

Compile and upload

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 Arduino and Galileo boards and with [Mifare-One RFID Tag](#) and the RFID card pictured above. All tests worked as expected.

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...
```

Results

G1/G2/Edison Compatible.

§



4 Adafruit* Ultimate GPS Logger Shield

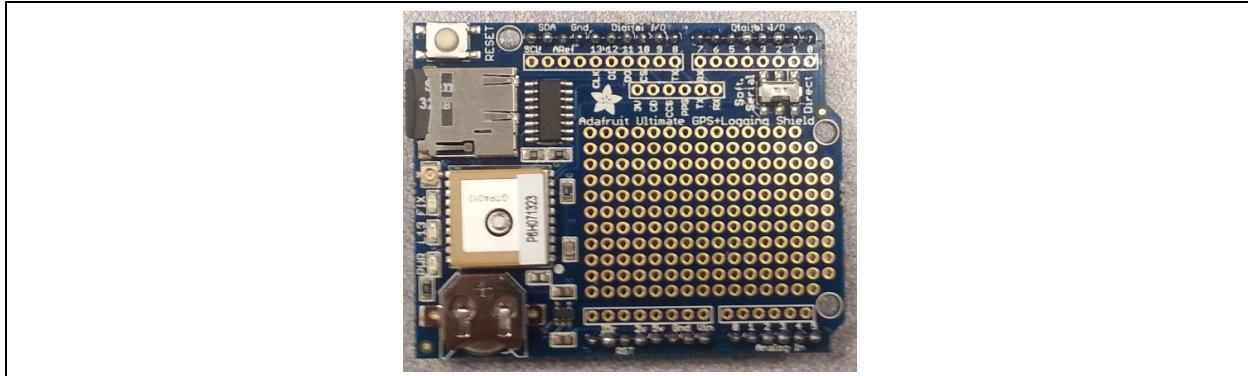
Use case

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. The card can be wired to use the Hardware Serial pins for use with the Galileo/Edison.

Key info	Links
URL	http://www.adafruit.com/products/1272
Library	https://github.com/adafruit/Adafruit-GPS-Library . https://github.com/adafruit/RTCLib . Dated: 8/18/14, 7/1/14

Hardware summary

Figure 18 Adafruit* Ultimate GPS logger shield



Key info	Description/links
Operating Voltage	Designed for 5V. Should work at 3.3V
IOREF	Present but not used.
Use VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
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	I2C Function (No wires required)
A4	I2C, SDA (Serial Data Line) Data
A5	I2C, SCL (Serial Clock Line) Clock
Device Address	I2C, RTClib.cpp defines the address as: #define DS1307_ADDRESS 0x68

The following pins used for the SD Card are as follows for the Arduino* board. On the Galileo, at this time there is no support for an SD cards that are present on a shield. See the [SD Card Library](#) in the Overview section.

The process to list files on the SD is documented in the '[Adafruit Data Logging Shield](#)' section. The sketch used is called 'ListFiles'.

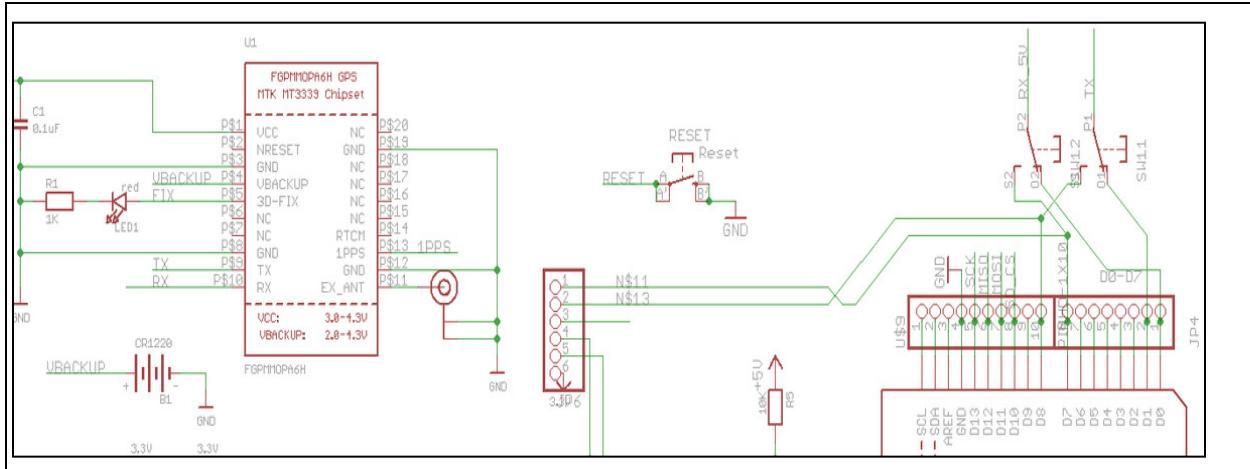
Shield Pin Name	SPI Function (No wires required)
D11	SPI, MOSI (Master Out Slave In)
D12	SPI, MISO (Master In Slave Out)
D13	SPI, Clock (CLK)
CS	SPI, Chip Select (CS), The sketch defines as: SD.begin(10)
D10	Must be left as an output (event if it is not used) in order for the SD to work.

The following pins are used for hardware and software serial. Since Galileo does not support software serial, D0 and D1 are used.

Pin name	Shield Pin/Function
D0	Hardware Tx
D1	Hardware Rx
D7	Connect to the GPS Rx pin on the Shield (Arduino Only for Softserial)
D8	Connect to the GPS Tx pin on the Shield (Arduino Only for Softserial)

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.

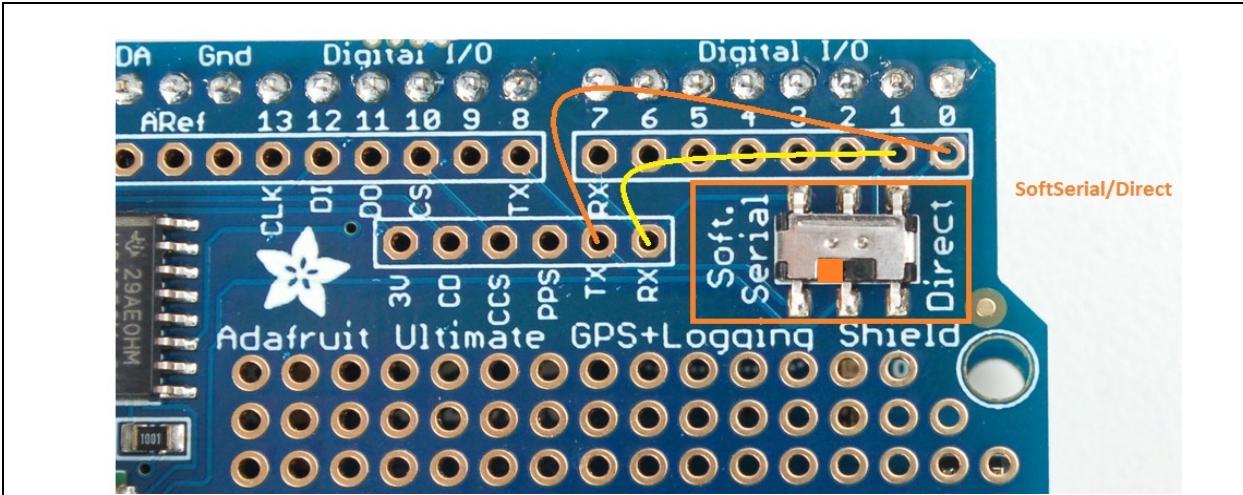
Figure 19 Adafruit Ultimate GPS logger shield schematic





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

Figure 20 Adafruit Ultimate GPS logger shield Tx/Rx connections



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
// This code demonstrates how to run the shield on Arduino and Galileo.
// 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)

// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Tx pin
TTYUARTClass* gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO, Rx pin
boolean bWaiting = true;
void setup() {
    gSerialStdPtr->begin(9600);
    gSerialTwoPtr->begin(9600);
    waitForUser(3);
}

void loop() {
    if(gSerialTwoPtr->available())
    {
        char c = gSerialTwoPtr->read();
        gSerialStdPtr->write(c);
        bWaiting = false;
    }
    if(bWaiting == true)
    {
        gSerialStdPtr->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);gSerialStdPtr->print(i);}
    gSerialStdPtr->println("");
}

```

The data displayed is raw and needs to be parsed.

```

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

```

Companion library

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

On Galileo, working with the companion library for the RTC no longer compiles due to the software serial references.

Results

G1/G2/Edison, GPS Compatible using hardware serial functions. For Galileo 2, using the second serial port was also functional.

G1/G2/Edison, Alternative Solution. Use [Galileo SD card](#) instead of the Shield SD Card.

G1/G2/Edison, RTC Compatible. Alternative Solution to use [Galileo Hardware Clock](#).

Next steps

- Test outside for GPS functionality.
- Utilize the Audio Jack Serial.

§

5 SeeedStudio® Bluetooth® Shield v1.1

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 021314 Provided, however, did not use.
Guide	http://www.seeedstudio.com/wiki/index.php?title=Bluetooth_Shield . 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=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.

Figure 21 SeeedStudio Bluetooth shield v1.1



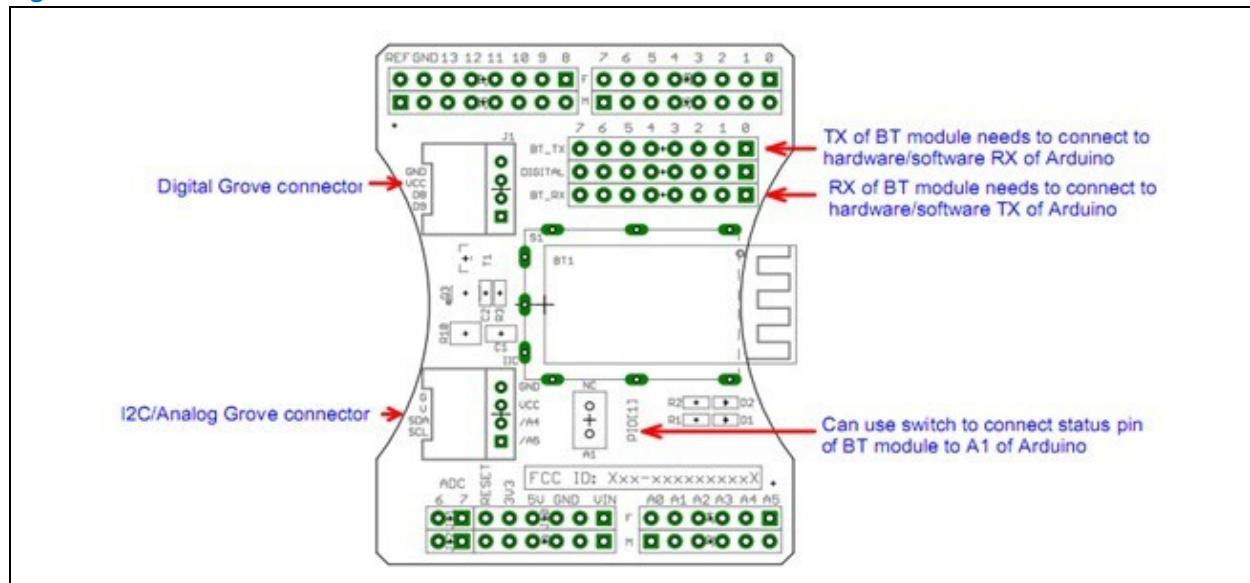
Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
IOREF	No
Use VIN as power source	No
Operating Voltage	5V
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 thru D7). Arduino, set to D7 (default) Galileo, set to D1
BT_Rx	Shield, Selectable pins for Rx (D0 thru D7). Arduino, set to D6 (default) Galileo, set to D0
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 22 SeeedStudio Bluetooth shield connectors



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 version of pairing will be demonstrated with simple printouts. Notice that this example uses two serial ports. On Arduino, softserial is used as the second port. On Galileo, /dev/ttySO is used as the second serial port. These settings have different jumper settings on the shield.

On Arduino, 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", D1 and D2 will flash 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)

// G A L I L E O
//TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Arduino
IDE Serial Monitor
//TTYUARTClass* gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO, Set
Shield BT_Rx jumper to 1, Set Shield BT_Tx jumper to 0

#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+n+STWMOD=0\r\n"); //set the bluetooth work in
slave mode
    gSerialTwoPtr->print("\r\n+n+STNA=SeeedBTSlave\r\n"); //set the bluetooth
name as "SeeedBTSlave"
    gSerialTwoPtr->print("\r\n+n+STOAUT=1\r\n"); // Permit Paired device to
connect me
    gSerialTwoPtr->print("\r\n+n+STAUTO=0\r\n"); // Autoconnection should be
```



Pairing Bluetooth for Arduino

```

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

```

Companion library

No library required.

Compile and upload

For Galileo, the same sketch is used. The sketch is constructed such that Arduino specific code can be commented out and Galileo-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.

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

Before	After
<pre> #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; </pre>	<pre> //#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; </pre>

The pairing LED works the same as it did on Arduino. However, there seems to be more output on the Galileo compared to the Arduino. 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
```

Results

G1/G2/Edison Compatible.

On Galileo 2, test with D2/D2 serial port. Edison/Galileo 1 only has one serial port.

Next steps

- Utilize the 'Grove' connections (I2C, digital) on the board.
- Investigate the "NC" option.

§



6 Cooking Hacks* eHealth Shield v2.0

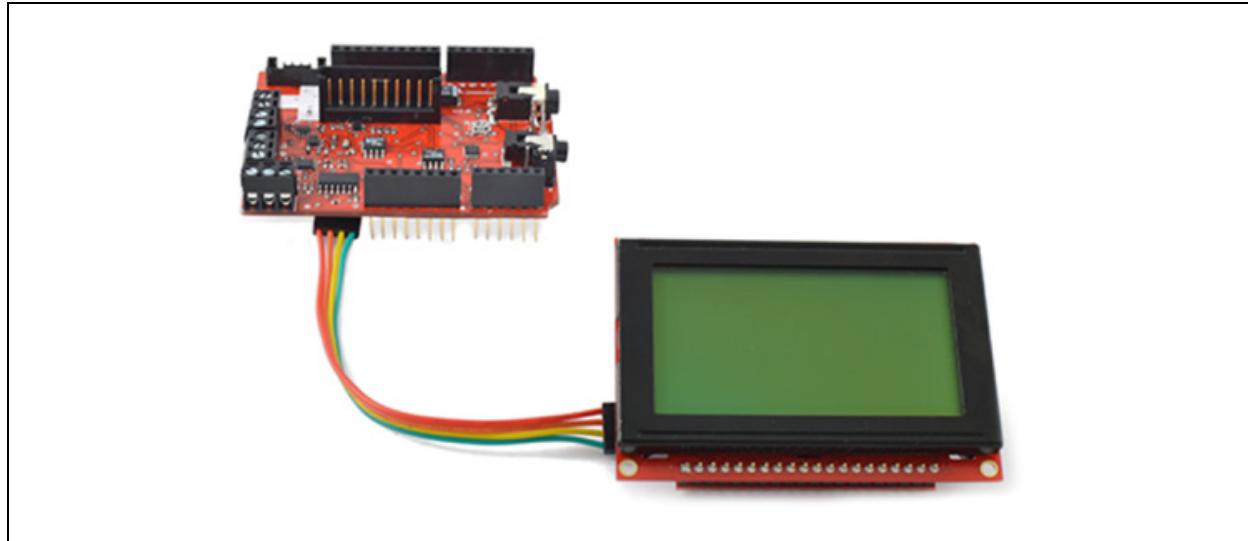
Use case

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	http://skin.cdn-libelium.com/frontend/default/cooking/images/catalog/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 23 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.

Hardware summary

Key info	Description/links
Operating Voltage	5V
Use VIN	No
Schematic	http://skin.cdn-libelium.com/frontend/default/cooking/images/catalog/documentation/e_health_v2/e-Health_v2.0.pdf
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)

Figure 24 Cooking Hacks eHealth shield connectors

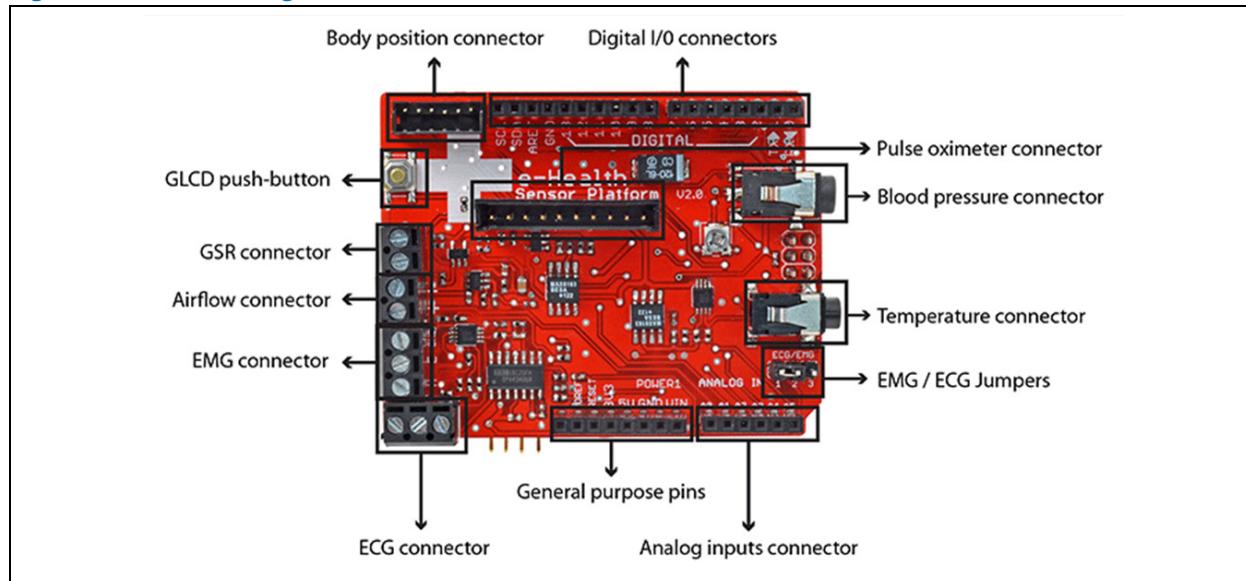
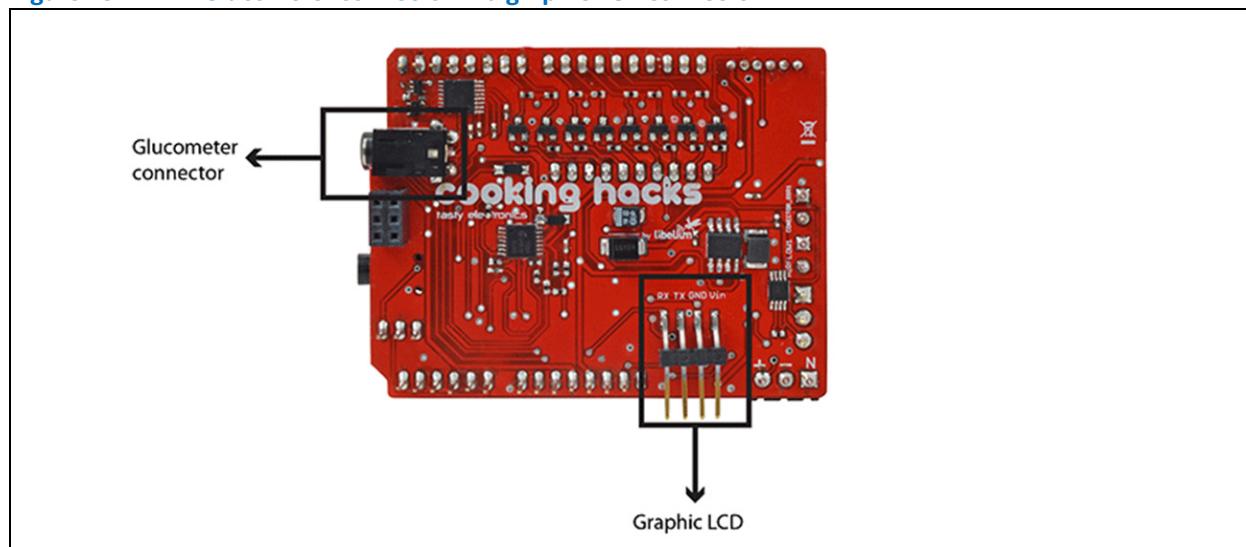


Figure 25 Glucometer connector and graphic LCD connector



Here is a photo of the sensors included when ordering the complete platform kit.

Intel® Galileo Board, Intel® Galileo Gen 2 Board, and Intel® Edison Board

Shield Testing Report

52

October 2014

Document Number: 330937-002

Figure 26 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

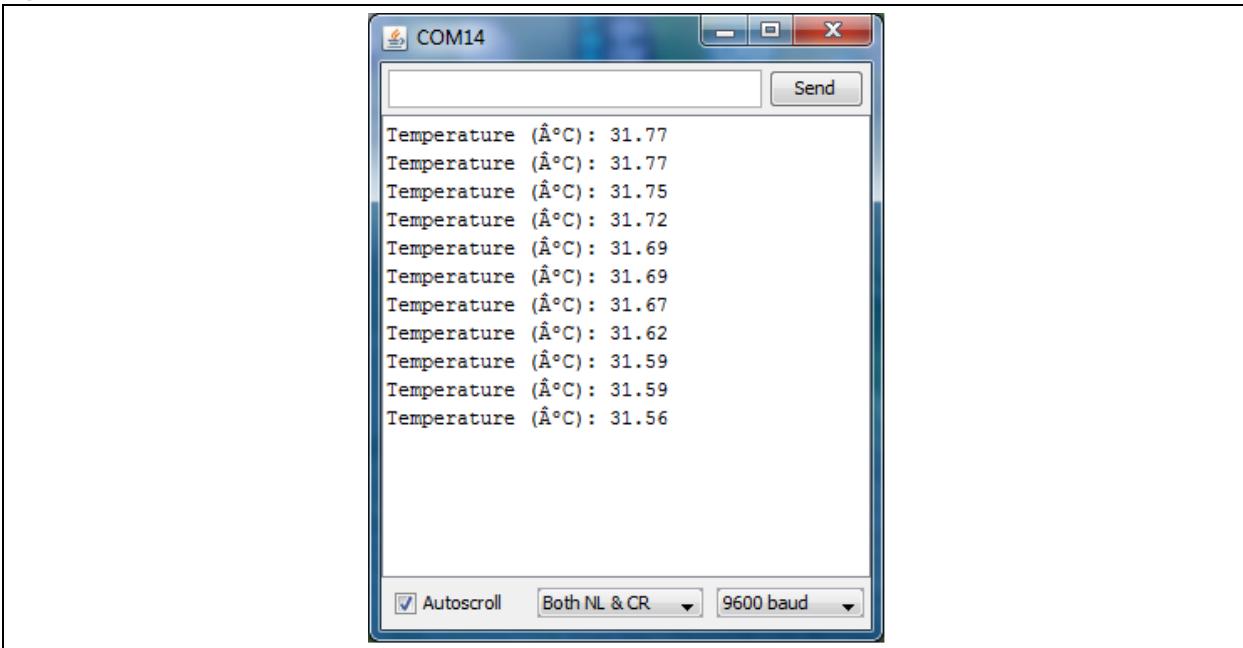
Companion library

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

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 27 Temperature connector output



Results

G1/G2/Edison Compatible.

Next steps

- Test with other sensors connected to the shield.

§

7 SeeedStudio® E-Ink Shield

Use case

These displays are relatively new. SeeedStudio 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 28 SeeedStudio E-ink shield



Hardware summary

Key info	Description/links
Operating Voltage	5V
IOREF	No
Use VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release) Not Supported
Edison Board Firmware	1.0.3 (WW37) Not Supported

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 29 SeeedStudio E-ink shield on Galileo 1



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

Companion library

The library is called "SeedEink" and uses the Arduino SPI library. There are two example sketches called 'displayCharacter' and 'displayChinese'. There was a suggestion to change the #defines for CS pins to make more compatible to Galileo.

Change Eink.h as follows:	
Before	After
#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;}	#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, however, calling the 'pinMode' and 'digitalWrite' is slower than AVR specific instructions.



Compile and upload

The included example sketch 'displayCharacter' compiles. It runs with no problem on the Arduino.

Due to the amount of time that the Galileo takes to configure or control a pin, it is suggested to avoid calling `pinMode()` every time that you need to handle the CS pin. Other changes were made to the library; however, these did not produce solid results. Garbage was being displayed; no clear characters.

Looking deeper into the library source code, there is a statement in the library that temporarily sets the SPI to LSB. The LSB-Mode in Galileo is not supported; therefore, we conclude that this shield is not supported.

Library implementation file: Eink.cpp

```
void E_ink::getCharMatrixData(INT16U unicode_Char)
{
    INT8U i;
    INT8U tempdata;
    INT32U Add=0;
    Add=calculateCharAddress(unicode_Char);
    GT_CS2_HIGH;
    delayMicroseconds(10);
    GT_CS2_LOW;
    SPI.transfer(0x03);
    SPI.transfer(Add>>16);
    SPI.transfer(Add>>8);
    SPI.transfer(Add);
    SPI.setBitOrder(LSBFIRST);
    for(i=0;i<16;i++)
    {
        tempdata=SPI.transfer(0x00);
        matrixdata[i]=(255-tempdata);
        delay(10);
    }
    SPI.setBitOrder(MSBFIRST);
    GT_CS2_HIGH;
}
```

Results

G1/G2 Not supported. See [SPI Issue related to SPI LSB-First](#).

§



8 SeeedStudio* NFC Shield V1.0

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

Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version) Not Supported
Edison Board Firmware	1.0.3 (WW37) Not Supported
IOREF	No.
Use VIN as power source	No.
Power from	5V pin (can use 3.3V or 5V).
Schematics	http://www.seeedstudio.com/wiki/File:NFC_Shield_Schematic.pdf

Typical Current: 100 mA

SPI interface using the ICSP connection hence, most Arduino* pins are available for other applications.

Built in PCB Antenna.

Supports both 3.3V and 5V operation using the TI* TxB0104 level translator.

Socket to connect other shields.

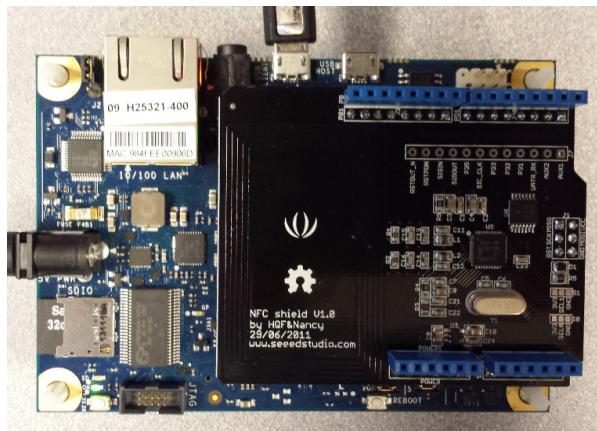
The maximum communication range of this NFC Shield is about 5 cm.

Not able to read/write ultralightC chip - can only Read its ID.

Pinout: None provided.



Figure 30 SeeedStudio NFC shield on Galileo 1



There is some concern about using the SPI interface on the Galileo due to the latency on the SS pin and the slope on the clock signal.

Companion library

The library is PN532_SPI that comes with five example sketches.

Arduino:

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:

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.

Modified Sketch: readAllMemoryBlocks

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

#define PN532_CS 10

PN532 nfc(PN532_CS);
```

**Modified Sketch:** readAllMemoryBlocks

```
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++)
                    {

```



```
Modified Sketch: readAllMemoryBlocks
//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);
}
```



Compile and upload

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

Galileo 1:

The Galileo IDE compiled the sketch and uploaded it correctly to the Galileo board. When run, the sketch could not find the NFC/RFID controller and stopped with the following message: "Didn't find PN53x board". It appears that the Galileo is not able to use the SPI interface to communicate to the NFC/RFID controller on the NFC Shield. An attempt to change the PN532_CS value from 10 to 9 produced the same results.

Looking deeper into the library source code, there is a statement in the library that temporarily sets the SPI to LSB. The [LSB-Mode](#) in Galileo is not supported; therefore, we conclude that this shield is not supported.

Library Implementation File: PN532.cpp

```
void PN532::begin()
{
    pn532_SPI.begin();
    /*The mode of the SPI interface should be set at mode0
       according to the datasheet of PN532 */
    pn532_SPI.setDataMode(SPI_MODE0);
    pn532_SPI.setBitOrder(LSBFIRST);
    /*Set the SPI frequency to be one sixteenth of the
       frequency of the system clock*/
    pn532_SPI.setClockDivider(SPI_CLOCK_DIV16);
    digitalWrite(_cs, LOW);
    delay(1000);
    /*Not exactly sure why but we have to send a dummy command to get synced
    up*/
    pn532_packetbuffer[0] = PN532_FIRMWAREVERSION;
    /*Ignore response!*/
    sendCommandCheckAck(pn532_packetbuffer, 1);
}
```

Results

G1/G2/Edison Not supported due to [SPI-LSB Issue](#).

Everything worked as expected on the Arduino. The library and sketch compiled on the Galileo but it was not able to communicate with the shield during initialization.

Next steps

- Use an in-between shield to move the CS pin.

§

9 EMAX* ES08A Analog Servo

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

Figure 31 EMAX ES08A Analog Servo



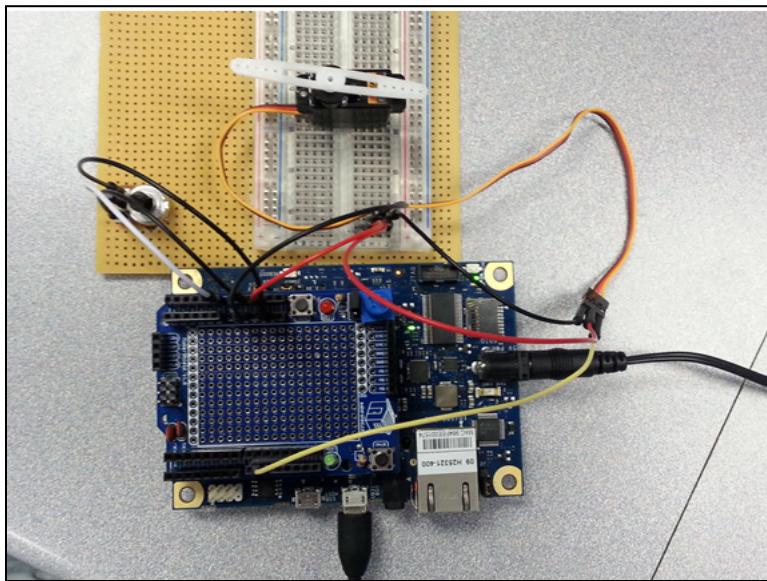
Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Operating Voltage	5V
Use VIN	Depends on the motor shield, for this sketch power was from 5V pin
Weight	8.0 g
Dimensions	23.0 × 11.5 × 24.0 mm
Stall Torque	At 4.8V: 1.5 kg/cm, at 6.0V: 1.8 kg/cm
Speed	At 4.8V: 0.12 sec/60 degrees, 0.10 sec/60 degrees
Pulse width range	1.5 to 1.9 ms

Wire	Function
Orange	Pulse control – connect to D9
Red	Power – connect to 5V
Brown	Ground – connect to GND

Figure 32

EMAX ES08A Analog Servo on Galileo 1



Companion library

No library required.

Compile and upload

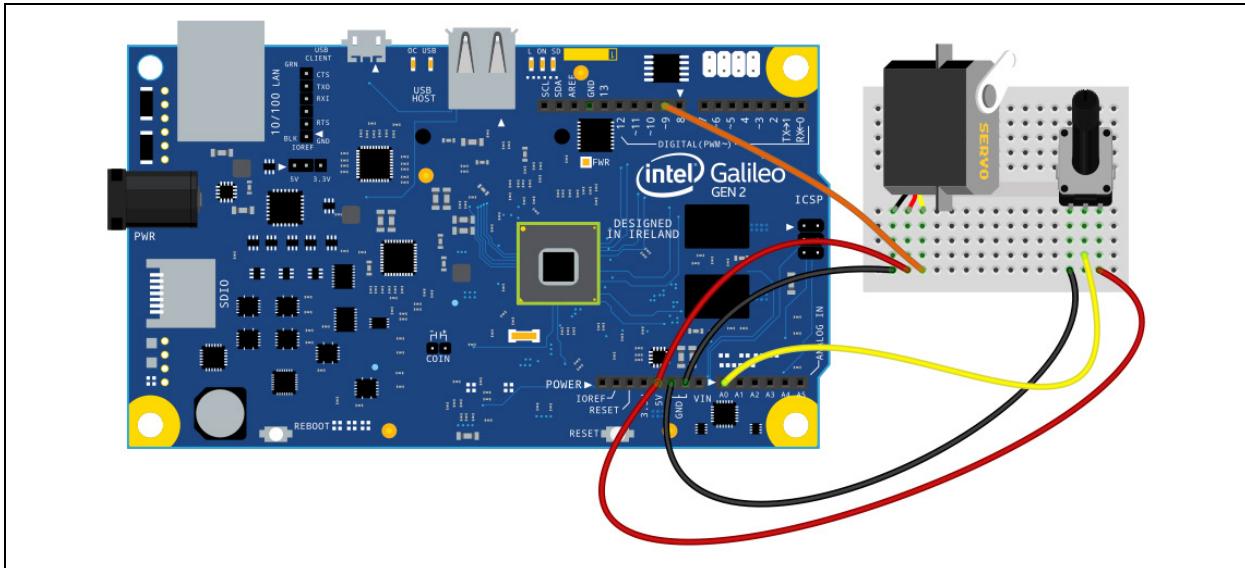
Connect the Servo to the Galileo and run the demos that come with the Arduino version 1.0.5 r2. The Arduino Galileo installation did not have the demos.

Connections:

- Brown wire to GND
- Red wire to 5V
- Orange wire to D9
- 10 Kohm potentiometer wiper to A0
- Right side of pot to 5V
- Left side of pot to GND



Figure 33 Connecting an Analog Servo EMAX ES08A



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

Example 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 potpin = 0; // analog pin used to connect the potentiometer
int val; // variable to read the value from the analog pin

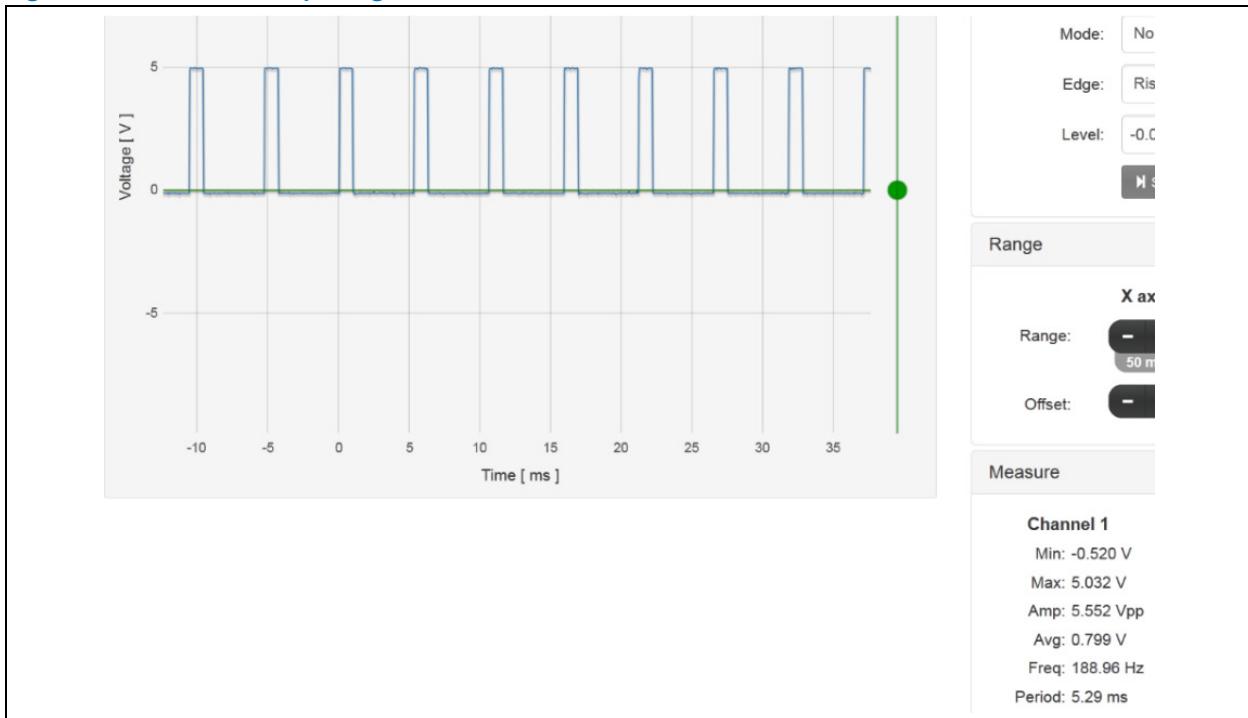
void setup()
{
    myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop()
{
    val = analogRead(potpin); // 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
}
```

Galileo 1: Here is the pulse that Galileo is generating. Frequency should be around 50 Hz instead of 188 Hz.



Figure 34 Galileo 1 pulse generation



Galileo 2: The frequency is close to 50 Hz, which provides smooth operation.

Figure 35 Galileo 2 pulse generation

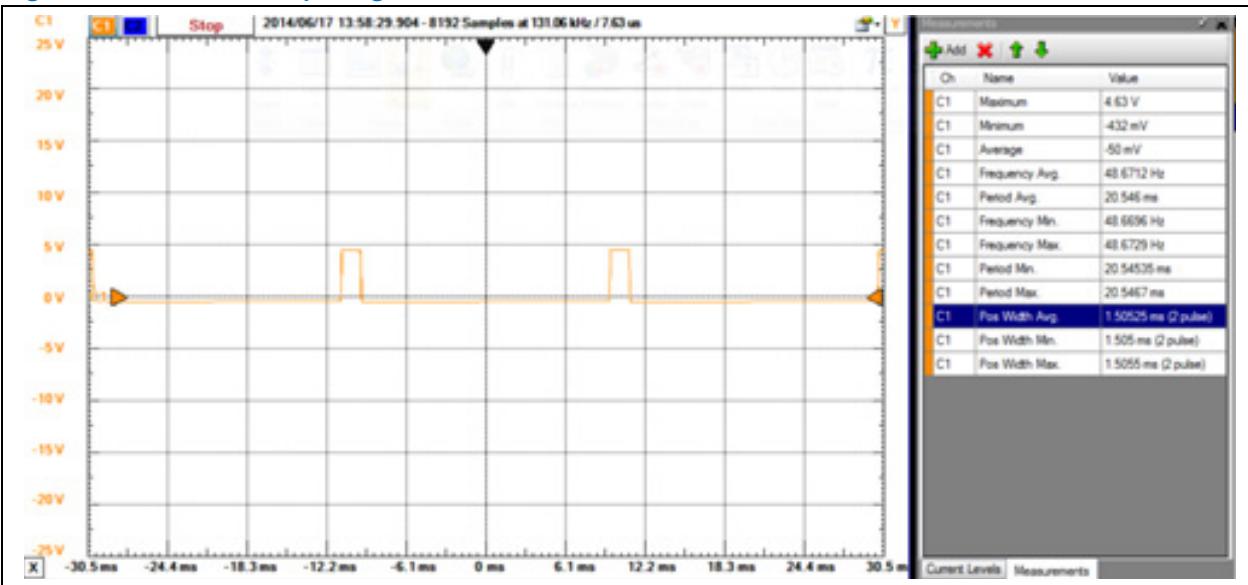
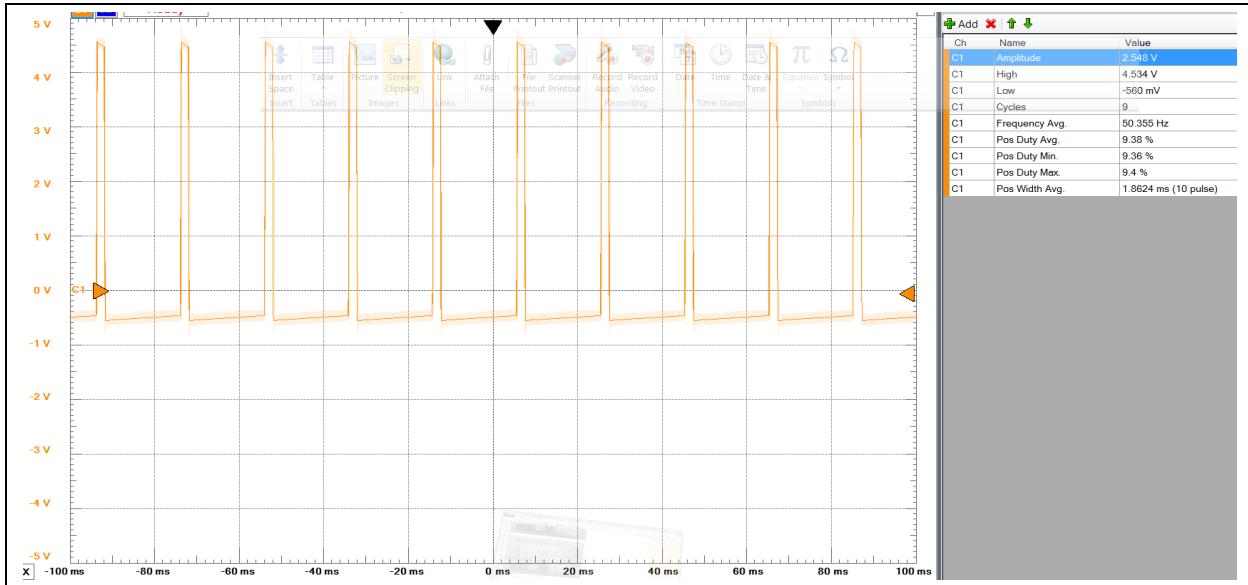


Figure 36 Edison pulse generation

Results

- Galileo 1, not Compatible. The servo motor cannot handle the PWM from Galileo.
- Galileo 2 Compatible.
- Edison Compatible.

Next steps

- Use this motor with servo shields.

§

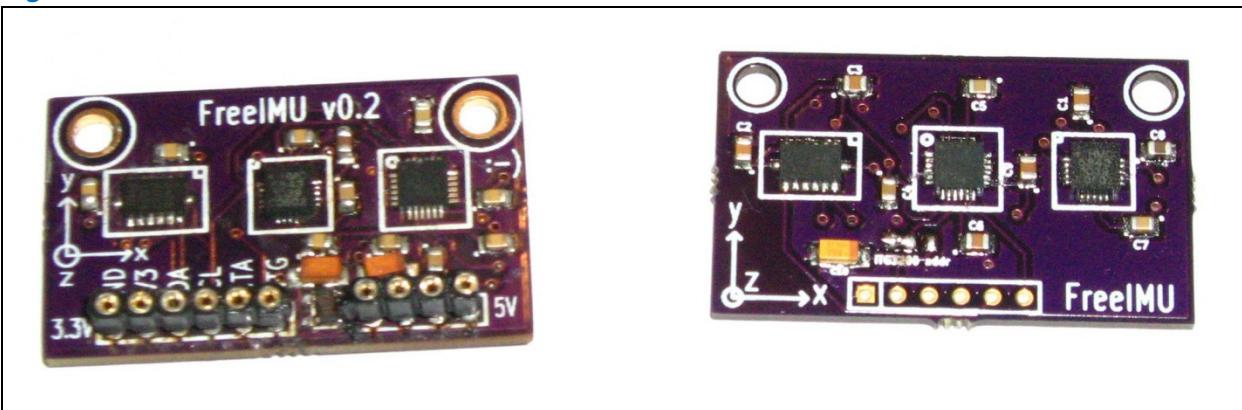
10 FreeIMU*

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 37 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 Galileo or Arduino*.

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.

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	I ² C bus



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.

Results

Inconclusive, since the board is not available.

Next steps

- Obtain one of the list boards above that supports the FreeIMU library.

§

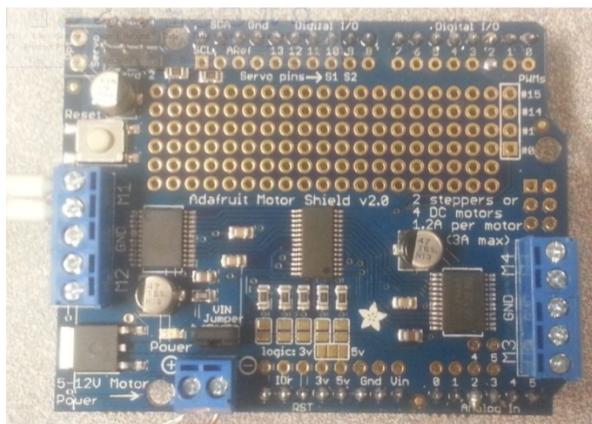
11 Adafruit* Motor Shield v2.0

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 on-board. This chip handles all the motor and speed controls over I2C. Only two pins (SDA and SCL) are required to drive the multiple motors. Since it is I2C, it is possible to connect any other I2C 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

Figure 38 Adafruit Motor Shield v2.0



Hardware summary

Key info	Function
Galileo Firmware	1.0.3 (Release Version)
Edison Firmware	1.0.3 (WW37)
IOREF	No
VIN	Yes, selectable
Power for motors	Either 5V VIN, or 5V to 12V via a power source connected to the power terminal.
Power for logic	3.3V or 5V 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)
I2C bus	<p>Connect to SDA and SCL pins.</p> <p>It is possible to stack multiple shields, however, each shield must have a unique I2C address:</p> <ul style="list-style-type: none"> 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. On first generation Galileo, if you need more than 5 V for the motors, use the external power supply. (Gen 2 and Edison can handle up 12 V.) Power for logic is solderable for either 3.3 or 5 V.

Note: Galileo 1 Only: If using an external power supply, remove the VIN jumper on the Galileo board or damage to the Galileo board can occur if the external power supply is generating over 5 V. Read the section on VIN in the jumper section.

Figure 39 Remove Galileo 1 VIN jumper if using an external power supply



Compile and upload

Adafruit_MotorShield.cpp and Adafruit_PWM_Servo_Driver.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:

```
#define WIRE Wire
```

DC Motor Test:

Connections:

- connect a DC motor to each of the terminals and test the DC motor sketch
- the 5V 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.

**Sketch: DC Motor Test**

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWM_Servo_Driver.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");

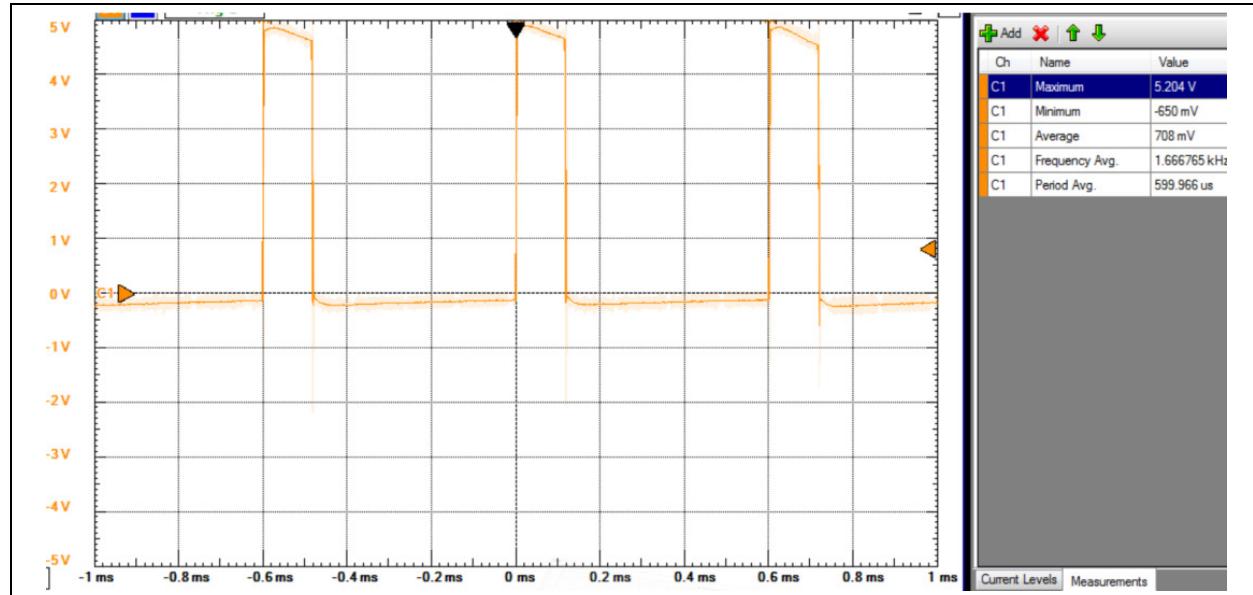
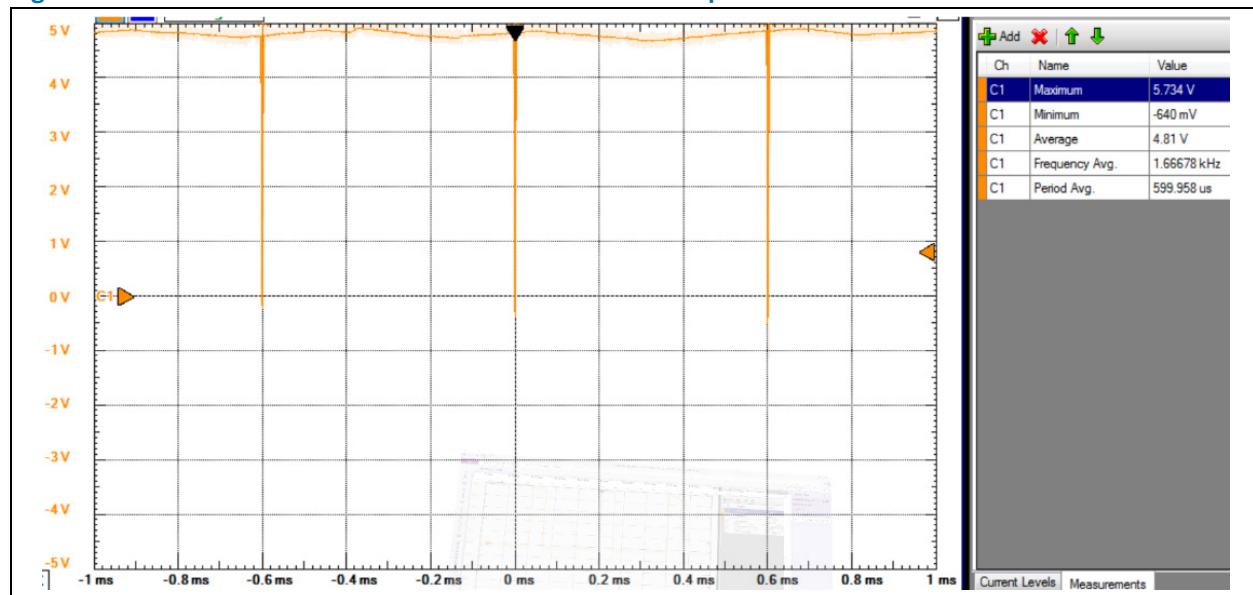
    myMotor->run(FORWARD);
    for (i=0; i<255; i++) {
        myMotor->setSpeed(i);
        delay(10);
    }
    for (i=255; i!=0; i--) {
        myMotor->setSpeed(i);
        delay(10);
    }

    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);
    }
}
```

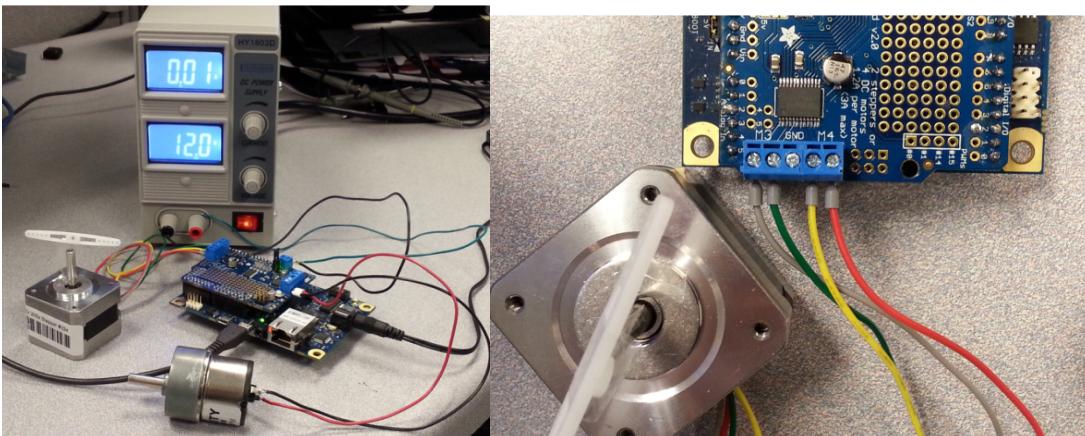
Sketch: DC Motor Test

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

Figure 40 Pulse width while connected to DC motor speed set at 50**Figure 41** Pulse width while connected with DC motor speed set at 255

**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, 12V motor. The DC motor is a 12V geared motor. Use external power supply 12V.

Figure 42 Connections for a stepper motor test on Galileo 1**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_PWM_Servo_Driver.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)
```



```

Stepper Test
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

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

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);
}

```

Results

- DC motors for G1/G2/Edison: Compatible.
- Edison shield will only work when using external power. If using internal power, Edison does not send power to VIN or motor ports, under investigation.
- Stepper motors G1/G2/Edison: Stepper motors were successful.

Next steps

- Test with Servos. Need to cut trace and add a connector. If using servos with Edison some additional steps are required from the [PWM](#) section in order to use pins 9 and 10 for PWM.



12 Arduino* GSM Shield

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

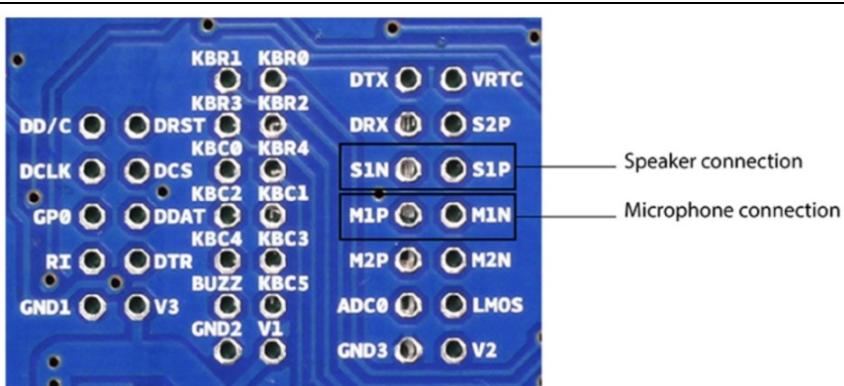
http://arduino.cc/en/uploads/Main/Quectel_M10_AT_commands.pdf. For voice calls, it is necessary to add a speaker and microphone to hear and speak to the other party. This will require soldering.

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 Arduino. Library installs for Galileo, but it is not functional yet.

Figure 43 Arduino GSM shield



Figure 44 Speaker and microphone connectors



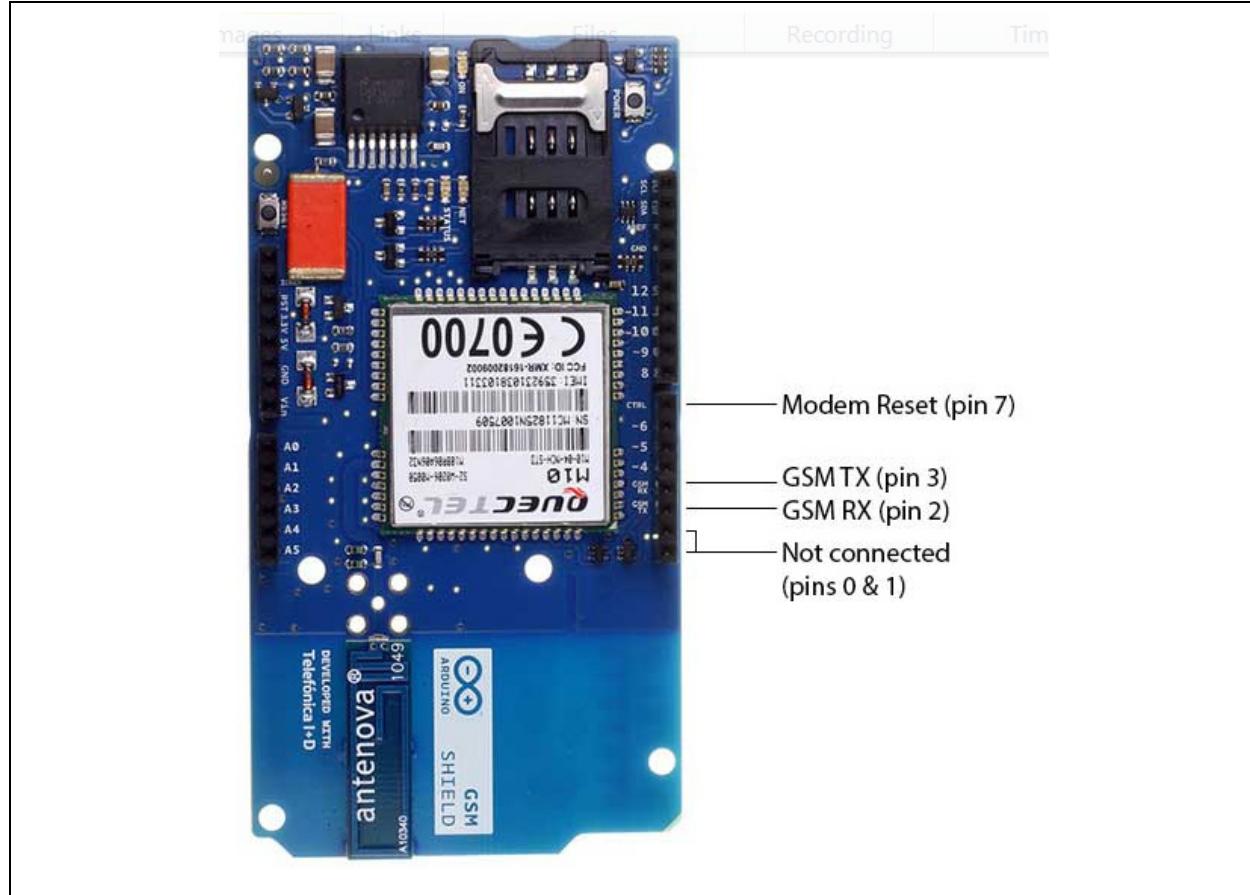
Hardware summary

Key info	Description/links
Operating Voltage	5V
IOREF	Not used
Use VIN	Yes
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Schematic	http://arduino.cc/en/uploads/Main/arduino-gsm-shield-06-reference-design.zip

Pin name	Function
GSM Rx	UART Rx – connected to D2
GSM Tx	UART Tx – connected to D3
CTRL	Modem reset – connected to D7

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 45 Modem and Tx/Rx connectors



Companion library

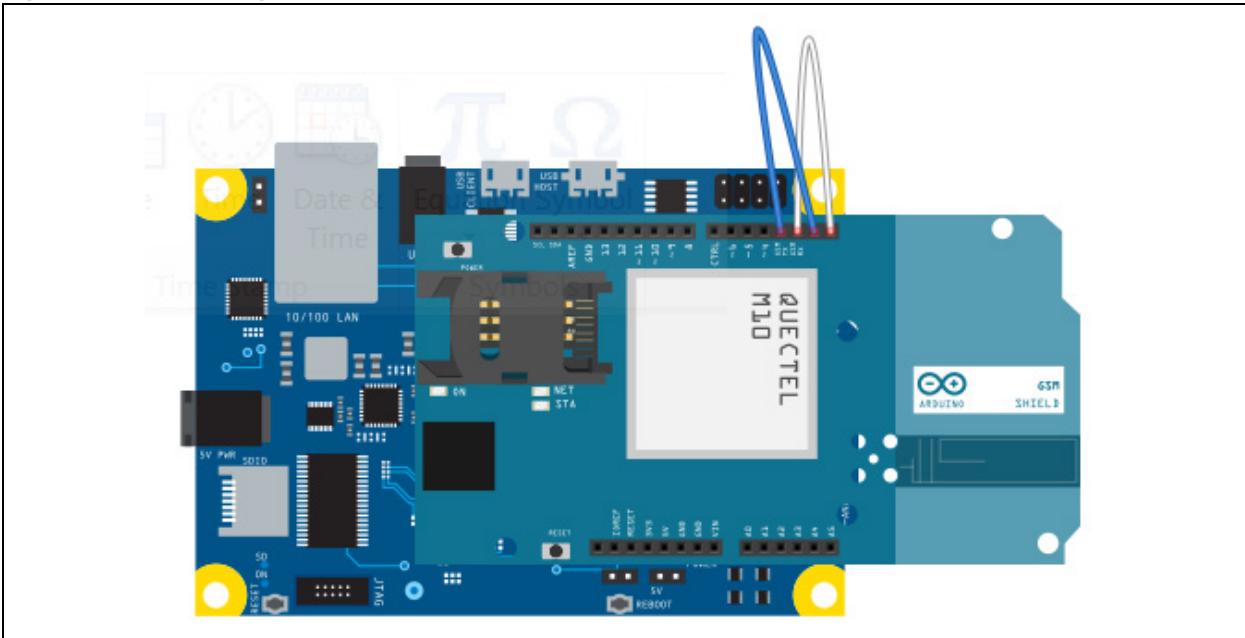
The GSM library is included with Arduino* IDE 1.0.4 and later. There was also a GSM library included with the Galileo IDE, but the GSM sketches would not compile. The Arduino GSM library would compile and the sketches would run successfully on the Arduino card. The library relies on the Software Serial library for communication between the modem and the Arduino.

There is no support for Software Serial on the Galileo, so it is not possible to support the Galileo.

Compile and upload

Since compiling the GSM library for Galileo was not possible, setting up a hardware serial connection was attempted. The transmit of the shield (D2) was connected to the D0 (receive) pin of the Galileo and the receive of the shield (D3) was connected to the D1 (transmit) pin of the Galileo. 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 Galileo and wire the transmit and receive connections.

Figure 46 Wiring the Arduino GSM shield



Insert an unlocked SIM card. Power up and connect the Galileo USB to the PC. Push the power button until the status light and ON light are illuminated. The net light should also blink.

Figure 47 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.

```
Serial1.println("AT+CMGS=\\"1555555555\\\"");  
Serial1.println("ATD + 1555555555;");
```
- 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.)

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

```
#include <String.h>

char data[256];

void setup()
{
    Serial1.begin(9600);
    Serial.begin(9600);      // the GPRS baud rate
    delay(1000);
}

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;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}

void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
    mode
    delay(1000);
    Serial1.println("AT+CMGS=\"15555555555\"");
    delay(1000);
    Serial1.println("Hello from Galileo?");
    delay(1000);
    Serial1.println((char)26); //the ASCII code of the ctrl+z is 26 (0x1A)
    delay(1000);
```



```
Sketch: Issue AT commands over serial link
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.println("ATD + 15555555555;");//dial the number
    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?");
}
```

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

```
delay(3000);
ShowSerialData();

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

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

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

// detach from GPRS newtork
Serial.println("AT+CGATT=0");
delay(1000);
ShowSerialData();
}
```

Results

G1/G2/Edison Compatible.

Next steps

- Solder wires for headset and microphone connection to test the audio quality.

§

13 Arduino* TFT Screen

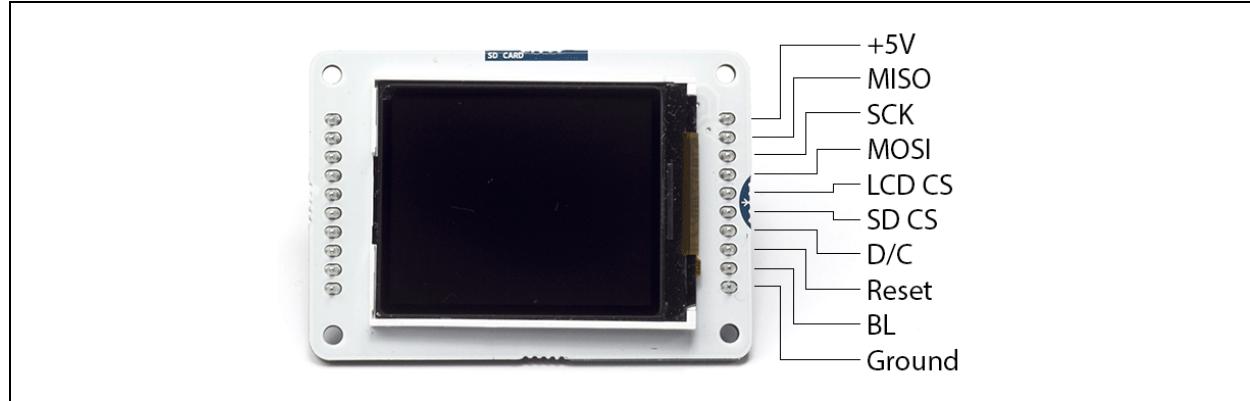
Use case

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	Include with Arduino, although Arduino v1.0.5 does not work. It produces horizontal lines. Obtain an Arduino nightly build (04/14/14, 05/14/14, 09/4/14).

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 48 **Arduino TFT screen pins**

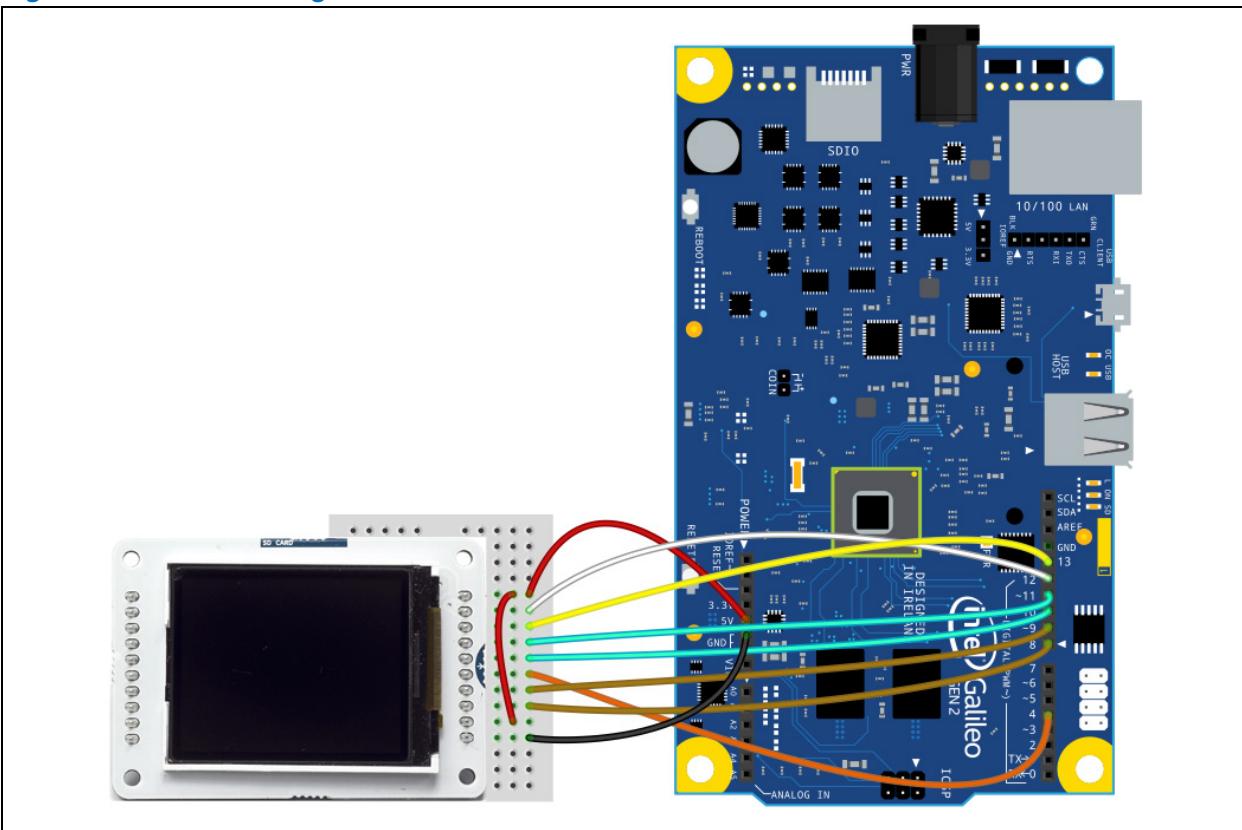


Hardware summary

Key info	Description/links
Operating Voltage	5V
IOREF	No
Use VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release) Not Supported. No TFT Library Available
Edison Board Firmware	1.0.3 (WW37) Not Supported. No TFT Library Available
Schematics	http://arduino.cc/en/uploads/Main/LCD_Rev-4.zip

**Pin connections:**

TFT Board	MC Board
+5V	5V
MISO	pin 12
SCK	pin 13
MOSI	pin 11
LCD CS	pin 10
SD CS	pin 4
D/C	pin 9
RESET	pin 8
BL	+5V
GND	GND

Figure 49 Connecting an Arduino TFT screen to Galileo 2

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

Due to the lack of performance of Galileo SPI, it was not possible to test the display.

In addition, there is no support for an SD card connected to the SPI interface ([see Overview](#)).



Companion library

The library comes with the IDE; however, the latest release of the Arduino IDE was not functional. Obtain a night build IDE that functions.

Compile and upload

Arduino:

The TFT library is included with Arduino IDE 1.0.5 and later. This shield and sketch do not work correctly on the Arduino board when compiled with Arduino IDE "1.0.5-r2". There appear to be horizontal lines in the display. They do work when compiled with IDE "Arduino nightly" build from the primary Arduino Site (tested versions are above).

The following 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 'Sample Sketch' header.)

Sample 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
PIImage logo;

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



Sample Sketch

```
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 accessed, 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
```

**Sample Sketch**

```

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);
}

```

Galileo/Edison:

The TFT library is not included with the Galileo IDE. The wrapper file "avr/pgmspace.h" must be added to allow the library to compile. Due to the lack of performance of the SPI interface, it was not possible to test the module.

Results

G1/G2/Edison Inconclusive. No TFT library available in Galileo/Edison.

Worked with the Arduino board. Not compatible with Galileo.

Next steps

- Find an alternative TFT library

§



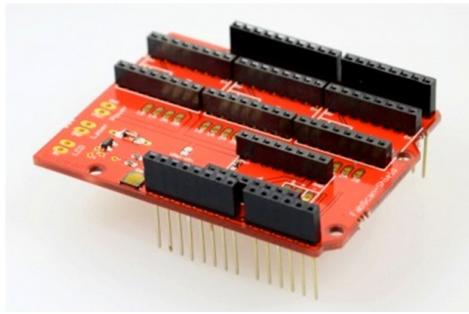
14 FabScan* 3D Scanner

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

Figure 50 FabScan 3D scanner



Hardware summary

Key info	Description/links
Operating Voltage	Logic 5V; motors and laser can operate at voltage from external supply
Use VIN as power source	Yes or external supply if connected
Galileo Firmware	1.0.3 (Release Version)
Edison Firmware	1.0.3 (WW37)
Schematics	https://github.com/watterott/FabScan-Shield/tree/master/pcb

Companion library

N/A

Compile and upload

Two sketches were used to test the shield. This is a test sketch from <https://raw.githubusercontent.com/francisengelmann/FabScan/master/arduino/FabScanArduinoFirmware.pde>. The following sketch was tested without stepper motors since the stepper driver breakout board was not used. Connect the shield to the Galileo board and upload the following sketch. For Edison use the default [PWM](#) settings.

**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);
}
```



Example Sketch

```
pinMode(STEP_PIN_2, OUTPUT);

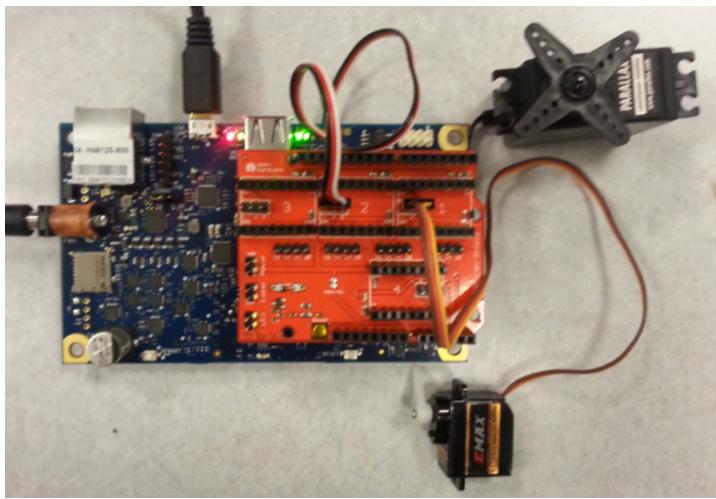
pinMode(ENABLE_PIN_3, OUTPUT);
pinMode(DIR_PIN_3, OUTPUT);
pinMode(STEP_PIN_3, OUTPUT);

//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 Galileo 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 ~5V, since there was no external power supply needed for these motors. The motors used were EMAX* ES08A, Tower Pro* MG995R, Parallax Continous.

Figure 51 **FabScan 3D scanner on Galileo 2**
**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

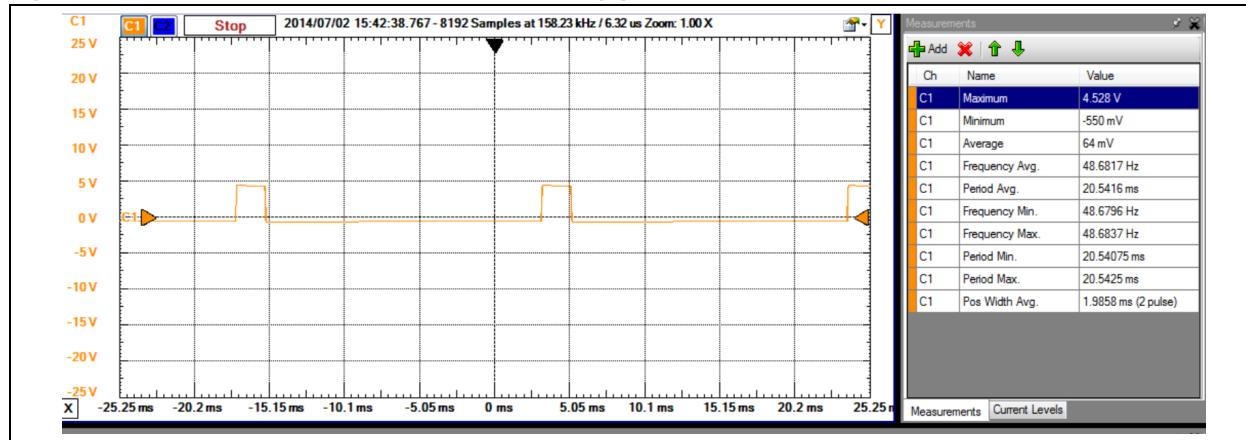
void setup()
{
    myservo1.attach(SERVO1); // attach the servos
    myservo2.attach(SERVO2);
}

void loop()
{
    for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
    {
        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);
        myservo2.write(pos);
        delay(15);
    }
}
```

```
Example Servo Sketch
{
    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
}
}
```

Galileo 2 only: 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 52 Galileo 2: Pulse width and frequency graph



Results

Tested using a sketch without connecting the stepper motor driver and lasers. Another sketch to control servo motors was tested.

G1, Compatible after version 1.0.3 (for perversions, SV1 motors would not turn with the Servo sketch). After version 1.03 the SV1 servo is jumpy due to degree of resolution and heats up. SV4 ran successfully on all versions.

Galileo Gen 2, 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. A servo motor was connected to port three, and the sketch was updated to use pin 12. Servo motors connected to port 3 would not run. There was continuity between the port three control pin and pin 12. According to the BOM of some FabScan kits, it would be possible to create a 3D scanner with one stepper motor.

Edison, Compatible. SV1 (low cost servo) did not function, SV5 and SV7 worked as expected. Low cost servos tend to be problematic.

Next steps

- Connect hardware for stepper motor and laser if possible. This would require the stepper motor control breakout boards.



15 RedBearLabs* Nordic BLE Shield

Use case

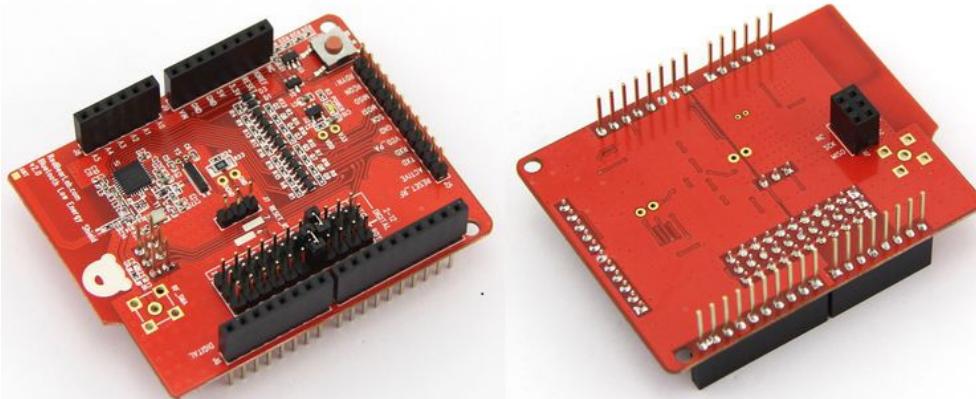
The BLE (Bluetooth* Low Energy) shield is designed to work with Arduino* boards and allows you to connect your Arduino board with other BLE Central devices, like smartphones and tablets. The BLE shield can operate with 3.3V or 5V; therefore, it should work with a lot of Arduino-compatible boards. This 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 Galileo board.

Key info	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 and BLE shield together allows you to:

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

Figure 53 RedBearLabs Nordic BLE shield





Hardware summary

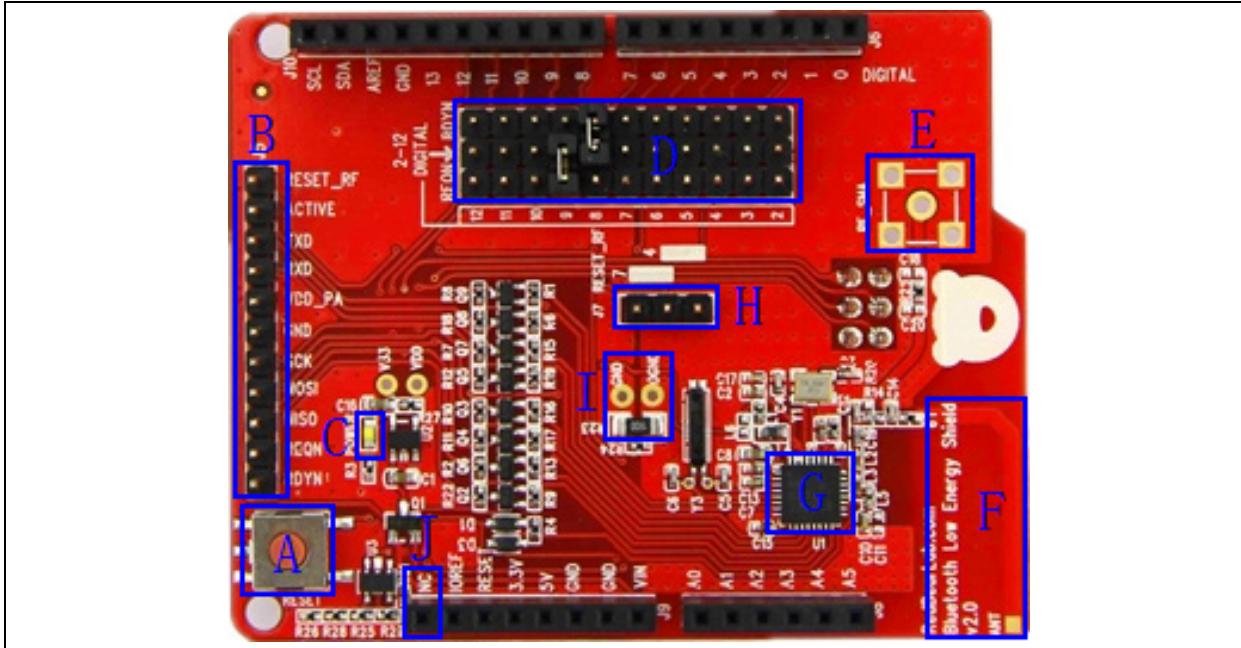
Key info	Description/links
Operating Voltage	Designed for 3.3V and 5V
IOREF	Present but not used
Use VIN as power source	No
Power	5V or VIN, automatically selected.
Galileo Firmware	1.0.3 (Production Release)
Edison Firmware	1.0.3 (WW37)
Schematics	https://github.com/RedBearLab/BLEShield/commit/306dd34feef285cdebdb75dd9724e8decc9a66c9
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 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	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 54 RedBearLabs Nordic BLE shield layout



Companion library

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

Arduino:

The exact procedure to get this running on Arduino is a little confusing. Some sites reference outdated libraries called 'BLEShield' and others reference 'nRF8001'. The "BLE" library works as is on the Arduino. 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 Arduino.

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

**Galileo/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.

```
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 Galileo. Edit the "hal_platform.h" file that is part of the library. There are board-dependent definitions, but not one for Galileo. 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.

```
Modification for hal_platform.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.

```
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>
```

In Arduino, 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 Galileo (since the processor is not Atmel). Galileo sets a '#define' for the keyword that enables a successful compile, but it will not work. Arduino gives very little space for program space and data; however, Galileo 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.).



```
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(""));
```

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.

Compile and upload

Arduino:

Compiling on Arduino 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.

Galileo/Edison:

An attempt to make this board functional on Arduino 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. 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 Galileo.

Results

G1, G2, Edison, Inconclusive.

Next steps

- Look for a less complicated example.
- Work on fixing all of the Serial/Serial1 notations.

§



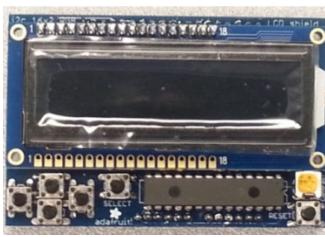
16 Adafruit* I2C RGB LCD Shield Kit with 16 × 2 Display

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 Galileo. 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 . dated 091713
Guide	https://learn.adafruit.com/rgb-lcd-shield/using-the-rgb-lcd-shield

Figure 55 Adafruit* I2C RGB LCD shield kit with 16 × 2 display



Hardware summary

Pin name	Function
VIN	No
Operating voltage	5V
Power for logic	5V
Schematic	https://github.com/adafruit/Adafruit-RGB-LCD-shield
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)

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: <code>#define MCP23017_ADDRESS 0x20</code>
GND	Ground
Power	5 V

Figure 56 Adafruit® I2C RGB LCD shield kit with 16x2 display on Galileo 2



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 Arduino® board. The demo will compile on Galileo with a couple of changes.

This site states to handing the including of avr/pgmspace.h differently (<https://communities.intel.com/thread/46076>), however, simply uncommenting it seems to be acceptable.

Change the following files as follows:

File: Adafruit_RGBLCDShield.cpp	
Before	After
<pre>#include "Adafruit_RGBLCDShield.h" #include <stdio.h> #include <string.h> #include <inttypes.h> #include <Wire.h> #ifndef __AVR__ #define WIRE Wire #else // Arduino Due #define WIRE Wire1 #endif</pre>	<pre>#include "Adafruit_RGBLCDShield.h" #include <stdio.h> #include <string.h> #include <inttypes.h> #include <Wire.h> #ifndef __AVR__ #define WIRE Wire #else // Arduino Due // #define WIRE Wire #endif</pre>

File: Adafruit_MCP23017.cpp	
Before	After
<pre>#include <avr/pgmspace.h> #include "Adafruit_MCP23017.h" #ifndef __AVR__ #define WIRE Wire #else // Arduino Due #define WIRE Wire1 #endif</pre>	<pre>//#include <avr/pgmspace.h> #include "Adafruit_MCP23017.h" #ifndef __AVR__ #define WIRE Wire #else // Arduino Due // #define WIRE Wire #endif</pre>

Compile and upload

The "hello world" demo compiles with the changes stated above.

- Plug the board directly onto the Galileo.
- For Galileo 1, move the J2 (SPI) jumper to the second and third pins (see [I2C Overview](#)). Galileo 2 and Edison do not have SPI jumpers.

Figure 57 Galileo 1 – SPI (J2) jumper



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

```
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!");
}
```



```

Sketch: 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);
        }
    }
}

```

Results

G1/G2/Edison Compatible.

For Galileo 1, moving the J2 jumper, as described above, enabled the shield to function.

Next steps

- None

§

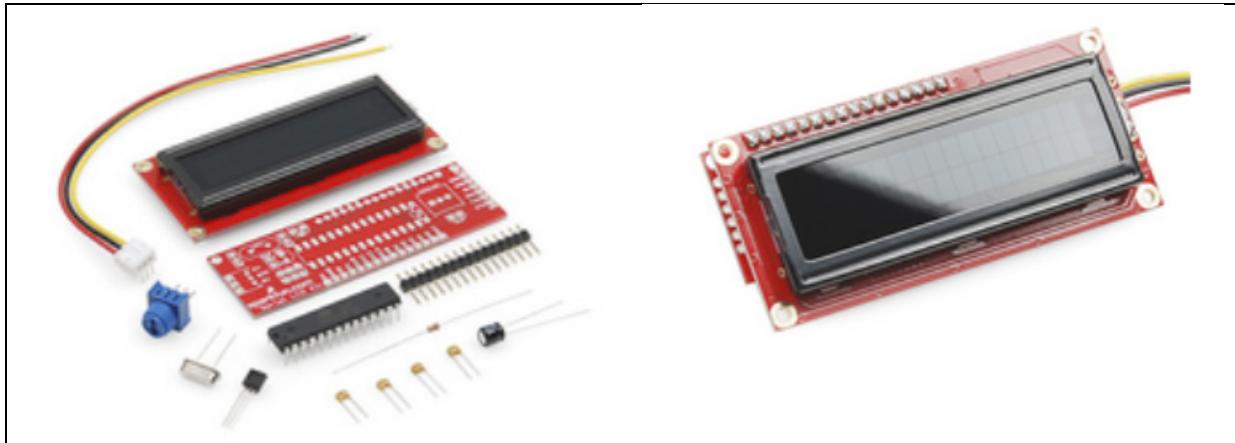
17 SparkFun* Serial-Enabled LCD Shield

Use case

This breakout board allows you to write on an LCD screen using only one pin of the Galileo. 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 58 SparkFun serial-enabled LCD shield



Hardware summary

Pin name	Function
VIN	No
Operating voltage	5V
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)

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 (5V 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 Galileo GND
5V	Power, connect to Galileo 5V
Rxl	Serial port Receiver pin, connected to the G1/G2 Tx pin (D1) Since Galileo 2 has another serial pin, connect to Tx pin (D3).
TxO	Serial port Transmit pin (not used)
DTR	Reset (not used)

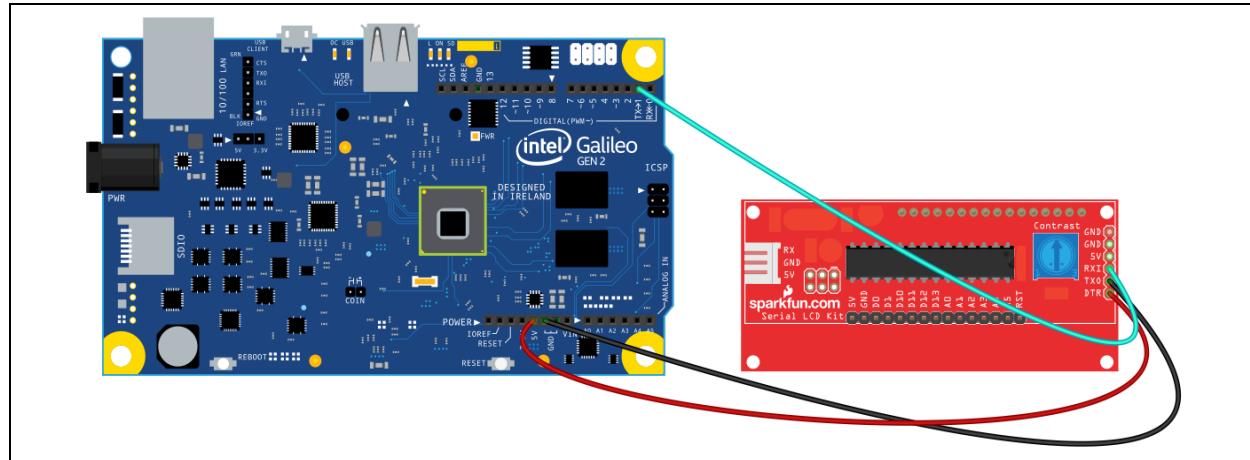
Companion library

No library required.

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 5V, GND and Rx to the Galileo board. Rx is connected to Tx (D1). The only thing to be changed is the 'lcd' declaration that uses the SoftwareSerial library for Arduino*. For the Galileo, the hardware serial is used.

Figure 59 Connecting a SparkFun® serial-enabled LCD shield



The sample code's serial port was changed as follows:

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

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
```

**Test Sketch**

```
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
    setLCD.Cursor(2); // Set cursor to the 3rd spot, 1st line
    lcd.print("Hello, world");
    setLCD.Cursor(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
        setLCD.Cursor(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);
        setLCD.Cursor(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
}
```



```
Test Sketch
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
    }
}
```



```
Test Sketch
    }
}
}

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.

Results

G1/G2/Edison Compatible.

Next steps

- Try to use the serial LCD with other shields that need an output.
- Try to use the temperature sensor and determine if it can interact with the Galileo.

§

18 SeeedStudio® CAN Bus Shield

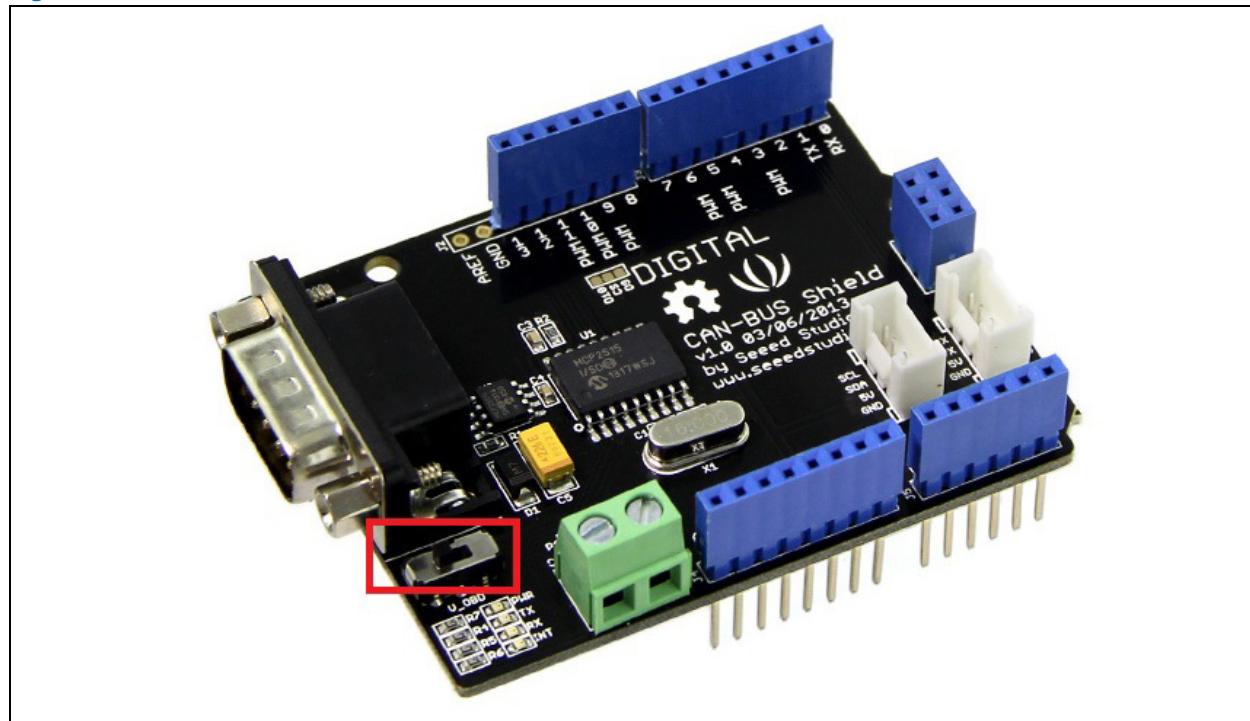
Note: This is on the "List of Supported Shields" (Link).

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® Due 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
Guide	http://upverter.com/seeedstudio/c71b8e78aef6dce5/CAN-BUS-Shield--v10/

Figure 60 SeeedStudio CAN bus shield



Hardware summary

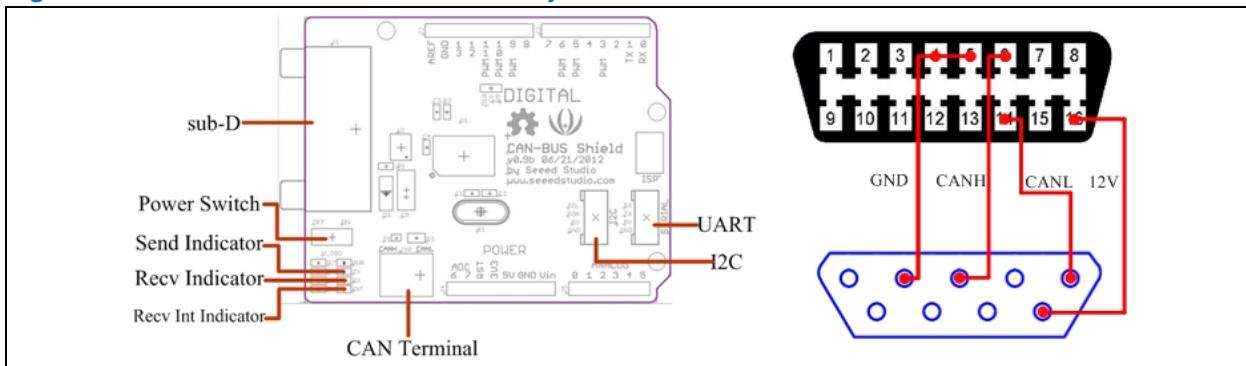
Key info	Description/links
Operating Voltage	Designed for 5 V
IOREF	Present but not used.
Use VIN as power source	No
Galileo Board Firmware	1.0.2 (Release Version) 1.0.3 Not tested. The shield is broken, therefore, could not be tested.
Edison Board Firmware	Not Tested. The shield is broken, therefore, could not be tested.
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).

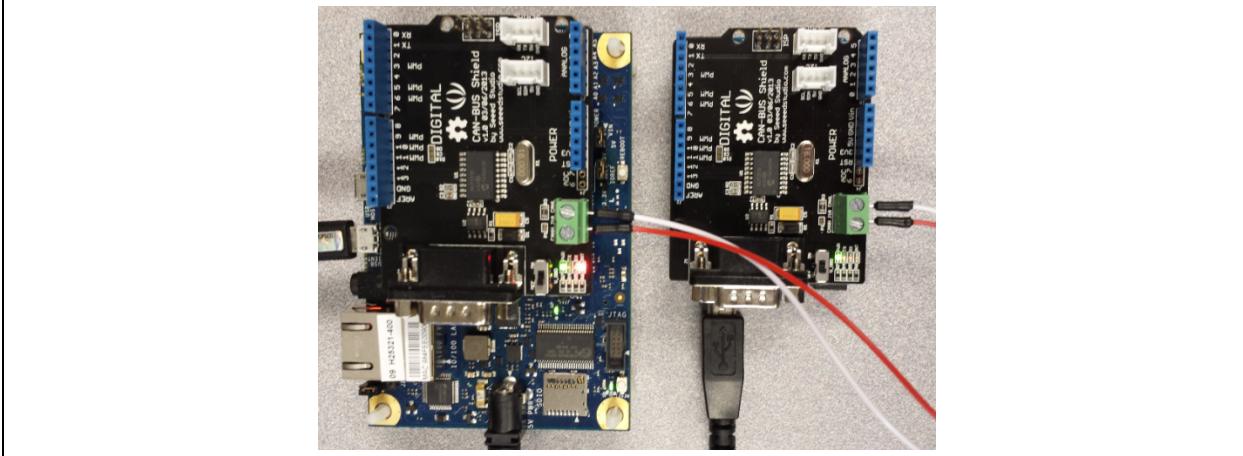
There are four LEDs: one indicating the presence of the power supply (PWR); two for the communication (Rx and Tx); the fourth LED is connected to the interrupt pin and indicates data is being received by the shield.

Figure 61 SeeedStudio 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 62 SeeedStudio CAN bus shield on Galileo 1 and Arduino



Companion library

The library uses the Arduino Core SPI library. It compiled without modification on both the Arduino and Galileo.

Compile and upload

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

Arduino to Arduino 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
01          2          3          4          5          6          7
```

Galileo “send” to Arduino “receive” Test:

One Galileo and one Arduino board communicating with each other via CAN bus. The Galileo board had the 'send' sketch loaded and the Arduino board had the receive sketch loaded.

The data shown above is displayed twice on the Arduino's serial monitor and then nothing else appears. The Tx and Rx LEDs on the Galileo "send shield" flash repeatedly. The Rx LED on the Arduino "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.

Galileo “receive” to Arduino “send” Test:

One Galileo and one Arduino board communicating with each other via CAN bus. The Galileo board had the receive sketch loaded and the Arduino board had the send sketch loaded. Setup messages appear on the Galileo's serial monitor and then nothing else. The Tx and Rx LEDs on the Arduino "send shield" flash repeatedly. The Rx LED on the Galileo "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 Galileo SPI library, attachInterrupt is rewritten with the following line:

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

In the Galileo Board User Guide, it is noted that Galileo SPI cannot act as a slave.

Note: The board has a native SPI controller; therefore, it will act as a master and not as an SPI slave. Therefore, it cannot be a SPI slave to another SPI master. It can act, however, as a slave device via the USB Client connector." <https://communities.intel.com/docs/DOC-22475>

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

Change receive.ino as follows:	
Before	After
<pre>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(n++); } }</pre>	<pre>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++); } }</pre>

Results

G1/G2 Compatible.

Galileo 1 sending is on par with Arduino; however, Galileo receives messages much more slowly. Galileo receives at a rate of about 200 messages per second, and Arduino receives at a rate of about 26,000 messages per second.

Galileo Gen 2 receives messages at a rate of about 1700 messages per second.

Next steps

- Scope the SPI and Interrupt pins on Galileo and compare them to the same pins on the Arduino board.

§

19 Arduino* Wi-Fi Shield

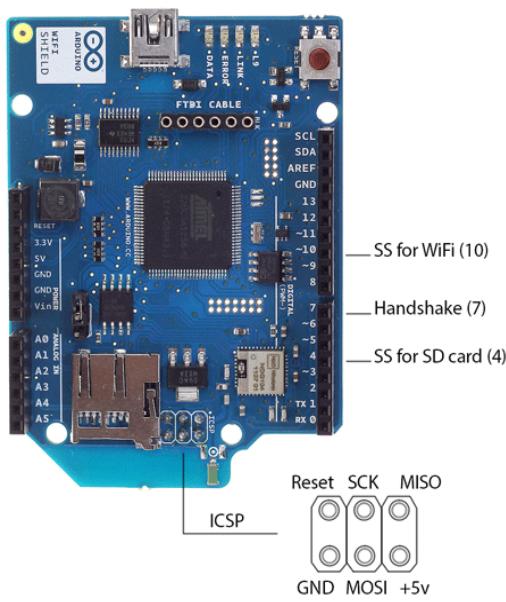
Note: Although this is on the “List of Supported Shields” ([Link](#)), we cannot verify it as compatible.

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. Galileo has the 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 Galileo 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 63 Arduino Wi-Fi shield



Hardware summary

Key info	Description/links
Operating Voltage	Designed for 5V
IOREF	Used, the shield can operate either at 3.3V and 5V.
Use VIN as power source	No.
Power	5V or VIN, automatically selected.
Galileo Firmware	1.0.3 (Production Release) Not Supported



Edison Firmware	1.0.3 (WW37) Not Supported
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
Wi-Fi 802.11b	HDG204

The AVR32UC3 and the HDG204 both work with 3.3V 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 Galileo 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.

Companion library

The Galileo library for the Wi-Fi miniPCI module has the same name and the same APIs of the official Arduino library. This makes it impossible to use the Arduino Wi-Fi shield library without conflicts.

There was an attempt to replace the Galileo version of the Wi-Fi library with the Arduino version. The code compiled, but it was not functional.

Compile and upload

This shield works with no problems on Arduino, but not on the Galileo. See [Overview Section](#) for information on Wi-Fi and SD cards on a shield.

Results

G1/G2/Edison Not compatible. See [SD Card Overview](#) and [Wi-Fi Overview](#) sections. An alternative is the [XBee Wifi Module](#).

Next steps

- Other reference documents stated to replace the Galileo Wi-Fi libraries with the 'Vanilla' version of Arduino from GitHub (<https://github.com/arduino/Arduino/tree/ide-1.5.x/libraries/WiFi>). This procedure could not be reproduced; therefore, this is categorized as 'Not compatible.'
- Update the shield firmware. This is recommended in the guide, since the shield firmware has changed since Arduino IDE 1.0.4.

§

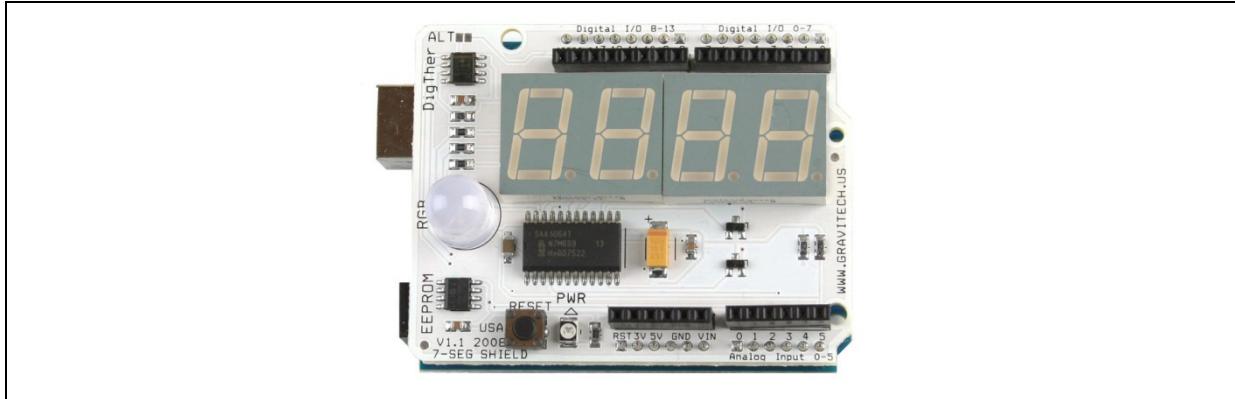
20 Gravitech* 7-Segment Shield

Use case

This shield is ideal for Galileo and Arduino* experimenting, temperature monitoring, etc. It is ideal for I2C basic electronic practice because the example code utilizes no libraries. There are also three I2C 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/SCHILD7/SEG7_SHIELD.zip This code required modification.
Guide	http://tronixstuff.com/2011/08/24/review-gravitech-7-segment-arduino-shield

Figure 64 Gravitech* 7-segment shield



Features of the board include:

- I2C four-digit, 7-segment driver (10 mm diameter)
- I2C temperature sensor
- I2C EEPROM
- PWM RGB LED
- Blue PWR LED
- Reset button
- All pins breakout



Hardware summary

Pin name	Function
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
VIN	Not present.
IOREF	Not present.
Power	5V only.
Schematics	http://site.gravitech.us/Arduno/SCHILD7/7SEG-SCHILD_schematic_v1_1.pdf

The shield mounts a digital temperature sensor (TMP75), an I2C EEPROM (24LC128), 4-digit display I2C controller, and RGB LED. The shield does not follow the latest Arduino 1.0 pinout, so it does not have the IOREF and the dedicated pins for the I2C interface. Mechanically, this is not a stackable shield. On the shield, there are 4.7 kohm pullup resistors for the I2C bus. Although the temperature sensor is functional, the heat of the shield and the Galileo interferes with the temperature readings.

Pin name	I2C Function (No wires required)
A4	I2C, SDA (Serial Data Line) Data
A5	I2C, SCL (Serial Clock Line) Clock
Device Address 1	I2C, defines 7-SEG address as: #define _7SEG (0x38)
Device Address 2	I2C, defines digital thermometer as: #define THERM (0x49)
Device Address 3	I2C, defines EEPROM address as: #define EEP (0x50)

Shield worked with the Galileo I2C jumper ([see I2C Jumper Overview](#)) in default and non-default position.

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

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 RGB 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.



```

Sketch: 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()

```



Sketch: Companion sketch modified as noted above

```
{  
    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();  
        }  
    }  
}
```



```

Sketch: Companion sketch modified as noted above
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 */
    }
}

```



```
Sketch: Companion sketch modified as noted above
*****
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--;
    }
}
```



Sketch: Companion sketch modified as noted above

```

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);
    }
}

```



Sketch: Companion sketch modified as noted above

```
}
```

```
}

***** 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");
}
```

Results

G1/G2 Compatible.

Edison Partially Compatible. Unable to write to the 7-segment display.

Next steps

- Test the EEPROM example in the [shield guide](#).

§

21 Adafruit* Data Logging Shield for Arduino*

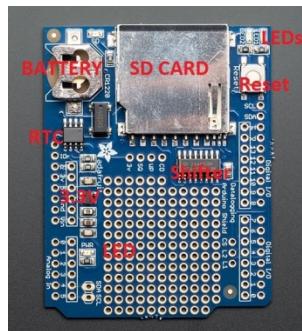
Note: This shield is on the "List of Supported Shields" ([Link](#)).

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 3.3V 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 or Galileo 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 Downloaded from GitHub 07/21/14
SD Library	None.

Figure 65 Adafruit data logging shield for Arduino



Hardware summary

Key info	Description/links
Operating Voltage	Designed for 5V.
IOREF	Present but not used.
Use VIN as power source	No.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Schematics	N/A
RTC	DS1307 Note: If the battery is not present, strange behavior will occur and a possibility of hanging the Arduino 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	I2C Function (No wires required)
A4	I2C, SDA (Serial Data Line) Data
A5	I2C, SCL (Serial Clock Line) Clock
Device Address	I2C, RTCLib.cpp defines the address as: <code>#define DS1307 ADDRESS 0x68</code>

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

On the Galileo, at this time there is no support for SD cards that are present on a shield ([see Overview Section](#)). Attempt to instantiate another object with a chip select (CS) value for the shield had no effect. The scope determines the new CS was not firing. Found that Galileo had replaced the SD library and the Galileo version of the library ignored the CS value.

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: <code>SD.begin(10)</code>
D10	Must be left as an output (even if it is not used) in order for the SD to work on Arduino.

Companion library

The RTC library requires a change for Galileo as follows:

RTClib.cpp Before	After
<pre>// Code by JeeLabs http://news.jeelabs.org/code/ // Released to the public domain! Enjoy! #include <Wire.h> #include "RTCLib.h" #ifdef __AVR__ #include <avr/pgmspace.h> #define WIRE Wire #else #define PROGMEM #define pgm_read_byte(addr) (*(const unsigned char *) (addr)) #define WIRE Wire #endif #define memcpy_P(dest, src, num) #endif</pre>	<pre>// Code by JeeLabs http://news.jeelabs.org/code/ // Released to the public domain! Enjoy! #include <Wire.h> #include "RTCLib.h" #ifndef __AVR__ #include <avr/pgmspace.h> #define WIRE Wire #else #define PROGMEM #define pgm_read_byte(addr) (*(const unsigned char *) (addr)) #define WIRE Wire #endif #define memcpy_P(dest, src, num) memcpy((dest), (src), (num))</pre>



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:

- Changing the **SPI** 'Wire1' references to 'Wire' in all the sketches. This change is required since the library itself was changed. If not done, compile errors will occur.
- The **serial port** baud rate to 9600. This is the Galileo/Arduino 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.
- Remove '**F()**' macros that uses PROGMEM. This converts it back to standard memory.
- It is suggested to use the [Galileo OnBoard-Clock](#) with a 3.3V battery to replace this board completely.

The outputs for a few selected sketches are:

Sketch Output: DateCalc
<pre>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</pre>

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.

Sketch Output: DS1307
<pre>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</pre>

Sketch Output: SoftRTC
<pre>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</pre>



Compile and upload

Real-Time Clock Test:

On Arduino, all examples in the RTC library worked with minor changes cited previously. See examples in the previous section.

On Galileo, the change required is changing the 'Wire1' object to 'Wire'.

SD Card Test:

There is no special library for the SD card (they are included with the Arduino/Galileo IDE).

The standard Galileo/Arduino SD Library includes the following examples:

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

The output below are from a couple of examples, however, the difference is that the SD card in the Galileo is being accessed (not the SD card from the shield).

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.gz 1441609 image-full-clanton.ext3 314572800 done!

Results

RTC: G1/G2/Edison Alternative, use the [Hardware Clock on Galileo](#).

SD Card: G1/G2/Edison Alternative, use the [SD Card on Galileo](#).

Overall: G1/G2/Edison, compatible for RTC but not compatible for SD Card.

Next steps

- None

§



22 LinkSprite* 3G + GPS Shield for Arduino*

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 Galileo board for data or files management. The 3G module supports using AT commands.

(See http://en.wikipedia.org/wiki/AT_command_set.)

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.

Figure 66 3G + GPS shield for Arduino





Hardware summary

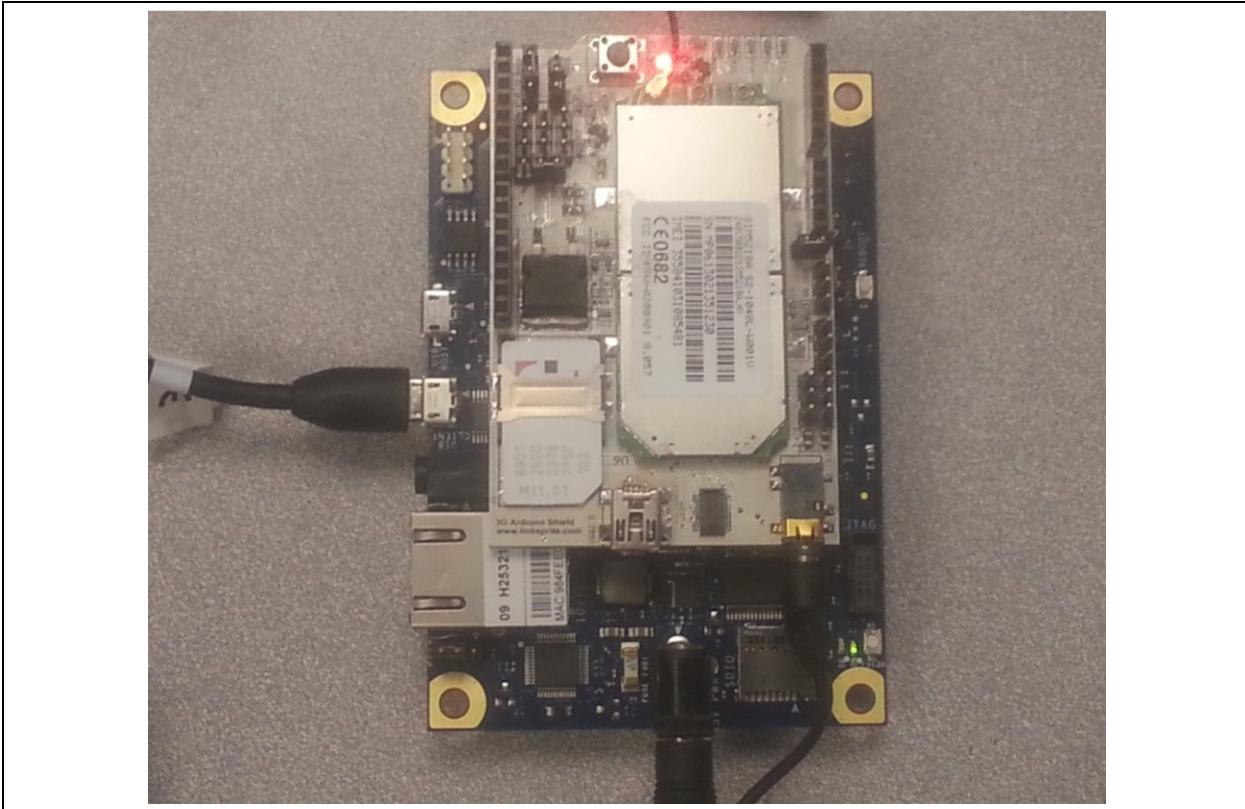
Key info	Description/links
Operating Voltage	5V
VIN as power source	No.
Audio Interface	2-in-1 audio connector for stereo headset.
SIM	Mini SIM card interface.
GSM module	SIMCOM® SIM5218 – quad band GSM/GPRS/EDGE and UMTS engine.
Antenna	Connection port for external antenna.
Galileo Board Firmware	1.0.2 (Release Version)
Edison Board Firmware	1.0.3 (WW31) Board did not power up properly, under investigation.
Schematics	http://www.cooking-hacks.com/index.php/documentation/tutorials/arduino-3g-gprs-gsm-gps

Companion library

No library required.

Compile and upload

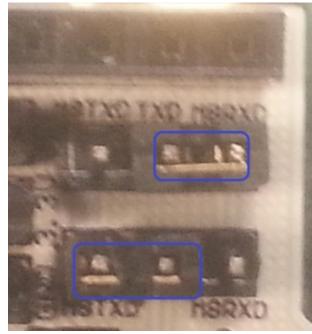
Figure 67 3G + GPS shield for Arduino on Galileo 1



Test the GSM capability.

- Insert unlocked SIM (mini SIM) card
- Connect cellular antenna with connection to the main antenna port
- Use two jumpers to set the serial settings as shown in Figure 68.

Figure 68 3G + GPS shield jumpers

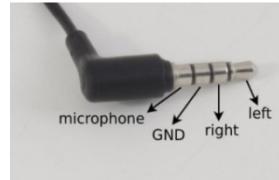


Do the following:

- Connect a 2-in-1 stereo headset to the headset connector.
- Attach the shield to the Galileo.
- Power up the Galileo.
- Connect Galileo USB to computer.

- Power up the shield by pressing the power button on the shield for two seconds.
- 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.


```
Serial1.println("AT+CMGS=\\"15555555555\\\"");  
Serial1.println("ATD + 15555555555;");
```
- Upload the following sketch that will enable sending text messages and making calls.
- Open the serial terminal from the IDE.
- 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.)





```
Sketch: Test voice calls in/out, text in/out, IP session
#include <String.h>

char data[256];

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

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;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}

void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
    mode
    delay(1000);
    Serial1.println("AT+CMGS=\"13182181167\"");
    delay(1000);
    Serial1.println("Hello from Galileo?");
```



```
Sketch: Test voice calls in/out, text in/out, IP session
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\"");
    delay(1000);
    Serial1.println("AT+CMGL=\"ALL\"");
}
void DialVoiceCall()
{
    Serial1.println("ATD + 13182181167");
    delay(100);
    Serial1.println();
}
void ShowSerialData()
{
    while(Serial1.available()!=0)
        Serial.write(Serial1.read());
}
void GPRS()
{
    Serial1.println("AT+CPIN?");
    delay(1000);
    ShowSerialData();

    Serial1.println("AT+CGREG?");
    delay(1000);
    ShowSerialData();

    Serial1.println("AT+COPS?");
    delay(1000);
    ShowSerialData();

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

    Serial1.println("AT+cgatt=1");
    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();
```



```
Sketch: Test voice calls in/out, text in/out, IP session
// 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();
}
```

Results

G1/G2 Compatible.

Edison Not Compatible. Shield will not power up, under investigation.

Inbound/outbound voice calls and texting were successful. It appears the GPS functionality should be supported by Galileo.

Next steps

- Get better audio quality by acquiring a better headset, moving further away to prevent feedback, using a different audio connection to the shield, or finding out if there are some parameters that can be set.
- Find a GPS antenna so the GPS functionality can be tested. After trying to run some GPS setup commands, it appears that GPS functionality is supported.

§

23 LinkSprite* ATWIN Quad-band GPRS/GSM Shield for Arduino*

Note: This is on the "List of Supported Shields" ([Link](#)).

Use case

This shield makes it possible to hook the Galileo 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. (See http://en.wikipedia.org/wiki/AT_command_set)

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. The shield supports AT commands through the serial interface.

Figure 69 LinkSprite ATWIN quad-band GPRS/GSM shield for Arduino

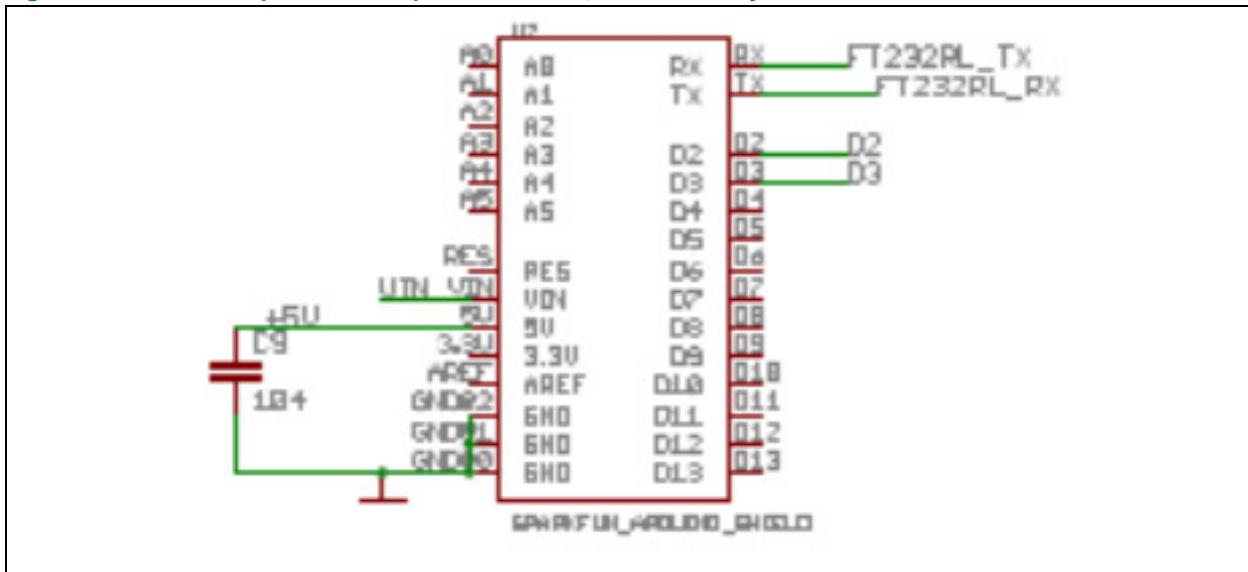




Hardware summary

Key info	Description/links
Operating Voltage	3.3V or 5V – jumper selectable.
IOREF	Present but not used.
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.
Galileo Board Firmware	1.0.2 (Release Version)
Edison Board Firmware	1.0.3 (WW31) Board did not power up properly, under investigation.
Schematics	https://s3.amazonaws.com/linksprite/Shields/ATWIN/schematic_of_ATWIN.rar

Figure 70 LinkSprite ATWIN quadband GPRS/GSM shield layout



Pin name	Function
Rx of hardware serial port	Input from Galileo - connected to D0.
Tx of hardware serial port	Output to Galileo - connected to D1.
5V	Connected to 5V.
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.



Companion library

No library required.

Compile and upload

Do the following:

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



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



- Set J2 so that Tx is connected to MRx.



- Power up the Galileo and connect to computer with the USB.
- 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.

```
Serial1.println("AT+CMGS=\\"15555555555\\\"");  
Serial1.println("ATD + 15555555555;");
```

- Upload the following sketch that will enable sending text messages and making calls.
- Open the serial terminal from the IDE and set rate to 9600.
- 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.)

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

```
#include <String.h>

char data[256];

void setup()
{
    Serial1.begin(9600);
    Serial.begin(9600);      // the GPRS baud rate
    delay(1000);
}

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;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}

void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
    mode
    delay(1000);
    Serial1.println("AT+CMGS=\"15555555555\"");
    delay(1000);
    Serial1.println("Hello from Galileo?");
    delay(1000);
    Serial1.println((char)26); //the ASCII code of the ctrl+z is 26 (0x1A)
    delay(1000);
}
```



```
Sketch: Issue AT commands over serial link
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.println("ATD + 15555555555;");//dial the number
    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?");
}
```



Sketch: Issue AT commands over serial link

```
delay(3000);
ShowSerialData();

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

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

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

// detach from GPRS newtork
Serial.println("AT+CGATT=0");
delay(1000);
ShowSerialData();
}
```

Results

G1/G2 Compatible.

Edison Not Compatible. Shield will not power up, under investigation.

Next steps

Test more with a data application.

§

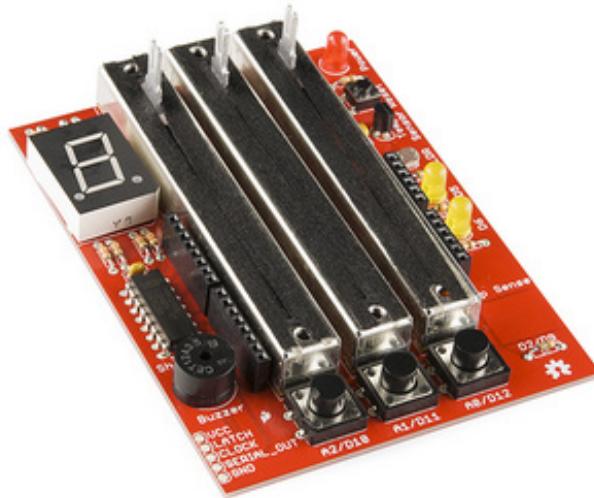
24 SparkFun* Danger Shield

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 71 Danger Shield



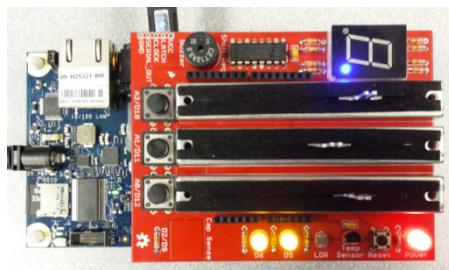
The shield's features include:

- Three linear slide potentiometers connected to the Arduino* 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 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.

Hardware summary

Key info	Description/links
Operating Voltage	5V
VIN as power source	No.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
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 72 Danger shield on a Galileo 1



Compile and upload

Arduino:

The 'Danger_shield' sketch was run on the Arduino. 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.

Galileo/Edison:

The 'Danger_shield' sketch was modified to function with the Galileo. 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.



```
Modified 'Danger sheield' Sketch:  
//RM4GALILEO, #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,B000001  
11,B01111111,B01101111  
};  
  
// 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
```



Modified 'Danger sheield' Sketch:

```
#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

//RM4GALILEO, 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);
    }

    //RM4GALILEO, 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
    {
```



```
Modified 'Danger sheield' Sketch:  
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);  
}
```



Modified 'Danger sheield' Sketch:

```
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
```



```

Modified 'Danger sheield' Sketch:
*   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);
    }
}

```

Results

G1/G2/Edison Compatible.

No capability for CapSense due to AVR calls.

Next steps

- The CapSense library seems to have many AVR related functions and would need to be reworked for Galileo/Edison.

§

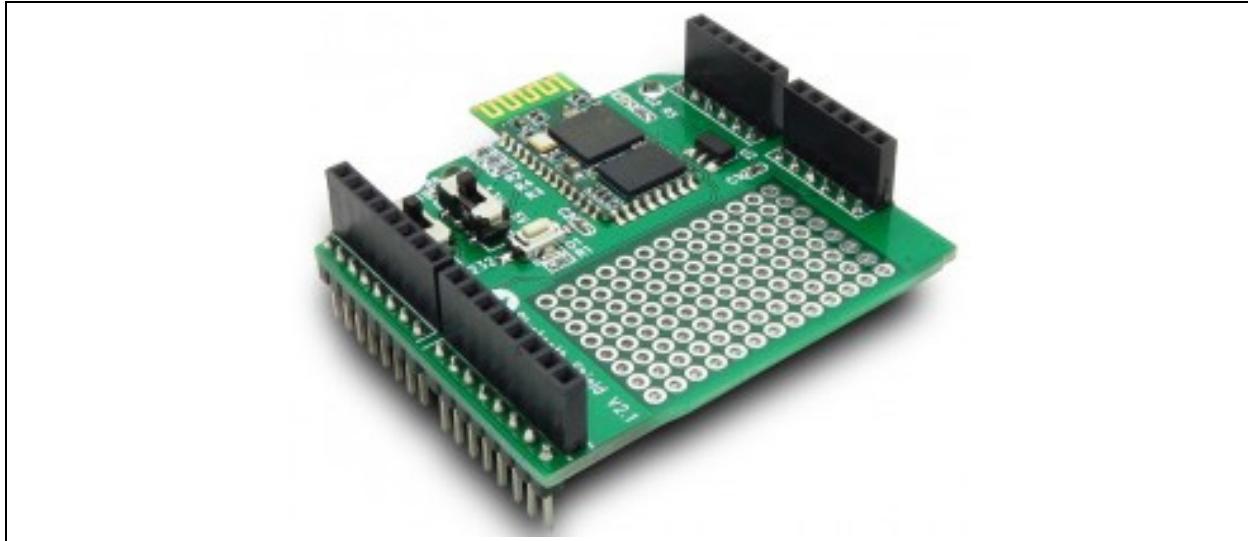
25 ITEAD* Bluetooth* Shield (Slave)

Use case

This BT shield (slave) is an HC-06 serial port Bluetooth module breakout board for Arduino*. You can directly use this BT shield with the Arduino 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=mobi.dzs.android.BluetoothSPP 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 is independent from the IDE. In this case, we are using an Android phone.

Figure 73 ITEAD* Bluetooth* shield (Slave)



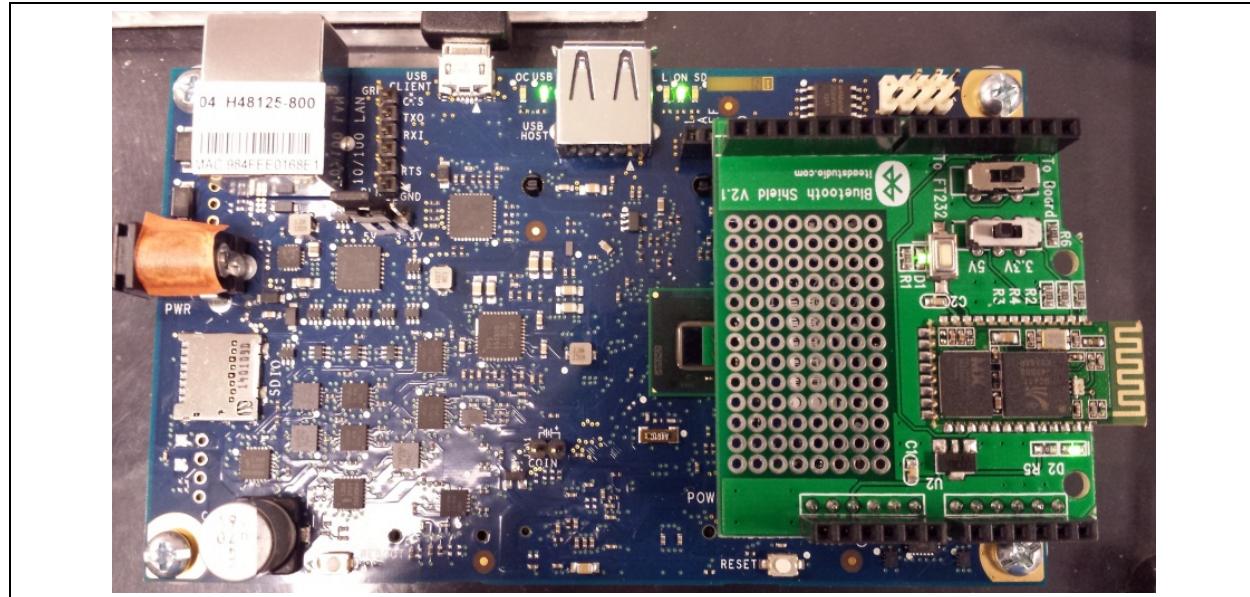
- Arduino UNO compatible
- Up to 10 m communication distance in house without obstacle
- UART interface (TTL) with programmable baud rate (SPP firmware installed)
- Default baud rate: 9600, data bits: 8, stop bit: 1, Parity: No parity
- Default PINCODE: "1234"
- A full set of configuration commands
- Onboard PCB antenna
- FCC ID certified

Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
IOREF	No
Use VIN as power source	No
Operating voltage	3.3V to 5V
Datasheet	DS_IM120417006_BT_Shield_Slave.pdf

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

Figure 74 ITEAD® Bluetooth® shield on Galileo 2



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 5V for the duration of this test. No sketch will download in 3.3V mode for Arduino.
- 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.
3.3V / 5V	Set to '5V' When set to '3.3V', 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.

Sketch: Led

```
// This code demonstrates how to run the shield on Arduino and Galileo.
// 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; // Galileo, /dev/ttyGS0, Tx pin
TTYUARTClass*   gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO, Rx pin
bool            gGalileo      = true;
void setup()
{
    pinMode(13,OUTPUT);
    gSerialStdPtr->begin(9600);
    gSerialTwoPtr->begin(9600);
    waitForUser(9);
}
void loop()
{
    char fromBT;
    if (gSerialTwoPtr->available() > 0 ) // From BT
    {
        // Read from BT
        fromBT = gSerialTwoPtr->read();
        // Process data
        if(fromBT == '1')
            digitalWrite(13,HIGH);
        else if(fromBT == '0')
            digitalWrite(13,LOW);

        // Echo back char received from BT
        if(gGalileo == true)
        {
            gSerialStdPtr->print(fromBT); // To IDE
            gSerialTwoPtr->print(fromBT); // To BT, echo
        }
    }
}
```



```
Sketch: Led
    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
    }
}
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("");
}
```

- 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.

Galileo usage is slightly different compared to Arduino because the UART on pins 0 and 1 is a different serial port than the one, which communicates with the IDE serial monitor. The UART on the Arduino communicates on digital pins 0 (Rx) and 1 (Tx) as well as with the computer via USB.

Results

G1/G2/Edison Compatible with microcontroller jumpered to 5V and 3.3V.

Next steps

- Test with the [ITEAD Bluetooth Master Shield](#).

§

26 SparkFun* MP3 Player Shield

Use case

MP3 files can be pulled from a microSD card and played using only one shield, effectively turning any Arduino* 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 75 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 based and complex. There is another guide that focused on getting this shield working on Galileo; however, it does not use any libraries.
Sample files	http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/MP3_Player_Files.zip These sample files come from the Sparkfun tutorial. They are called "track001.mp3" and "track002.mp3".

Hardware summary

Key info	Description/links
Operating Voltage	Designed for 5V. Should work at 3.3V.
IOREF	Present but not used.
Use VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematics	http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/MP3%20Shield-v13.pdf

Figure 76 SparkFun MP3 player shield on Galileo 1



On the Galileo, at this time there is no support for SD cards that are present on a shield; therefore, the files are read from the SDCard on the Galileo board.

We modified the “Galileo-based 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. If your SD card has Galileo 2 boot files, the files can still be copied to that SD Card.

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

```

Sketch: Sample Sequencer
#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

```



Sketch: Sample Sequencer

```
//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_HDATA0 0x08
#define SCI_HDATA1 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
```



```

Sketch: Sample Sequencer
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);

```



Sketch: Sample Sequencer

```
//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.
```



```

Sketch: Sample Sequencer
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
}

```



```
Sketch: Sample Sequencer
    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
```



```
Sketch: Sample Sequencer

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.

Results

G1/G2/Edison Compatible.

SPI performance seems to be dependent on the SPI.transfer() and SPI.transferBuffer() functions. Adjusting the arguments of these functions can make a difference. Increasing CHUNK_SIZE directly increases the transfer buffer. In the “Galileo-based 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.

Next steps

- Look further into SPI performance and workarounds for SPI transfer.





27 Mayhew Labs* Mux Shield

Note: This is on the "List of Supported Shields" ([Link](#)).

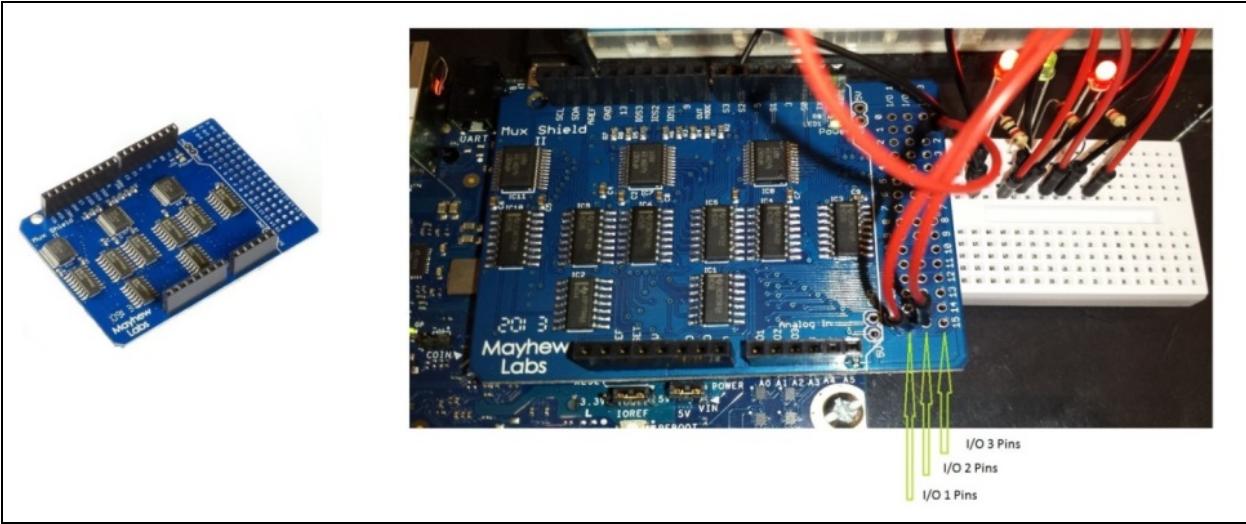
Use case

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

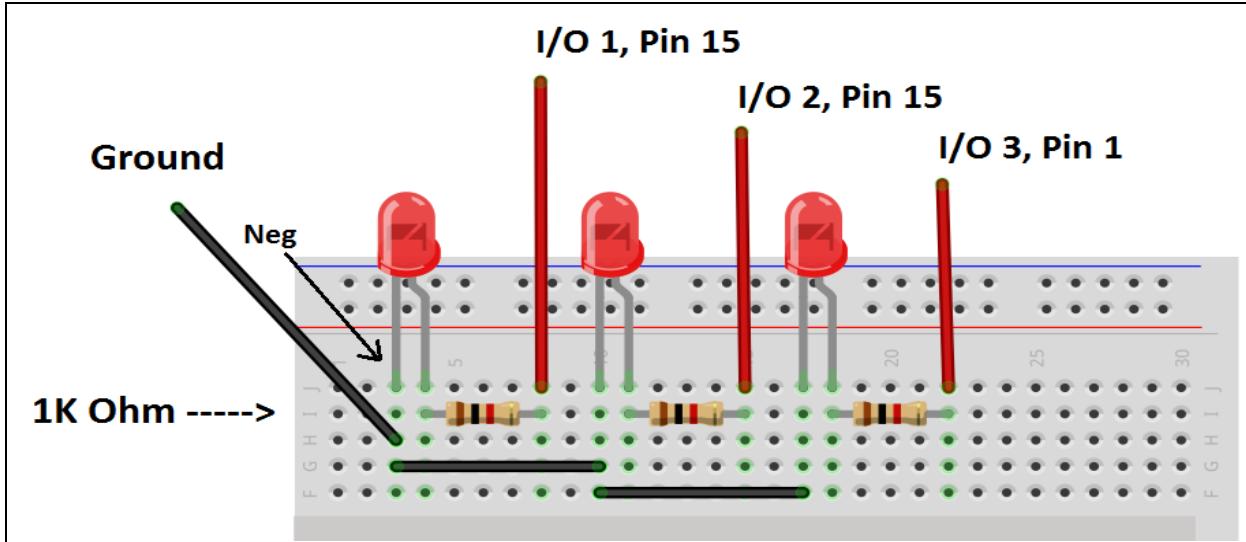
Hardware summary

Key info	Description/links
Operating Voltage	Designed for 5V
IOREF	No (present but not used).
Use VIN as power source	No.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Shift Registers	74HC595
Analog Multiplexers	74HC4067

Figure 77 Mux shield

Pin Name	Function
GND	All negative LED leads are connected to ground, 5V.
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 78 Mux shield LED wiring



Companion library

The library has four examples. This example used is the 'MuxShieldDigitalOut' sketch included in the library. The library works as expected for Arduino* and compiles on Galileo. However, some modifications are required in the library for the sketch to work on the Galileo.

The core resolution is that the initialization needs to happen twice. More specifically the second initialization needs to happen inside the 'setup' function passing key parameters. An exact copy of the constructor with parameters was copied into a method name 'init'. Add the new implementation into the library's implementation file and header file as follows:

```
MuxShield.cpp, Add method implementation
void MuxShield::init(int S0, int S1, int S2, int S3, int OUTMD,int IOS1, int
IOS2, int IOS3, int IO1, int IO2, int IO3)
{
    _S0 = S0;
    _S1 = S1;
    _S2 = S2;
    _S3 = S3;
    _OUTMD = OUTMD;
    _IOS1 = IOS1;
    _IOS2 = IOS2;
    _IOS3 = IOS3;
    _IO1 = IO1;
    _IO2 = IO2;
    _IO3 = IO3;

    pinMode(_S0,OUTPUT);
    pinMode(_S1,OUTPUT);
    pinMode(_S2,OUTPUT);
    pinMode(_S3,OUTPUT);
    pinMode(_OUTMD,OUTPUT);
    digitalWrite(_OUTMD,LOW);
    pinMode(_IOS1,OUTPUT);
    pinMode(_IOS2,OUTPUT);
    pinMode(_IOS3,OUTPUT);
}
```

```
MuxShield.h, header file before
public:
    MuxShield();
    MuxShield(int S0, int S1, int S2, int S3, int OUTMD,int IOS1, int IOS2,
int IOS3, int IO1, int IO2, int IO3);
MuxShield.h, header file after
public:
    MuxShield();
    MuxShield(int S0, int S1, int S2, int S3, int OUTMD,int IOS1, int IOS2,
int IOS3, int IO1, int IO2, int IO3);
    void init(int S0, int S1, int S2, int S3, int OUTMD,int IOS1, int IOS2,
int IOS3, int IO1, int IO2, int IO3);
```



Compile and upload

The 'MuxShieldDigitalOut' sketch needs to be modified to perform the second initialization. (The entire sketch is not shown.) The parameters passed are mapping to the same pins as the "MuxShield::MuxShield()" constructor; therefore, it is not clear why the values in the first constructor are not being accepted.

```
Sketch MuxShieldDigitalOut:
Before
void setup()
{
    //Set I/O 1, I/O 2, and I/O 3 as digital outputs
    muxShield.setMode(1,DIGITAL_OUT);
    muxShield.setMode(2,DIGITAL_OUT);
    muxShield.setMode(3,DIGITAL_OUT);
}
Sketch MuxShieldDigitalOut:
After
void setup()
{
    muxShield.init(2,4,6,7,8,10,11,12,A0,A1,A2);
    //Set I/O 1, I/O 2, and I/O 3 as digital outputs
    muxShield.setMode(1,DIGITAL_OUT);
    muxShield.setMode(2,DIGITAL_OUT);
    muxShield.setMode(3,DIGITAL_OUT);
}
```

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 deasserted. The LED shall light up from left to right, and then turn off from left to right.

Results

G1/G2/Edison Compatible.

Next steps

- Verify Analog examples (this uses a different shift register).

§

28 LinkSprite* Quad-band GPRS/GSM Shield

Use case

The LinkSprite SM5100B GSM/GPRS shield is ready for integration in various kinds of wireless phones and other wireless devices. It is possible to dial phone numbers and send text messages using AT commands. (See http://en.wikipedia.org/wiki/AT_command_set) The shield uses a quad-band low power consumption GSM/GPRS module SIM900 as well as a built-in compact PCB antenna.

Key info	Links
Product Info	http://www.cutedigi.com/Arduino-Shields/Quad-Band-Gprsgsm-Shield-For-Arduino-Cellular-Module-Included.Html
Library	No library.

Figure 79 Quad-band GPRS/GSM shield



Hardware summary

Key info	Description/links
Operating Voltage	3.3V
VIN as power source	No.
SIM	Mini SIM card interface.
GSM module	SM5100B http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Cellular/SM5100B-D_HW_spec_V1.0.0.pdf
Audio Interface	Not specified.



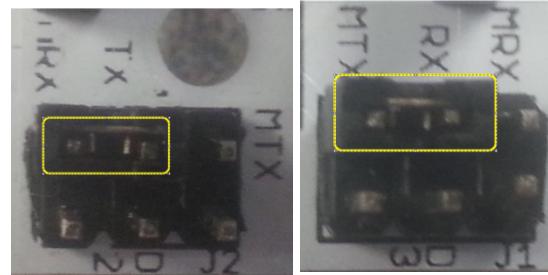
Pinout	Not specified.
Antenna	Antenna interface for external antenna.
Galileo Board Firmware	1.0.1 (Build 04232014) 1.0.3 Not tested since the shield is broken.
Edison Board Firmware	1.0.3 (Production Release) Not tested since the shield is broken.

Companion library

N/A.

Compile and upload

- Insert an unlocked SIM card on the underside of the shield. The SIM is a mini-SIM or 2FF size.
- Attach the shield to the Galileo. There is no extra wiring necessary.
- Set the Serial port select jumpers to the Hardware Serial position.



- Attach an antenna to the antenna connector.
- Connect the SM5100B chip.



- Power up the Galileo and connect to computer with the USB.
- 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.

```
Serial1.println("AT+CMGS=\\"15555555555\\");  
Serial1.println("ATD + 15555555555;");
```

- Upload the following sketch that will enable sending text messages and making calls.
- Open the serial terminal from the IDE.
- 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.)



Sample code

```
#include <String.h>

char data[256];

void setup()
{
    Serial1.begin(115200);
    Serial.begin(11500);      // the GPRS baud rate
    delay(1000);
    // power up the SIM900
    pinMode(9, OUTPUT);
    digitalWrite(9,LOW);
    delay(100);
    digitalWrite(9,HIGH);
    delay(500);
    digitalWrite(9,LOW);
    delay(100);
}

void loop()
{
    if (Serial.available())
        switch(Serial.read())
    {
        case 't':
            SendTextMessage();
            break;
        case 'r':
            ReceiveTextMessage();
            break;
        case 'd':
            DialVoiceCall();
            break;
        case 'a':
            AnswerVoiceCall();
            break;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}

void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
    mode
    delay(1000);
    Serial1.println("AT+CMGS=\"15555555555\"");
    delay(1000);
}
```



```
Serial1.println("Hello from Galileo?");  
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.println("ATD + 15555555555;");//dial the number  
    delay(100);  
    Serial1.println();  
}  
void ShowSerialData()  
{  
    while(Serial1.available()!=0)  
        Serial.write(Serial1.read());  
}
```

Results

G1/G2 Inconclusive.

Did not get this shield to work on Arduino* or Galileo. On a couple of occasions, valid text was being displayed in the serial terminal. The SIM card holder was broken so the SIM card was taped to the holder. There might be a problem with this card.

Next steps

Since there appeared to be a problem with this card, get another card that uses the SM5100B for testing.

§

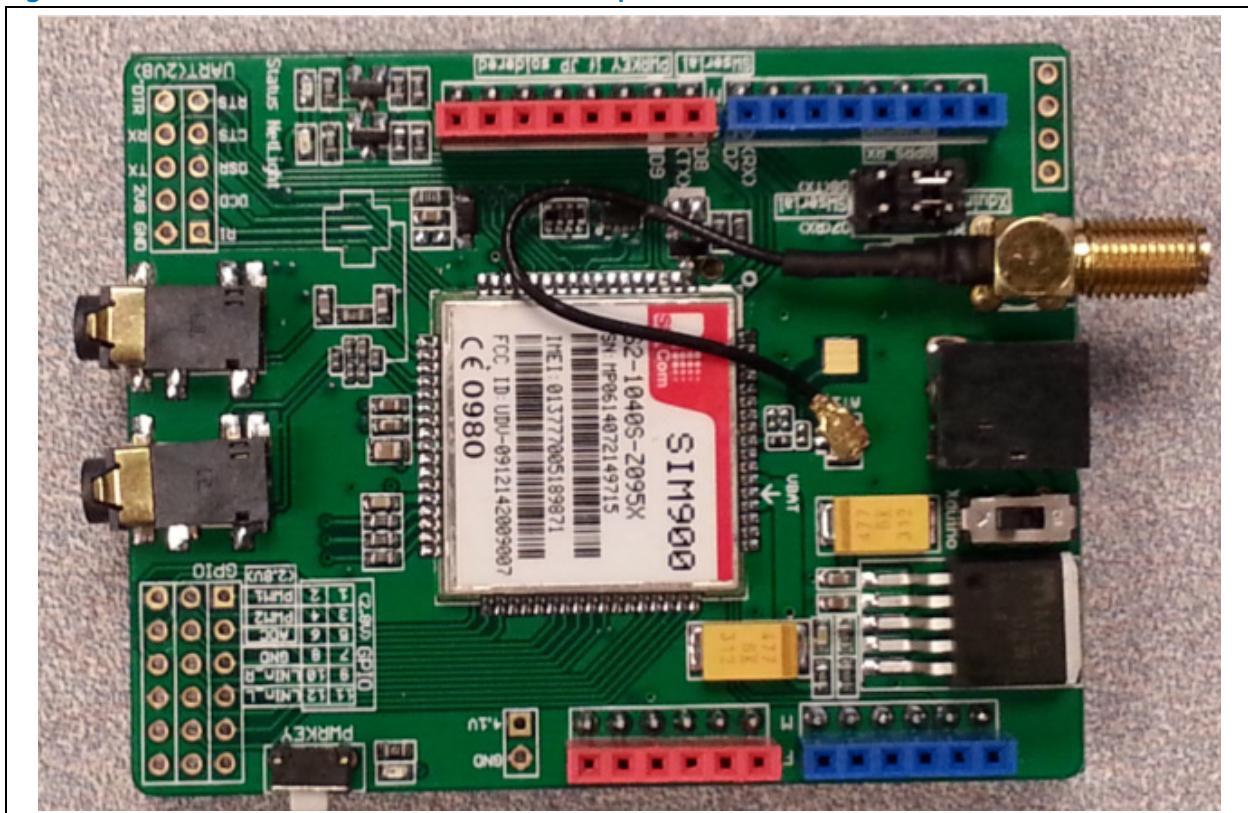
29 SIMCOM* Quad-band Mobile Sim900 – Arduino* Pack

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. (See http://en.wikipedia.org/wiki/AT_command_set.) The shield has an interface for an external antenna. The shield has a jumper selectable serial port so the Galileo is compatible with the hardware serial interface.

Key info	Links
Product Info	http://www.aliexpress.com/store/product/SIMCOM-SIM900-Quad-band-GSM-GPRS-Shield-Development-Board-for-Arduino/538175_1315631223.html
Library	No library. Use the serial communication interface to issue AT commands.

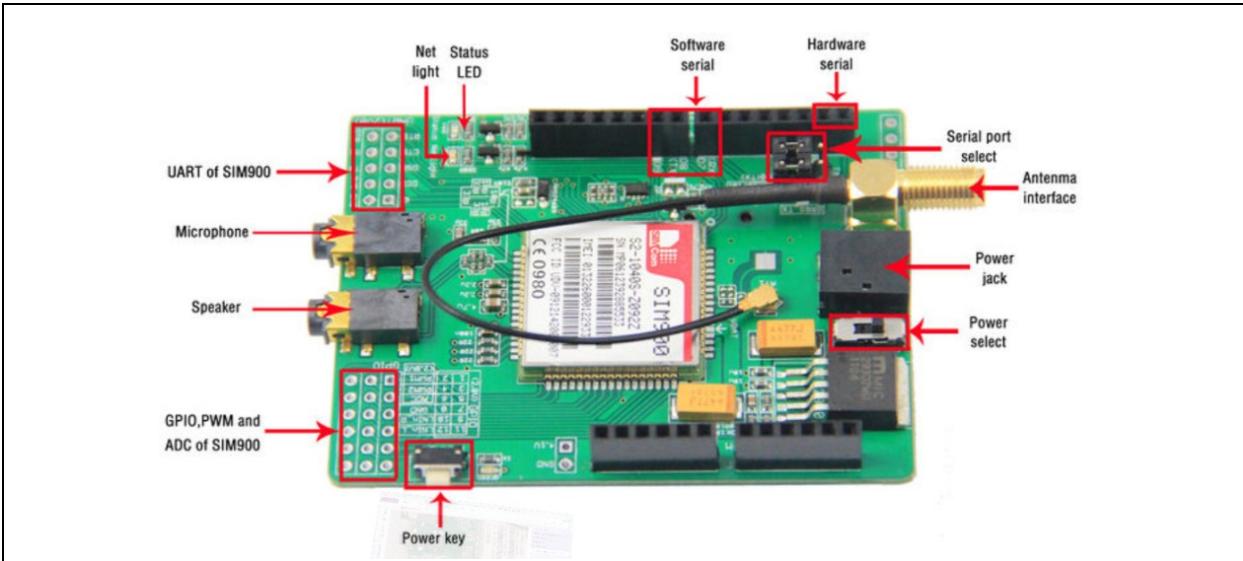
Figure 80 Quad-band Mobile Sim900 – Arduino pack



Hardware summary

Key info	Description/links
Operating Voltage	5V
VIN as power source	Yes.
Audio Interface	2 connections (microphone and speaker).
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. Antenna is provided with shield.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)

Figure 81 Quadband Mobile Sim900 layout



Pin name	Function
Rx of hardware serial port	Input from Galileo - connected to D0.
Tx of hardware serial port	Output to Galileo - connected to D1.
5V	Connected to 5V.
GND	Connected to GND.
D7	Only used if Software serial is selected. Galileo doesn't use SofwareSerial.
D8	Only used if Software serial is selected. Galileo doesn't use SofwareSerial.
D9	Used for software control of power-up/down of the SIM900.

Companion library

No library required.

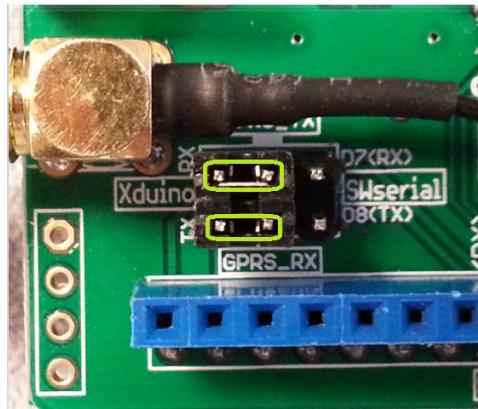
Compile and upload

Do the following:

- 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.
- Attach the shield to the Galileo. There is no extra wiring necessary.



- Set the Serial port select jumpers to the Hardware Serial position (Xduino).
- Power up the Galileo and connect to computer with the USB.



- 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.
- 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.

```
Serial1.println("AT+CMGS=\"15555555555\"");  
Serial1.println("ATD + 15555555555;");
```

- Upload the following sketch that will enable sending text messages and making calls.
- Open the serial terminal from the IDE and set rate to 9600.
- 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.)





```
Sketch: Issue AT commands over serial link
#include <String.h>

char data[256];

void setup()
{
    Serial1.begin(9600);
    Serial.begin(9600);      // the GPRS baud rate
    delay(1000);
}

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;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}

void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
    mode
    delay(1000);
    Serial1.println("AT+CMGS=\"15555555555\"");
    delay(1000);
    Serial1.println("Hello from Galileo?");
    delay(1000);
    Serial1.println((char)26); //the ASCII code of the ctrl+z is 26 (0x1A)
    delay(1000);
}
```



```
Sketch: Issue AT commands over serial link
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\"");
    delay(1000);
    Serial1.println("AT+CMGL=\"ALL\"");
    delay(1000);
}

void DialVoiceCall()
{
    Serial1.println("ATD + 155555555555");
    delay(1000);
    Serial1.println();
}

void ShowSerialData()
{
    while(Serial1.available() != 0)
        Serial.write(Serial1.read());
}

void GPRS()
{
    Serial1.println("AT+CPIN?");
    delay(1000);
    ShowSerialData();

    Serial1.println("AT+CGREG?");
    delay(1000);
    ShowSerialData();

    Serial1.println("AT+COPS?");
    delay(1000);
    ShowSerialData();

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

    Serial1.println("AT+cgatt=1");
    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?");
```

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

```

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();
}

```

Results

G1/G2/Edison Compatible.

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.

Next steps

Test more with a data application.

§



30 MCM* RS-232 Arduino* Shield

Note: This is on the "List of Supported Shields" ([Link](#)).

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://www.cutedigi.com/arduino-shields/rs232-shield-for-arduino.html
USB to Serial Cable	http://www.newegg.com/Product/Product.aspx?gclid=CMKtvaekb8CFQqlfgodmVkJA8g&Item=N82E1681207381&nm_mc=KNC-GoogleAdwords&cm_mmc=KNC-GoogleAdwords_-pla_-Serial+Cables_-_N82E16812107381&ef_id=UwmRrAAABAIRQ2K7:20140623223942:s
Library	None

Hardware summary

Key info	Description/links (No wiring required)
Operating Voltage	Designed for 5V
IOREF	Present but not used.
Use VIN as power source	No.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematics	http://www.cutedigi.com/pub/Arduino/arduino_RS232.pdf

Figure 82 RS-232 Arduino shield



This shield uses D0 and D1 on the Galileo/Arduino board with no jumpers to change the pin allocation. No wiring is required.



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 Galileo/Arduino 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. By default, Arduino and Edison sketches are persistent. By default, Galileo sketches are NOT persistent. To make sketches persistent on a Galileo board, set up the SD card (see [Getting Started Guide](#)). Once an SD card is formatted and configured with a bootloader, it can be inserted into the SD card slot on the Galileo board. When a sketch is downloaded to the Galileo, the sketch will be written to the SD card and will be available on the next power cycle.

This test will communicate from the Galileo 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 Galileo IDE. The setting for this test is 9600 baud (8, N 1) for both COM ports.

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 Galileo. The serial writes (in the sketch) are used to verify that the data transmitting through the RS-232 shield to the PC.

Unfortunately, serial ports (in the sketch) are mapped slightly different in Arduino and [Galileo](#). This difference may give the impression that the shield is not functional. The example was written such that sketch can run on both Arduino and Galileo boards. Pointers to the serial objects were necessary to minimize the changes in the sketch to run on the Arduino.

Sketch: Example Sketch

```
// This code demonstrates how to run the shield on Arduino and Galileo.
// A R D U I N O
//HardwareSerial* gSerialStdPtr = &Serial; // Arduino Uno, Rx(D0) Tx(D1)
//HardwareSerial* gSerialOnePtr = &Serial; // Arduino Uno, Rx(D0) Tx(D1)

// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Arduino
IDE Serial Monitor
TTYUARTClass* gSerialOnePtr = &Serial1; // Galileo, /dev/ttySO, Tx pin on
the Shield

int gCnt; // Loop iterations
int gLed = 13; // Add led blink for visual verification (Pin 13)
void setup()
{
    gSerialStdPtr->begin(9600);
    gSerialOnePtr->begin(9600); // Initialize serial port
    pinMode(gLed, OUTPUT); // Initialize the digital pin as an output
    gCnt = 0;
}

void loop()
```



```
Sketch: Example Sketch

{
    if(*gSerialStdPtr)
    {
        gSerialStdPtr->print("GSO> Blink ");
        gSerialStdPtr->println(gCnt);
    }
    if(*gSerialOnePtr)
    {
        gSerialOnePtr->print("SO> Blink ");
        gSerialOnePtr->println(gCnt);
    }

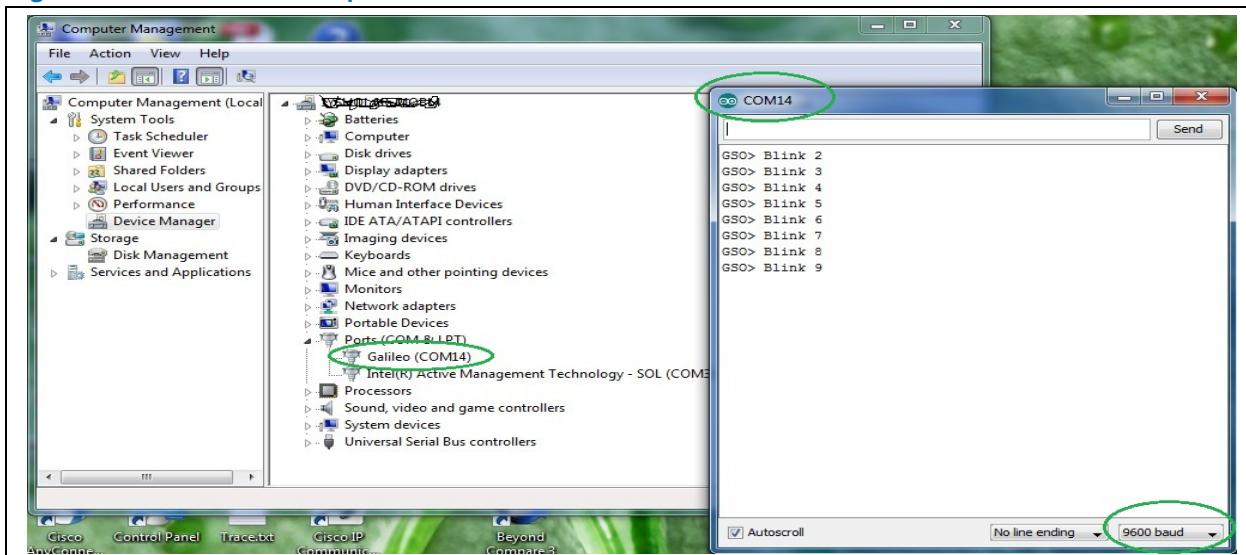
    digitalWrite(gLed, HIGH); // Turn the LED on (HIGH is the voltage level)
    delay(1000*1); // Wait in seconds

    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 Galileo board.
- Power up the Galileo (wait the appropriate time).
- Connect the USB cable (used for sketch download) to the Galileo.
- Deploy the sketch to the Galileo 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 Galileo 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 Galileo board.

Figure 83 Galileo COM port test results – without the RS-232 Arduino shield



The first test was successful on the Galileo. The LED was blinking and the output was displayed on the serial port accessed from the Galileo IDE. This test validated the functionality of the program.

Figure 84 Connecting the RS-232 Arduino shield on Galileo 1



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 Galileo 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.

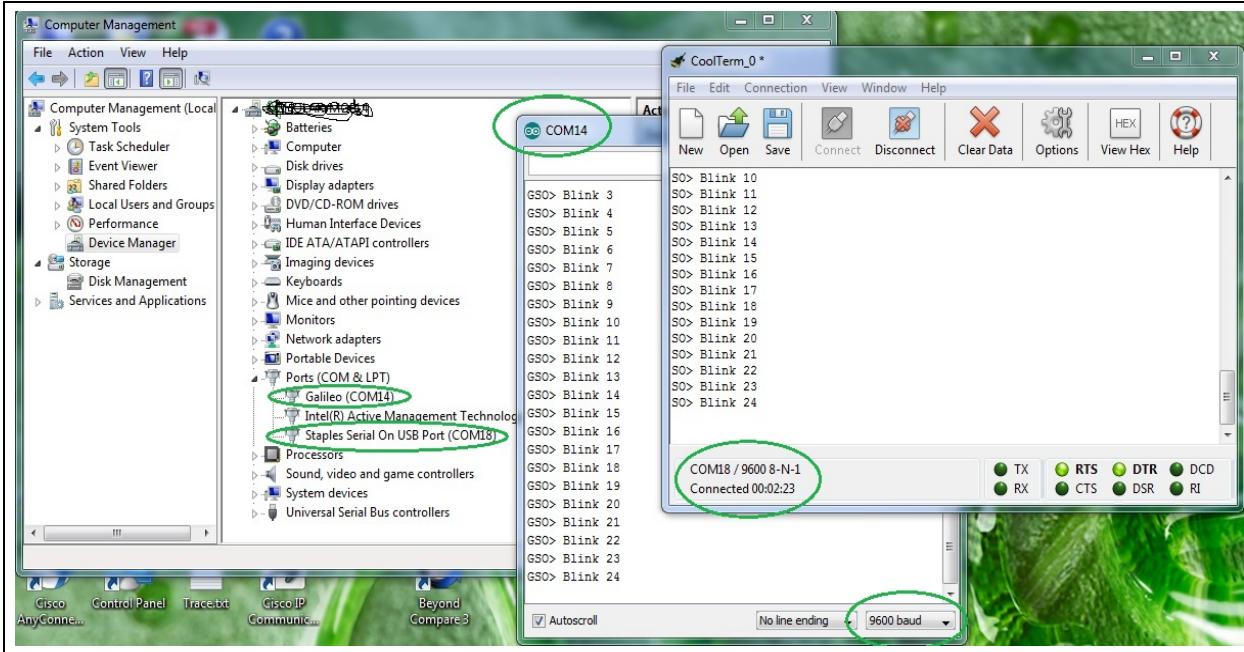
- Ensure that the Galileo is powered down. (SD card shall be in the SD card slot.)
- Ensure the USB cable (used for sketch download) is not attached.

Note: The program downloaded in the first test is still present on the SD.

- Attach the RS-232 shield to the Galileo.
- Attach the 'Serial to USB' adapter to the shield.
- Power the Galileo 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 Galileo. On the PC side, a COM port will be assigned to this connection.
- Start the Galileo IDE and bring up the serial port. Display for Galileo IDE Serial (/dev/ttyGS0) 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 85 Galileo 1 COM port test results – with the RS-232 Arduino shield



The second test showed a blinking LED. The serial port on the RS-232 shield is 'Serial1' (/dev/ttys0 in Linux*). The standard serial on the Galileo IDE is 'Serial' (/dev/ttys0 in Linux*). The result shows that data is being transmitted on both serial ports.

This sketch can also be run on the Arduino. Comment out the serial pointers for the Galileo and uncomment the serial pointers for the Arduino. Rerun the test as described. The results will show all outputs go to both serial ports.

Results

G1/G2/Edison Compatible. The shield is hardcoded to serial port D0/D1, therefore, it was not possible to test Galileo 2 on D2/D3

Next steps

- In a sketch, read data from the RS-232 serial port and display it on the Galileo IDE.
- Attempt to connect through the audio jack.

§

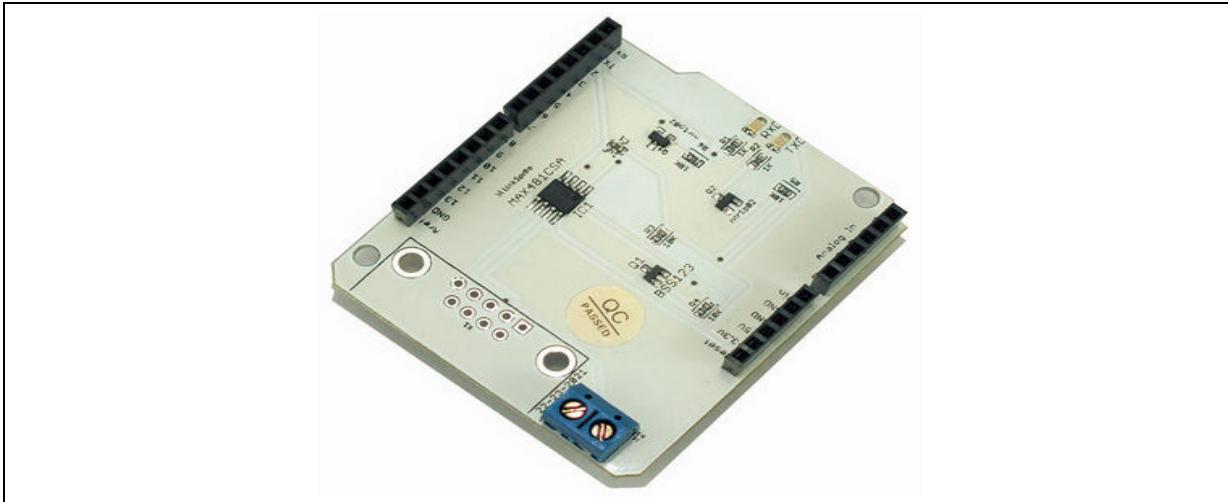
31 LinkSprite® RS-485 Shield

Use case

RS-485 is a standard communication port for field bus. Arduino® only has a USB port and a TTL UART interface. In order to talk to a device that has RS-485 bus, we can add an RS-485 port to Arduino using this RS-485 shield. Even though the RS-485 is sometimes thought as an "archaic" protocol, it will allow up to 32 devices to communicate through 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/
Library	Not required.
Guide	http://linksprite.com/wiki/index.php5?title=RS485_Shield_for_Arduino

Figure 86 LinkSprite RS-485 shield



Hardware summary

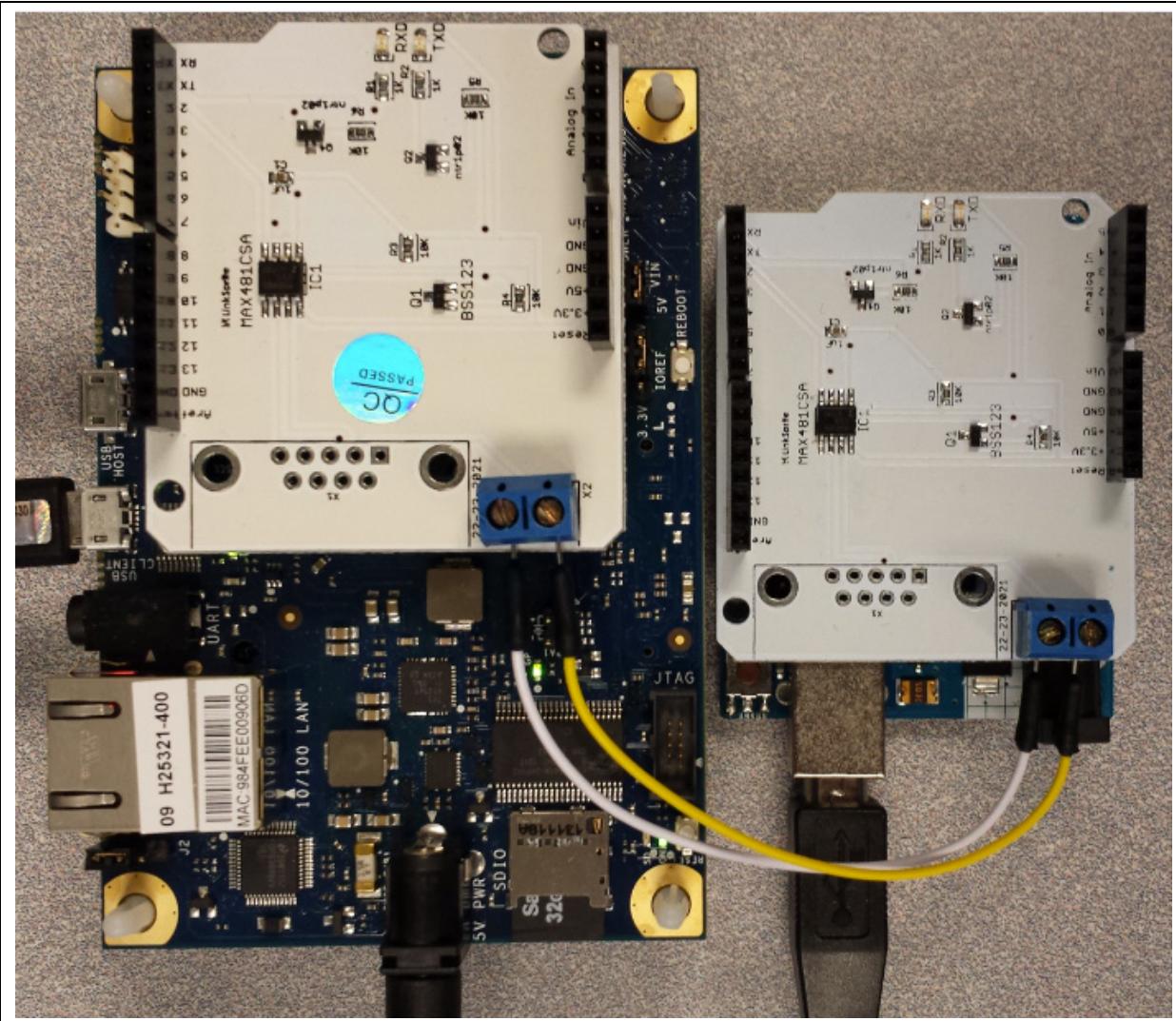
Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
IOREF	No.
Use VIN as power source	No.
Operating voltage	3.3V to 5V
Schematics	https://s3.amazonaws.com/linksprite/Shields/RS485/RS485_schematics.pdf

Connections used for testing:

- X2-1 Arduino (transmitter) to X2-1 Galileo (receiver)
- X2-2 Arduino (transmitter) to X2-2 Galileo (receiver)



Figure 87 LinkSprite RS-485 shield on a Galileo 1



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 board. This also means that the RS-485 shield and the USB port on the Arduino Uno 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 the Arduino board.



Compile and upload

The purpose of the following sketches is to test RS-485 communication between an RS-485 shield mounted on an Arduino (acting as a transmitter) and an RS-485 shield mounted on a Galileo (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 (transmitter) is as follows:

```
Sketch: Example Sketch
//#include <SoftwareSerial.h>
//SoftwareSerial mySerial(0,1);
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 Galileo (receiver) is as follows:

```
void setup()
{
    Serial.begin(9600);
    Serial1.begin(9600);
}

void loop()
{
    while(Serial1.available() > 0)
    {
        char c = Serial1.read();
        Serial.write(c);
    }
}
```

Results

G1/G2/Edison Compatible.

Next steps

- None



32 SeeedStudio* Relay Shield

Note: This shield is on the “List of Supported Shields” ([Link](#)).

Use case

This relay switch enables the Galileo or Arduino* 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 88 SeeedStudio relay shield

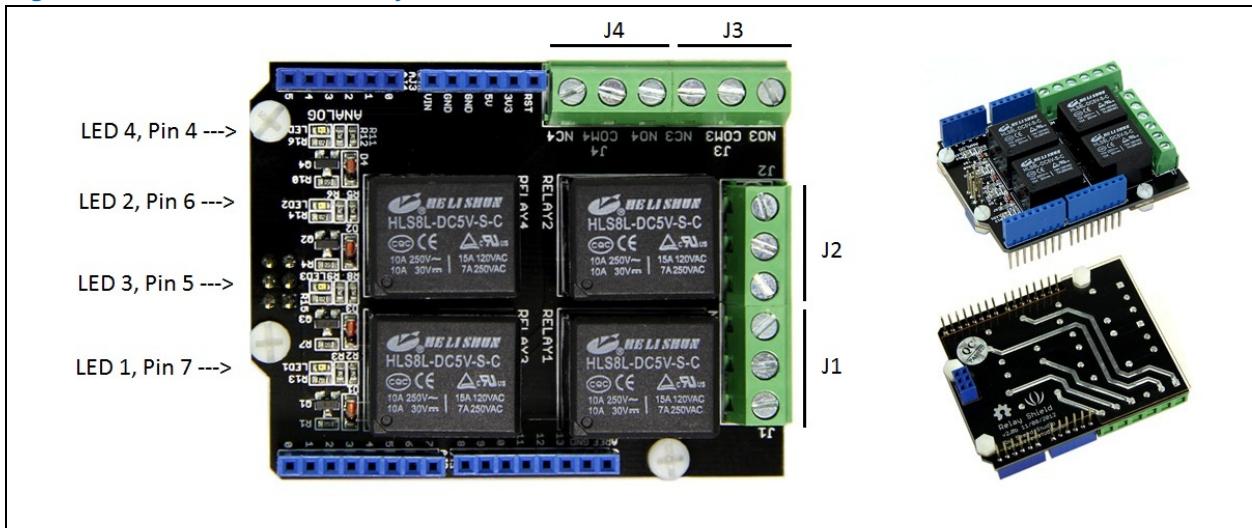
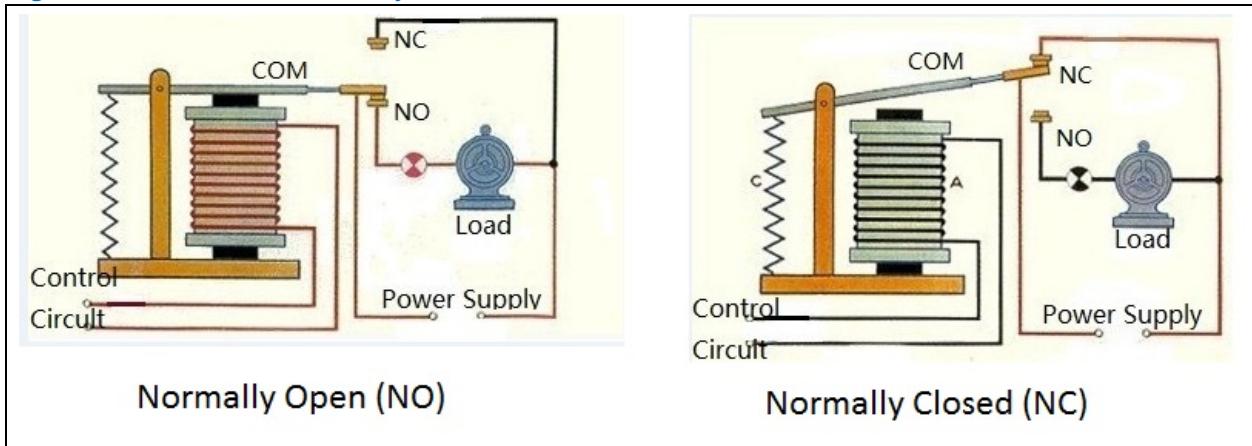


Figure 89 SeeedStudio relay shield NO and NC states





Hardware summary

Key info	Description/links
Operating Voltage	Designed for 5V.
IOREF	Present but not used.
Use VIN as power source	No.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Schematics	N/A.
Relay	HLS8L-DC5V-SC

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.

Compile and upload

Attach the shield to the Galileo 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.

Sketch: 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);
}
```

Results

G1/G2/Edison Compatible.



Next steps

Add a non-5 V device to the JN interface port.

§

33 SeeedStudio® GPRS Shield

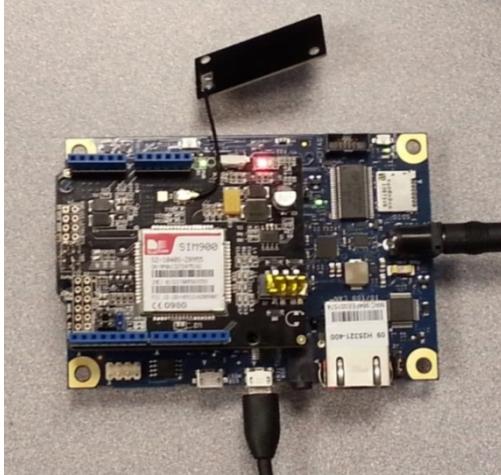
Note: This is on the "List of Supported Shields" ([Link](#)).

Use case

This shield makes it possible to hook the Galileo / Arduino® 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. (See http://en.wikipedia.org/wiki/AT_command_set.) The shield uses a SIMCOM® SIM900 quad-band 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
Library	None.
AT Commands	http://en.wikipedia.org/wiki/AT_command_set

Figure 90 SeeedStudio GPRS shield on Galileo 1



Hardware summary

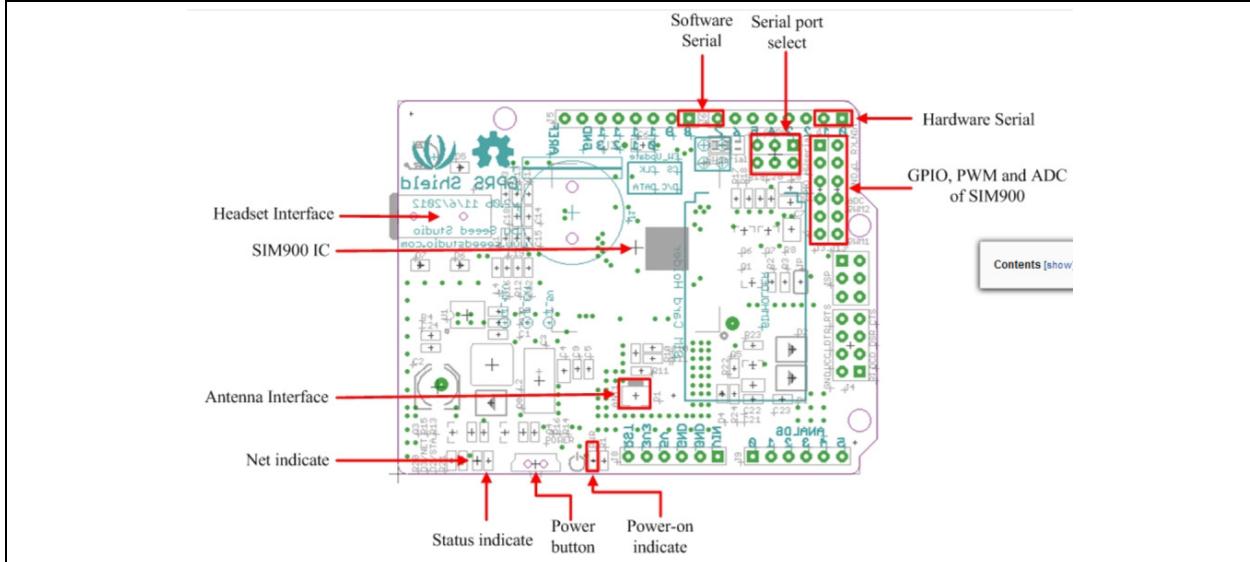
Key info	Description/links
Operating Voltage	5V
VIN as power source	Yes.
Audio Interface	2-in-1 headset jack.
SIM	Mini SIM card interface.
GSM module	SIM900 http://www.simcom.us/product_detail.php?cid=1&pid=37
Antenna	Antenna interface for external antenna. PCB antenna is provided with the shield.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)



Pin name	Function
Rx of hardware serial port	Input from Galileo - connected to D0.
Tx of hardware serial port	Output to Galileo - connected to D1.
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 91 SeeedStudio GPRS shield layout



Companion library

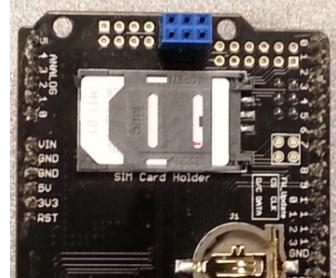
No library required.



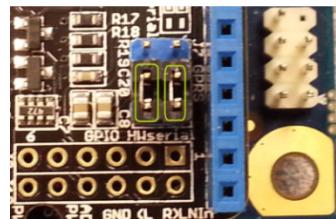
Compile and upload

Do the following:

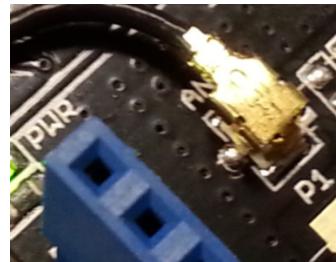
- Insert an unlocked SIM card on the underside of the shield. The SIM is a mini-SIM (2FF size).
- Attach the shield to the Galileo. There is no extra wiring necessary.



- Set the Serial port select jumpers to the Hardware Serial position. The default position is Software Serial.



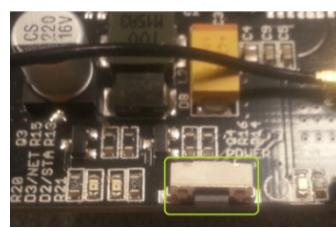
- Attach the PCB antenna that is supplied with the shield.



- Connect a 2-in-1 headset to the headset connector on the shield.
- Power up the Galileo and connect to computer with the USB.



- Press the power button on the shield for two seconds. This step can be skipped if using a sketch that turns on the power, such as the sketch that follows.



- The red status light should illuminate, and the green net status light should blink once every 3 seconds.
- 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.

```
Serial1.println("AT+CMGS=\"1555555555\"");  
Serial1.println("ATD + 1555555555;");
```

- Upload the following sketch that will enable sending text messages and making voice calls.



- Open the serial terminal from the IDE at 19200 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.)

```
Sketch: Issue AT commands over serial link
#include <String.h>

char data[256];

void setup()
{
    Serial1.begin(19200);
    Serial.begin(19200);      // the GPRS baud rate
    delay(1000);
}

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;
    }
    if (Serial1.available())
        Serial.write(Serial1.read());
}

void AnswerVoiceCall()
{
    Serial1.println("ATA");
}
```



```
Sketch: Issue AT commands over serial link
void SendTextMessage()
{
    Serial1.print("AT+CMGF=1\r");      //Because we want to send the SMS in text
mode
    delay(1000);
    Serial1.println("AT+CMGS=\"15555555555\"");
    delay(1000);
    Serial1.println("Hello from Galileo?");
    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\"");
    delay(1000);
    Serial1.println("AT+CMGL=\"ALL\"");
}
void DialVoiceCall()
{
    Serial1.println("ATD + 15555555555"); //dial the number
    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();
```



```
Sketch: Issue AT commands over serial link
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 network
Serial1.println("AT+CGATT=0");
delay(1000);
ShowSerialData();
}
```

Results

G1/G2/Edison Compatible.

Next steps

- Use better headset and microphone to test audio quality.

§

34 SeeedStudio® Solar Charger Shield v2

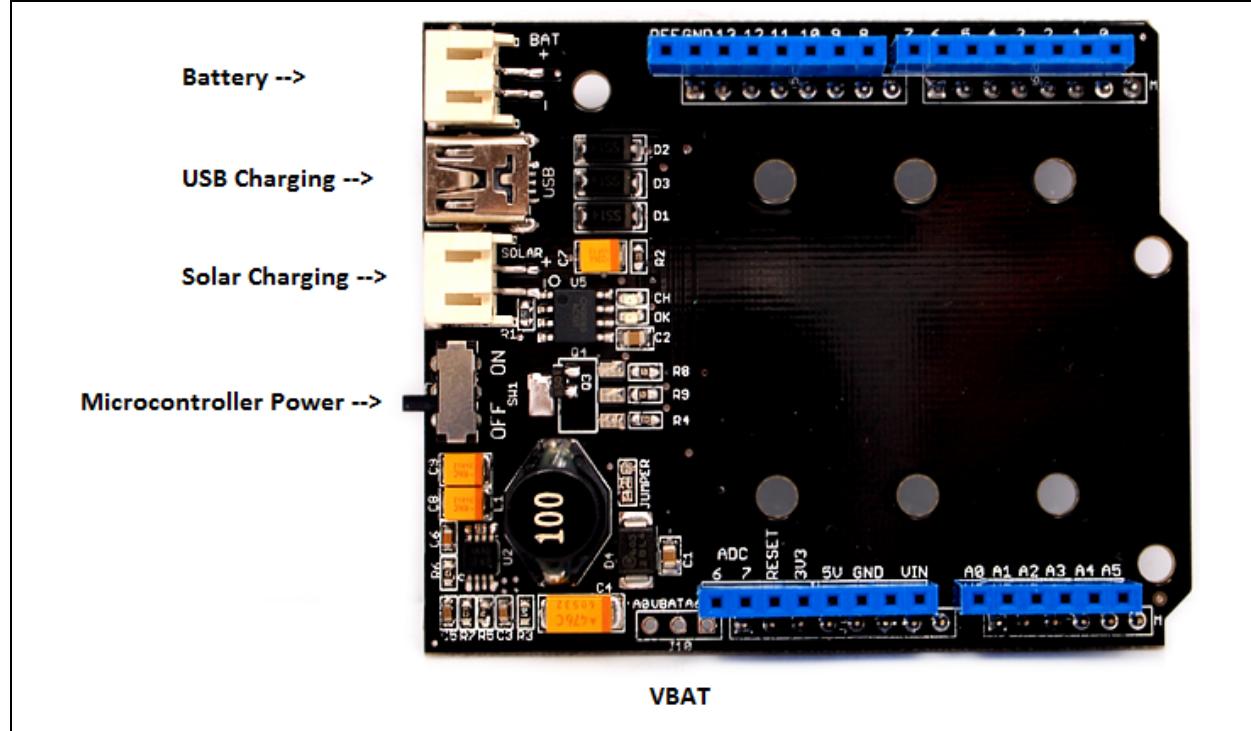
Note: This is on the "List of Supported Shields" ([Link](#)).

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 2.7V 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 700mA. A USB connector is also useful to charge the battery. The shield features short-circuit protection, battery status indicator, and 5V 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 92 Seeed Studio® Solar Charger Shield v2





Hardware summary

Key info	Description/links
Operating Voltage	2.7V to 5V
VIN as power source	No
VBAT	Pin to measure the output of the charging circuit of the battery.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	Not Supported since power generates 5V.
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 93 Seeed Studio® Solar charger shield v2 with Solar Panel



Companion library

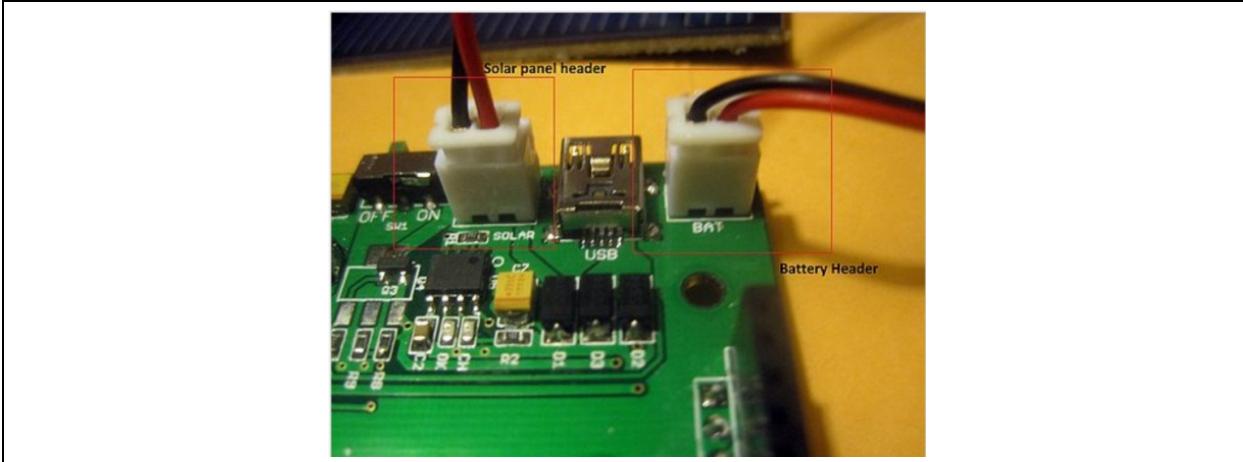
No library required.

Compile and upload

The solar shield does not need to be connected to the Galileo to begin charging. It is recommended to give proper charging to the battery before connecting it to the microcontroller.

Connect the solar panel and the battery as shown in Figure 94.

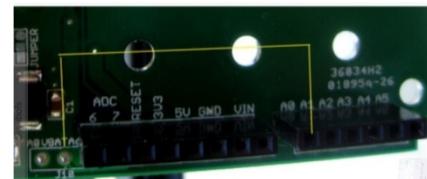
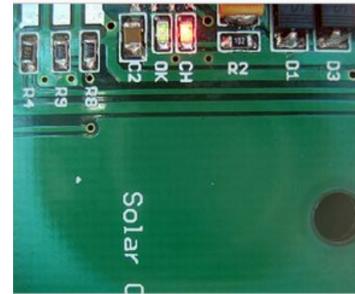
Figure 94 Seeed Studio solar charger shield connections



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 Galileo 1. Connect VBAT pin (center pin) on the shield to pin A0 of the charger shield. Power-up the Galileo with the standard power supply.





Use the following sketch to measure the voltage of the battery.

```
Sketch: Sketch Name or Sketch Description
/*
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
```



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
```

Perform at your own risk! Last, we can verify that the shield with a battery can power up the microcontroller. Send the standard 'Blink' program and insure 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 the Galileo 1 board with a fully charged battery. Once the shield is attached, power on the board by setting the shield power to 'ON'. When the Galileo is fully booted, the LED 13 should be blinking.

Results

G1 Compatible. G1 support 5 V power supply. Galileo 2 and Edison supports 7 to 15 V power supply. It is not recommended to use this shield on a G2/Edison.

Charging batteries is not dependent on Galileo. 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.

Next steps

- Tests to determine if voltage measurements from the battery are accurate.



35 MIFARE-One* RFID Tag 13.56 MHz (Keyfob)

Use case

The MIFARE-One RFID tag (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.

Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
IOREF	N/A.
Use VIN as power source	N/A.
Operating voltage	N/A.

Figure 95 MIFARE-One RFID tag 13.56 MHz (keyfob)



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.

Note: 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.

Results

G1/G2/Edison Compatible.

§



36 DFRobot* 2-Amp Motor Shield

Note: This is on the "List of Supported Shields" ([Link](#)).

Use case

The DFRobot Arduino* Compatible Motor Shield (2A) uses the L298P chip which will drive two 5V to 12V DC motors with maximum current of 2A. This shield can be directly mounted onto the Galileo. The motor shield can use the VIN power supply or an external power source when the motor current exceeds the limits provided from the Galileo (5V). 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.

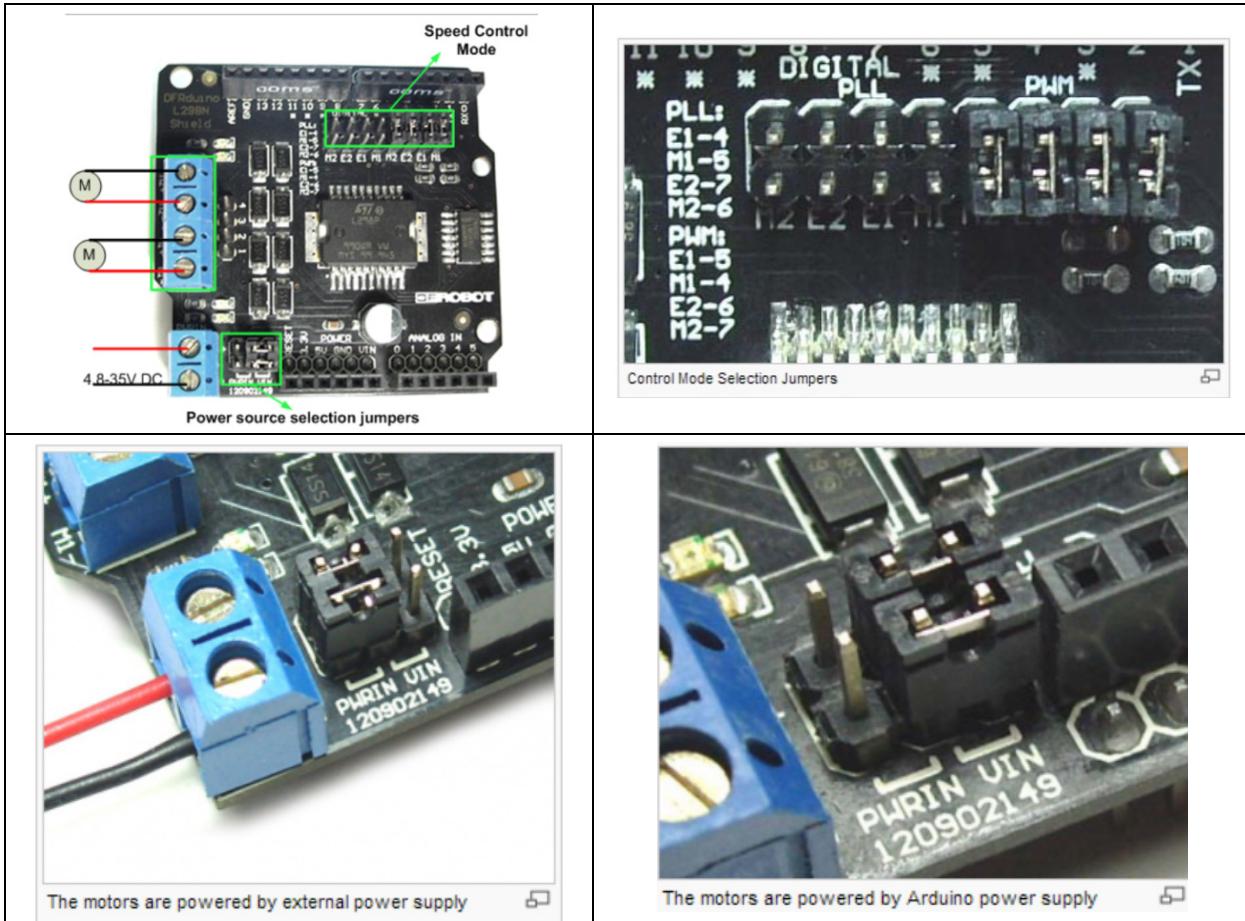
Key info	Links
Product Info	http://www.dfrobot.com/index.php?route=product/product&keyword=DRI0009&category_id=0&description=1&model=1&product_id=69#.U49LucRDuCl
Library	None.
Guide	http://www.dfrobot.com/wiki/index.php?title=Arduino_Motor_Shield_(L298N)_(SKU:DRI0009)

Hardware summary

Key info	Description/links
Operating Voltage	5V to 12V (logic control 5V from Galileo).
Maximum Current	2A per channel.
IOREF	Yes.
Use VIN	Optional with jumper or use an external power supply if over 5V needed.
Motor Controller	L298P http://www.st.com/web/en/catalog/sense_power/FM142/CL851/SC1790/SS1555/PF63147
Galileo Board Firmware	1.0.3 (Production Version)
Edison Board Firmware	1.0.3 (WW37)
Control Mode	Pulse width modulation (PWM) or phase locked loop (PLL) – set by jumper.
Motors	DC1, DC2

Figure 96

Connecting a 2-amp motor shield using an external power supply



Companion library

No library required.

Compile and upload

Note: Galileo 1 Only: Remove the VIN jumper on the Galileo card or damage to the Galileo card will occur.
Read the *VIN Jumper Overview* section.

For Edison use the default *PWM* settings.

Example code from this link:

[http://www.dfrobot.com/wiki/index.php?title=Arduino_Motor_Shield_\(L298N\)_SKU:DRI0009](http://www.dfrobot.com/wiki/index.php?title=Arduino_Motor_Shield_(L298N)_SKU:DRI0009)

**Sketch: PWM Speed Control**

```
//Arduino PWM Speed Control:  
int E1 = 5;  
int M1 = 4;  
int E2 = 6;  
int M2 = 7;  
  
void setup()  
{  
    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);  
    }  
}
```

Sketch: PLL Speed Control

```
//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 97

Galileo 1: PWM mode at 100 (out of 255) speed with a DC motor using external 6 V for power.
Reading taken from pin D5.

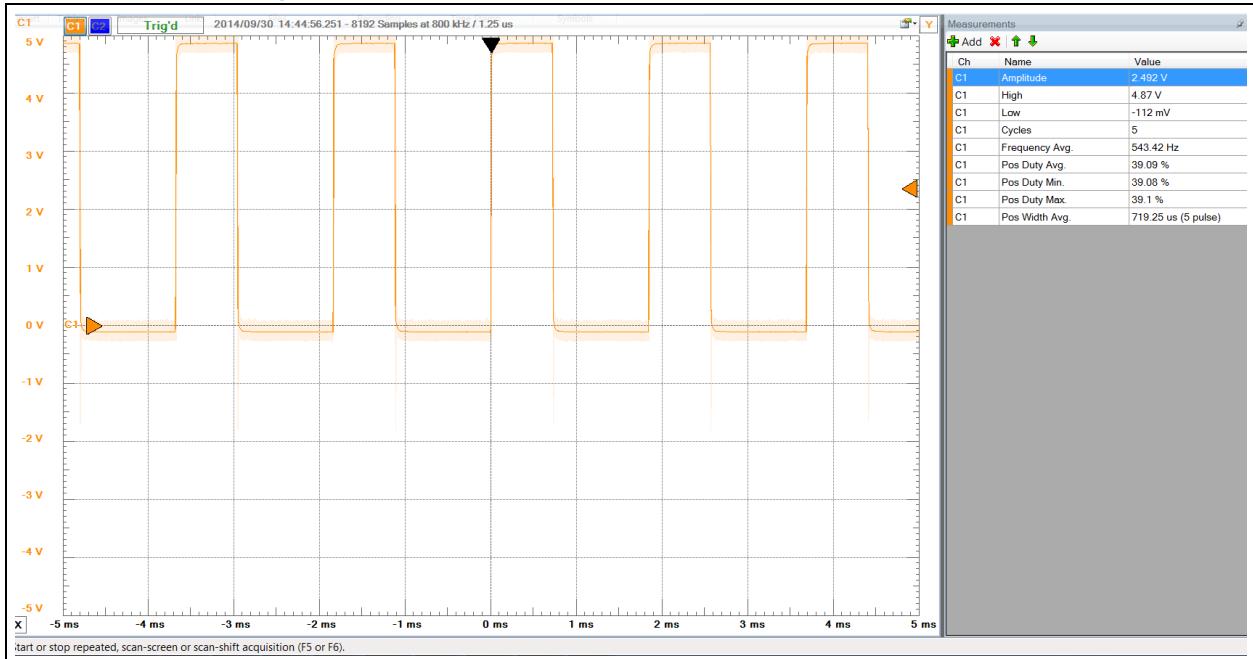


Figure 98

Edison: PWM mode at 100 (out of 255) speed with a DC motor using external 6 V for power.
Reading taken from pin D5.

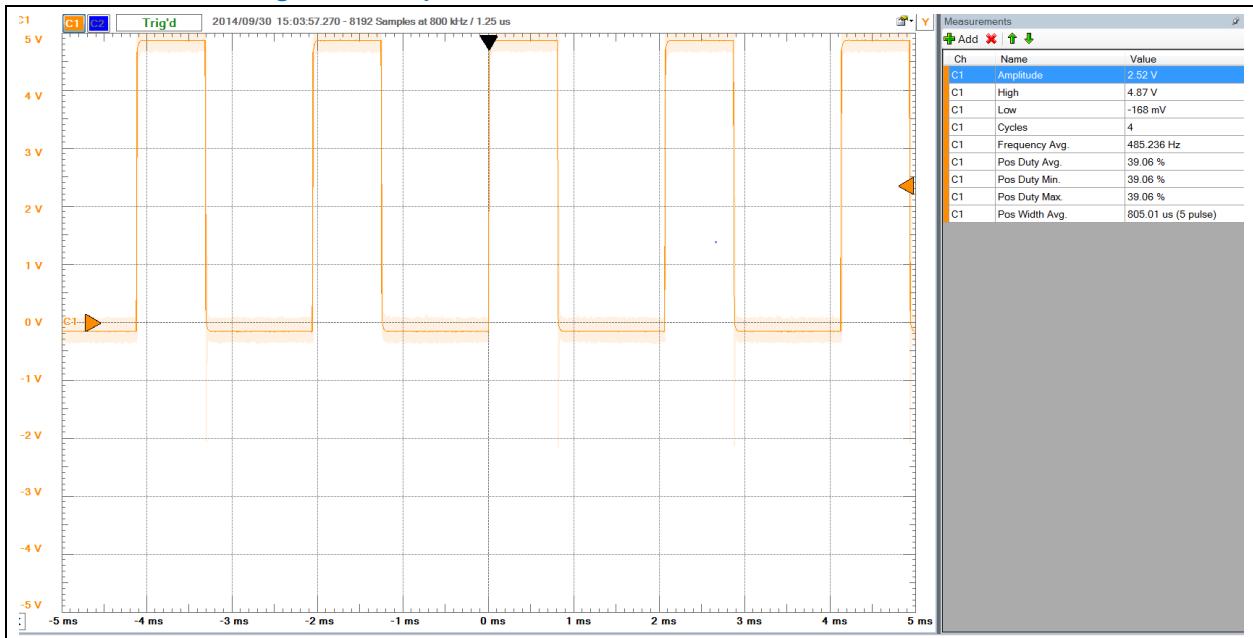


Figure 99 Edison: PLL mode at 100 (out of 255) speed with a DC motor using VIN, motor doesn't turn.
Reading taken from pin 4.

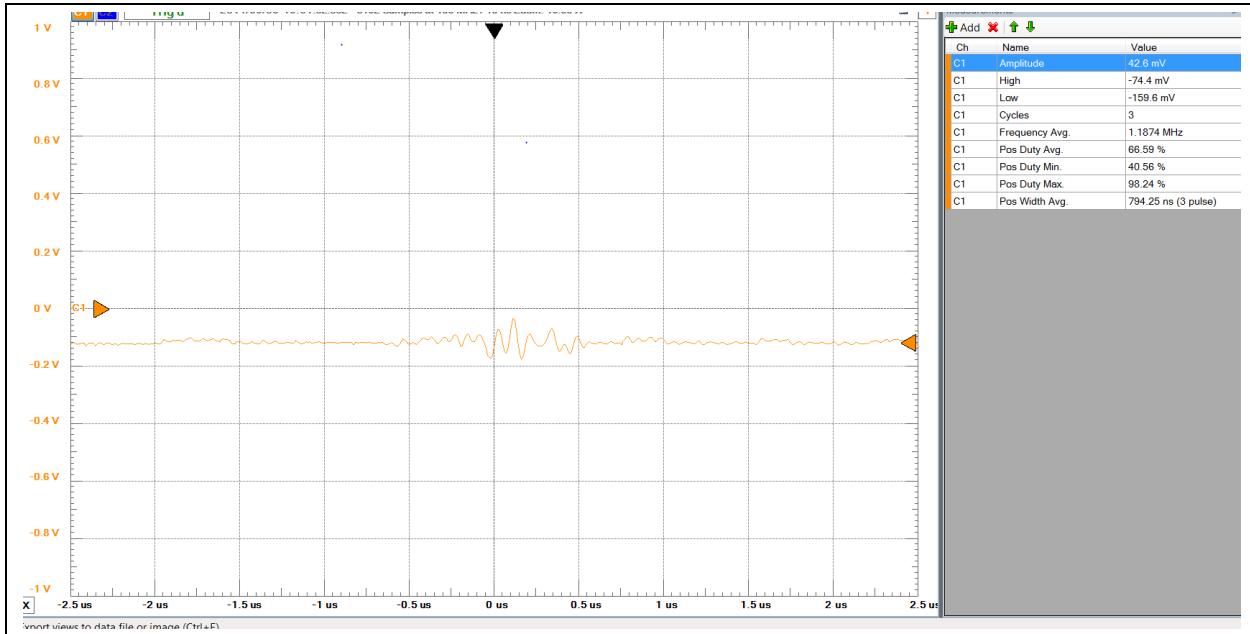
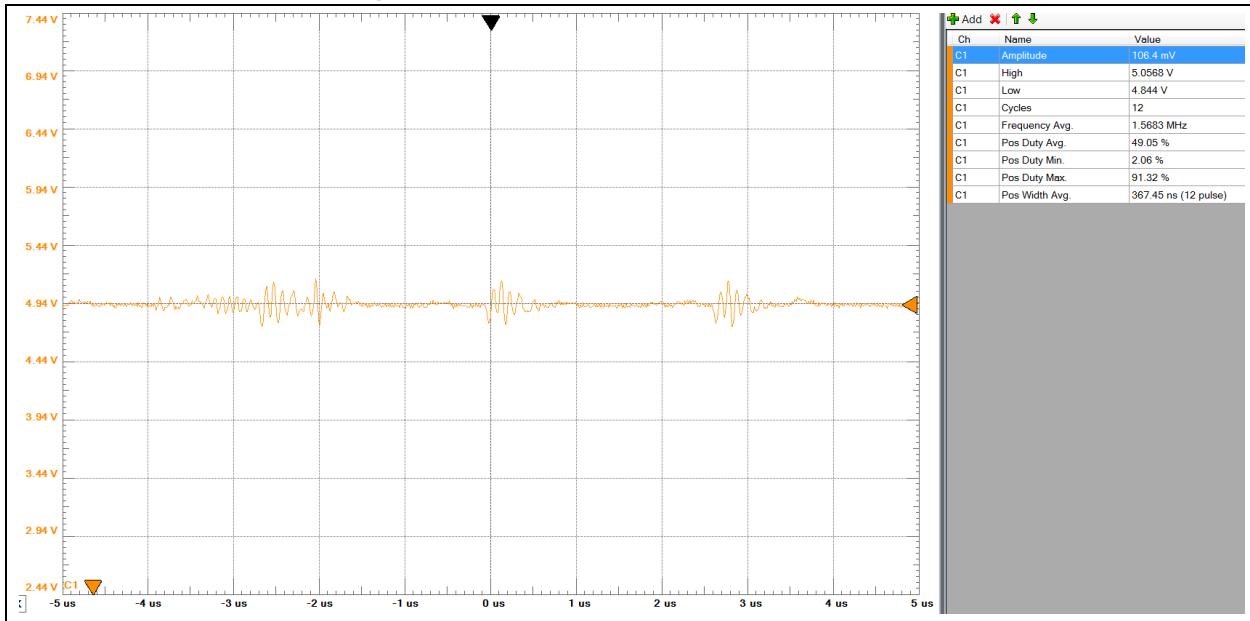


Figure 100 Arduino: PLL mode at 255 (out of 255) speed with a geared DC motor with no resistance applied. Reading taken from pin 4.





Results

G1/G2/Edison Compatible with PWM mode (using VIN or an external 9V/11V power source).

G1/G2/Edison Not Compatible with PLL mode. Works on Arduino but not on Galileo Galileo1/Galileo2 and Edison. Tested with VIN power with an external power source.

Next steps

Try to see if PLL mode will work on the Galileo. After searching for information on PLL for this motor, there is not a lot of information on how it is implemented or if it is really useful. It would be nice to be able to have a dyno meter for small motors so that the speed and power could be gauged to see if the PLL mode is working properly on the Arduino. Maybe the feature is not needed on the Galileo.

§



37 Renbotics* Servoshield v2.0

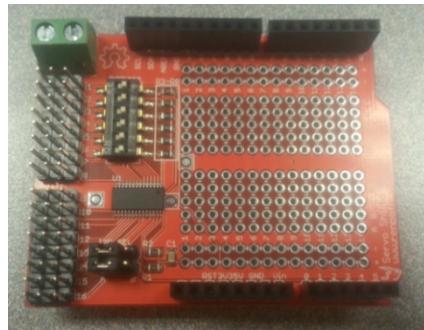
Note: This is on the "List of Supported Shields" ([Link](#)).

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 I2C 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	http://www.renbotics.com/files/servoshield2.zip
Guide	http://www.renbotics.com/files/RenboticsServoShield2Rev1.pdf

Figure 101 Renbotics Servoshield v2.0



Hardware summary

Key info	Description/links
Operating Voltage	5V for shield, voltage for motor comes from an external source.
Maximum Current	2A per channel.
Use VIN	No
Motor Controller	NXP* PCA9685 http://www.nxp.com/documents/data_sheet/PCA9685.pdf
Galileo Board Firmware	1.0.2 (Production Release)
Edison Board Firmware	1.0.3 (WW31) Shield does not power-up properly, under investigation.
Pinout	Not specified, but there is a jumper selection for which pins I2C should use.

Companion library

The library works with no modifications.

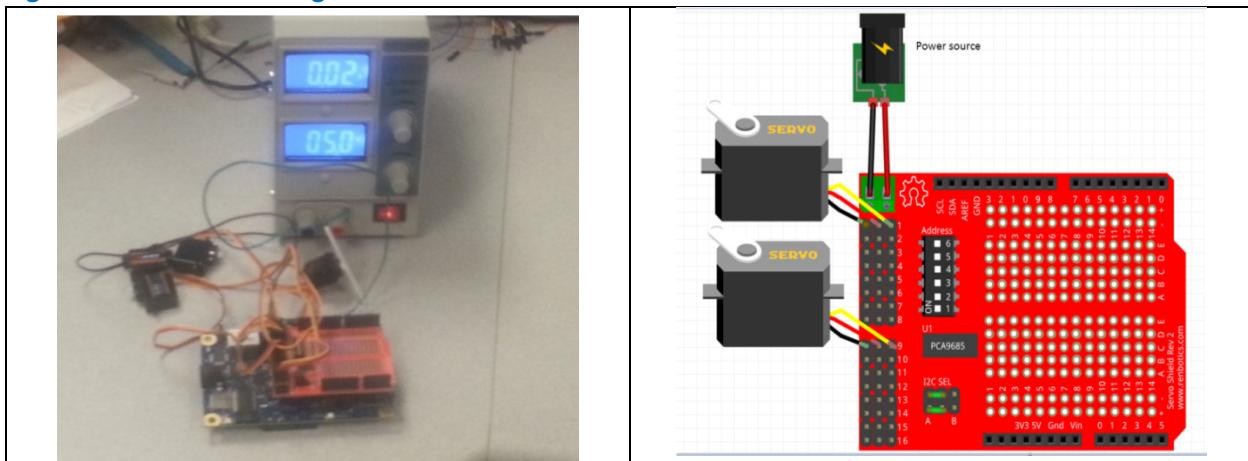
Compile and upload

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 50 and 60 Hz. 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 clockwise.

Jumper the I2C SEL to the A side of the pins as shown in the diagram. Connect the shield to a 5V 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.

Note: Galileo 1 Only: Remove the VIN jumper on the Galileo card or damage to the Galileo card will occur. Read the VIN Jumper Overview section.

Figure 102 Connecting the Renbotics Servoshield on Galileo 1



Sketch: Run up to 16 servos in sweeping fashion

```
#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");
}
```



Sketch: Run up to 16 servos in sweeping fashion

```
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();
}
```

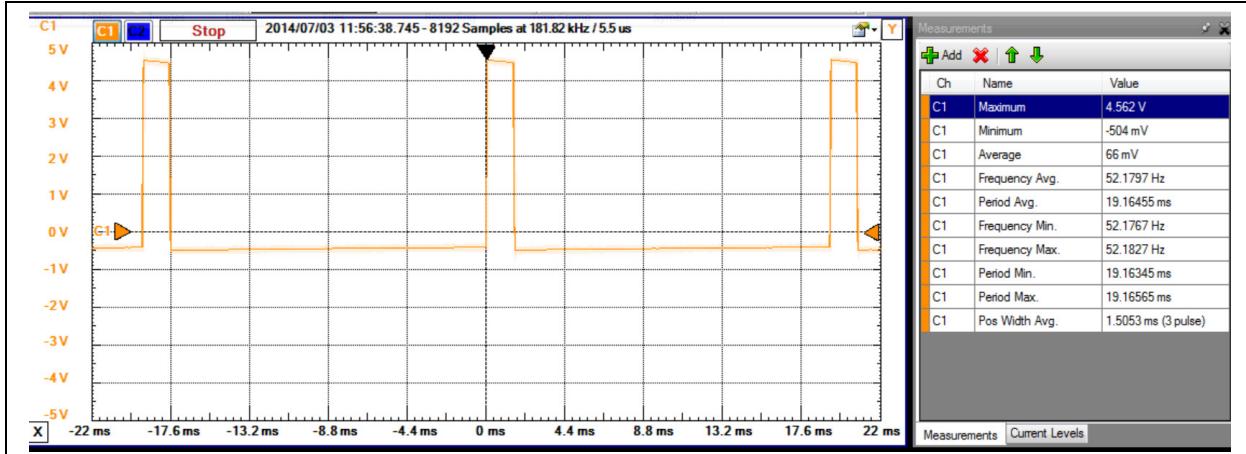
Galileo 1: The pulse frequency is 52 Hz while the pulse width is within the 1 to 2 ms range. The ranges tested were within valid ranges for servos.

Figure 103 Galileo 1: Frequency 52 Hz



Galileo 2: Pulse width and frequency is within permissible ranges and servos are running smoothly.

Figure 104 Galileo 2: Frequency within permissible ranges



Results

G1/G2 Compatible. Tested with 6 ES08A servo motors attached. The PWM stayed between 1.0ms and 2.0ms.

Edison Not Compatible. VIN=11.86V, Pin=4.9V, PWM measuring 0V with multimeter. Board does not power up properly. When a scope was attached to the SDA and the sketch was downloaded again, the board started to function. The issue is still under investigation.

Next steps

- Load with other types of servo motors.

§

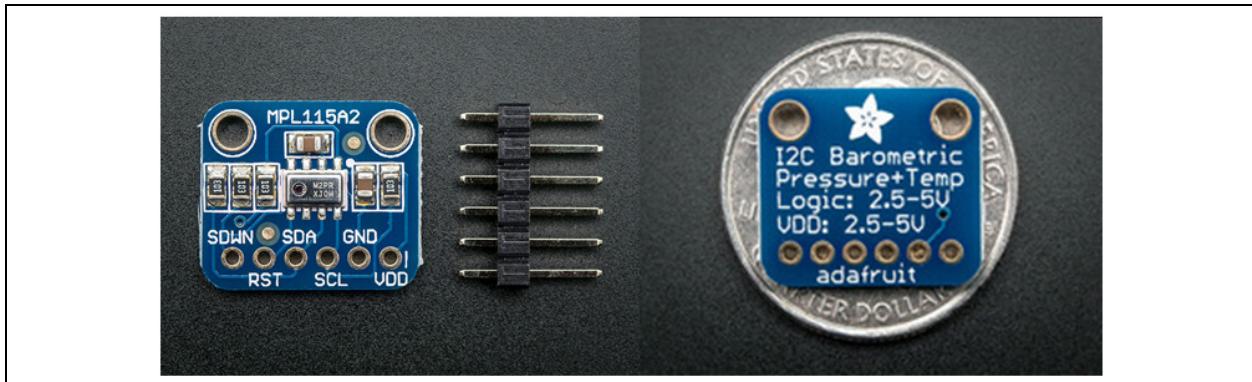
38 Adafruit* MPL115A2 Barometric Pressure/ Temperature I2C Sensor

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 , dated 11/18/12 Name: Adafruit_MPL115A2

Figure 105 MPL115A2 barometric pressure/temperature I2C sensor



Hardware summary

Key info	Description/links
Operating Voltage	3.3V or 5V
Use VIN	No.
Galileo Board Firmware	1.0.2 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematic	http://www.adafruit.com/datasheets/MPL115A2.pdf

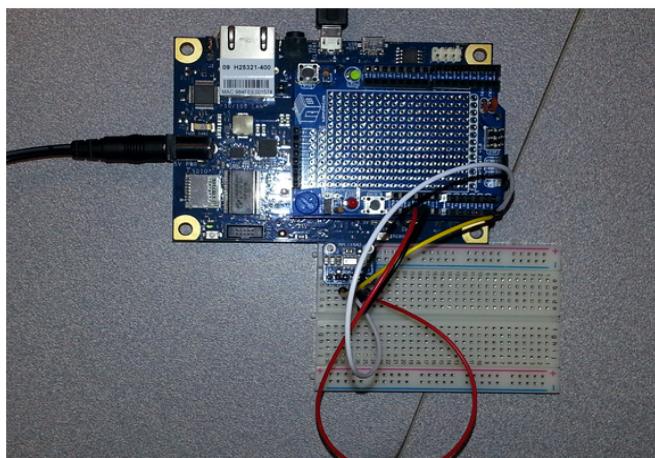
Pin name	Function
SDWN	Shutdown, not used in this example.
RST	Reset, not used for this example.
SDA	I2C, Serial Data Line - connect to A4.
SCL	I2C, Serial Clock Line - connect to A5.



Adafruit* MPL115A2 Barometric Pressure/ Temperature I2C Sensor

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 2.375V to 5.5V - connect to 5V for this example.

Figure 106 MPL115A2 barometric pressure/temperature I2C sensor on Galileo 1



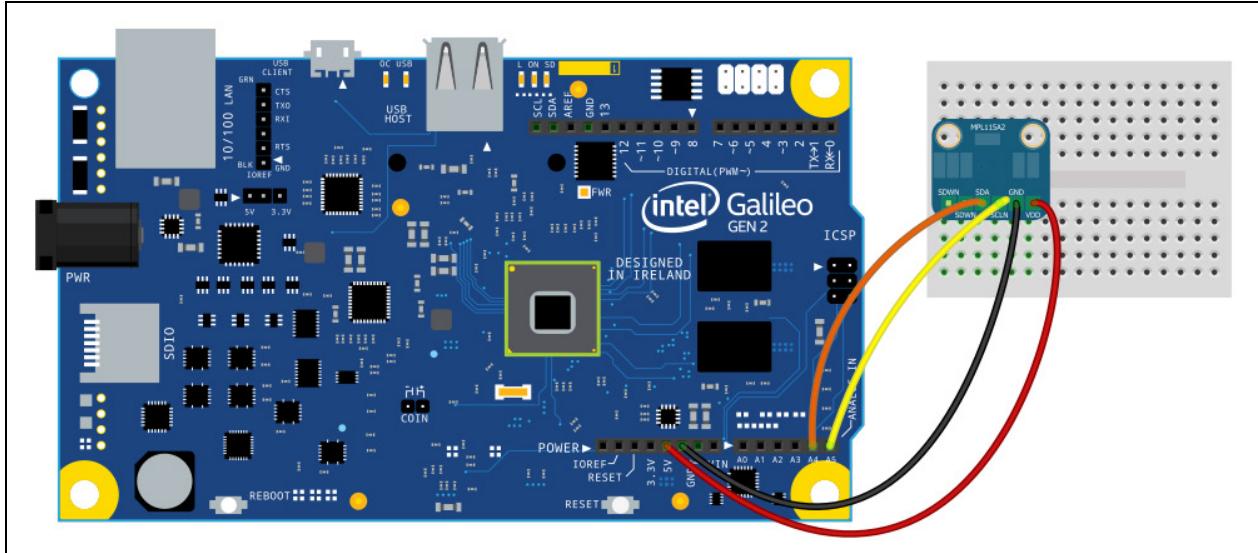
Companion library

The companion library compiles on both Arduino* and Galileo. Two sketches are present in the library called 'getpressure' and 'getPT'.

Compile and upload

Connect the sensor as follows:

Figure 107 Connecting an MPL115A2 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.

```
Sketch: getPT
#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);
}
```

Results

G1/G2/Edison Compatible.

Next steps

- Test in known atmospheric conditions.





39 Adafruit* Electret Microphone Amplifier MAX4466

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

Figure 108 MAX4466 electret microphone amplifier



Hardware summary

Key info	Description/links
Operating Voltage	3.3 or 5V.
Use VIN	No.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
MAX4466	Low-Noise Microphone Amp Datasheet.
CMA-4544PFW	Electret Capsule Microphone Datasheet.

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)

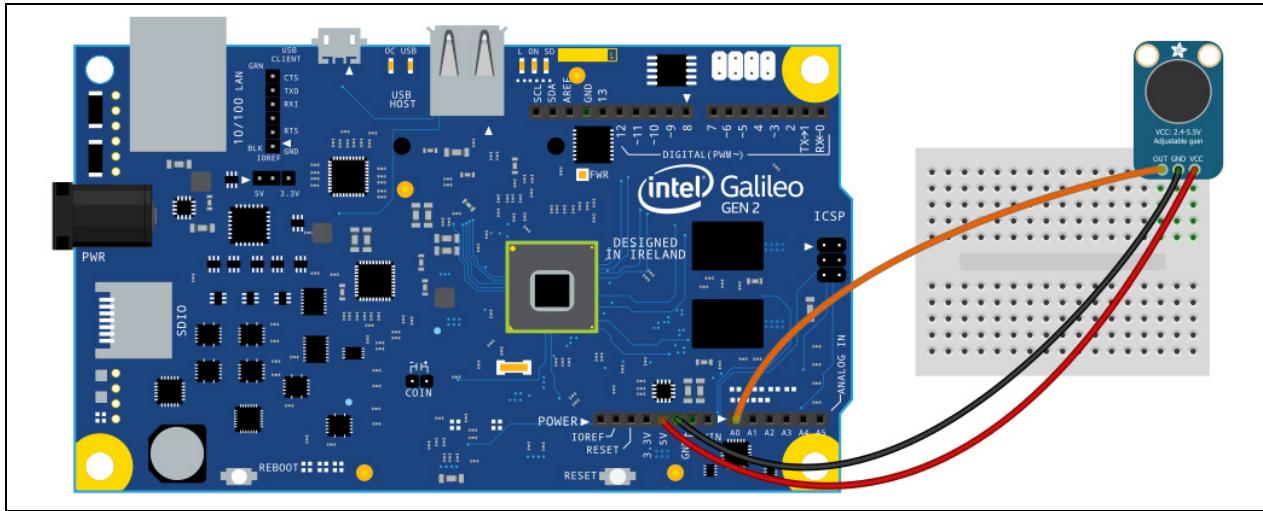
Companion library

No Library.

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 109 Connecting an MAX4466 electret microphone amplifier



Sketch: 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;

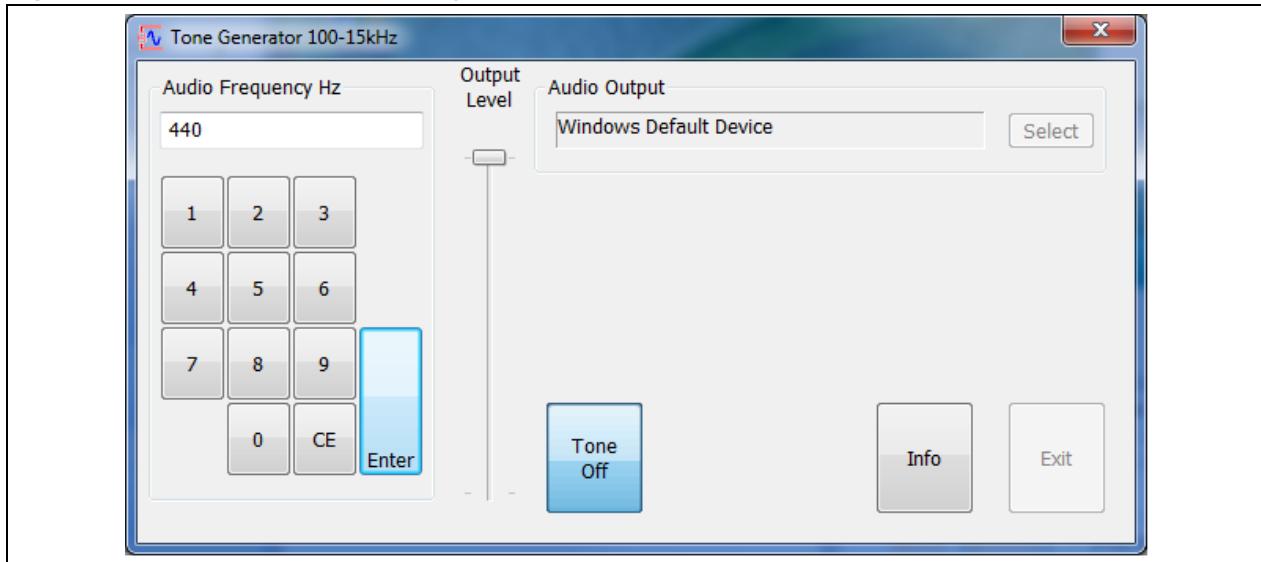
    // collect data for 50 mS
    while (millis() - startMillis < sampleWindow)
    {
```

**Sketch:** Sound Level Sketch

```
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);
}
```

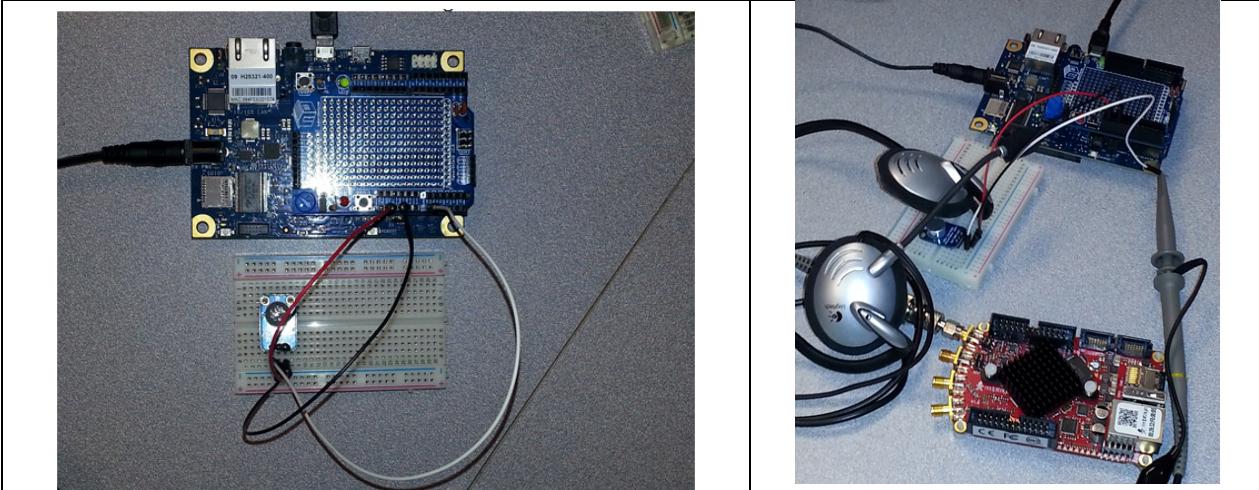
Next, analyze the signal with an oscilloscope by connecting OUT from microphone to the oscilloscope probe. Start the Tone Generator program and set the frequency as 440 Hz. Click the 'Enter' button to start the tone.

Figure 110 Tone Generator display



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 111 Watch the levels change in the serial console

Connect OUT to an oscilloscope and play a signal through the headphones to the microphone (Figure 112).

Figure 112 Oscilloscope signal

Results

G1/G2/Edison Compatible.

Next steps

- Work with the gain feature to see how the amplifier works.

§

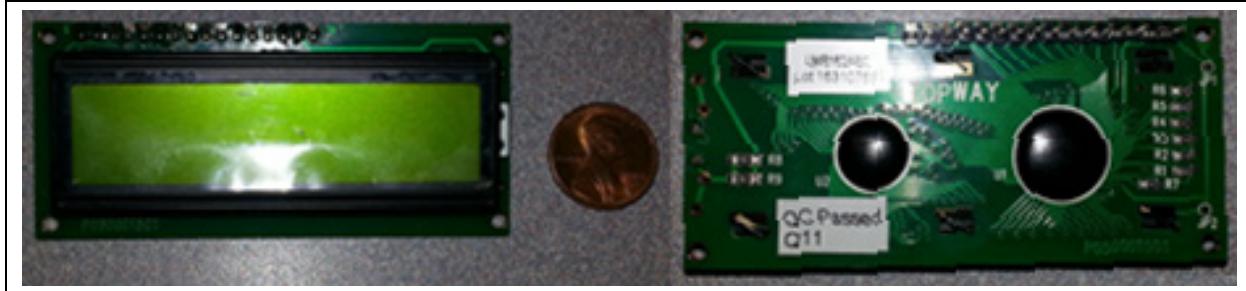
40 Topway* LCD LMB162ABC

Use case

The LCD is a 16×2 character (5×8 dot) panel. It supports all the LCD demos in the Galileo 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	http://topwaylcd.en.ecplaza.net/lcd-module-topway-lmb162abc--250557-1759603.html http://www.topwaydisplay.com/Pub/Manual/LMB162ABC-Manual-Rev0.2.pdf
Library	No extra library, the LCD library that installs with Galileo is used.

Figure 113 Topway LCD LMB162ABC



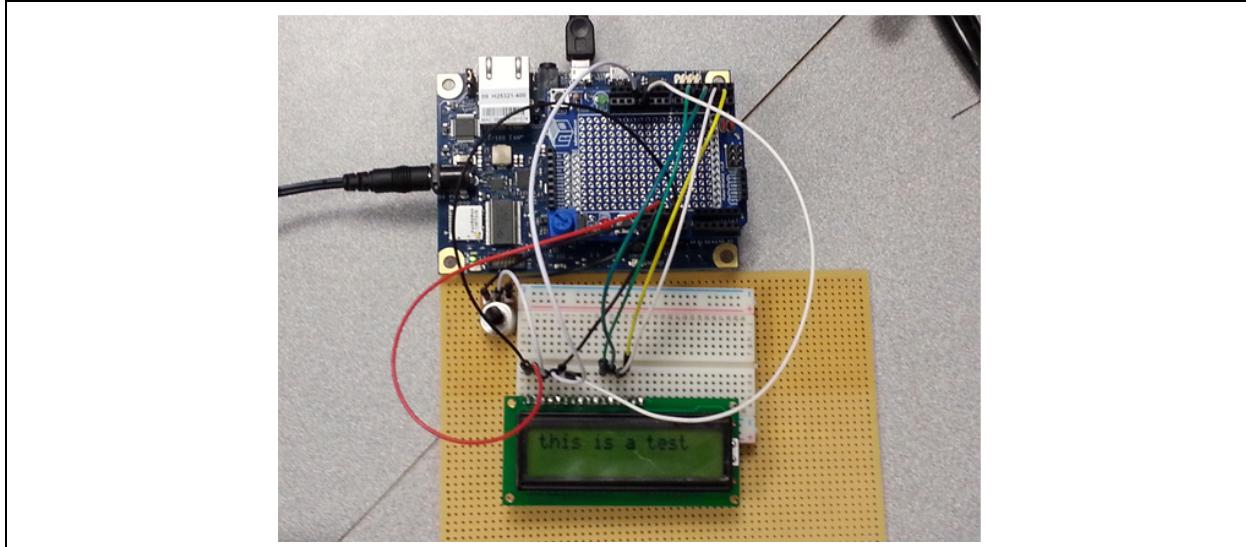
Hardware summary

Key info	Description/links
Operating Voltage	5V
Use VIN	No.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
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 5V.
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

LCD Pin Name	Function/Connection
07 thru 10 not used	Data Bus – connect for 4 bit mode.
11 thru 14	In 4 bit mode, only DB4 thru DB7 are used
DB4 – DB7	11->D5, 12->D4, 13->D3 14->D2 for 4 bit mode In 8 bit mode, DB0 thru DB7 are used.
BLA	Backlight positive supply – not using for this example.
BLK	Backlight negative supply – not using for this example.

Figure 114 Topway* LCD LMB162ABC on Galileo 1



Companion library

No additional library required. This LCD uses the “LiquidCrystal” library included in the IDE.

Compile and upload

Connect the LCD to the Galileo for 4 bit mode. This connection includes a potentiometer to control the contrast for the display.

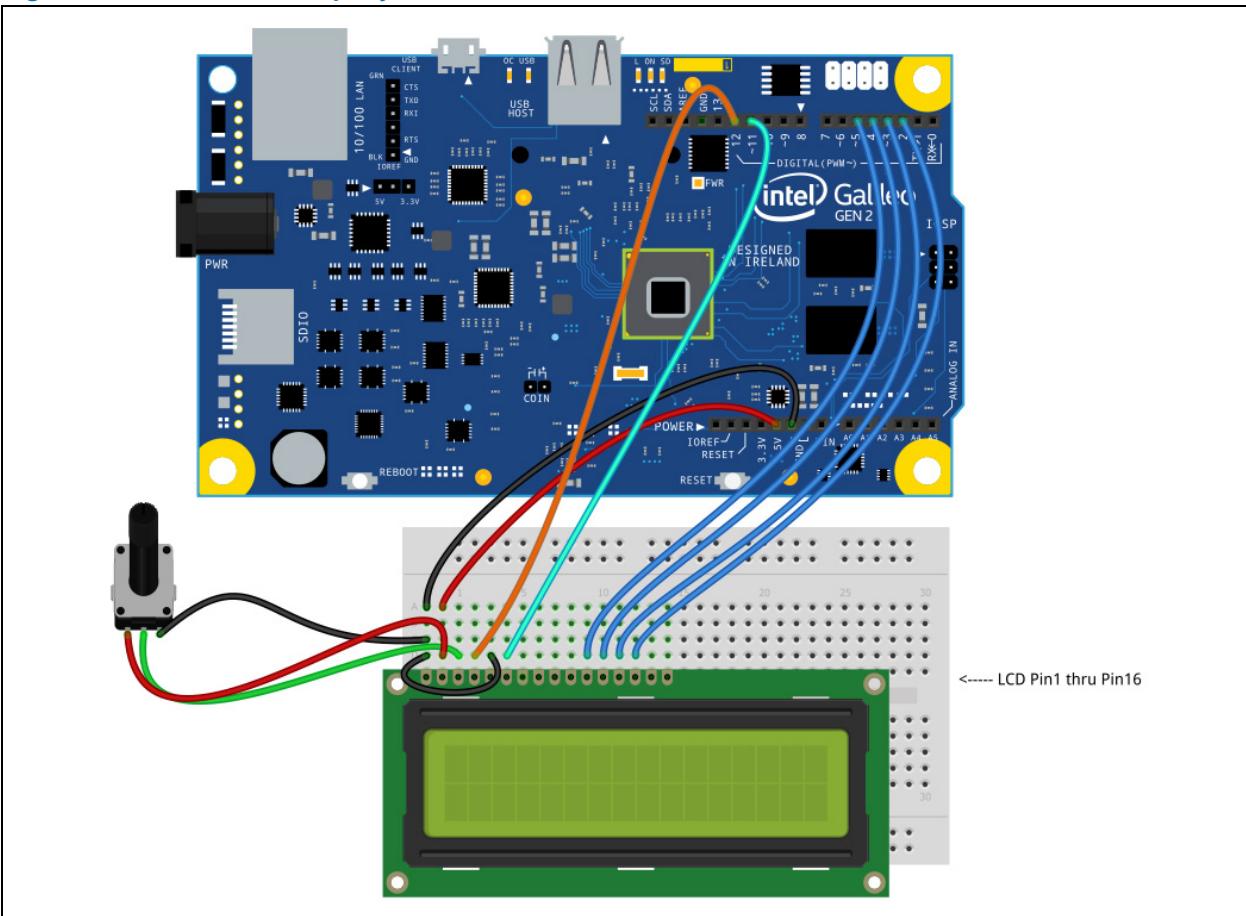
Connections with Potentiometer:

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 5V.
- Other end (not middle) of potentiometer to GND.



Figure 115 Galileo 2 Topway* LCD LMB162ABC Connection



Run the demos that come with the Galileo version IDE. Load the "HelloWorld" sketch. A counter and a text 'Hello World' are displayed.

Results

G1/G2/Edison Compatible.

Next steps

- Test using 8 bit mode.



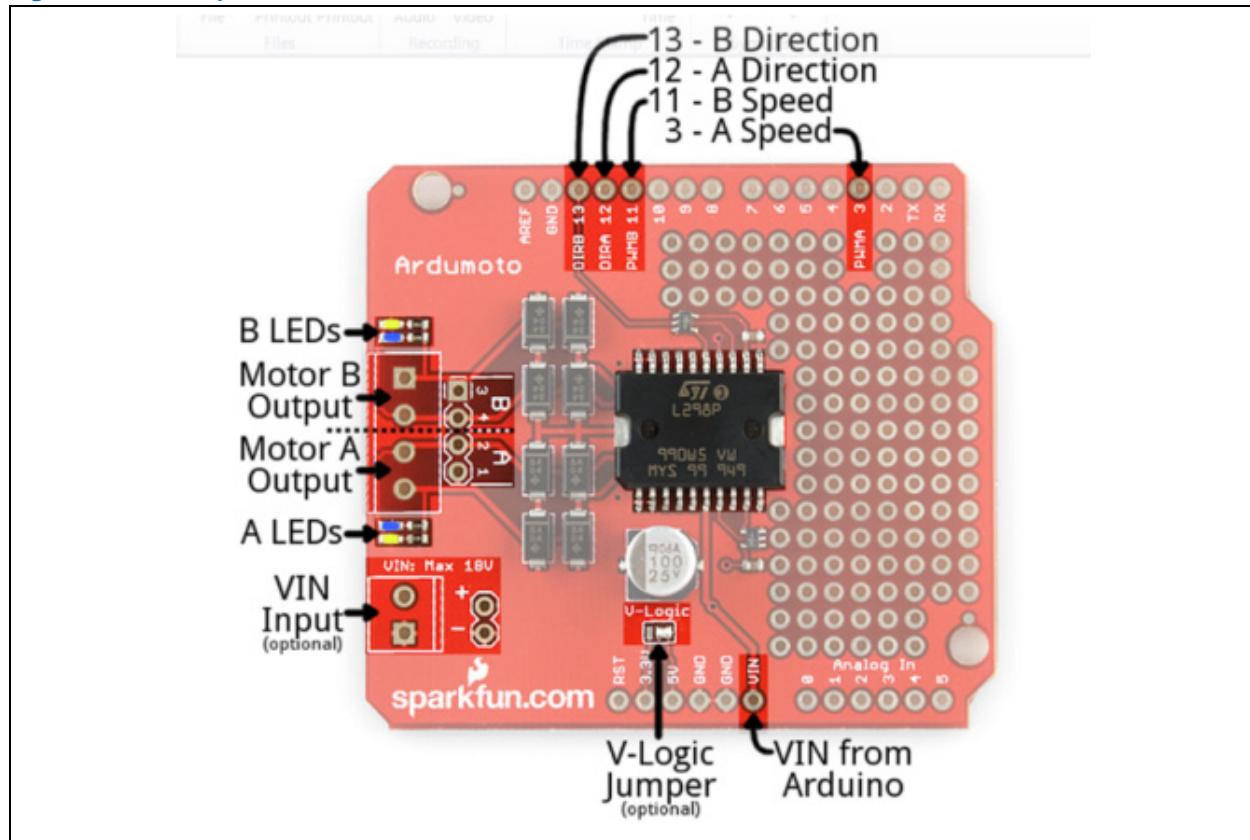
41 SparkFun® Ardumoto Motor-Driver Shield

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 amps 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.
Motors Used	DC1

Figure 116 SparkFun® Ardumoto motor-driver shield



Connect the shield to the Galileo board, jumper the Galileo VIN to the shield VIN, and connect each motor. Set the V-logic jumper (on the shield) to 5V.

The board takes its power from the same VIN line as the Galileo board. Motors tend to draw a lot of current so driving the motor from the Galileo 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 counterclockwise and can use PWM (pulse width modulation) to control the speed.

Hardware summary

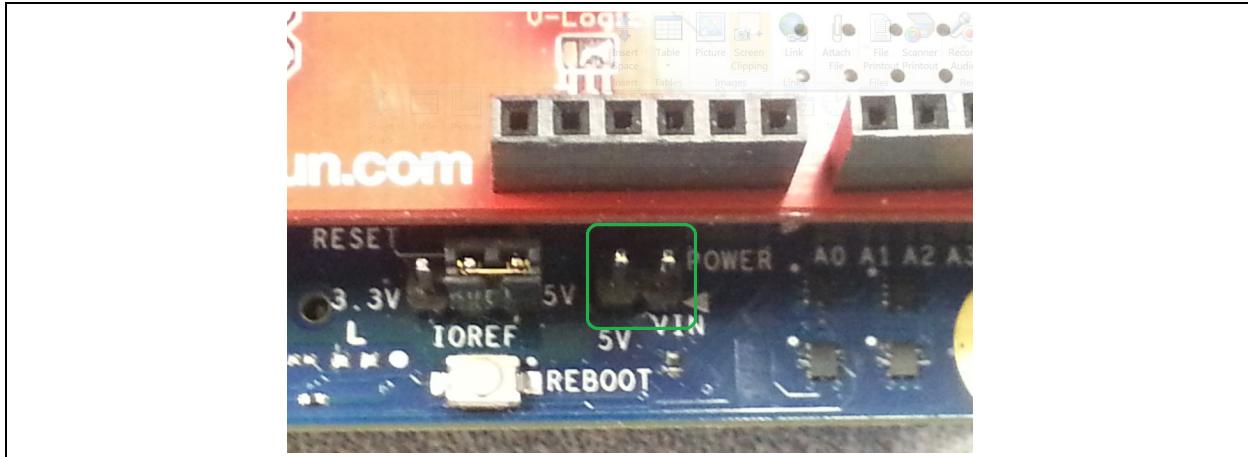
Key info	Description/links
Operating Voltage	5V
Use VIN	Yes, can also hook up an external source to the shield's VIN connection.
Motor Driver	L298 H-bridge https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematic	https://www.sparkfun.com/datasheets/DevTools/Arduino/Ardumoto_v13.pdf

Pin name	Function
PWM A	PWM signal to control the speed of motor A (0=off, 255=max speed) to Galileo pin D3.
PWM B	PWM signal to control the speed of motor B (0=off, 255=max speed) to Galileo pin D11.
DIR A	Control the direction of rotation of motor A (High=CW, Low=CCW) to Galileo pin D12.
DIR B	Control the direction of rotation of motor B (High=CW, Low=CCW) to Galileo pin D13.

Compile and upload

Note: Galileo 1 Only: If using an external power supply, remove the VIN jumper on the board or damage to the Galileo board will occur. Read the [VIN Jumper Overview](#) section.

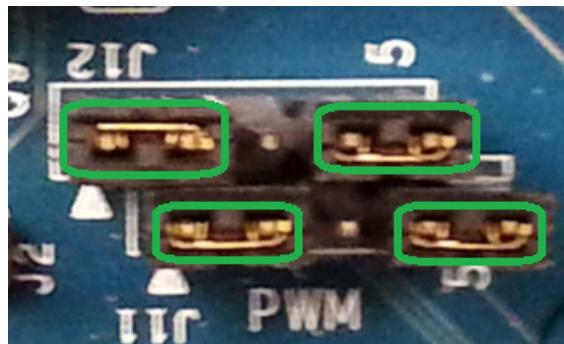
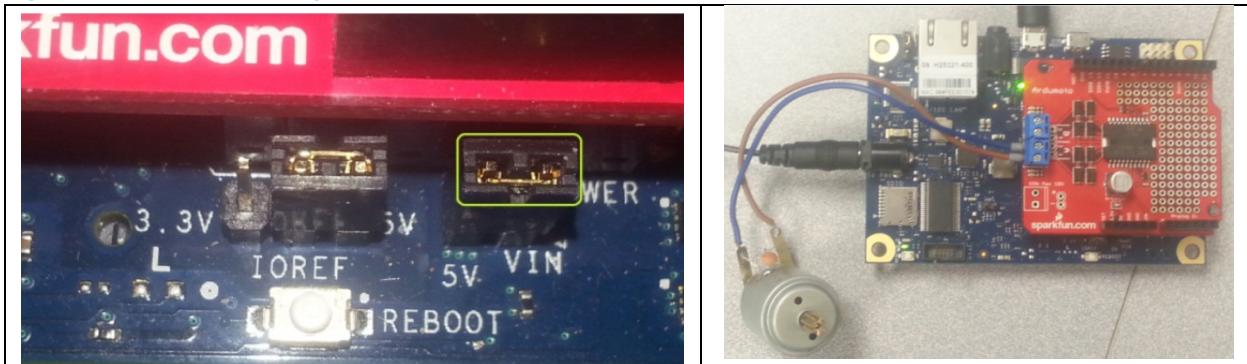
Figure 117 Remove the VIN jumper on the Galileo 1 board



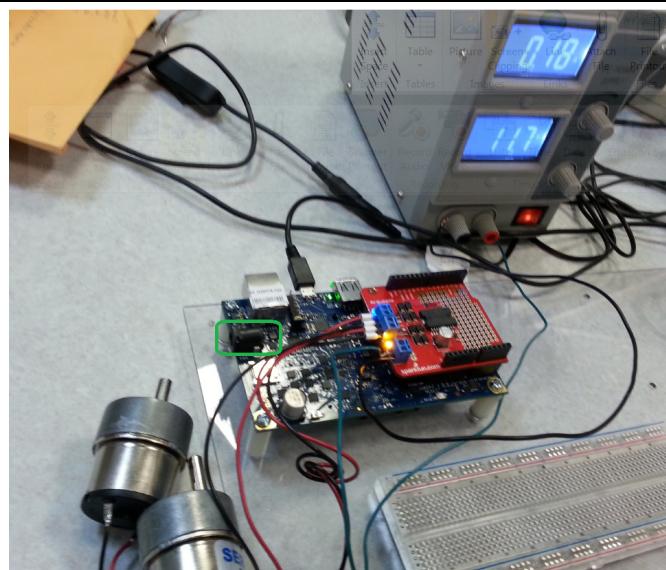
The SparkFun sketch will run on the Galileo without any modifications, but might need a change for Edison as noted below. http://dlnmh9ip6v2uc.cloudfront.net/downloads/Ardumoto/ardumoto_example.ino

Galileo 1: If using Galileo 1 VIN for motor power, connect VIN jumper on the Galileo so the two pins are connected. The motors must be 5V or less and have a current draw of less than 1 amp, or damage to the Galileo card may occur.

Edison: The pins used for PWM are 3 and 11 for motor terminals A and B. By default Edison is set to use pins 3, 5, 6 and 9. See the [PWM section](#) to set the Edison PWM swizzle jumpers so that pins 3 and 11 can be used for PWM. The setPwmSwizzler(3, 5, 10, 11) function call will also need to be called in the sketch when changing from the default PWM pins.

Figure 118 Edison PWM Swizzle Jumper Settings used for Ardumoto**Figure 119** Connecting the Ardumoto motor-driver shield on Galileo 1

Galileo Gen 2 Only: Notice that the Galileo Gen 2 (below) power supply is not connected since the external power supply is providing power to the Galileo and motors.

Figure 120 Connecting the Ardumoto motor-driver shield on Galileo Gen 2

The example below is a modification of the program in the link. The program was modified 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.

**Sketch:** Modified 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);
```

**Sketch:** Modified Sketch

```
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
```

**Sketch:** Modified Sketch

```
analogWrite(pwm_a, 0);      //set both motors to run at 100% duty cycle
(fast)
analogWrite(pwm_b, 0);
}

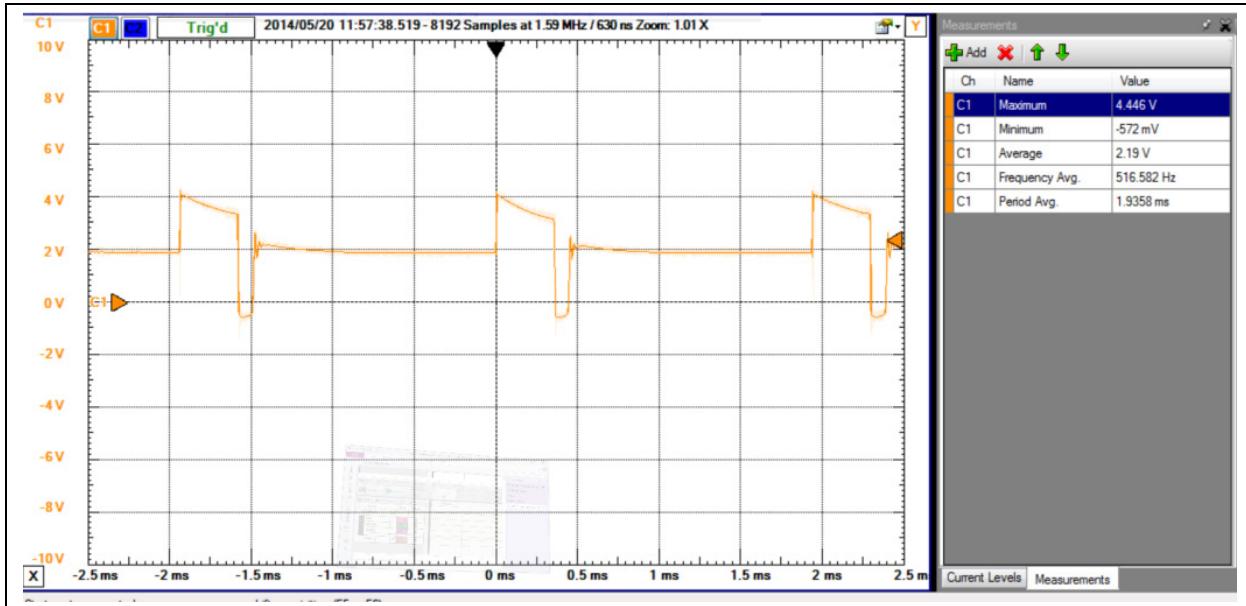
void fadein()
{
    // fade in from min to max in increments of 5 points:
for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5)
{
    // sets the value (range from 0 to 255):
    analogWrite(pwm_a, fadeValue);
    analogWrite(pwm_b, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    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)
}
```

Galileo 1 oscilloscope: 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. This was using the Galileo power supply to provide power for a 5V motor. This reading was taken from the output from the shield to the motor.

Figure 121 First generation Galileo oscilloscope output

Results

G1/G2/Edison Compatible.

The shield was tested to verify that it could run with two motors, change speeds, and change direction.

For Edison, the fade in appears to be jumpy, however, functional.

Next steps

- Test with other shields.

§



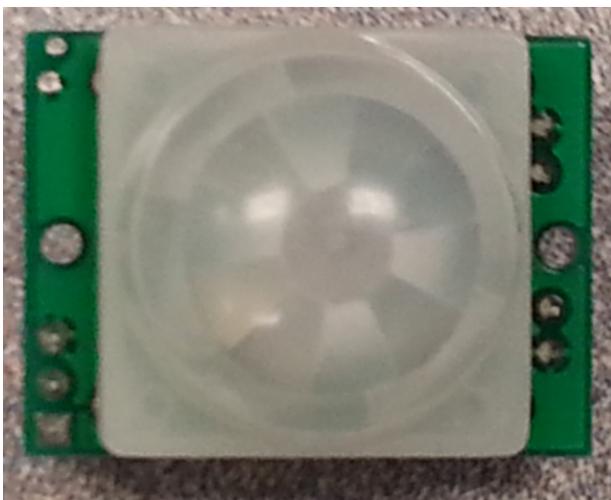
42 DFRobot* Digital Infrared Motion Sensor

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 122 Digital infrared motion sensor



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
IOREF	No.
Use VIN as power source	No.
Operating voltage	5V

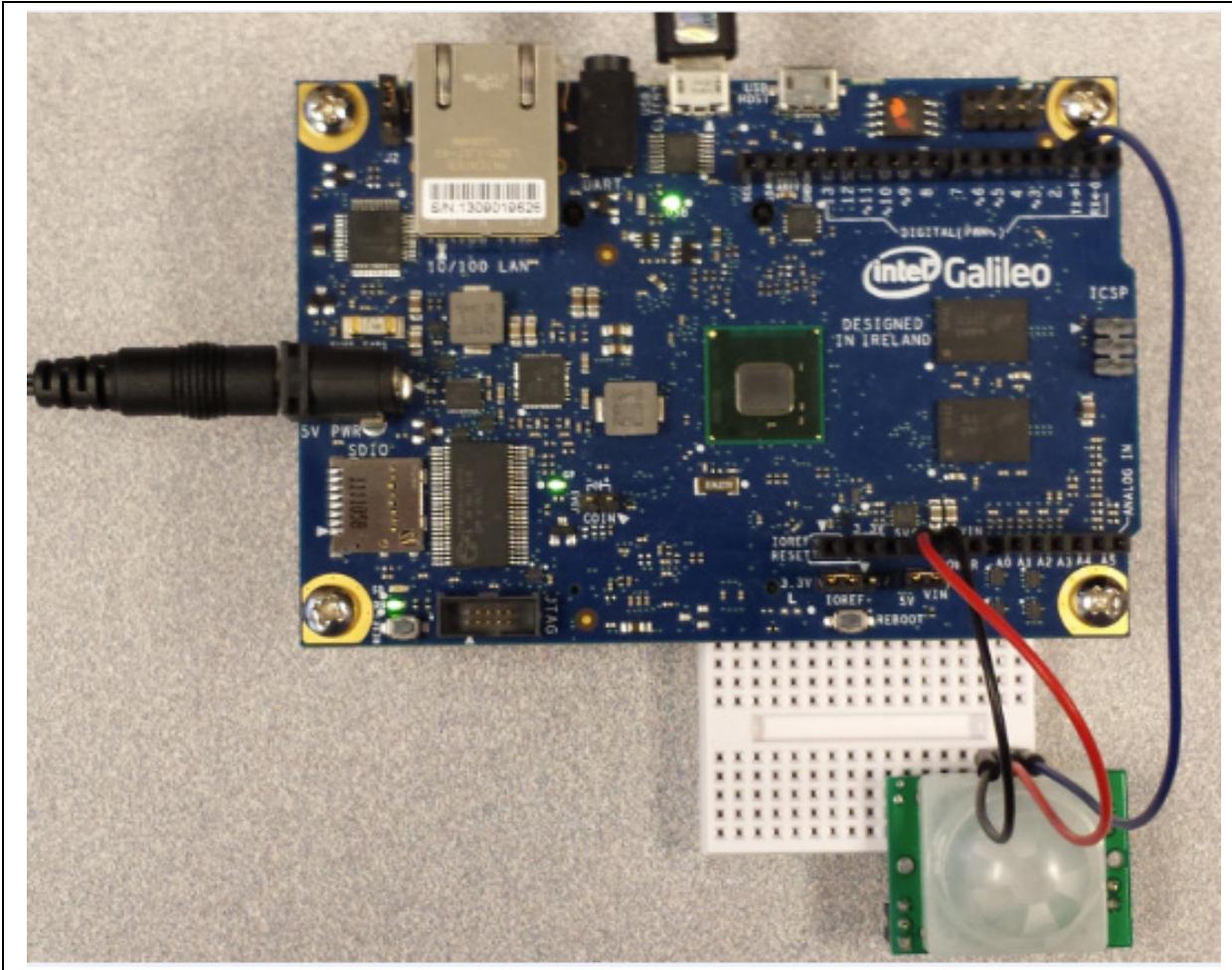
Pin name	Function
-	Connect to GND.
+	Connect to 5V .
OUT	Connect to D2.



Figure 123 Digital infrared motion sensor pins



Figure 124 Connecting a digital infrared motion sensor on Galileo 1



Compile and upload

Set the jumper for high or low sensitivity depending on the application. The following example is jumpered for High (H) sensitivity.

Figure 125 High sensitivity jumper



Wire up the sensor as displayed above and download the example sketch. LED-13 on the Galileo board will turn on when motion is detected. LED-13 will turn off when motion is no longer detected.

The sketch on the Galileo is as follows:

```
Example Sketch
// 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
byte senseMotion = 0;        // current state of motion detector
byte lastSenseMotion = 0;    // last state of motion detector
void setup()
{
    // set the digital pin directions
    pinMode(ledPin, OUTPUT);
    pinMode(motionPin, INPUT);
    digitalWrite(ledPin, LOW);
}
void loop()
{
    // Now watch for motion
    senseMotion = digitalRead(motionPin);
    if (senseMotion != lastSenseMotion)
    {
        digitalWrite(ledPin, senseMotion); // HIGH or LOW
        lastSenseMotion = senseMotion;
        delay(1000);
    }
}
```

Results

G1/G2/Edison Compatible.

Next steps

- None

§



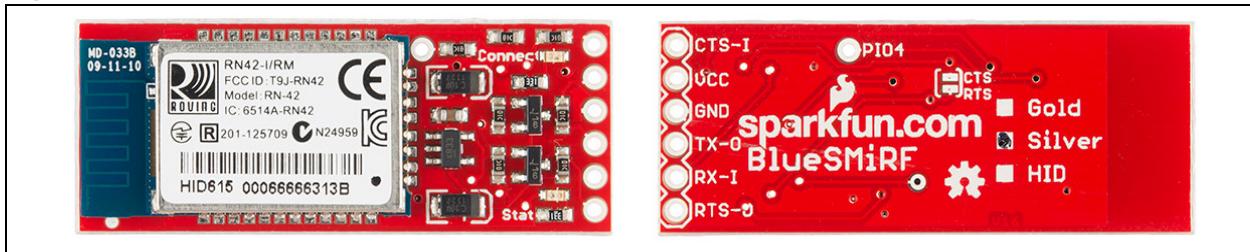
43 SparkFun® BlueSMiRF Silver – Bluetooth® Modem

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 the Galileo board or Arduino® 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-bluesmif/hardware-overview

Figure 126 SparkFun BlueSMiRF Silver Bluetooth modem



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (Build WW37)
IOREF	No.
Use VIN as power source	No.
Operating voltage	3.3V 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.

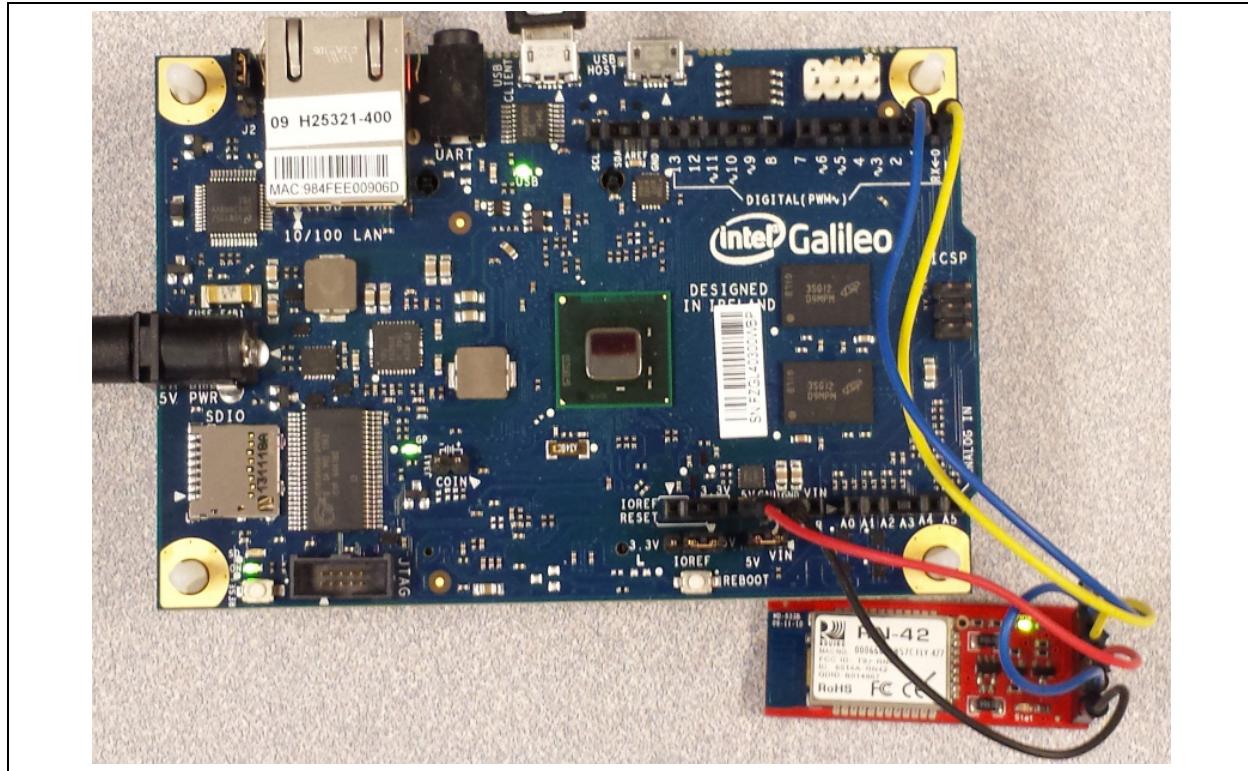


Rx-I	Serial receive	Input	This pin receives serial data from another device. It should be connected to the Tx of the other device.
Tx-O	Serial transmit	Output	This pin sends serial data to another device. It should be connected to the Rx of the other device.
VCC	Voltage supply	Power In	This voltage supply signal is routed through a 3.3V 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-O connected to Galileo D0
- Rx-1 connected to Galileo D1
- VCC connected to Galileo 5V
- GND connected to Galileo GND
- RTS-0 and CTS-1 connected to each other

Figure 127 Connecting BlueSMiRF Silver to Galileo 1



On the Arduino board you can choose the pins that you want to use for the Serial Communication because Software Serial is being used. However, on the Galileo pins D0 and D1 shall be used (with the Tx and Rx swapped between the RN-42 and the Galileo 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.



Compile and Load

Pins D0 and D1 on the Galileo board can be used (with the Tx and Rx swapped between the RN-42 and the Galileo board). "Serial1" provides access to these pins; "Serial" is used for debugging purposes. To use the second serial port on Galileo 2, use "Serial2" with pin D2/D3.

The purpose of the test is to test the serial communication between the module and the Galileo and test the communication with another Bluetooth device. An Android® smartphone and an app called "Bluetooth SPP" were used as the second Bluetooth device.

The sketch used to test the Galileo is as follows:

```
Sample Sketch
// Written by: David P. Lawrence 4/17/2014
// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Arduino
IDE Serial Monitor
TTYUARTClass* gSerialOnePtr = &Serial1; // Galileo, /dev/ttyS0, Tx pin on
the Shield

int ctr = 0;
char string[50];

void setup()
{
    gSerialStdPtr->begin(9600);
    delay(1000*3);
    gSerialOnePtr->begin(9600);
    gSerialStdPtr->println("Setup-Done");
}

void loop()
{
    // If bluetooth received characters show them on the serial monitor
    if (gSerialOnePtr->available())
    {
        int index = 0;
        // Build a string out of the chars until there are no more to receive
        while (gSerialOnePtr->available())
        {
            char rcvData = (char)gSerialOnePtr->read();
            gSerialStdPtr->print(rcvData);
            string[index] = rcvData;
            index++;
        }
        gSerialStdPtr->println();
        string[index] = 0;
        gSerialStdPtr->print("Received Bluetooth data: ");
        gSerialStdPtr->println(string);
    }

    // If characters were typed in the serial monitor send them to bluetooth
    if (gSerialStdPtr->available())
    {
```

**Sample Sketch**

```
int index = 0;
// Build a string out of the chars until there are no more to send
while (gSerialStdPtr->available())
{
    char sendData = (char) gSerialStdPtr->read();
    gSerialOnePtr->print(sendData);
    string[index] = sendData;
    index++;
}
gSerialOnePtr->println();
string[index] = 0;
gSerialStdPtr->print("Sent Bluetooth data: ");
gSerialStdPtr->println(string);
}

ctr++; // "heartbeat" to show loop running
if (ctr > 2000000000) ctr = 0;
if ((ctr % 10000000) == 0)
{
    gSerialStdPtr->print("LoopCnt: ");
    gSerialStdPtr->println(ctr);
}
}
```

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 the Galileo 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-bluesmif/example-code-using-command-mode>

Results

G1/G2/Edison Compatible.

Because the Galileo does not support the SoftwareSerial library, the wiring and the sketch must be different from what is used on the Arduino board. This device tested successfully on Galileo 2 second serial port (D2/D3). Galileo 1 and Edison only has one serial port.

Next steps

- None



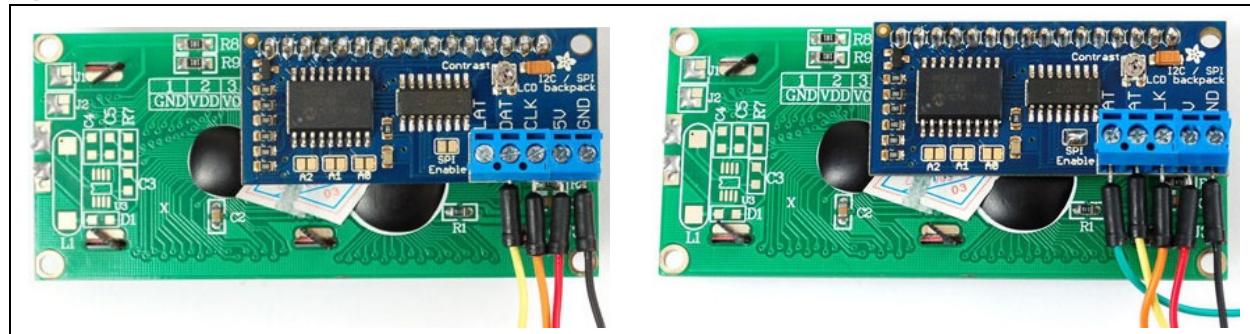
44 Adafruit* LCD Backpack w/I2C, SPI

Use case

This shield is a backpack (add-on circuit) that reduces the number of pins without a lot of expense. By using simple I2C and SPI input/output expanders, it reduces the number of pins (only 2 pins are needed for I2C) 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 I2C. https://github.com/adafruit/LiquidCrystal , dated 100513 For Galileo, 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 128 Adafruit* LCD backpack w/I2C and SPI

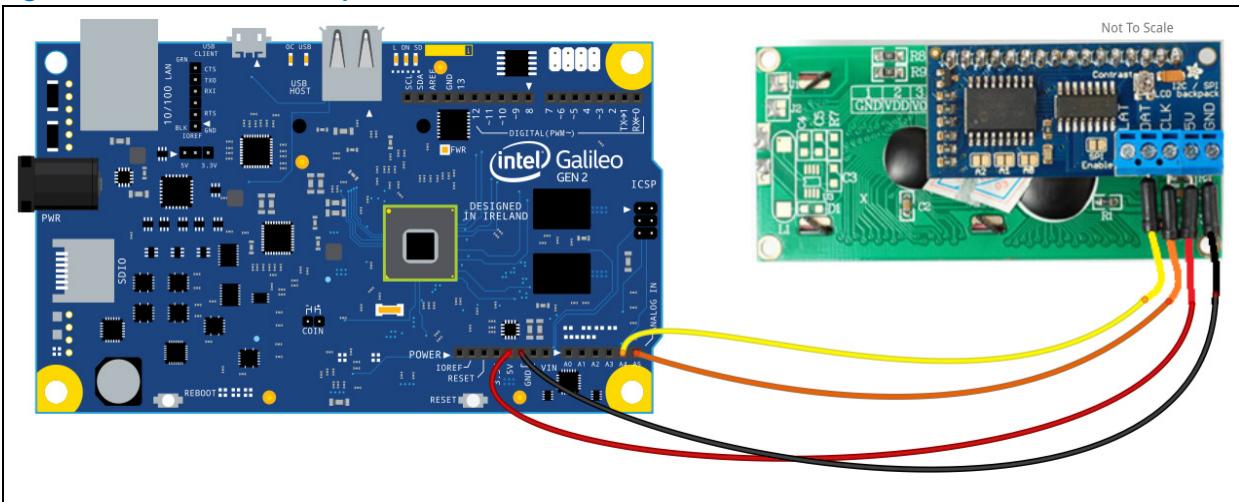


Hardware summary

Key info	Description/links
Operating Voltage	5V
Use VIN	No
Galileo Board Firmware	1.0.3 (Build Release Version)
Edison Board Firmware	1.0.3 (WW37)
Schematic	https://github.com/adafruit/i2c-SPI-LCD-backpack/blob/master/i2cspilcdbackpacksch.png

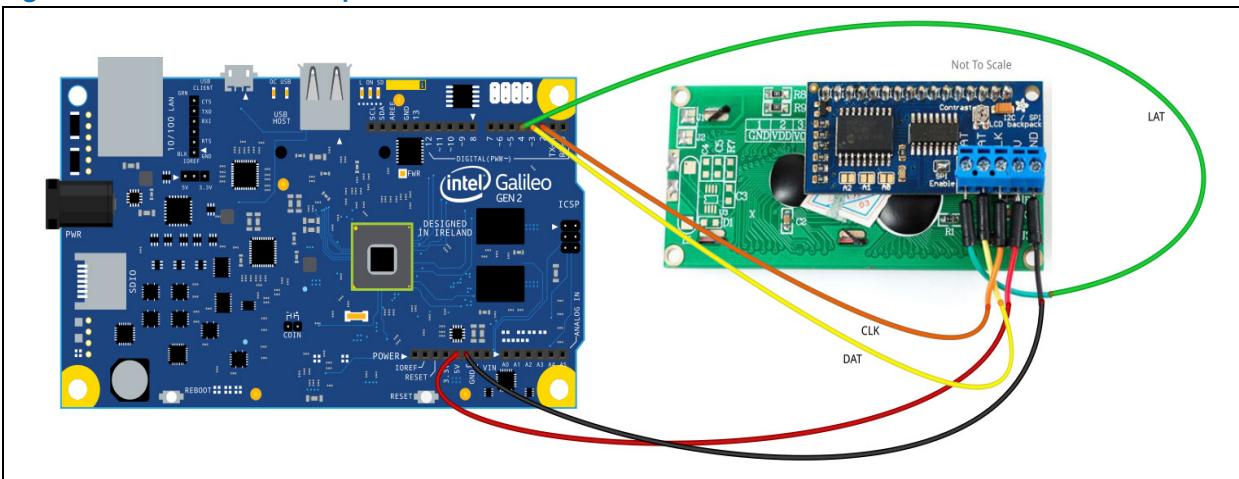


Figure 129 I2C LCD Backpack on Galileo 2.



Breakout Pin Name	I2C Function/Connection
GND	Ground – connect to GND.
5V	Power – connect to 5V.
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	On Galileo 1, the default jumper setting is the first and second pin on the board. Change the jumper to the second and third pin (see I2C Jumper Overview). The jumper must be in this position to instantiate the object: <code>LiquidCrystal lcd(0); // Pass the I2C Jumper</code> The LCD backpack has solder jumpers (A0 thru 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 130 SPI LCD Backpack on Galileo2





Breakout Pin Name	SPI Function/Connection
GND	Ground – connect to GND.
5V	Power – connect to 5V.
CLK	Clock – connect to D2 for SPI.
DAT	Data – connect to D3 for SPI.
LAT	Latch – connect to D4 for SPI.

Companion library

The library is a replacement for the built-in LiquidCrystal library. Backup the original library and replace it with the new Adafruit LiquidCrystal library or rename the library. Upon first compile for Galileo, an error will occur because a "shiftOut" function is missing.

Edit the LiquidCrystal.cpp file to include the "shiftOut" prototype near the top of the file. The "shiftOut" implementation is provided in the sketch.

Change LiquidCrystal.cpp as follows:
Before
#if ARDUINO >= 100 #include "Arduino.h" #else #include "WProgram.h" #endif
After
#if ARDUINO >= 100 #include "Arduino.h" #else #include "WProgram.h" #endif // Prototype void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val);

The library does not work for SPI as is. The SPI needs to be initialized twice; therefore, the SPI constructor needs to be copied as a method and called by the sketch.

In LiquidCrystal.cpp, copy SPI constructor and make it a method.
Before
LiquidCrystal::LiquidCrystal_AF(uint8_t data, uint8_t clock, uint8_t latch) { // Code is not shown }
After
LiquidCrystal_AF::LiquidCrystal_AF(uint8_t data, uint8_t clock, uint8_t latch) { // Code is not shown } void LiquidCrystal::init_spi(uint8_t data, uint8_t clock, uint8_t latch) { // Copy code from constructor to this method. }



The header file needs a prototype also.

In LiquidCrystal.h, add a prototype
Before
LiquidCrystal_AF(uint8_t data, uint8_t clock, uint8_t latch);
After
LiquidCrystal_AF(uint8_t data, uint8_t clock, uint8_t latch); void init_spi(uint8_t data, uint8_t clock, uint8_t latch);

Compile and upload

I2C:

Connect the LCD as described in the I2C 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'.

Note: For Galileo, the sketch has been edited to include an extra "shiftOut" function.

Sketch: HelloWorld_i2c
/* 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 "LiquidCrystal.h" // Connect via i2c, default address #0 (A0-A2 not jumpered) LiquidCrystal 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);



```
Sketch: HelloWorld_i2c
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);
    }
}
```

SPI:

To enable SPI on the backpack, solder over "SPI Enable" next to the LAT pin. The SPI sketch needs to be modified by adding the 'shiftout' function and calling the 'init_spi' method such that the library is initialized again. On Galileo 1, the LCD is function, however, it seems to lag.

```
Sketch: HelloWorld_SPI
// include the library code:
#include "Wire.h"
#include "LiquidCrystal.h"

// Connect via SPI. Data pin is #3, Clock is #2 and Latch is #4
LiquidCrystal lcd(3, 2, 4);

void setup() {
    lcd.init_spi(3,2,4); // For Galileo, initialize twice
    // 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);
```



```
Sketch: HelloWorld_SPI
```

```
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);
    }
}
```

Results

- G1/G2/Edison, compatible for I2C and SPI with changes to the library

§



45 DFRobot* SPI LCD ST7920, 12864ZW

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, namely 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 16x16 dots 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 131 SPI LCD ST7920, 12864ZW



Hardware summary

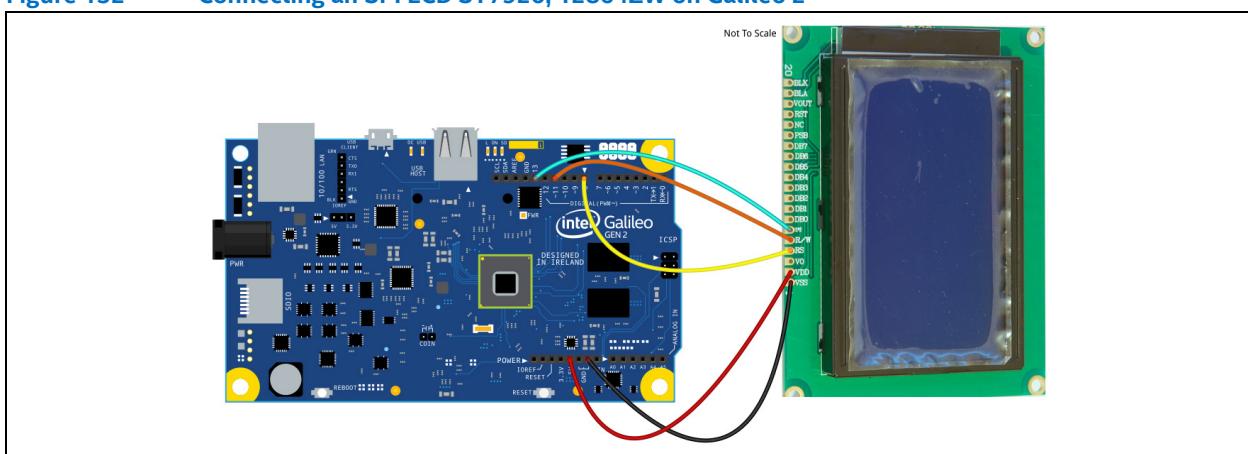
Key info	Description/links
Operating Voltage	2.7V to 5.5V
Use VIN	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Controller	ST7920
Schematic	In datasheet: http://www.dfrobot.com/image/data/FIT0021/ST7920.pdf



Breakout Pin	Function/Connection
VSS	Ground – connect to GND.
VDD	Power – connect to 5V
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 anti-noise characteristics on the OSC line will result in malfunction, or adversely affect the clock's duty ratio.

Figure 132 Connecting an SPI LCD ST7920, 12864ZW on Galileo 2



Companion library

No additional library required. Uses libraries included in the IDE.

Compile and upload

The following sketch compiles and uploads without issue.

The clock divider was modified. While the original sketch had clock division at 128, output was more consistent when set to 64. This is likely due to SPI limitations on the Galileo. Characters on various parts of the screen did not light up consistently which could mean a bad LCD. The brightness was on the dark side. Galileo 2 worked with both clock divisions.

Change initial SPI_LCD sketch as follows:

Before

```
SPI.begin();
SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE0);
SPI.setClockDivider(SPI_CLOCK_DIV128);
```

After

```
SPI.begin();
SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE0);
SPI.setClockDivider(SPI_CLOCK_DIV64);
```



```
Sketch: SPI LCD
/*
 * SPI_LCD 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 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_DIV64);

    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);
}
```



Sketch: SPI LCD

```
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);
```



```
Sketch: SPI LCD
    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);
}
```

Results

G1/G2/Edison compatible.

Works on the Galileo some of the time, output is inconsistent where strange characters appear on G1. Does not work on the Arduino* in its current configuration. Seen inconsistent behavior when the R/W (MISO) pin was not connected.

Next steps

- Add potentiometer brightness control.
- Look for methods to achieve consistent output.

§

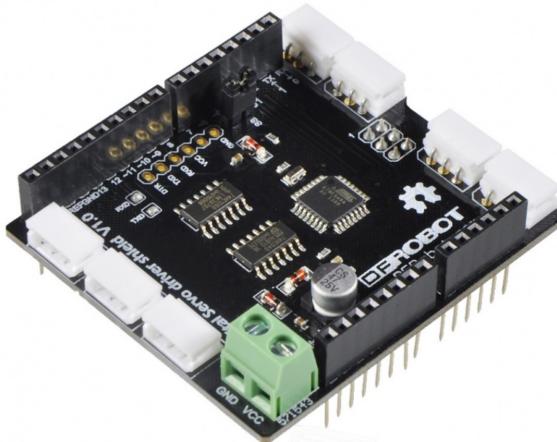
46 DFRobot* Digital Servo Driver Shield, v1.0

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 Galileo 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

Figure 133 DFRobot digital servo driver shield, v1.0

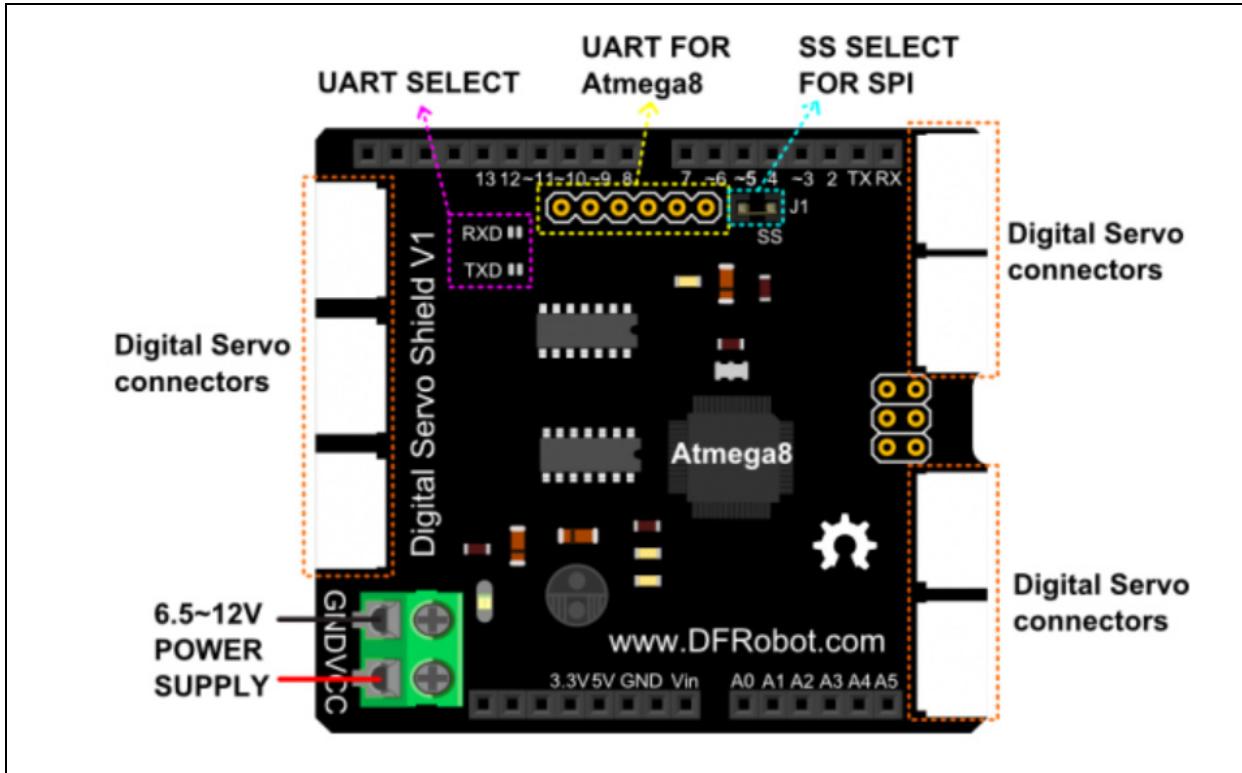


Hardware summary

Key info	Description/links
Galileo Voltage	5V – disconnect VIN jumper to use Galileo.
Motor Voltage	6.5V to 12V from external power source.
VIN as power source	No. This shield requires higher voltage than the Galileo can supply. It is necessary to remove the VIN jumper from the Galileo card because Galileo card will break when using an external power supply with the VIN jumper installed.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
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.



Figure 134 DFRobot digital servo driver shield layout



Pin name	Function
D10	SPI - Chip select - SS
D13	SPI - Clocking
D11	SPI - MOSI
D12	SPI - MISO

Companion library

CDS55Servo library uses SPI for communication.

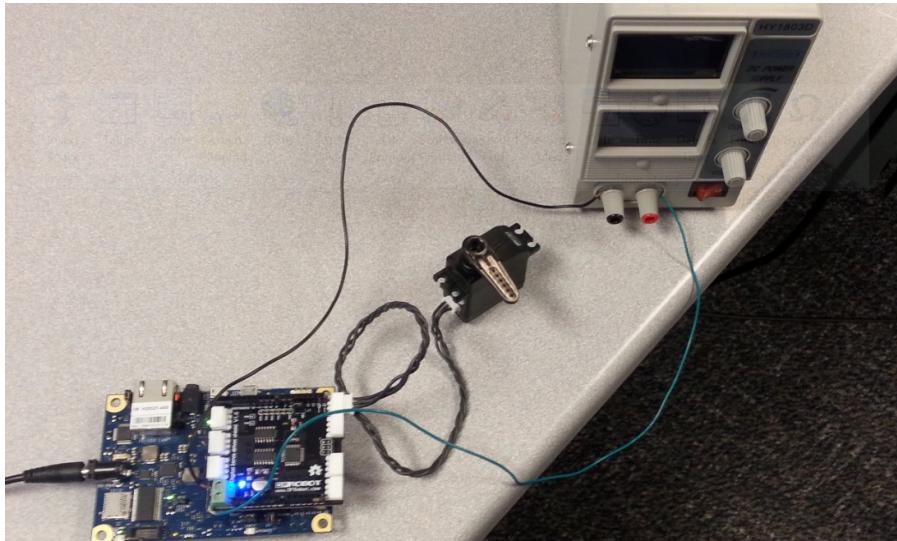
Compile and upload

This sketch will set the servo running continuously clockwise and counter clockwise at a certain velocity.



Figure 135

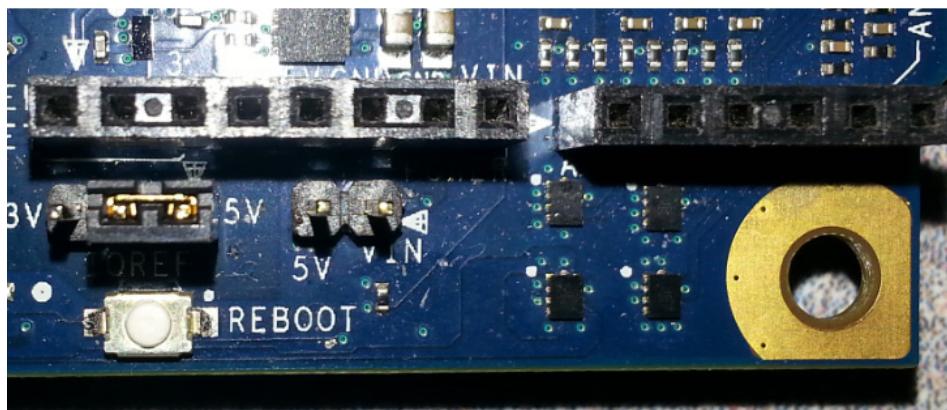
Connecting the DFRobot digital servo driver shield



Note: Galileo 1: Remove the VIN jumper on the Galileo card or damage to the Galileo card will occur. Read the [VIN Jumper Overview](#) section about this jumper.

Figure 136

Galileo 1 - VIN jumper



Servo Sketch

```
#include <SPI.h>
#include <ServoCds55.h>
#include "pins_arduino.h"
ServoCds55 myservo;

void setup (void)
{
  Serial.begin (115200);
  digitalWrite(SS, HIGH);
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV8);
}

void loop () {
```

**Servo Sketch**

```

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':
                myservo.rotate(1, -150);
                //delay(3000);
                break;
            case 'u':
                myservo.rotate(1, 150);
                break;

            case 's':
                //myservo.setVelocity(0);
                myservo.rotate(1, 0);
                break;
        }
    }
}

```

Results

G1/G2 Compatible. Appears to work with demo and the example sketch in this report. Need to understand how the API works in more detail to be able to use.

Edison Compatible. Must use external power supply the Edison will not power the motors and VIN is running around 2V.

Next steps

- Try to work with the other API functions to get exact positions and other features.
- Study how to change the IDs to operating two different servos.

§



47 ThingM® BlinkM RGB LED

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 I2C interface. Multiple BlinkMs can be strung together on an I2C bus, allowing for some amazing light displays.

BlinkMs are also programmable. With ThingMs sequencer software, it is possible to create a mix of colors with different time slices, upload that sequence to the Blink, and then let it rip.

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

Hardware summary

Key info	Description/links
Operating Voltage	5V
VIN as power source	No.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Schematics	In datasheet, page 45: http://thingm.com/fileadmin/thingm/downloads/BlinkM_datasheet.pdf

Pin name	Function
-	Ground – connect to GND.
+	Power – connect to 5V.
c	Clock – connect to SCL.
d	Data – connect to SDA.

Companion library

No additional library required. Uses libraries included in the IDE.

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.

Sketch: BlinkM Example

```
/*
 * 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();
    }
}
```



Sketch: BlinkM Example

```
        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);
    }
}
```

Results

G1/G2/Edison compatible.

§

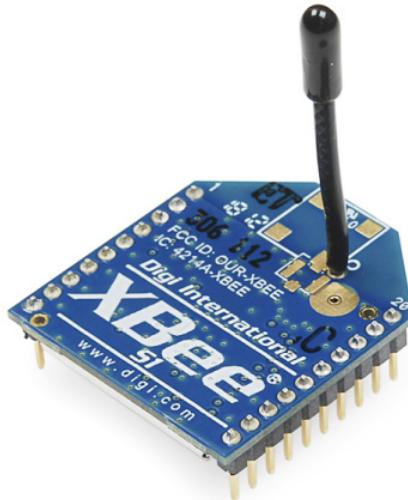
48 XBee® S1 Module w/Arduino® Wireless Shield

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 user 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 proto shield 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 137 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).



Hardware summary

Key info	Description/links
Operating Voltage	5V
VIN as power source	No.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Datasheet	https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf
Schematics	None.

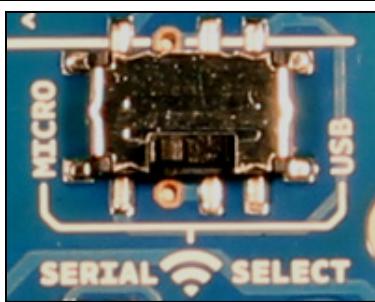
Pin name	Function
D0	"Serial" Rx pin for Arduino, "Serial1" Rx pin for Galileo.
D1	"Serial" Tx pin for Arduino, "Serial1" Tx pin for Galileo.

The module communicates through the existing hardware serial ports. It was not clear, at this time, if other serial ports are available through the proto shield for debugging purposes.

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 board.

Figure 138 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 a Galileo. This means that the Galileo board (using this shield) cannot be used to configure the XBee module. For this test we used an Arduino board to configure the XBee module.

Query steps:

- Insert XBee module into the header in the proper direction.
- The switch "Serial Select" on the proto shield to the "USB" position.

An 'EmptySketch' program shall be downloaded to the Arduino board. If any other program is present, the module will not query.

Sketch: EmptySketch
void setup() {}
void loop() {}

- Selecting the 'discovery' option in the XCTU program.

All known COM ports on the PC will be displayed. Select the COM ports related to the connected Arduino shields. The X-CTU program will query the module through the selected COM ports. The following example shows that two XBee modules are discovered.

Figure 139 Devices discovered output

Devices discovered:

 <input checked="" type="checkbox"/>	Port: COM6 - 9600/8/N/1/N - AT Name: MAC Address: 0013A20040B36318
 <input checked="" type="checkbox"/>	Port: COM4 - 9600/8/N/1/N - AT Name: MAC Address: 0013A2004063B163

The two modules have different firmware version that did 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).

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	Hi and lo different
ATMY	ATMY	
0	1	Different Net Address



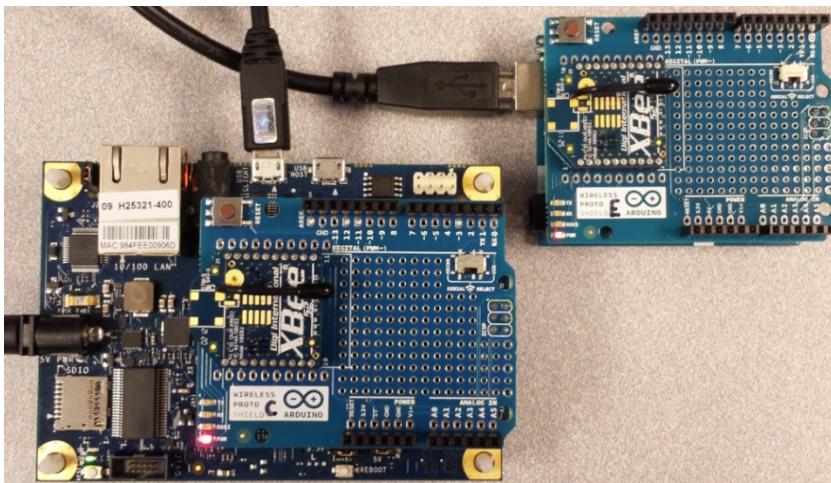
ATDH	ATDH	Dest High
0	0	
ATDL	ATDL	Dest Low
1	0	DL will talk to MY
ATBD	ATBD	
3	3	
ATVR	ATVR	Firmware Version
10E6	10EC	
ATHV	ATHV	
1744	1745	

Normal Operation:

For normal operation on the Galileo board, the 'Serial Select' switch shall be in the "MICRO" position. **This means that the XBee S2 module cannot be configured and a Galileo/Arduino sketch cannot be downloaded.**

There are no other pins involved in the shield. Galileo was tested 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 140 Connecting an XBee S1 module on Galileo 1 and Arduino



Compile and upload

Two XBee S1 modules were configured as described in the 'Module Setup' table. One board is downloaded a sender sketch and the other is downloaded with a receiver sketch.

To test, the End Point Module was placed on the Arduino (with the switch set to USB) and an empty sketch was run. The Controller Module was placed on the Galileo (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 Galileo 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.



```
Sketch: Sender
// MODE: Sender
// A R D U I N O
//HardwareSerial* gSerialShieldPtr = &Serial; // Rx pin 0, Tx pin 1:
Arduino Uno (Serial)
// G A L I L E O
TTYUARTClass* gSerialShieldPtr = &Serial1; // Rx pin 0, Tx pin 1: Galileo
(/dev/ttySO=Serial1)

int gLed = 13;
void setup()
{
    gSerialShieldPtr->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 with the shield attached.
- Set "Serial Select" to "MICRO" to start receiving.

```
Sketch: Receiver
// MODE: Receiver
// A R D U I N O
HardwareSerial* gSerialStdPtr      = &Serial; // Rx pin 0, Tx pin 1: Arduino
Uno (Serial)
HardwareSerial* gSerialShieldPtr = &Serial; // Rx pin 0, Tx pin 1: Arduino
Uno (Serial)
// G A L I L E O
//TTYUARTClass* gSerialStdPtr      = &Serial; // Rx pin 0, Tx pin 1:
Galileo      (/dev/ttyGSO=Serial )
//TTYUARTClass* gSerialShieldPtr = &Serial1; // Rx pin 0, Rx pin 0:
Galileo      (/dev/ttySO =Serial1)

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, observe that LED13 is blinking on the Arduino. This means that the proper characters was transmitted from the Galileo to Arduino via the XBee module. As each character is received, it is buffered until the receiver removes it. If the sender is powered down, LED 13 will continue to blink until all characters are extracted from the buffer. The LED will be solid ON or OFF depending on the last character received.

Now make the Arduino the sender and the Galileo the receiver. There will be a timing issue where the Arduino is sending faster than the Galileo can receive. Powering down the sender will show that the Galileo will blink a bit longer than the previous example.

Make Galileo the sender again, and the Edison the receiver. Data will be sent between the two boards.

Results

G1/G2/Edison compatible.

Next steps

- Determine if there additional serial ports can be used for debugging. There is no obvious wiring for softserial/other Galileo serial port.
- Determine if the entire XBee library works on the Galileo method.
- Attempt to query modules on a nonstrict network.
- Attempt to upgrade firmware. Perhaps this will fix the query issue.

§

49 ITEAD* Bluetooth* Shield, Master/Slave

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 141 ITEAD* Bluetooth* shield (Master)



Hardware summary

Key info	Description/links
Operating Voltage	5V DC
VIN as power source	No.
Galileo Board Firmware	1.0.2 (Production Release) 1.0.3 Not tested since the board is dead.



Edison Board Firmware	1.0.2 (WW31) 1.0.3 Not tested since the board is dead.
Device Type	BR/EDR Bluetooth.
Class of device	1f00
AT Firmware Version	2.0-20100601
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.

Shield Pin Name	Function (no wiring required)
Tx/Dn Jumper	Shield, Dn is selectable for Tx (D0, D1, D4 through D7). Arduino, set to D7 (default) Galileo, set to D1
Dn/Rx Jumper	Shield, Dn is selectable pins for Rx (D0, D1, D4 through D7). Arduino, set to D6 (default) Galileo, set to D0
CMD/DAT Switch	Set to 'CMD' to configure and query the state of the shield. Set to 'DAT' for processing data being transmitted from the Rx/Tx
Status Led	Slave: PAIRABLE, Fast Blink PAIRED, Steady Blink CONNECTED, Blink-Blink-Pause Master: INITIALIZED, INQUIRING, Fast Blink PAIRED,

Companion library

No library is required; however, a PC serial port program is required to configure the shield. Configuration was not possible on a Galileo/Edison. The possible reason is probably due to the multiplexing of the D0/D1 serial ports.

Using an Arduino board, download an empty sketch to the Master Shield. This insures 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. Enter the following commands to query default settings.

**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 attempted to setup were as follows.

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 gather is 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.

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.



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* 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 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:

AT Commands: The Master connection to the slave	
AT+STATE?	
+STATE:INITIALIZED	
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.



```

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 = &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; // Galileo, /dev/ttyGS0, Tx pin
TTYUARTClass*   gSerialTwoPtr = &Serial1; // Galileo, /dev/ttySO, Rx pin
bool            gGalileo      = true;
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, however, it was garbled. The results are inconclusive and more debugging is required.

Results

G1/G2/Edison InConclusive as a Master. It was functional as a slave for some versions of firmware.

Next steps

- Determine if you ned to write a sketch to connect (the example above used an empty sketch) or match brands.
- Determine if additional configuration needs to be done for the Master/Slave.
- Write a sketch such that the shield can be configured on Galileo/Edison.

§

50 Adafruit* Digital Accelerometer

Use case

This is a low power, 3-axis MEMS accelerometer module with both I2C and SPI interfaces. There are 4 sensitivity ranges from +/- 2G to +/- 16G. It supports output data rates ranging from 10hz to 3200Hz 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 I2C	https://learn.adafruit.com/adxl345-digital-accelerometer
Library I2C	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 142 ADXL345 Digital Accelerometer



Hardware summary

Key info	Description/links
Operating Voltage	3.3V - 5V
Use VIN	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)

The breakout board has connection for I2C and SPI.

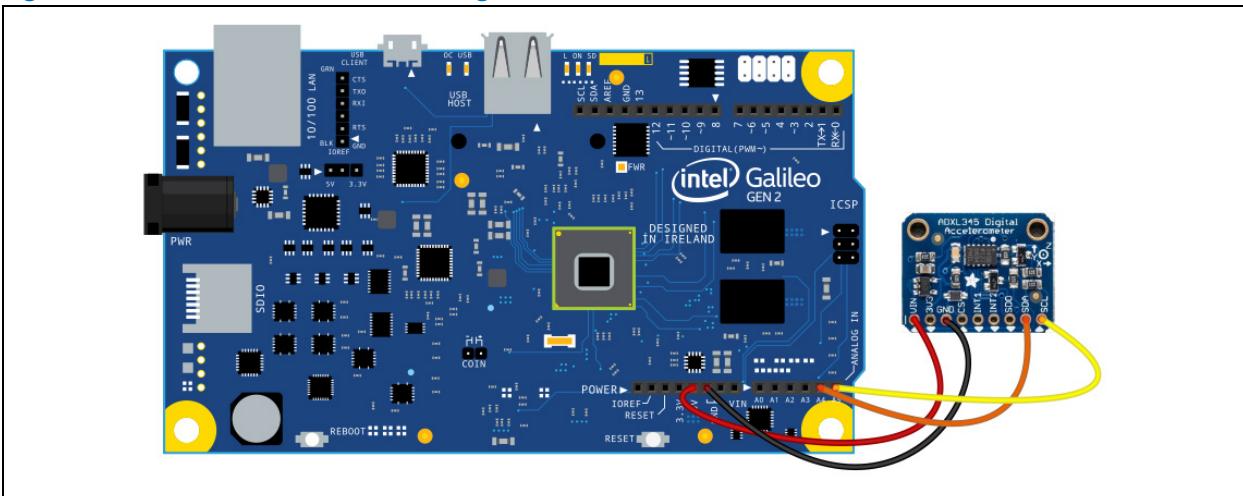
I2C Configuration:

I2C is a four wire connection. See the diagram below that uses the SDA and SCL pins on the Galileo board.

I2C Pin Connections	
Breakout Pin	Function
VIN	Power, connect to 5V
GND	Ground, connect to GND
SDA	Data Line, connect to SDA or A4 Address in Adafruit_ADXL345_U.h file: <code>#define ADXL345_ADDRESS (0x53)</code>
SCL	Clock Line, connect to SCL or A5



Figure 143 I2C Connection for the Digital Accelerometer to Galileo 2

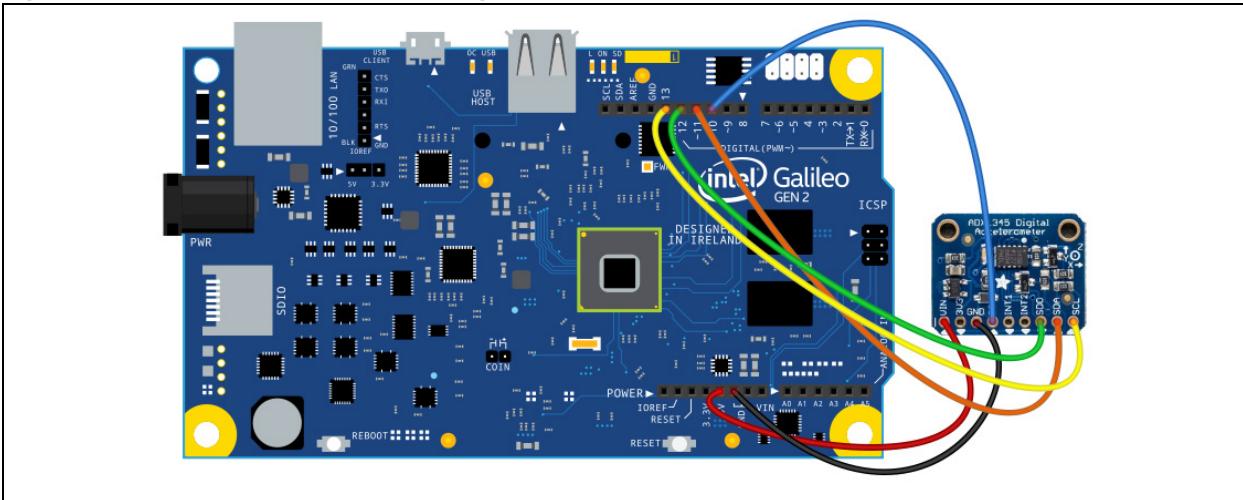


SPI configuration

SPI is a five wire connection. At this time, the library that comes with the breakout board only supports I2C. Sparkfun tutorial, see link above, was used to test for SPI.

SPI Pin Connections	
Pin name	Function
VIN	Power, connect to 5V
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 144 SPI Connection for the Digital Accelerometer to Galileo 2





Companion library

I2C Configuration:

Two libraries are required since once library inherits from another. The libraries are called 'Adafruit_ADXL345_U' and 'Adafruit_Sensors'. The first library is used in the example sketch and has one provided example called 'sensorstest' which is an I2C example.

This library does not compile on Galileo. The Arduino compiler seems to allow non-defined virtual function while the Galileo compiler disallows them. Changes to header files resolved the compile issue.

Changes were made to the parent class to better describe the present implementation. The subclass does not define an 'enableAutoRange' method; therefore, the 'virtual' keyword was removed from the parent class. The parent class did not define a 'getEvent' and a 'getSensor', therefore, the virtual function was changed to a pure virtual function since the subclass defined it. This sketch was still functional on Arduino with these changes.

Change Adafruit_Sensor.h as follows:
Before
// These must be defined by the subclass virtual void enableAutoRange(bool enabled) {}; virtual void getEvent(sensors_event_t*); virtual void getSensor(sensor_t*);
After
// These must be defined by the subclass void enableAutoRange(bool enabled) {}; virtual void getEvent(sensors_event_t*) = 0; virtual void getSensor(sensor_t*) = 0;

SPI Configuration:

The sketch, from the SPI Overview link, compiles and download with no problems.

Sketch: Sparkfun SPI version
//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 DATA_X0 = 0x32; //X-Axis Data 0 char DATA_X1 = 0x33; //X-Axis Data 1 char DATA_Y0 = 0x34; //Y-Axis Data 0 char DATA_Y1 = 0x35; //Y-Axis Data 1 char DATA_Z0 = 0x36; //Z-Axis Data 0 char DATA_Z1 = 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;

**Sketch:** Sparkfun SPI version

```
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){
```



```
Sketch: Sparkfun SPI version
//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 address 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 Chip's Select pin high to end the SPI packet.
  digitalWrite(CS, HIGH);
}
```

Compile and upload

Running the I2C 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
```



Sketch Output for 'sensortest'					
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					
-10,-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					

Results

G1/G2/Edison Compatible for I2C

G1/G2/Edison Compatible for SPI

Next steps

- None

§

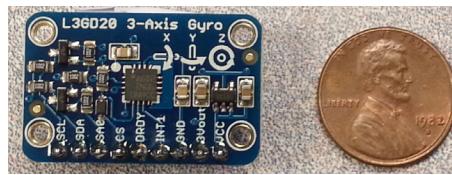
51 Adafruit* L3GD20 Gyro

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 I2C 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

Figure 145 L3GD20 3-Axis Gyro



Hardware summary

Key info	Description/links
Operating Voltage	3.3 or 5V.
Use VIN	No.
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
L3GD20 Datasheet	Low-Noise Microphone Amp Datasheet.

Sketches can use the SPI or I2C interface so some of the pins have a dual purpose in the descriptions.

Pin name	Function
SCL	I2C clock/SPI clock - Connect to SCL for I2C or the Galileo pin used for SPI clock (pin 13).
SDA	I2C data/SPI data input - Connect to SDA for I2C or the Galileo pin used for MOSI (pin 11).
SA0	I2C least significant bit of device address/SPI MISO – Connect to ?? for I2C or the Galileo pin used for MISO (pin 12)
CS	SPI chip select – Connect to Galileo 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 5V



Companion library

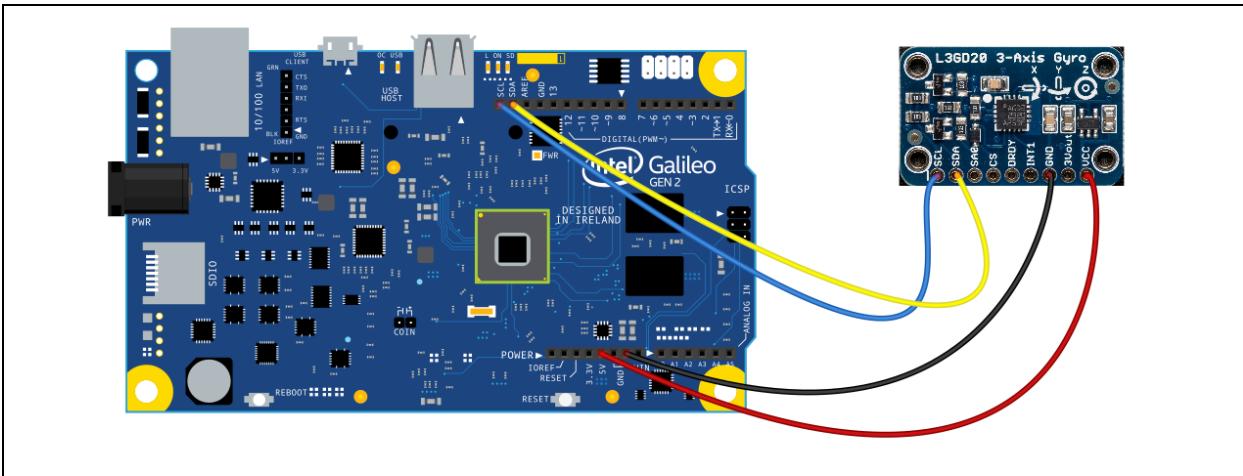
The sample sketch works on Galileo when the I2C sketch option is used. SPI mode uses for the L3DG20 uses LSB (least significant byte) so this will not work with the Galileo.

Compile & Test

Make the following L3DG20 pin connections:

- SCL to SCL (or A5) on Galileo
- SDA to SDA (or A4) on Galileo
- VCC to 5V on Galileo
- GND to GND on Galileo

Figure 146 Connecting an L3DG20 Gyro using the I2C interface



The following sketch was taken from the library installation. Uncomment the `#define USE_I2C` line for using the I2C interface and wiring configuration.

Upload the sketch and the board on different axes to see the data change for all 3 axes.

```
Sketch: Adafruit L3GD20 test sketch that is 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
 ****
```



Sketch: Adafruit_L3GD20_test sketch that is installed with the library

```
#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);
}
```

Twist the board on different axes and notice the values of angular rate changing according to the pivoting axis.



Figure 147

Serial Terminal Output

The screenshot shows a Windows-style terminal window titled "Serial Terminal Output". The window has a "Send" button at the top right. The text area contains the following repeated data:

```
X: 572 Y: 572 Z: 573  
X: 572 Y: 572 Z: 572  
X: 572 Y: 572 Z: 573  
X: 572 Y: 572 Z: 572  
X: 572 Y: 572 Z: 573  
X: 572 Y: 572 Z: 573
```

Results

I2C – G1/G2/Edison Compatible.

SPI – No library.

Next steps

- Verify the data output and try to find an SPI library.

§



52 Datan* Analog Feedback Micro Servo - Metal Gear

Use case

This is a metal-gear '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 148 Analog Feedback Servo



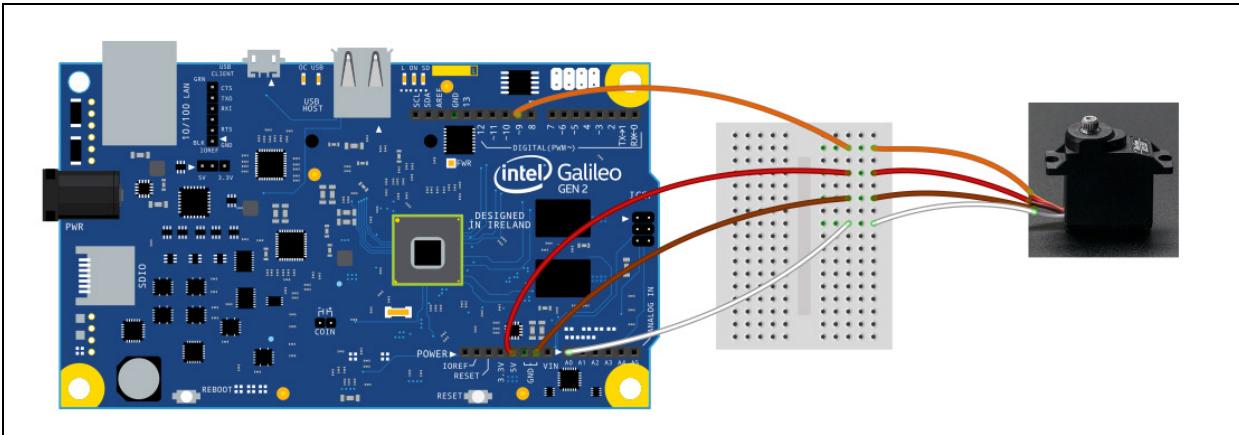
Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Use VIN as power source	No
Operating voltage	6V
Torque	25 oz*in/1.8*cm @6V
Speed	0.1 sec/60° @6V

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 149 Connecting the Analog Feedback Micro Servo to Galileo 2



Companion library

No library required

Compile and upload

Connections:

- White wire (feedback) to A0
- Orange wire (servo) to D9
- Red to 5V
- Brown to GND

Run the following on the Galileo is as follows and the servo should sweep through 180 degrees and show the position in the serial terminal.

```
Example Sketch to Calibrate the feedback and sweep 180 degrees
#include <Servo.h>

Servo myservo;

// 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
```



```

Example Sketch to Calibrate the feedback and sweep 180 degrees
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);
}

```



Example Sketch to Calibrate the feedback and sweep 180 degrees

```
maxDegrees = maxPos;  
delay(2000); // make sure it has time to get there and settle  
maxFeedback = analogRead(analogPin);  
}
```

Results

G1 Not compatible

G2/Edison Compatible

Next steps

- See if there are firmware changes for G1 that will provide a better PWM

§

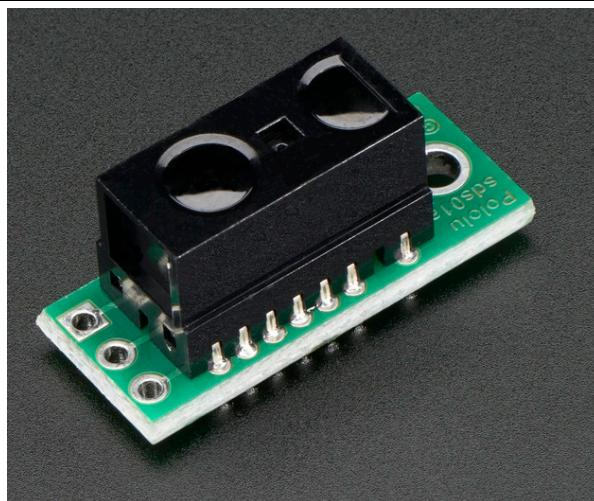
53 Sharp* Digital Distance Sensor w/Pololu* Carrier

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 150 Sharp* Distance Sensor

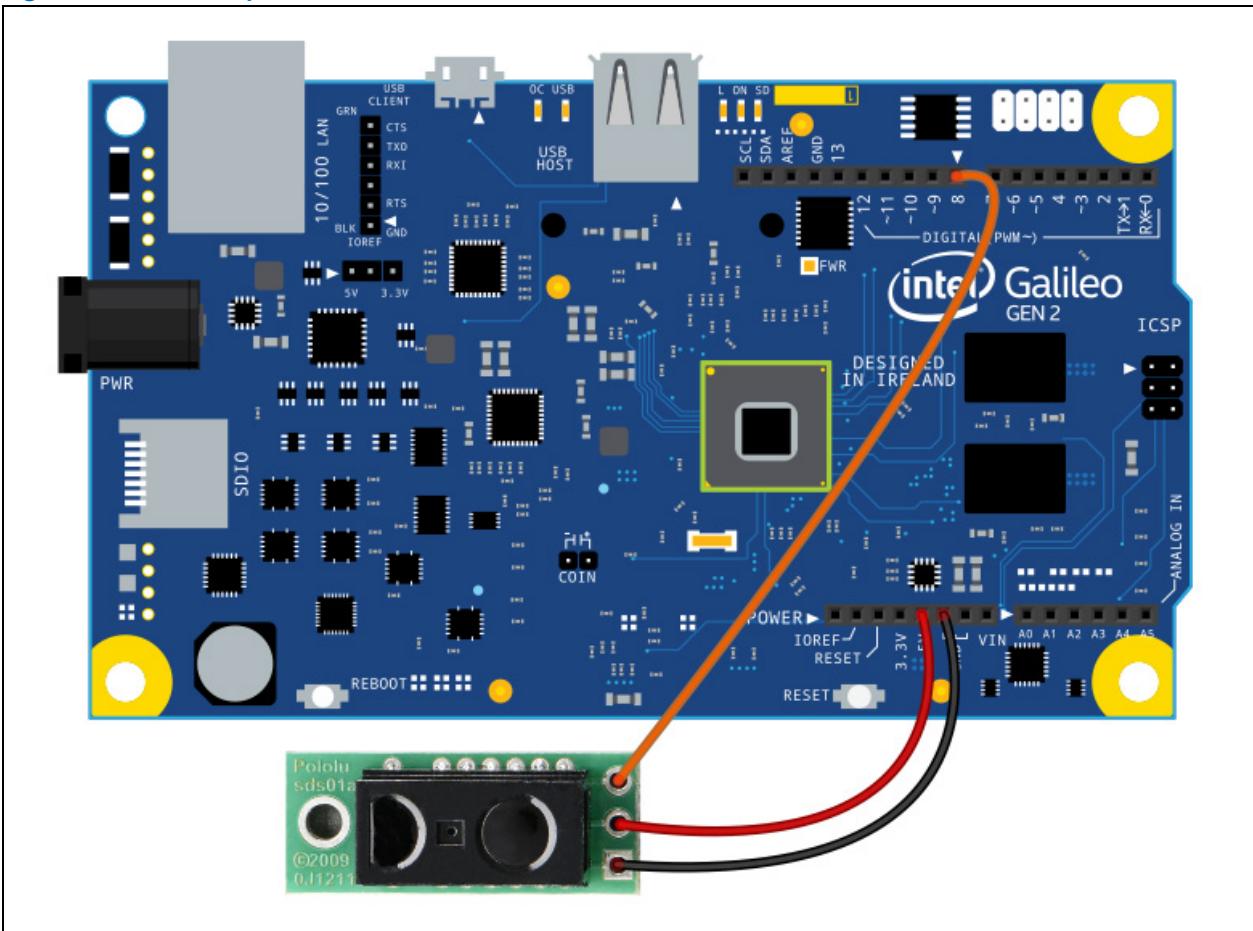


Hardware summary

Key info	Description/links
Operating Voltage	2.7 to 6.2 V
VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematics	None
Sensor Datasheet	http://www.pololu.com/file/0J154/GP2Y0D810Z0F.pdf
Distance	0.8" to 3.9"



Figure 151 Sharp* Distance Sensor on Galileo 2



Breakout Pin	Function with board connections
GND	Ground, connect to GND
VIN	Voltage, connect to 5V
OUT	Digital, connect to D8

Companion library

No Library required.



Compile and upload

The example sketch below 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 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 an 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.

Results

G1/G2/Edison Compatible.

Next steps

- None



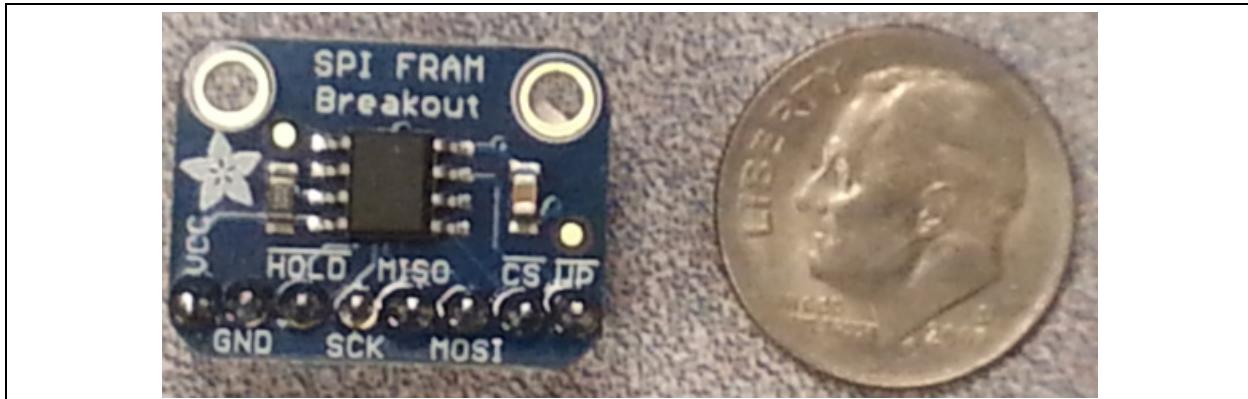
54 Adafruit* Nonvolatile FRAM Breakout SPI

Use case

FRAM, or Ferroelectric Ram is similar to Dynamic random-access memory except with a ferroelectric layer instead of a dielectric layer. This gives it stable handling for writing non-volatile memory fast. With the SPI FRAM breakout board FRAM storage can be added to a Galileo project for lower power usage and faster write performance. It's excellent for low-power or inconsistent-power datalogging 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	Library Date: 5/22/14

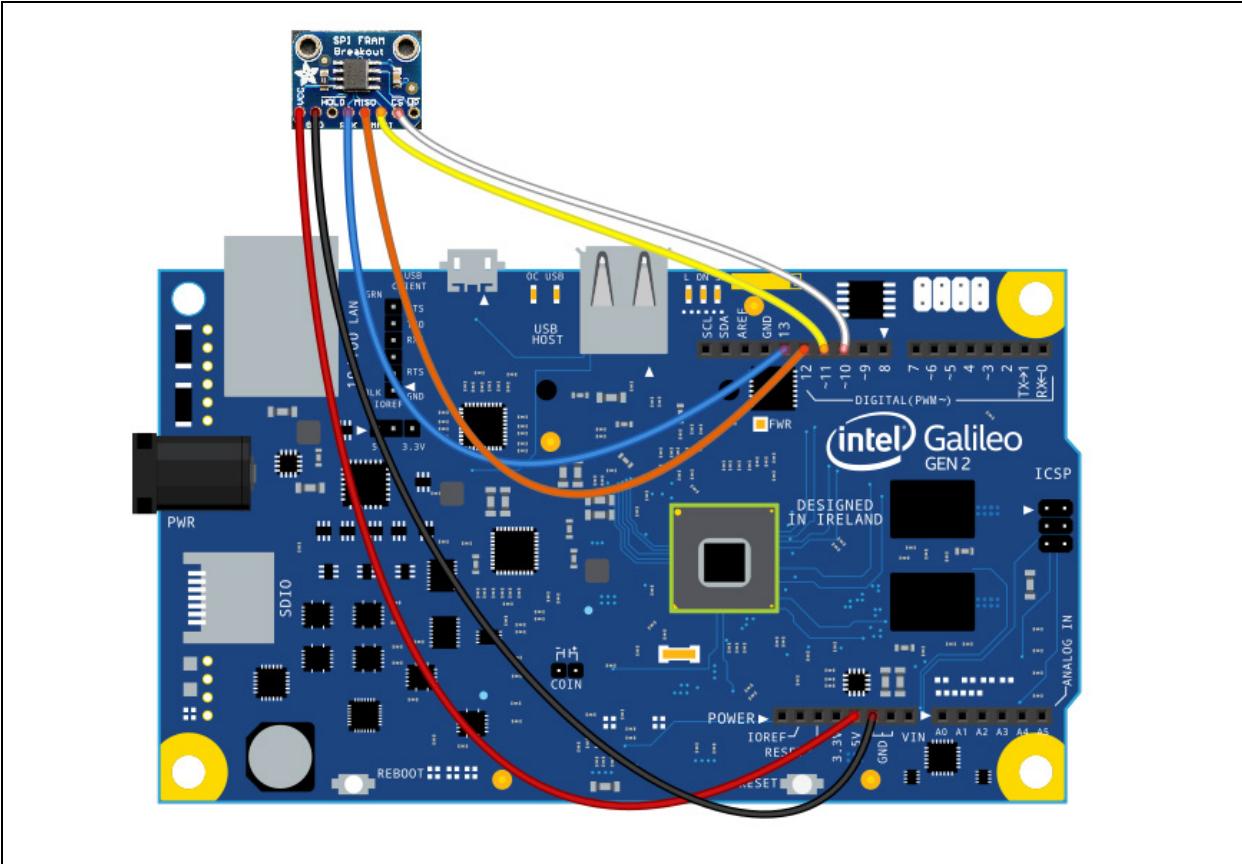
Figure 152 SPI FRAM Breakout



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Use VIN as power source	No
Operating voltage	3.3V or 5V

Figure 153 Connecting the FRAM Breakout to Galileo 2



Pin	Function
VCC	Connect to 5V.
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

Companion library

The library will compile after commenting a header file from Adafruit_FRAM_SPI.

```
// #include <util/delay.h>
```

The MB85RS64V sketch that is installed with the library will compile and run



Compile and upload

Install the library and make the modification from the previous section. Wire the Galileo 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).

```
Sketch to 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);
  }
}
```



Sketch to read/write to FRAM

```
// 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) {

}
```

Results

G1 Not Compatible. Extremely slow for logging data real time this might not be a good option. Only tested with 100 bytes which was taking 20 – 30 seconds, therefore, it would not be compatible for most applications.

G2 Compatible – 5.830 seconds to write/read all 8192 bytes on the FRAM using Galileo Gen2. The Arduino UNO ran the same sketch in 15.168.

Edison Compatible – 3.444 seconds



Next steps

- Find an application that can use this breakout board for logging or storing other data.

§

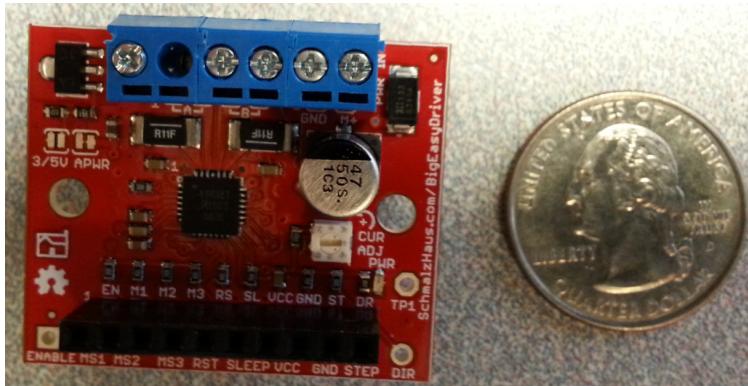
55 Sparkfun* Big Easy Driver v1.2

Use case

The Big Easy Driver is a stepper motor driver board for bi-polar stepper motors up to 1.4A/phase. It is based on the Allegro A4983 stepper driver chip. It is a chopper microstepping driver which defaults to 16 step microstepping mode. It can take a maximum motor drive voltage of around 35V and includes on-board 5V/3.3V 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

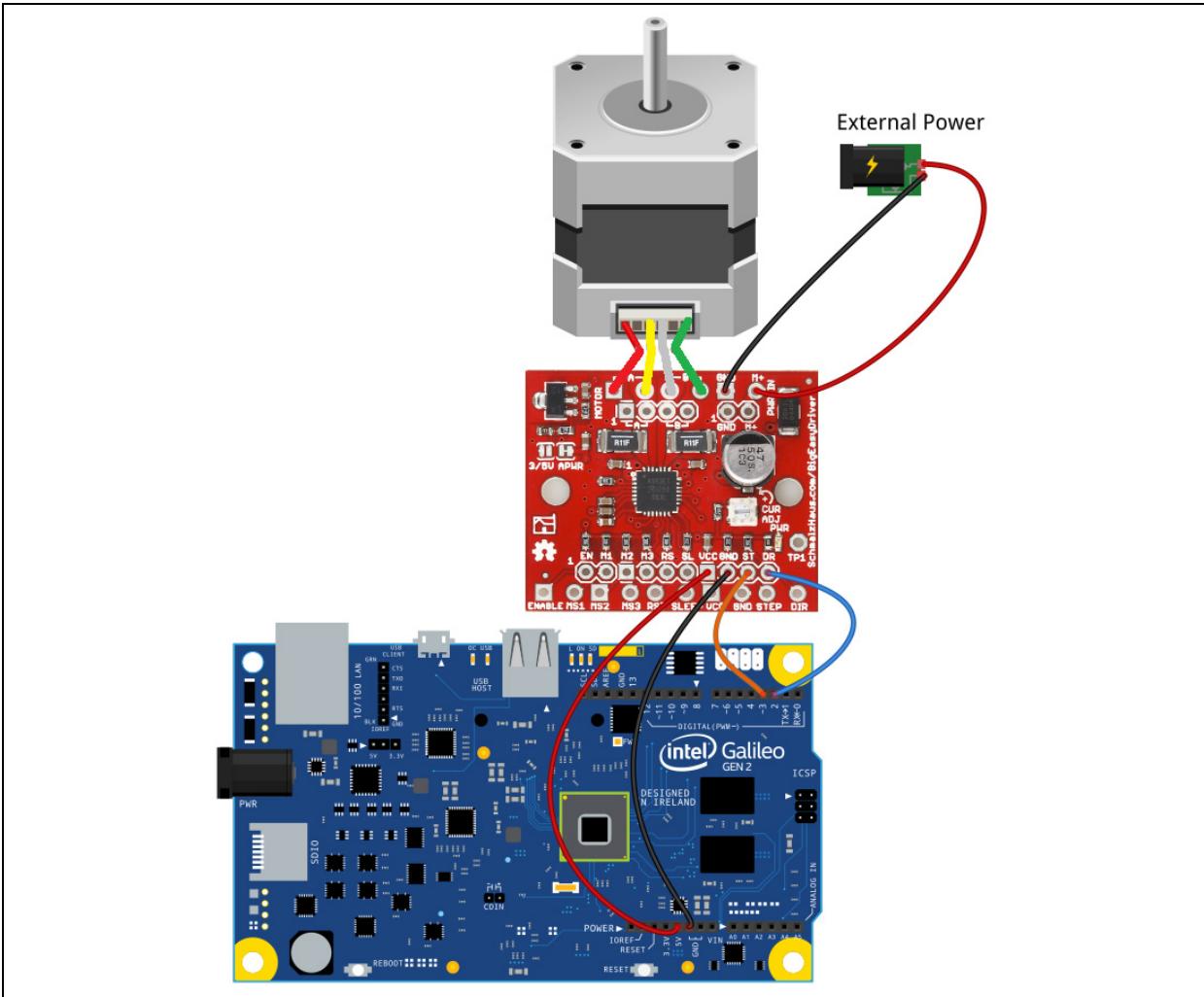
Figure 154 Big Easy Driver Board v1.2



Hardware summary

Key info	Description/links
Galileo Board Firmware	1.0.3 (WW37)
Edison Board Firmware	1.0.3 (Production Release)
Use VIN as power source	No
Operating voltage	3.3V or 5V
Schematic	http://www.schmalzhaus.com/BigEasyDriver/v1_2/BigEasyDriver_v12_sch.pdf

Figure 155 Big Easy Driver (parts are not to scale) connection diagram to Galileo 2



Breakout Pin	Function
DR	Direction – connect to D2
ST	Stepper – connect to D3
GND	Connect to GND
VCC	Connect to 5V.
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

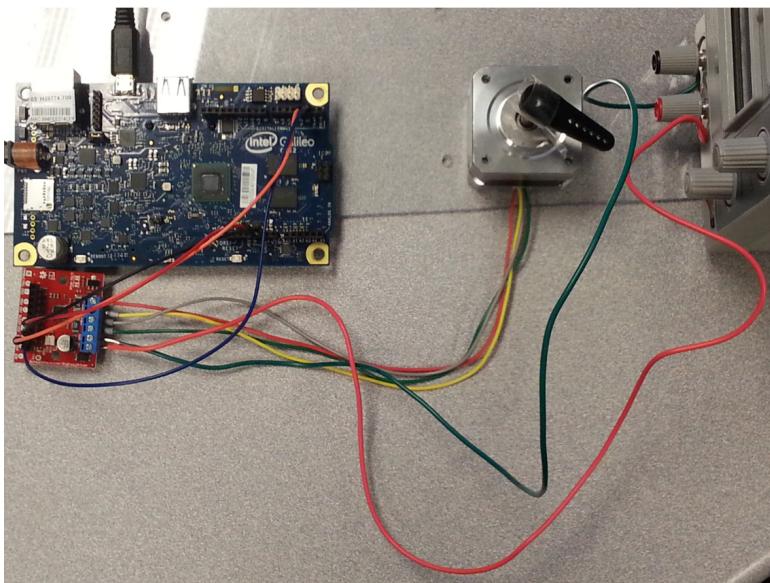
Companion library

No Library

Compile and upload

The physical connections are show as follows:

Figure 156 Connecting the Big Easy Driver using a NEMA017 Servo to Galileo 2



Note: Galileo 1: Remove the VIN jumper on the Galileo card or damage to the Galileo card will occur. Read the [VIN Jumper Overview](#) section about this jumper.

The following sketch 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 ease up on the current.

```
Example Sketch to step motor forward and backward
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
}
```

**Example Sketch to step motor forward and backward**

```
// 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);
}
```

Results

G1 Not compatible. The motor wasn't stable and was overheating.

G2/Edison Compatible.

UniPolar steppers configured to run as BiPolar was not functional on this breakout for all three boards. These UniPolar steppers worked as BiPolar on the [Adafruit Motor Shield](#).

Next steps

- Use with a unipolar stepper motor

§

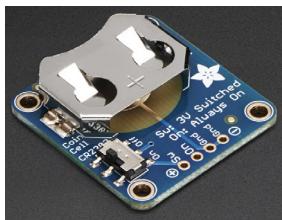
56 Adafruit* Coin Cell Battery

Use case

The breakout board uses a coin cell battery holder, soldered to the board, to provide power to the 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

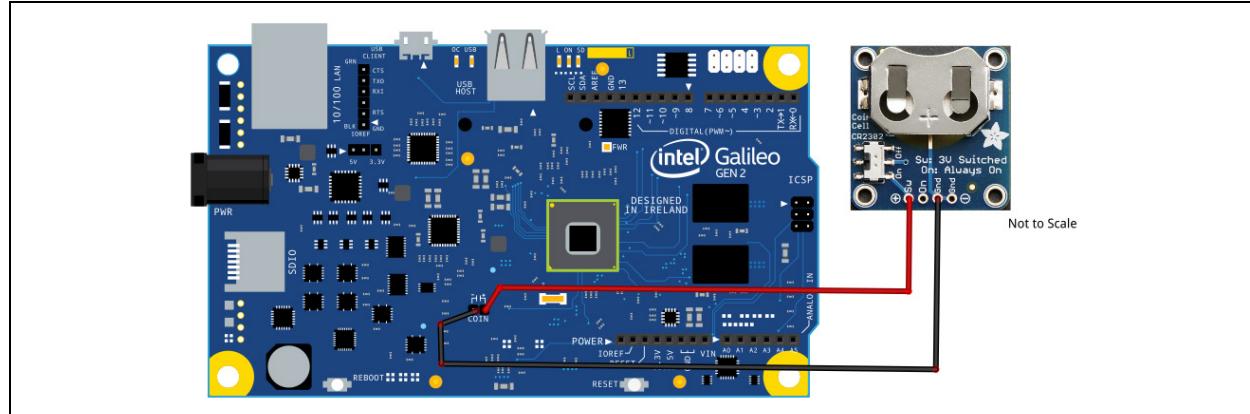
Figure 157 Coin Cell Battery Breakout Board



Hardware summary

Key info	Description/links
Operating Voltage	3.3V
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 exists.
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)

Figure 158 Coin Cell battery on a Galileo 2





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. Insure that the switch is turned on so that the clock is provided power.

Companion library

No library required.

Compile and upload

The sample sketch makes a Linux call to obtain the current date and time. The default time is set for January 2001. The sample 'Sketch 1' will display the current time in the IDE serial port.

```
Sketch 1: 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/ttyGSO"); // Current Time
    gSerialStdPtr->println("HW Clock");
    system("hwclock > /dev/ttyGSO"); // 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("");
}
```

The sample 'Sketch 2' 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.

```
Sketch 2: Set date and time in Linux
// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGSO, Tx pin
```



Sketch 2: Set date and time in Linux

```

void setup()
{
    gSerialStdPtr->begin(9600);
    waitForUser(5); // Give user time to open serial
    terminal

    gSerialStdPtr->println("SW Clock");
    system("date > /dev/ttys0"); // Current Time
    delay(1000*2);
    gSerialStdPtr->println("HW Clock");
    system("hwclock > /dev/ttys0"); // Current Time
    delay(1000*2);

    gSerialStdPtr->println("Set Software Clock");
    system("date 070408002014 > /dev/ttys0"); // Software Clock
    delay(1000*2);

    gSerialStdPtr->println("Set Hardware Clock");
    system("hwclock --systohc --utc > /dev/ttys0"); // Hardware Clock
    delay(1000*2);
}

void loop()
{
    system("date > /dev/ttys0"); // Current Time
    system("hwclock > /dev/ttys0"); // 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 Sketch1 again to prevent the clock to be set when the board powers up.

Turn the physical power off on the board. Remember to disconnect the USB cable also to insure the board receives no power.

Restart the board and load 'Sketch1' (if required). The clock should retain its time.

Results

G1/G2/Edison Compatible.

NOTE: On Edison, the hardware clock is properly set. However, on boot up, the software clock is not reset to the hardware clock. This is being investigated

Next steps

- Investigate why software clock is not being set based on hardware clock.



57 Sparkfun* Line Sensor Breakout

Use case

The 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 159 Line Sensor Breakout Analog (left)/Digital (right)



Hardware summary

Key info	Description/links
Operating Voltage	5 V
VIN as power source	No
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
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)

Sparkfun* Line Sensor Breakout



The method

Figure 160 Analog Line Sensor on a Galileo 2

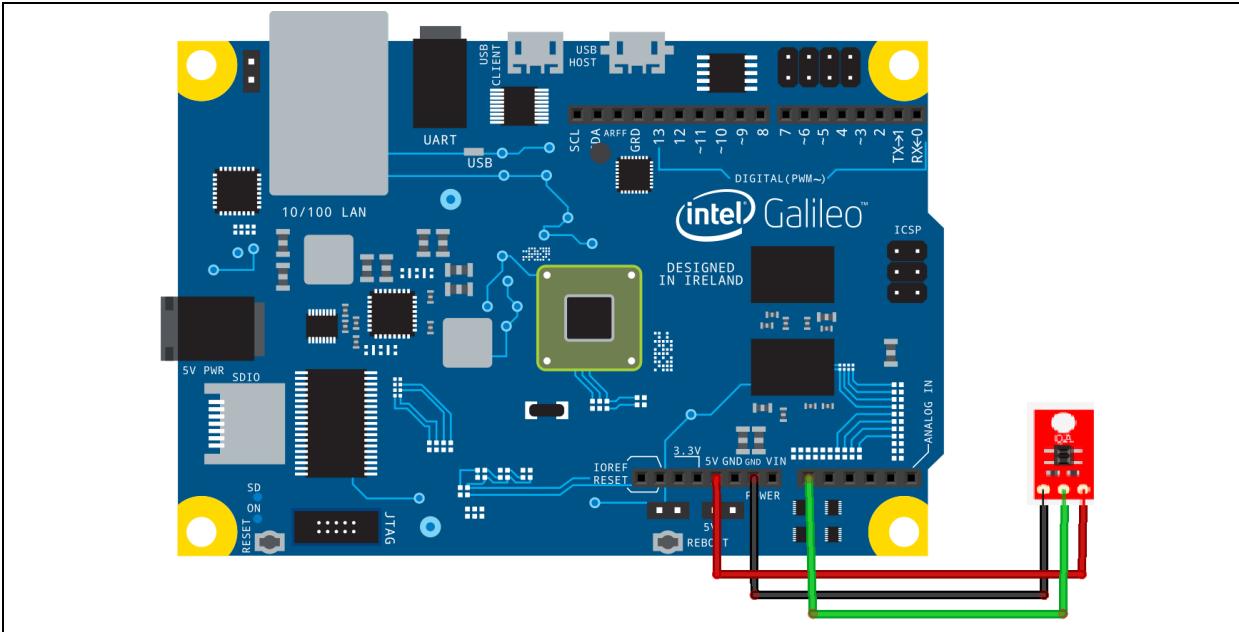
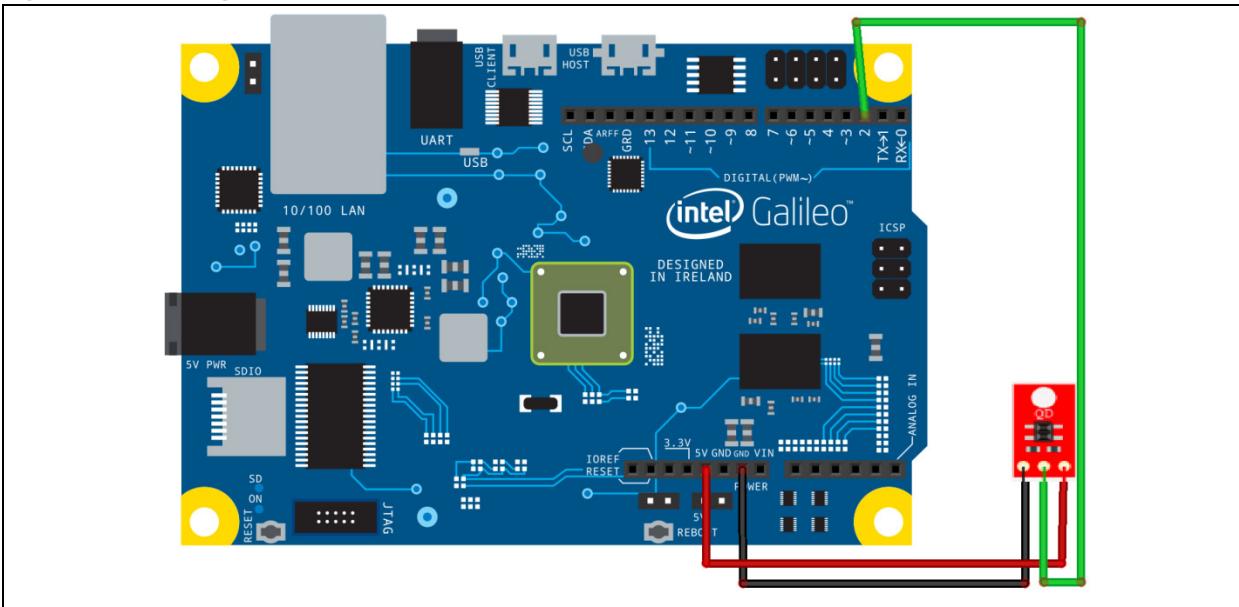


Figure 161 Digital Line Sensor on a Galileo 2



Breakout Pin	Function
VCC	Power, connect to 5V
OUT	Analog: Signal, connect to A0 Digital: Signal, connect to D2
GND	Ground, connect to GND



Companion library

None.

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. 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.

Analog Version

```
//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 then 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.

Digital Version

```
//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);
}
```



```
Digital Version
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;
}
```

Results

G1/G2/Edison Digital, Not Compatible. This sensor is a single wire device that is switching between INPUT and OUTPUT on the same pin. Suspect this is a Galileo timeout issue.

G1/G2/Edison Analog, Compatible. The values generated are not the same as the Arduino, however, scanning across a dark line has a consistent behavior.

Next steps

- None.

§

58 Parallax* Ping* Ultrasonic Distance Sensor

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.

For Galileo, the single I/O pin was changed to two pins to compensate for the speed requirements. The pin number was also changed so that Galileo fast pins could be utilized.

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 162 Ping* Ultrasonic Distance Sensor (28015, REV A)



Hardware summary

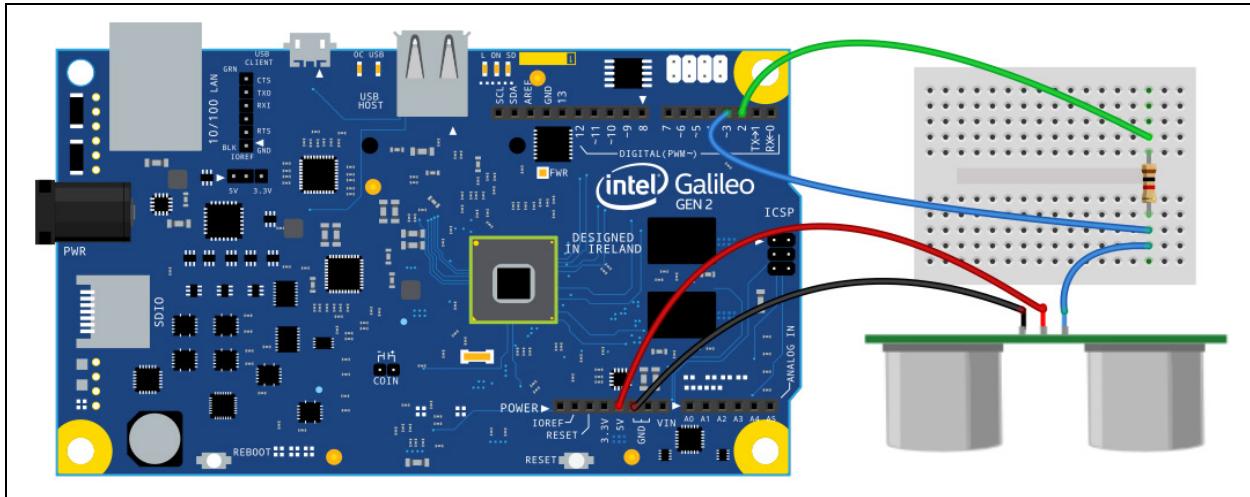
Key info	Description/links
Operating Voltage	5V
VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
Schematics	http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/Arduino/Shields/Danger_Shield-v17.pdf
Datasheet	http://www.jameco.com/Jameco/Products/ProdDS/282861.pdf
Distance	0.8 inches To 3.3 yards



The distance sensor was not functional under Galileo since the 'pulseIn()' function was timing out as the pin type was being changed from OUTPUT to INPUT. The reason for this timeout was that this is a one-wire device. The 'pinMode()' function took too long to respond. The length of time cannot be avoided because Galileo takes a long time to perform an I2C translation with the IO expander.

A resolution was to change the device from a single pin to a multi-pin. One pin is the OUTPUT and the other pin is the INPUT. The sensor signal wire was re-wired to include a resistor to provide a clear HIGH or LOW signal.

Figure 163 Galileo 2 connection for distance sensor



Sensor Pin Name	Function and Galileo connection
GND	Ground, connect to GND
5V	Power, connect to 5V
SIG	Signal, connect to board, then to D3 Signal, connect to 1 kohm resistor D3-onboard then to D2 (see above for visual configuration)

Companion library

No companion library required, however, there is a fix required for the firmware Release Version (1.0.2) of the pulseIn.cpp file. This file is located in the '..\hardware\arduino\x86\cores\arduino' directory.

In the 'pulseIn' function an 'if' and 'else' statement exists for PLATFORM_NAME "GalileoGen2". The 'pinMode' calls needs to be commented out for pin 2 and pin 3. This is the 'else' portion of the code as follows:

pulseIn.cpp for Galileo Firmware 1.0.2	
Before	After
<pre> else { switch(pin) { case 2: fastPin = GPIO_FAST_ID_QUARK_SC(0x40); pinMode(pin,INPUT_FAST); highValue = 0x40; break; } } </pre>	<pre> else { switch(pin) { case 2: fastPin = GPIO_FAST_ID_QUARK_SC(0x40); // pinMode(pin,INPUT_FAST); highValue = 0x40; break; } } </pre>



pulseIn.cpp for Galileo Firmware 1.0.2	
Before	After
<pre>case 3: fastPin = GPIO_FAST_ID_QUARK_SC(0x80); pinMode(pin, INPUT_FAST); highValue = 0x80; break; default: highValue = 1; fastMode = false; break; }</pre>	<pre>case 3: fastPin = GPIO_FAST_ID_QUARK_SC(0x80); // pinMode(pin, INPUT_FAST); highValue = 0x80; break; default: highValue = 1; fastMode = false; break; }</pre>

There is also an API change from the Galileo Firmware 1.0.2 to 1.03. The 'fastGpioDigitalWrite' API was changed to 'fastDigitalWrite'. The example sketch reflects this change. The old API does not exist in the Edison firmware.

Compile and upload

The example sketch was modified to move the pin(s) to the Galileo fast pin(s). Another fast pin was added to change to a two wire solution. The switching of pin modes is no longer performed in the 'loop' function.

Since the PING sensor is normally pulled low (on the sensor end), the better method is to use resistor compared to a diode. When the PING sensor response with its HIGH pulse, the output pin, which is LOW, is weaker since there is a 1K resistor between.

Sketch: G1/G2/Edison Sample Sketch
<pre>/* 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 #ifdef GALILEO</pre>



```

Sketch: G1/G2/Edison Sample Sketch
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);

```



Sketch: G1/G2/Edison Sample Sketch

```
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;
}
```

Results

G1/G2/Edison Compatible with described modifications.

An alternative distance sensor is the [HC-SR04 Sensor](#) that uses a two wire solution.

Next steps

- None

§

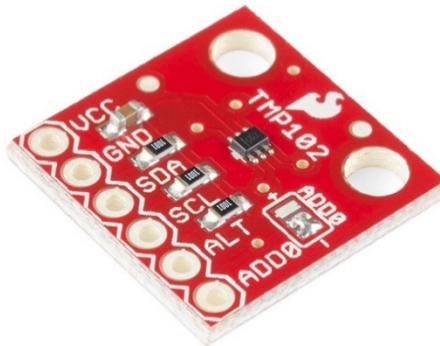
59 Sparkfun* Digital Temperature Breakout

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 I2C interface.

Key info	Links
Order/Product Info	https://www.sparkfun.com/products/11931
Guide	http://bildr.org/2011/01/tmp102-arduino/
Library	None

Figure 164 Sparkfun* TMP102 Digital Temperature Sensor

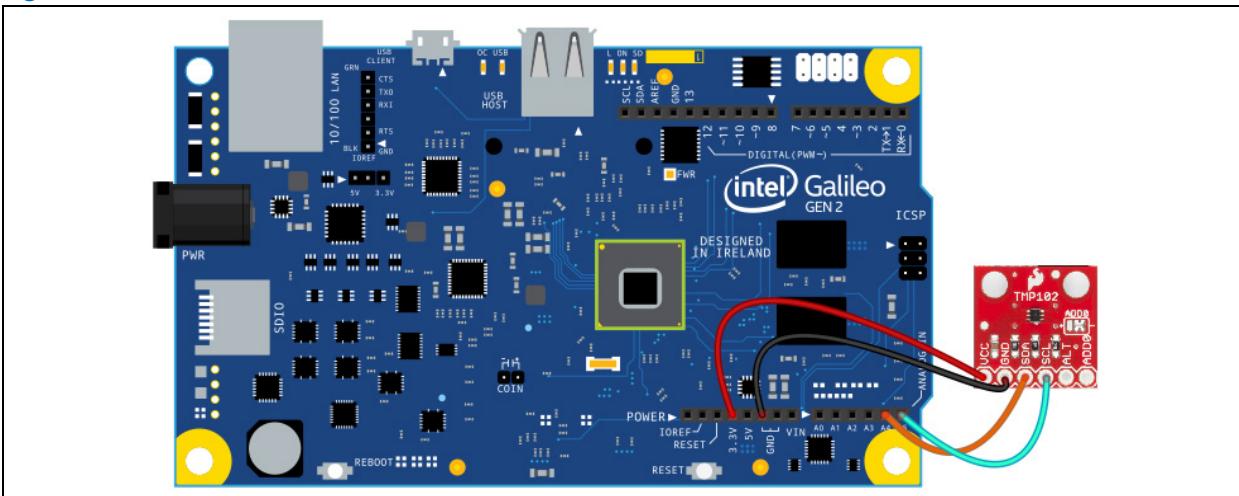


Hardware summary

Key info	Description/links
Operating Voltage	3.3V
VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)
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 165 TMP102 breakout on a Galileo 2



Breakout pin	Function
VCC	Power, connect to 5V
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 3.3V uses I2C Address 0x49

Companion library

Not required.

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.

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
/////////////////////////////
```



```

Guide Example Sketch
#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 honored?
    {
        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

```

Results

G1/G2 Compatible.

Edison Not Compatible. Temperature reads are not correct. More specifically, read request is not returning requested bytes.

Next steps

- None

§



60 XBee Wi-Fi Module w/Arduino Wireless Proto Shield

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

Figure 166 XBee Wi-Fi Module



Hardware summary

Key info	Description/links
Operating Voltage	5V
VIN as power source	No
Galileo Board Firmware	1.0.3 (Production Release)
Edison Board Firmware	1.0.3 (WW37)

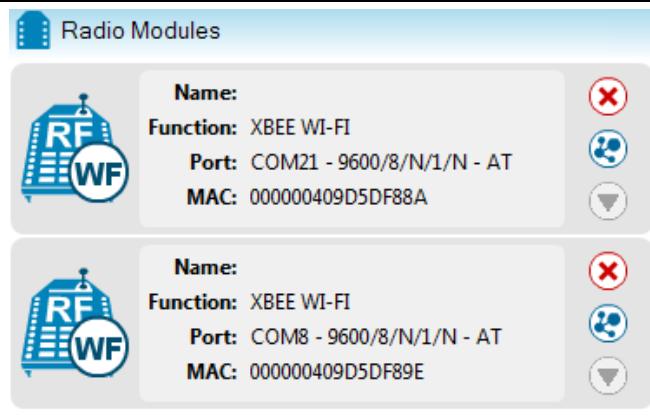
**Figure 167 XBee* Wi-Fi Module shield with Arduino* and Galileo 2**

- Attach module to shield and attach shield to Arduino. This module cannot be queried or configured on a Galileo/Edison. Use Arduino to configure the module to the router.
- Set proto-shield switch to "USB". If not set to 'USB', the configuration software will not detect it and a sketch cannot be downloaded.
- Download an empty sketch.

Empty Sketch

```
void setup() { }
void loop() { }
```

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 Arduinos.

Figure 168 Name of shield layout, connections, "on a Galileo", etc.

- Select "Added selected devices" to add module to the left screen
- If not previously setup
- Select "Scan for Access points in the vicinity"
- Select "Linksys", Enter password if necessary. In this case, we are connecting to a Linksys router.
- Select "Connect", enable 'Save the SSID configuration in the module'
- Once the module is connected, it will be supplied an IP address.
- Once connected, select the device which will display the Radio Configuration.
- Key Network/Serial Information is shown in the table below.



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]

- 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 setup, the baud rate needs to be set. The appropriate baudrate is set shown here.
- Save the module settings.
- Connect the shield to a Galileo/Edison board. One shield will remain on the Arduino.

Companion library

No library is used. The module will communicate through the serial port.

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.

```
Sender
// Sender
// A R D U I N O
//HardwareSerial* gSerialStdPtr = &Serial;
//HardwareSerial* gSerialOnePtr = &Serial;

// G A L I L E O
TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Arduino
IDE Serial Monitor
TTYUARTClass* gSerialOnePtr = &Serial1; // Galileo, /dev/ttyS0, Tx pin on
the Shield

char* gPromptPtr = "TX> ";
int gCtr = 0;
int gTXcnt = 0;

void setup()
{
    gSerialStdPtr->begin(9600);
    gSerialOnePtr->begin(9600);
```

**Sender**

```

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.

Receiver

```

// Receiver
// A R D U I N O
HardwareSerial* gSerialStdPtr = &Serial;
HardwareSerial* gSerialOnePtr = &Serial;

// G A L I L E O
//TTYUARTClass* gSerialStdPtr = &Serial; // Galileo, /dev/ttyGS0, Arduino
IDE Serial Monitor
//TTYUARTClass* gSerialOnePtr = &Serial1; // Galileo, /dev/ttySO, Tx pin
on the Shield

char* gPromptPtr = "RX> ";
int gCtr = 0;
int gBTcnt = 0;
void setup()
{
    gSerialStdPtr->begin(9600);
}

```

**Receiver**

```
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 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.

It is not clear what happens if the IP address expires. Meaning, does the module automatically accept a new IP address?

Results

G1/G2/Edison Compatible.

Next steps

- None.

§

61 Ultrasonic Ranging Module HC-SR04

Use case

The HC-SR04 provides 2cm – 400cm non-contact measurement function with a ranging accuracy that can reach 3mm. 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 2 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 169 Ultrasonic Ranging Module HC-SR04

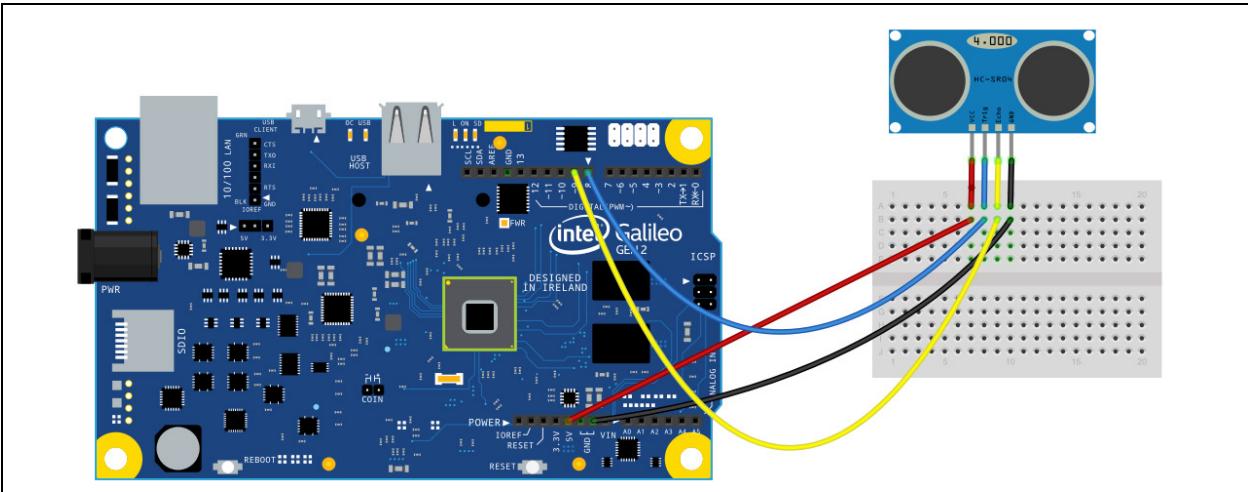


Hardware summary

Key info	Description/links
Operating Voltage	5V
Working Current	15 mA
VIN as power source	No
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Distance	2 cm to 4 meters



Figure 170 Galileo 2 connection for distance sensor



Sensor Pin Name	Function and Galileo connection
GND	Ground, connect to GND
5V	Power, connect to 5V
Trig	Trigger pulse – input – connected to D8
Echo	Echo pulse – output – connected to D9

Companion library

No companion library required.

Compile and upload

The example sketch 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.

```
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
```



```
Distance sensor sample sketch from fritzing.org
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
}
```

Results

G1 not compatible

G2/Edison Compatible.

Next steps

- None

§

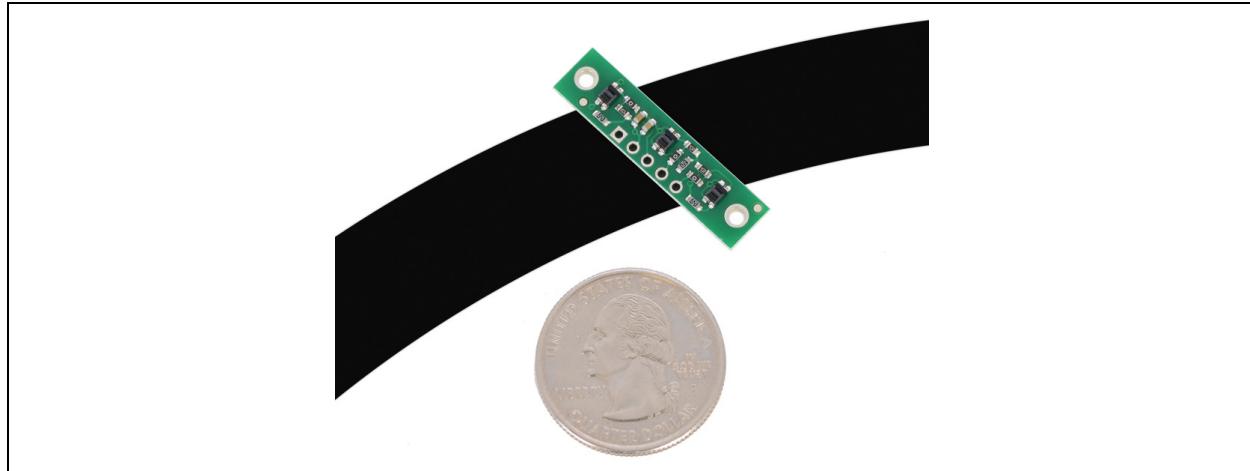
62 Pololu QTR-3A Reflective Sensor Array

Use case

The compact module packs three IR LED/phototransistor pairs onto a 1.25" x 0.3" 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	QTR-3A product info
Guide	Arduino Library Guide
Library	Arduino Library

Figure 171 QTR-3A Reflective Sensor on a $\frac{3}{4}$ " line

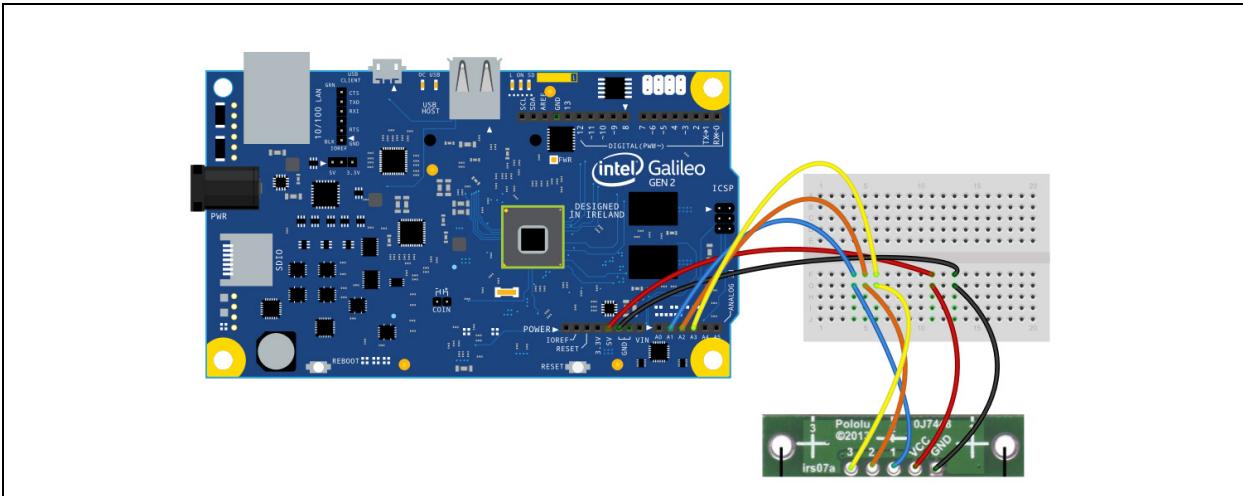


Hardware summary

Key info	Description/links
Operating Voltage	5 V
Working Current	50 mA
VIN as power source	No
Galileo Board Firmware	1.0.3 (Release Version)
Edison Board Firmware	1.0.3 (WW37)
Optimal sensing distance	0.125 in. (3 mm)
Maximum recommended sensing distance	0.25 in. (6 mm)



Figure 172 Galileo 2 connection for reflective sensor



Sensor Pin Name	Function and Galileo connection
GND	Ground, connect to GND
VCC	Power, connect to 5V
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

Companion library

Arduino library required. 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.

Compile and upload

Use the wiring diagram above and upload this sketch. Open a serial terminal and move sensor across a 3/4" 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
```



```

Reflective sensor sample sketch using raw sensor values
#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);
}

```

Results

G1/G2/Edison Compatible.

Next steps

- None

