Philosophy

We have turn key boxes for data acquisition like the Vernier's *LabQuest Mini* or Pasco's *PASPortal*. These tools are fabulous but they insulate students from the essence of the data acquisition process. One alternative is to go completely 'luddite' and only use rulers and stopwatches. The problem with this strategy is that it limits the systems we can study pushing us further from apparent realities. Nevertheless this is a useful place to start. What I would like to do is to pull back the curtain a bit and let students build some of the data acquisition tools needed for this purpose. To this end I notice that Vernier together with SparkFun have designed a shield that plugs into an inexpensive micro-controller, the Arduino Uno. With a specialized library we can have students do more than drag and drop graphical elements in specialized software but build from well defined modules the pieces that make a data acquisition system for an experiment.

Technology

Arduino Uno

The Arduino microcontroller is a type of microcomputer which comes in many flavors. The device that is most popular and most useful for our purposes is the Arduino Uno although many of the modules outlined here will work with other variants (Leonardo, Yun, or any newer model which has the same header pattern as the Uno). The arduino line of microcontrollers are computers that are programmed with a fairly easy to learn language and can send and receive electrical signals with simple commands. With an appropriate set of library calls and template programs it should be simple to build small applications that gather and organize measurements. This is a brief outline of the capabilities of the Uno

The electrical characteristics of the Uno are at the right. The key elements to pay attention to are the fact that it has 14 DIO pins, 6 Analog input pins (many double as DIO pins) with 6 PWM pins.
The logical pin list maps the external pins to the internal processor.

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	**14 (of which 6 provide PWM output)**
PWM Digital I/O Pins	**6**
Analog Input Pins	**6**
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Each of the 14 digital pins on the Uno can be used as an input or output, using **pinMode()**, **digitalWrite()**, and **digitalRead()** functions. They operate at 5 volts. Each pin can provide or receive 20mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50kΩ. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller simply providing an input resistor of 1kΩ inline with

the signal.

In addition, some pins have specialized functions:

Serial: DIO0 (RX) and DIO1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the **attachInterrupt()** function for details.

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the **analogWrite()** function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

I2C: A4 or SDA pin and A5 or SCL pin. Support I2C communication using the Wire library.

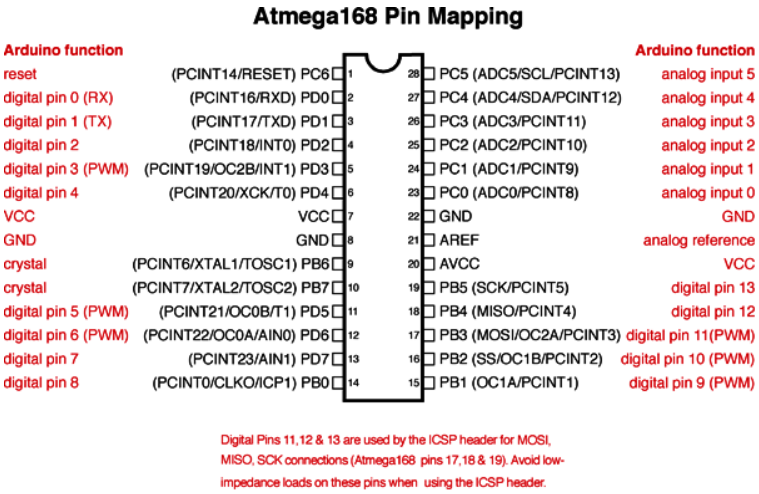
The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function.

There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Two pins are inaccessible as they are needed for Serial communication. DIO0 and DIO1 are Rx and Tx respectively. DIO2-DIO13 are available for output and input. DIO3,5,6,9-11 are available for PWM which means that the computer, through a simple call, can output a rapid series of pulses. Take this output and drop it across a capacitor and we get a voltage output proportional to the value sent to the command. The processor is also capable of responding to interrupts. When electrical signals on those special ports change the microcomputer stops what it is doing and runs a small fast program you can write.

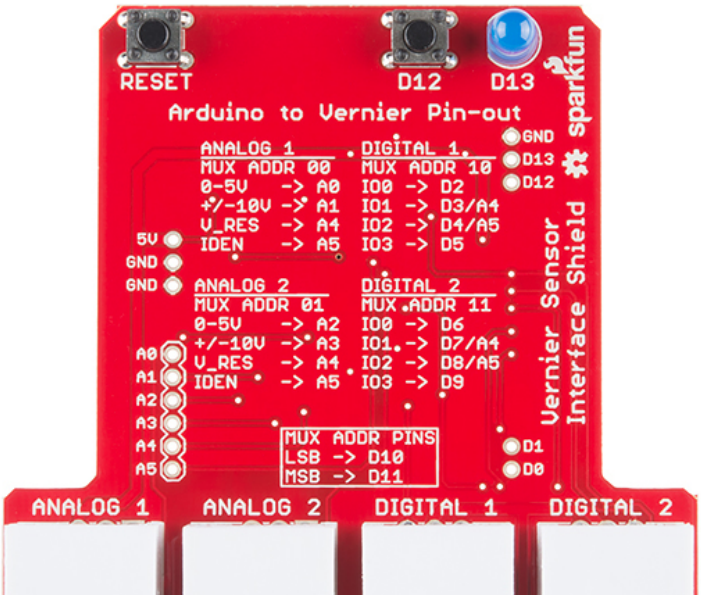


Vernier Shield

Shields are electronic add-ons to Arduinos that make it easy to develop interfaces to outside equipment. There are modules that control relays, motors, and other devices. This shield provides an interface to the Vernier Probes. Much of the documentation for this device is right on the card but here is a more detailed breakout:

The shield can take up to 4 sensors, 2 analog, 2 digital. The board is set to condition the signals from the probes and make them ready for what the Arduino can handle. In addition there is some additional electronics to detect the identity of the probes attached to the shield.

D10 and D11 run a multiplexer which allow the Arduino to detect what probe is connected[1].

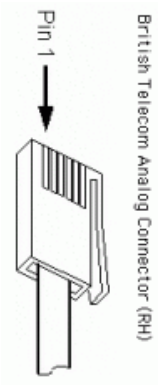


Probes

A colleague of mine once told me if you can turn anything you measure into a voltage or current you can automate the data taking. The probes that plug into the ports on the shield are just this. They take a measurement and turn it into a voltage (any current can be turned into a voltage with a precision resistor). There are two classes of devices Digital (things that simple switch between on and off, in the form of a 5V or 0V signal) or Analog (something whose voltage can vary continuously between 0-5V or ±10V).

Analog Input - BTA - Right Hand

The pins of the BTA connector are defined as such:



PIN #	Pin Name	Description
1	Analog Sensor Output (-10V to +10V)	Used with a number of Vernier voltage probes. This is wired through a scale and shifting op-amp circuit so that the Arduino can read it on a scale of 0 - 5V.
2	GND	Ground.
3	Vres	Resistance reference. 15K pull-up resistor ties this pin to 5V to use as a voltage divider between Pin 6 and GND.
4*	AutoIDENT	Most sensors have a unique resistor that is tied between this pin and GND. Vernier uses this to identify the sensor. (not supported on all sensors)
5	Power	5 VDC
6	Analog Sensor Output (0V to -5V)	Primary sensor output for most analog sensors including light, temperature, force, pressure, pH, etc...

http://www.vernier.com/support/sensor-pinouts/

[1] A multiplexer (MUX) is an electronic switch that allows multiple inputs to be funneled to one pin. Since the 'detection' of attached devices is a low priority, the MUX allows the Arduino to choose inputs for detecting the attached probe.

To extend the use of Vernier equipment, they also offer a few voltage probes that allow for direct voltage measurements between +/- 6V, +/- 10V, and +/- 30V. On each of these probes, the signals are each tied to Pin 1 and and GND of the BTA connector.

Digial Input - BTD - Left Hand

Digital sensors are any of the devices that return a signal that is either on (5V) or off (0V). Common digital sensors include: motion detector, photogate, radiation monitor, and the rotary motion sensors.

These sensors each have a somewhat unique pin-out definition. The following table outlines the pin assignments used for the Vernier digital connector.

PIN #	DEFAULT	MOTION DETECTOR	PHOTOGATE	RADIATION MONITOR	ROTARY MOTION
1	IO1	Echo	Input	Count	CCWcount
2	IO2	Init			CWcount
3	IO3	AutoIDENT	AutoIDENT	AutoIDENT	AutoIDENT
4	PWR	PWR	PWR	PWR	PWR
5	GND	GND	GND	GND	GND
6	IO4				X4res

Arduino Shield Pin Assignments

To maximize the flexibility of using all of the Vernier sensors, we have made the following pin assignments on our shield. Many of the Vernier sensors use I2C for identification and calibration data. We use a multiplexer to “share” (multiplex) pins A4 and A5 between all four connectors. The multiplexer is controlled with Pins 10 (LSB) and Pin 11 (MSB). See the section on multiplexing for more information.

Pin assignments for analog ports

	Analog 1	Analog 2	Description
MUX Control Address	00	01	Pins 10 (LSB) and 11 (MSB) control a multiplexer for A4 and A5.
Analog Signal (O - 5V)	A0	A2	Most analog sensors will interface to this pin. Use A0 for Analog 1 and A2 for Analog 2.
Analog Signal (-10V - +10V)	A1	A3	The shield has a built-in circuit to scale and shift input voltages from -10V to +10V to a range of 0V to 5V for pins A1 and A3.
V_res	A4*	A4*	Resistance reference. 15K pull-up resistor ties this pin to 5V to use as a voltage divider between Pin 6 and GND.
AutoIDEN	A5*	A5*	A 10K pull-up resistor ties these pins to 5V. The measured voltage drop across this pin uniquely identifies the sensor.

Pin assignments for digital ports

	Digital 1	Digital 2	Description
MUX Control Address	10	11	Pins 10 (LSB) and 11 (MSB) control a multiplexer for A4 and A5.

IO1 (BTD Pin 1)	2	6	Signal pin used for the photogate, motion detector echo, radiation count, and CCW rotary motion count.
IO2 (BTD Pin 2)	3 / A4*	7 / A4*	Trigger pin for the motion detector and I2C data (SDA) for sensor ID
IO3 (BTD Pin 3)	4 / A5*	8 / A5*	I2C clock (SCL) for sensor ID
IO4 (BTD Pin 6)	5	9	Used for rotary motion sensor to increase sensitivity.

*Pins A4 and A5 are shared across all four connectors. In order to properly access the BTA and BTD connector pins, you will need to interface the analog multiplexer circuit.

The above information is from the SparkFun site and attempts to provide all capabilities in a shield. We will throw away the ‘autoident’ capabilities because we know (or should) know what we plug into our ports. When a student cobbles the acquisition code they should have a general idea of what they are doing. The communication protocol developed below reflects this important point. The student should be concerned with timing and character of the data not getting a plug and play experience where the data magically appears.

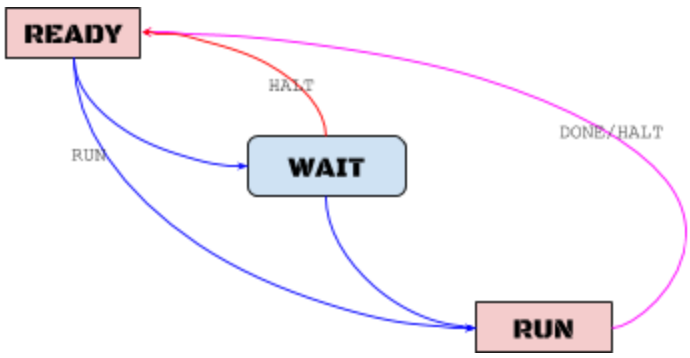
Develop a protocol

After quite a bit of testing and prototyping I feel it is best to use the Arduino as a local controller and pass the information out to a system master which can do higher level things with the data. For this to work we need a protocol to communicate to the Arduino which uses electrical signals to manage the shield and sensors. Let the master process the data. Communication takes place over the pseudo serial port of the USB connection. The arduino uses the SerialEvent interrupt driven character capture toolkit. The arduino passes information back depending on its state and mode.

A typical sequence might look something like the two following examples
Digital timing of a pendulum:

HALT is sent to arduino to make sure it is an a READY state. $^{A}c_{K}$ is received when arduino is ready. DIGMDE1 command is sent with PROPerty H2H or L2L set (for this example the exact model doesn't matter). $^{A}c_{K}$ is received when arduino is set to go. The GO command is given ($^{A}c_{K}$ is received) and 4 byte data values will be returned representing the time in µseconds since the last interval. This will continue indefinitely until a HALT message is received to stop the process.
Gathering voltage measurements:

HALT is sent to arduino to make sure it is in READY state. $^{A}c_{K}$ is received when arduino is ready. ANAMDE1 command is sent with PROPerty 0x0B (50ms timing) and a trailing 0 (no data stored.) $^{A}c_{K}$ is received to indicate all is well. [Optionally we can also send a TRIGMODE command with the bytes that will set a start based on a button push, a correct mode setting would also be rewarded with an $^{A}c_{K}$] The GO command is given ($^{A}c_{K}$ is received) and 6 byte data values will be returned at every set interval representing the time in µseconds since the trigger point (immediately, button push or signal value) and 2 bytes for each analog input. If the mode is to read only from one ADC then the other ADC's value is automatically set to 0 and never actually read (reading from ADCs takes time and can interfere with fast data acquisition)



STATES	READY	Waiting for commands and mode/property changes, can execute immediate commands
	WAIT	Entered with a RUN command with a mode that requires a trigger, will

		respond to a STATUS command
	RUN	Entered with a RUN command with no trigger mode, responds with data streams (depends on mode)

MODE TIMER (D1 or D2) — Gathers timing data from either digital inputs.

PROPS TIMING

H2H Time rises to a rises
H2L Time rises to a falls
L2H Time falls to a rises
L2L Time falls to a falls
ANY Any transition

RESP DATA — 4 byte values are returned whenever the conditions allow a timing signal. The 4 byte values represent the time difference between successive events.

MODE ANALOG (A1, A2, BOTH) — Gathers analog data data from either one or both analog inputs. When this mode is set information is also sent as to how the timing should be handled.

PROPS INTERVAL — Take values at or close to a time interval. See table for time intervals.

NPTS — The faster modes require that the Arduino store up to 255 values before transmitting them back otherwise this value is ignored.

RESP DATA — 8 byte values are returned at each time interval. 4 byte timing and 2 bytes for channel 1 and 2.

code	hex	period		freq		*store internally?*	*max values*
1	0x01	10	µs	100000	Hz	Y	255
2	0x02	20	µs	50000	Hz	Y	255
3	0x03	50	µs	20000	Hz	Y	255
4	0x04	100	µs	10000	Hz	Y	255
5	0x05	500	µs	2000	Hz	Y	255
6	0x06	1	ms	1000	Hz	Y	255

7	0x07	2	ms	500	Hz	Y	255
8	0x08	5	ms	200	Hz	Y	255
9	0x09	10	ms	100	Hz	N	*unl*
10	0x0A	20	ms	50	Hz	N	*unl*
11	0x0B	50	ms	20	Hz	N	*unl*
12	0x0C	100	ms	10	Hz	N	*unl*
13	0x0D	500	ms	2	Hz	N	*unl*
14	0x0E	1	s	1	Hz	N	*unl*
15	0x0F	2	s	0.5	Hz	N	*unl*
16	0x10	5	s	0.2	Hz	N	*unl*
17	0x11	10	s	0.1	Hz	N	*unl*
18	0x12	20	s	0.05	Hz	N	*unl*
19	0x13	100	s	0.01	Hz	N	*unl*
240	0xF0	MAN	When button is pressed			N	*unl*

Table of timing values. Triggering at a speeds faster than 10ms requires that the Arduino store values internally and report later. The second byte sets the maximum number of values to keep before reporting. MAN will record a value whenever the button is pressed.

MODE	ANALOG (A1, A2, BOTH)	Gathers analog data data from either one or both analog inputs. 4 byte values are sent which represent the time difference between successive events.
PROPS	TRIGGER	Optional command can be issued to set a trigger mode for the the analog data gathering. See table for values. Triggers can be set for rising above or descending below a particular set point. A trigger can also be set for a button press.

code	hex	TriggerModes	
0	0x0000	FREE	free run (no trigger)
1	0x0001	RISING	When analog value rises above setpoint
1024	0x0400		
4097	0x1001	FALLING	When analog value falls below setpoint
5120	0x1400		

Codes for Digital timing modes for a single timer