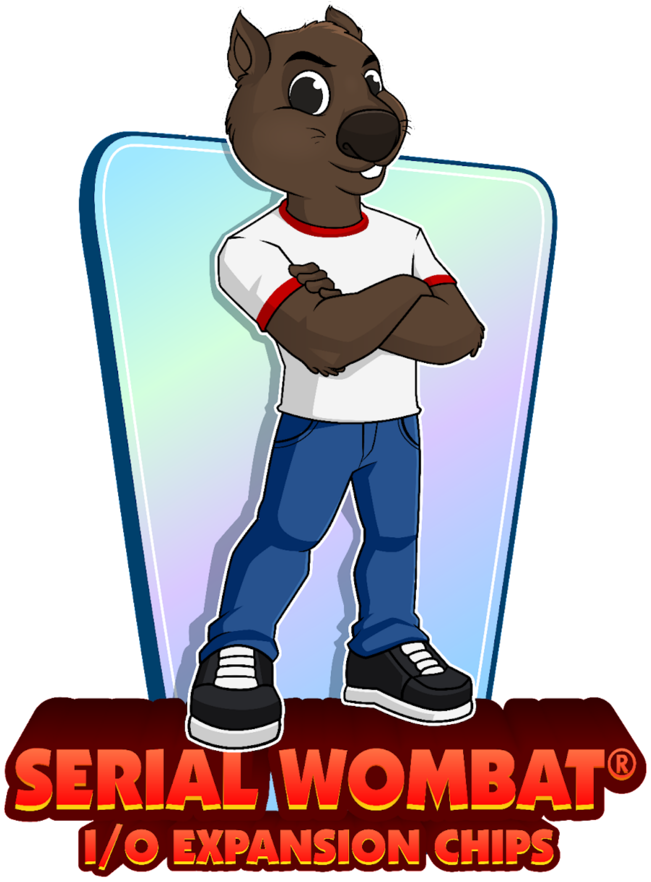
Serial Wombat 4B Chip User Guide

Version 2.0.3_A (Corresponds to Serial Wombat 4B firmware version 2.0.3)



An Open Source Project

Created by
Broadwell Consulting Inc.

Table of contents:

Overview

The Serial Wombat 4B chip family is designed to add smart I/O capability to Arduino or other systems capable of communicating over I2C. Each Serial Wombat 4B chip adds up to 4 I/O pins (1 is input only, and 3 are Input/Output Capable).

The Serial Wombat 4B firmware is open source under MIT license available here:

https://github.com/BroadwellConsultingInc/SerialWombat

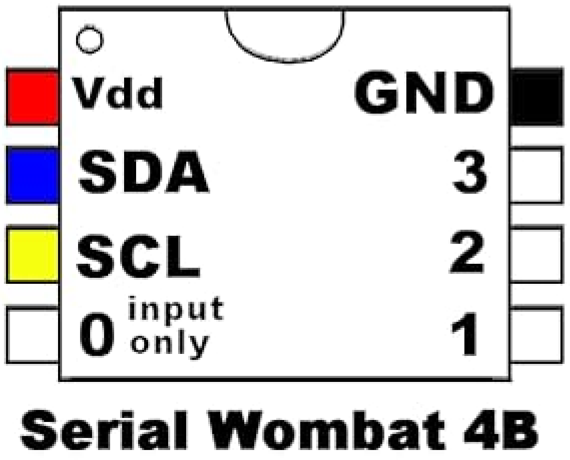
The Serial Wombat 4B chip is a peripheral chip that is commanded by a host device. It is not a device that runs downloaded user code directly. An Arduino library is available to control the Serial Wombat 4B chip from an Arduino host. The I2C communication protocol is available for user that wish to interface to the Serial Wombat 4B chip from other platforms.

The Serial Wombat 4B firmware is heavily commented using Doxygen compatible commenting. The compiled Doxygen documentation is available here:
https://broadwellconsultinginc.github.io/SerialWombat/sw4AB/index.html
This documentation exposes the internal workings of the Serial Wombat chip and its protocol. This documentation is typically not needed in order to use the Serial Wombat 4B chip from Arduino due to the availability of a wrapper library.

Each Serial Wombat pin runs an individual state machine every 1mS allowing that pin to solve common embedded systems problems. Pin modes can be mixed and matched (for example, two debounced inputs, an analog input, and a servo output).

The Serial Wombat 4B chip supports the following pin modes:

- Digital Input (with optional pull-up) on all pins
- Digital output (with optional open drain mode) on 3 output capable pins
- Button Debouncing (with optional pull-up) on all pins
- Pulse Timer (with optional pull up) on all pins
- Servo output on 3 output capable pins
- Analog Input (10 bit) on 3 A/D capable pins
- UART Receive (up to 115200 bps, one per chip) on any one of 4 pins
- UART Transmit (up to 115200 bps, one per chip) on any one of 3 output pins
- Protected Output on 3 output capable pins
- PWM Output on 3 output capable pins
- Rotary/Quadrature Encoder input (up to 2 encoders using 2 pins each)
- Watchdog Output on 3 output capable pins



Serial Wombat 4B Pinout

Shown above is the pinout of the Serial Wombat 4B chip. In this document pins 0 to 3 represent the 4 I/O capable pins (pins 4 through 7 on the package pinout). Pin 0 is input only, and Pins 1-3 are Input/Output/Analog Input capable.

The Serial Wombat 4B chip can be powered from 3.0 to 5.5v. Lower voltages are possible, but have not been tested to be compatible with 400kHz clock speeds.

The I2C bus can run at any voltage between 3.0v and the Serial Wombat chip supply voltage. External pull-up resistors are required. 2200 ohm resistors are suggested. I2C clock frequencies up to 400kHz are supported. Preprogrammed chips are available that respond to I2C addresses 0x6C, 0x6D, 0x6E, and 0x6F. The Serial Wombat 4B chip utilizes I2C clock stretching as defined in the I2C specification. Host systems controlling the Serial Wombat chip must support clock stretching (note that the Raspberry Pi built in I2C does not support clock stretching).

The Serial Wombat 4B chip is open-source firmware running on a Microchip PIC16F15214. See the datasheet of that part for additional electrical specifications.

The Serial Wombat 4B utilizes an internal phase-locked-loop RC oscillator to generate its internal clock. The nominal value of this clock is 32MHz, but may vary up to 2% based on manufacturing variation (variation may increase beyond 2% below 0 deg. C or above 60 deg. C). Since all timing done on the chip is based on this clock, absolute timing values such as PWM frequency, servo pulse, pulse measurement, UART bits, etc may vary by up to 2%.

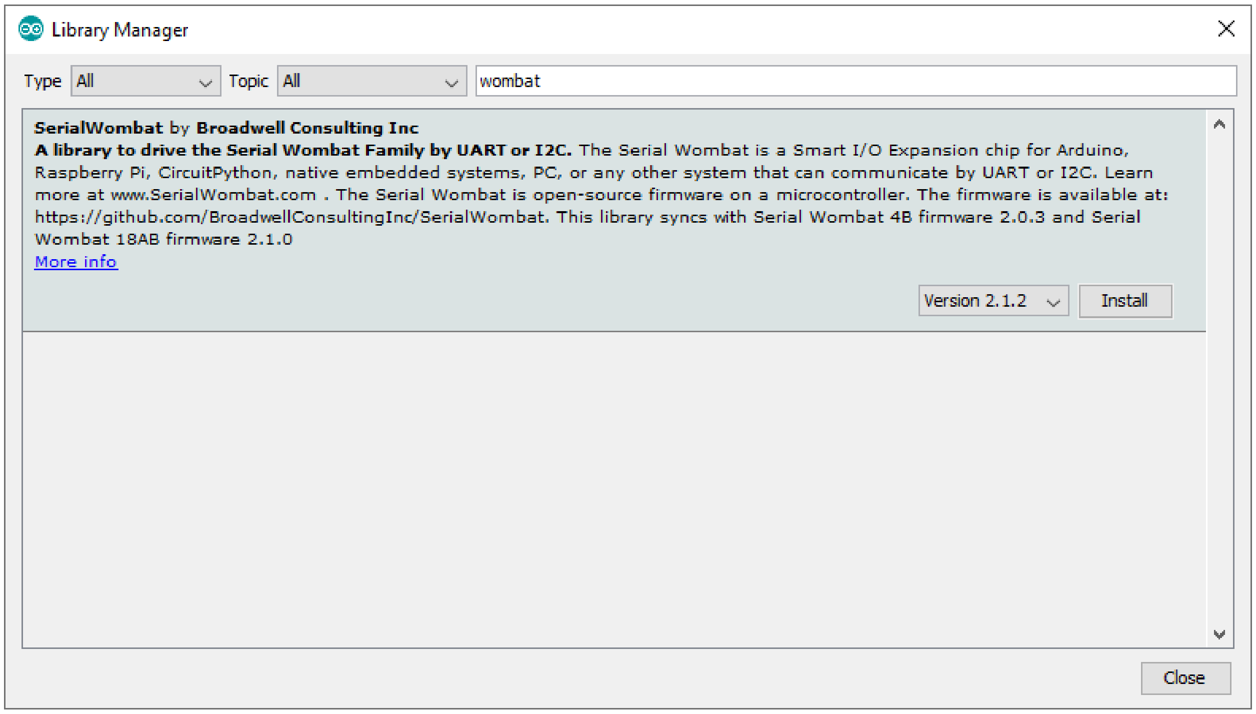
Arduino Library

An Arduino library is available that abstracts the communication protocol used by the Serial Wombat family of chips. The examples shown in this document assume that the Arduino library is used. The source code for the Serial Wombat Arduino library is available here: https://github.com/BroadwellConsultingInc/SerialWombatArdLib

The library is heavily documented using Doxygen in-line comment documentation. A compilation of this documentation is available here:

https://broadwellconsultinginc.github.io/SerialWombatArdLib/

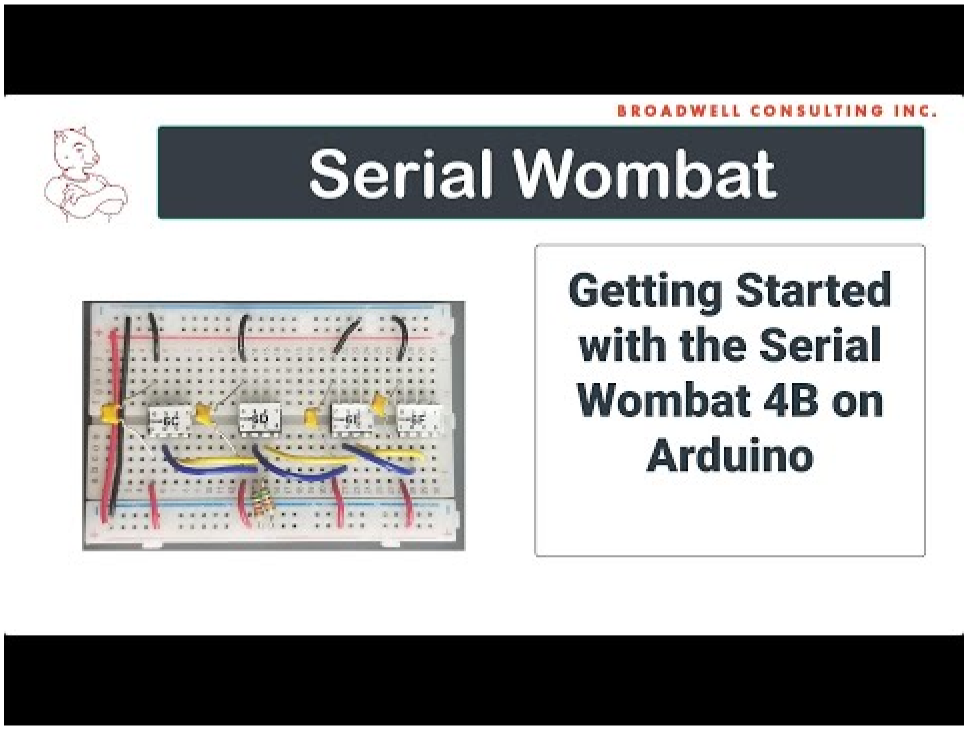
The Arduino library can be installed using the Arduino library manager:



SerialWombat by **Broadwell Consulting Inc**
A library to drive the Serial Wombat Family by UART or I2C. The Serial Wombat is a Smart I/O Expansion chip for Arduino, Raspberry Pi, CircuitPython, native embedded systems, PC, or any other system that can communicate by UART or I2C. Learn more at www.SerialWombat.com . The Serial Wombat is open-source firmware on a microcontroller. The firmware is available at: https://github.com/BroadwellConsultingInc/SerialWombat. This library syncs with Serial Wombat 4B firmware 2.0.3 and Serial Wombat 18AB firmware 2.1.0
More info

Some Arduino based interfaces are available directly from the SerialWombatChip class in the library, such as pinMode, digitalWrite, digitalRead, analogWrite, and analogRead. These provide a convenient way for Arduino programmers to get started quickly with the Serial Wombat 4B chip. Over time, it is recommended that programmers transition to using the native Serial Wombat interfaces shown in the examples and videos.

A getting started video which includes this procedure is available on YouTube:


BROADWELL CONSULTING INC.
Serial Wombat
Getting Started with the Serial Wombat 4B on Arduino

Getting Started With the Serial Wombat 4B chip and Arduino using I2C

Python / MicroPython Library

A Python and MicroPython library is available which provides equivalent interfaces to the ones in the Arduino library. Most videos and examples shown are in C++ for Arduino, but are easily ported to Python.

The library is available here:

https://github.com/BroadwellConsultingInc/SerialWombatMicroPython

An introductory video is available here:

https://youtu.be/bbBO5n_Ef-I

Circuit Construction

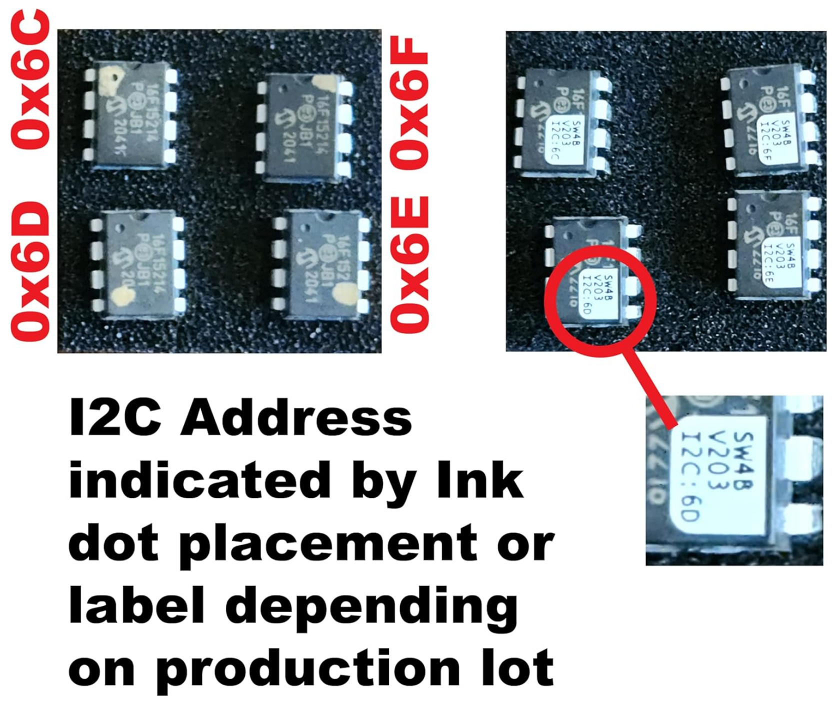
Before starting, consider subscribing to the Serial Wombat YouTube Channel:
https://www.youtube.com/@SerialWombat

and Instagram:
https://www.instagram.com/serialwombat/

When bugs are discovered or fixed or new firmware or library updates are made available a post will be made on these platforms.

The Serial Wombat 4B chip requires power and ground to be attached to the pins as shown in figure 1 above. Additionally, a 100nF ceramic decoupling capacitor must be connected across power and ground. Appropriate capacitors are included with the Serial Wombat 4B kits created by Broadwell Consulting Inc. sold on Amazon.

Each Serial Wombat 4B chip has a constant I2C address programmed in its firmware. In Serial Wombat kits created by Broadwell Consulting Inc. the I2C address can be determined either by a paint marking or by a decal on the chip:


0x6C
0x6D
0x6F
0x6E
SW4B
V203
I2C:6D
I2C Address indicated by Ink dot placement or label depending on production lot

Serial Wombat 4B kits created by Broadwell Consulting Inc. include user-applyable labels which indicate pin functions and show the I2C address of the chip.

I2C lines should be pulled up with appropriate resistors. Appropriate values vary with application and clock speed. 2200 ohm pull up resistors are suggested as a starting point. Reliance on internal Arduino or other micros' pull ups instead of discrete resistors may cause communication errors.

The I2C pull up voltage may be lower than the Serial Wombat chip system voltage (e.g. 3.3V I2C bus and 5.0V Serial Wombat chip voltage). This may prevent I2C operation at 400kHz clock.

Multiple Serial Wombat Chips can be used in the same circuit as long as they (and all other I2C devices) have unique addresses. Each chip should have its own 100uF capacitor.

The Serial Wombat 4B chip can function on input voltage from 3.0 to 5.5V. Input voltage as low as 2.5V can be provided, but I2C frequency should not exceed 100kHz. I2C and I/O pin voltages should not exceed the Serial Wombat 4B chip's source voltage.

The Serial Wombat 4B chip requires a stable power supply to function properly. It is suggested that loads which may cause supply instability (such as relays, servos, motors, or other loads with large inductive or capacitive values) be powered from a separate power supply from that used for the Serial Wombat 4B chip, as power fluctuations caused by these loads may trip the Serial Wombat 4B chip's internal low-voltage reset circuit.

Pin Modes

There are multiple pin modes which can be selected for each pin.

Pin 0 supports only digital input modes (digital input, pulse timer, Button Debouncing, UART RX, Quadrature encoder).

Pins 1-3 support digital input modes, digital output modes (digital output, servo, PWM, UART TX, protected output, watchdog), and Analog Input mode.

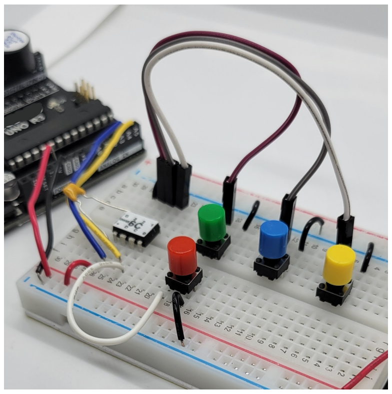
Digital Input Modes

Digital GPIO Input Pin Mode

The Digital Input Pin mode allows the host to determine if the pin is logic high or logic low. Inputs are Schmitt Trigger inputs with a low value of 0.2 x System Voltage and high value of 0.8 x System Voltage. See the PIC16F15214 datasheet for more information on logic levels.

Digital inputs can be configured to use internal pull up resistors within the chip.

Debounced Input Pin Mode



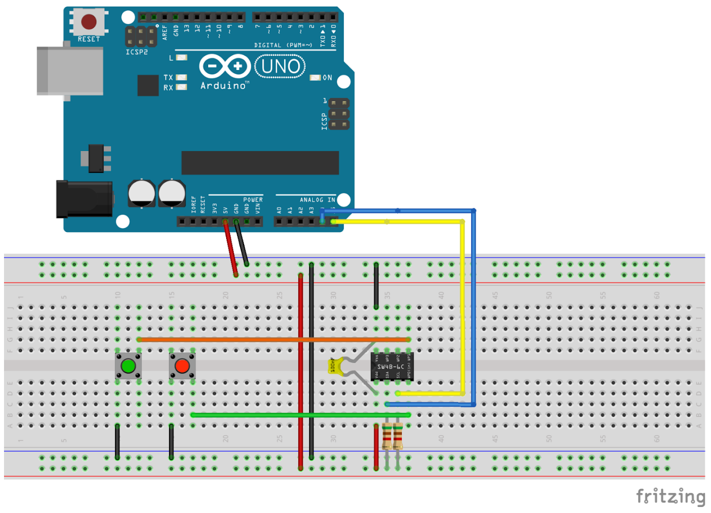
Debounced Input pin mode is designed to facilitate button and other switch style inputs which may oscillate before settling to a constant value.

The Debounced Input pin mode monitors an input pin 1000 times per second and reports back a value only after it has stabilized for a specified period of time.

The Debounced Input pin mode also counts debounced transitions and records how long in milliseconds the pin has been in the present state.   This allows easy creation of user interfaces or pulse counters without the need to constantly query the Serial Wombat chip.

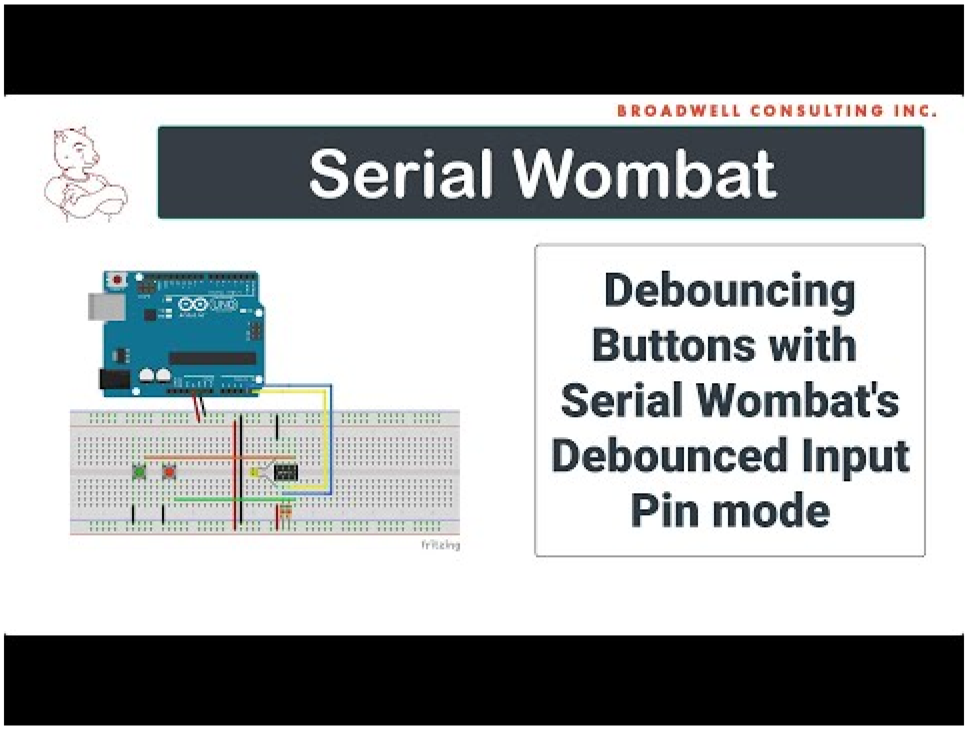
The Debounced Input pin mode allows inversion of the signal so that inputs can report “true” when the input is low, such as when the pin is connected to ground through a button. This can make interfaces that produce a low input when active more intuitive to process.

The Debounced Input pin mode can enable an internal pull up resistor in the Serial Wombat chip which allows a typical button or switch to be used with no additional components when connected to ground.



A wrapper class is available on Arduino which can increment or decrement a variable at increasing speeds based on how long a button is held down.

A video tutorial on this pin mode is available here:
https://www.youtube.com/watch?v=R1KM0J2Ug-M


BROADWELL CONSULTING INC.
Serial Wombat
Debouncing Buttons with Serial Wombat's Debounced Input Pin mode
fritzing

Here's an Arduino example of the Debounced Input pin mode from the Arduino library examples:

```
#include <SerialWombat.h>

SerialWombatChip sw;    //Declare a Serial Wombat
SerialWombatDebouncedInput redButton(sw);
SerialWombatDebouncedInput greenButton(sw);


// This example is explained in a video tutorial at: https://youtu.be/R1KM0J2Ug-M

void setup() {
  // put your setup code here, to run once:

  {//I2C Initialization
    Wire.begin();
    sw.begin(Wire,0x6C);  //Initialize the Serial Wombat library to use the primary I2C
port, SerialWombat is address 6C.
  }

   redButton.begin(0);
   greenButton.begin(1);
   Serial.begin(115200);
}

void clearTerminal()
{
  Serial.write(27);       // ESC command
  Serial.print("[2J");    // clear screen command
  Serial.write(27);
  Serial.print("[H");     // cursor to home command
}

int greenTransitions = 0;
int redTransitions = 0;

void loop() {
    clearTerminal();

    redButton.readTransitionsState();
    redTransitions += redButton.transitions;

    greenButton.readTransitionsState();
    greenTransitions += greenButton.transitions;

    Serial.print(greenTransitions);
    Serial.print(" ");
    Serial.println(greenButton.readDurationInTrueState_mS());

    Serial.print(redTransitions);
    Serial.print(" ");
    Serial.println(redButton.readDurationInTrueState_mS());


    delay(50);

}
```

Pulse Timer Pin Mode

The Serial Wombat Pulse Timer pin mode is useful for timing pulses such as RC Servo pulses, or reading PWM frequency and duty cycle.

The Serial Wombat Pulse Timer pin mode keeps track of the length of the most recent complete high segment, the most recent complete low segment, and the number of pulses measured.

The Serial Wombat chip can measure pulses in either millisecond or microsecond units. The user should select the correct units based upon pulse length. Measurements with a maximum value of less than 65535uS should use microsecond mode. Measurements with a maximum value longer than 65535 uS should use millisecond mode.

This pin mode has a 1uS precision and 2% accuracy (due to internal FRC variation from part to part).

Warning: Care must be taken when using this pinmode with high frequency (> 5 kHz) signals or pins that may be left floating (without pull-up enabled) on the Serial Wombat 4B chip because the Serial Wombat 4B chip uses an interrupt internally to capture transition timestamps. Excessively frequent pin transitions may cause the interrupt handler to starve the main processing loop, impacting function of all pin modes and communications.

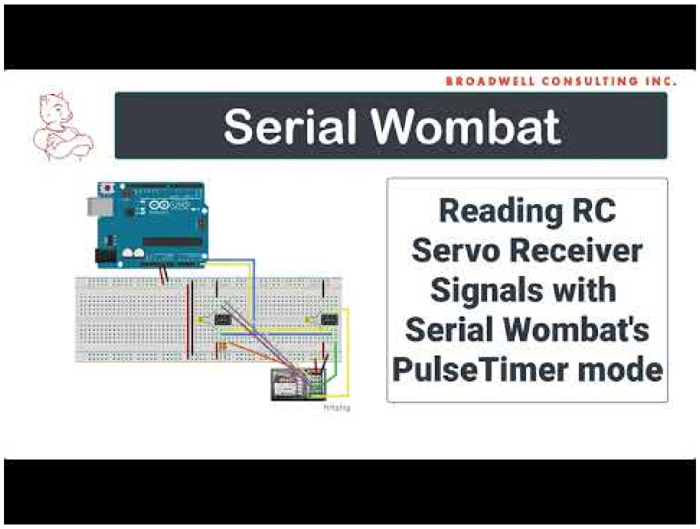
The Serial Wombat Protocol and Arduino library supports requesting both high and low times in a single transaction. This allows the most recent high and low times to be read together, which is important when calculating a PWM duty cycle. However, either the high time or low time may be the most recently measured value depending on when the request is made which may cause variation in duty cycle or frequency calculation for quickly changing PWM values.

The number of measured pulses increments for each high/low combination. By reading this value twice over a given period of time, the host can calculate an approximate frequency of a signal. The measured pulses value overflows from 65535 to 0 without notice.

A hardware overflow flag is available which indicates if pulse transitions are occurring more frequently than the Serial Wombat 4B chip's hardware can measure them, leading to inaccurate measurements. An exact frequency at which this occurs cannot be specified, as it varies depending on how many pins are configured to the Pulse Timer pin modes, and the aggregate number of pulses per second across all of those pins. Only pins which are configured to Pulse Timer pin mode contribute to interrupt loading.

A video tutorial on this pin mode is available here:

https://www.youtube.com/watch?v=YtQWUub9gYw


BROADWELL CONSULTING INC.
Serial Wombat
Reading RC Servo Receiver Signals with Serial Wombat's PulseTimer mode
fritzing

Here's an Arduino example of the Pulse Timer pin mode from the Arduino library examples which reads the high time of 4 channels of an R/C servo receiver:

```
#include <SerialWombat.h>

SerialWombatChip sw;    //Declare a Serial Wombat chip
SerialWombatPulseTimer steering(sw);
SerialWombatPulseTimer throttle(sw);
SerialWombatPulseTimer button(sw);
SerialWombatPulseTimer thumbSwitch(sw);

// This example is explained in a video tutorial at: https://youtu.be/YtQWUub9gYw

void setup() {
  // put your setup code here, to run once:

  {//I2C Initialization
    Wire.begin();
    sw.begin(Wire,0x6C);  //Initialize the Serial Wombat library to use the primary I2C port,
SerialWombat is address 6C.
  }

   steering.begin(0);
   throttle.begin(1);
   button.begin(2);
   thumbSwitch.begin(3);

   Serial.begin(115200);
}

void clearTerminal()
{
    Serial.write(27);       // ESC command
  Serial.print("[2J");    // clear screen command
  Serial.write(27);
  Serial.print("[H");     // cursor to home command
}

int i;
void loop() {
    clearTerminal();
    Serial.println(steering.readHighCounts());
    Serial.println(throttle.readHighCounts());
    Serial.println(button.readHighCounts());
    Serial.println(thumbSwitch.readHighCounts());

    delay(50);

}
```

UART Receive Pin Mode

The UART Receive Pin mode allows the Serial Wombat 4B chip to receive UART data which can then be transferred back to the host over I2C. The Serial Wombat 4B chip can support 1 UART input pin at a time. The UART Transmit and Receive pin modes must communicate at the same baud rate.

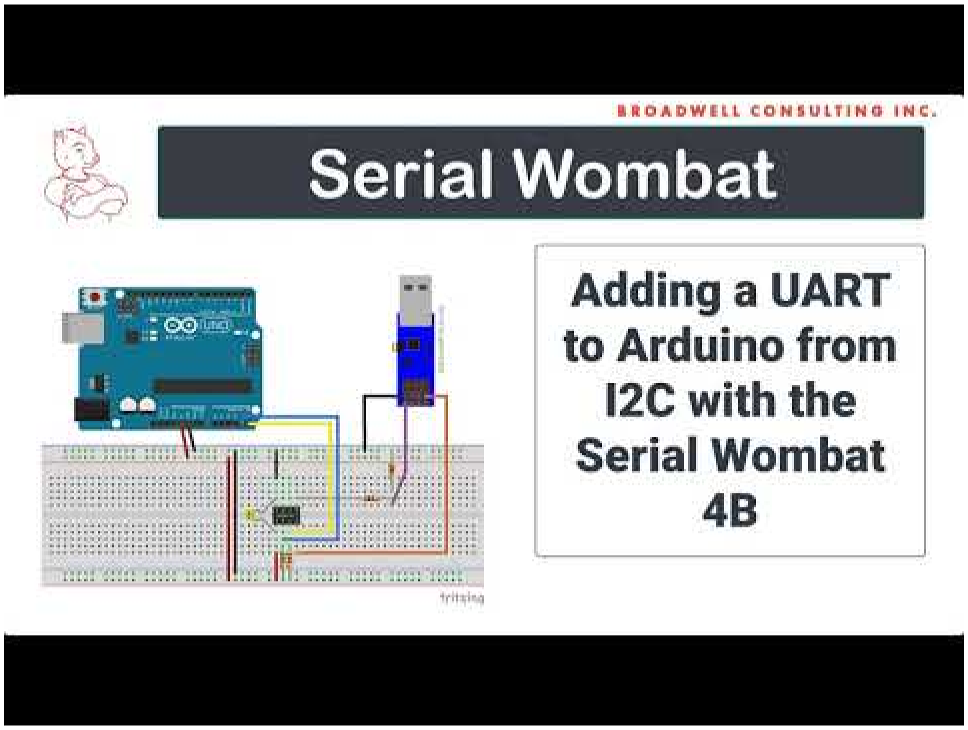
UART Format of 8 bits, no parity and 1 stop bit is required. Baud rates of 300,1200,2400,4800,9600,19200,38400,57600 and 115200 bits per second are supported.

The UART pin mode may have limitations compared to a UART integrated into the host. When using higher baud rates the host must keep up with the incoming data to prevent data loss. The Serial Wombat Protocol is effectively half duplex when communicating over I2C between the host and the Serial Wombat chip. Using the Serial Wombat 4B chip's capability of running at 400kHz I2C bus speed will increase the maximum throughput of the UART pin mode.

The SerialWombat 4B Chip has an onboard 128 byte buffer for UART reception to allow the host to do other things in addition to monitoring the UART interface.

On Arduino the UART Receive pin mode wrapper inherits from the Stream class allowing a Serial Wombat chip based UART to be interfaced using the same methods as accessing a UART integrated into the host.

A video tutorial of the Serial Wombat 4B chip's UART bridge capabilities is available here:


BROADWELL CONSULTING INC.
Serial Wombat
Adding a UART to Arduino from I2C with the Serial Wombat 4B
fritzing

https://youtu.be/C1FjcaiBYZs

Quadrature / Rotary Encoder Pin Mode

A class that uses two Serial Wombat input pins to read quadrature encoder input.

The Serial Wombat Quadrature Encoder Pin Mode configures two pins on the Serial Wombat chip to work together to read quadrature encoder inputs.
By offloading the reading of an encoder to the Serial Wombat chip, it makes it easy for the host to track multiple encoders at once. The host need only periodically retrieve the net change in rotary encoder position from the Serial Wombat chip rather than monitoring for every signal change.

The quadrature encoder is capable of running in either polled interrupt/DMA driven modes. Polled mode is recommended for manual inputs such as rotary encoder knobs. It polls at 1 kHz which is fast enough for most applications.

Interrupt driven mode on the Serial Wombat 4A/4B is capable of correctly decoding very fast signals. However, the signals need to be properly filtered in order to eliminate any bouncing.

The SerialWombatQuadEnc can make use of the Serial Wombat chip's built in pull-up resistors to make connecting a rotary encoder knob very simple. Debouncing is available which prevents additional transitions from being measured for a specified number of mS after a transition.

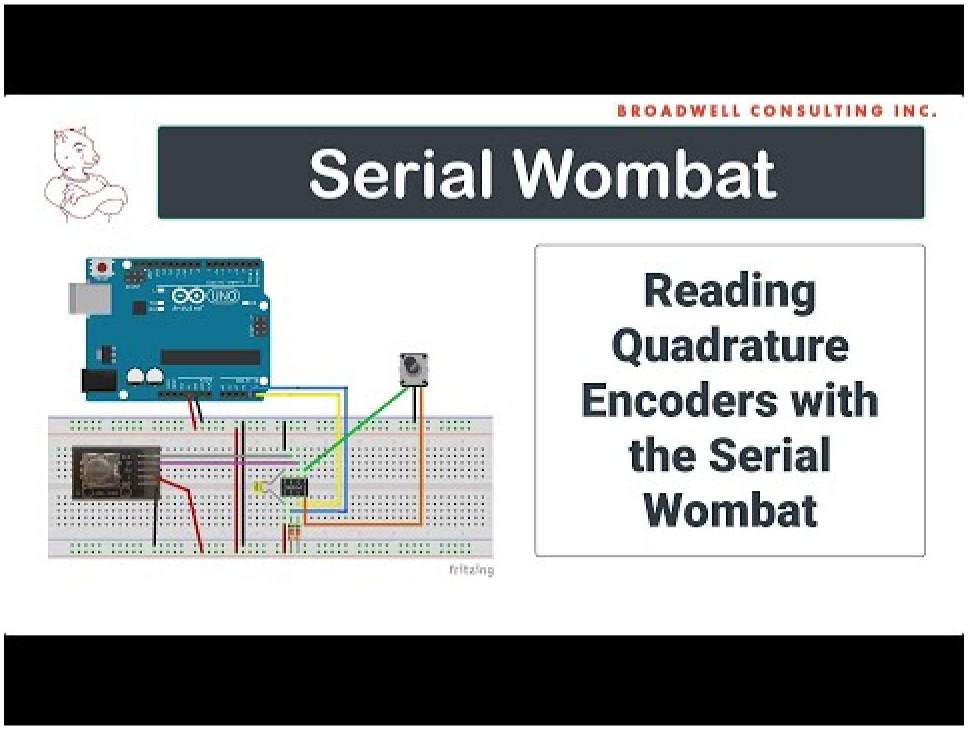
Rotational direction measurement can be changed by switching the "pin" and "second pin" parameters in the begin call.

The reported position can be changed on low to high transitions of "pin", high to low transitions, or both transitions. This allows knobs that make and break connection on each click/detent and knobs that either make or break connection on each detent to report one change per detent to the host.
The default mode for simple initialization is to measure both, which will result in 2 increments per detent for encoders that make and break connection on each detent.

Warning
Care must be taken in interrupt mode on Serial Wombat 4A/4B chips when using this pinmode with high frequency (> 5 kHz) signals or pins that may be left floating on the Serial Wombat 4A / 4B because the Serial Wombat uses an interrupt internally to capture transitions between state machine updates. Excessively frequent pin transitions may cause the interrupt handler to starve the main processing loop, impacting function of all pin modes and communications.
The Serial Wombat chip can be queried for overflow frames. If overflow frames are occuring, then the system is overloaded.

The Serial Wombat 4B chips can measure a maximum of 8 transitions per mS across all pulse input pins. More frequent transitions than this may result in pin mode malfunction.

A video tutorial of the Serial Wombat 4B chip's Quadrature Encoder capabilities is available here:


BROADWELL CONSULTING INC.
Serial Wombat
Reading Quadrature Encoders with the Serial Wombat
fritzing

https://youtu.be/_wO8cOada3w

The following code shows Initializing two Quadrature/Rotary encoders on a Serial Wombat chip and reading them periodically.

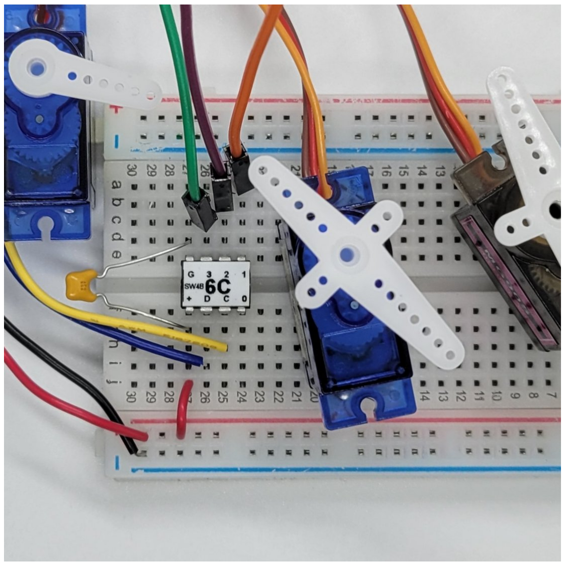
```
#include <SerialWombat.h>

SerialWombatChip sw6C;    //Declare a Serial Wombat chip
SerialWombatQuadEnc qeBasic(sw6C);
SerialWombatQuadEnc qeWithPullUps(sw6C);

// This example is explained in a video tutorial at: https://youtu.be/_wO8cOada3w



void setup() {
  // put your setup code here, to run once:

  { //I2C Initialization
    Wire.begin();
    sw6C.begin(Wire, 0x6C); //Initialize the Serial Wombat library to use the primary I2C port,
SerialWombat is address 6C
  }
  qeBasic.begin(0, 1);  // Initialize a QE on pins 0 and 1
  qeWithPullUps.begin(2, 3);  // Initialize a QE on pins 2 and 3
  Serial.begin(115200);
}

void loop() {
  Serial.print(qeBasic.read());
  Serial.print(" ");
  Serial.print(qeWithPullUps.read());
  Serial.println();
  delay(50);
}
```

Output Modes

Digital Output Pin Mode

The Digital Output pin mode allows the host to set the pin high or low. The maximum current sunk or sourced per pin should not exceed 25mA. See the PIC16F15214 datasheet for detailed electrical specifications.

Servo Output Pin Mode



The Serial Wombat 4B chip can drive up to 3 standard RC servos with 1uS precision. This significantly improves on the 180 available positions available through a standard Arduino servo call.

The Servo pin mode outputs a pulse every 20mS.   The minimum and maximum pulse lengths are specified when the pin mode is initialized (544uS and 2400uS maximum by default).

The host then provides a 16 bit value between 0 and 65535 which scales the pulse between minimum and maximum length.

A reverse option can be specified at initialization which causes 0 to generate a maximum length pulse, and 65535 to generate a minimum length pulse. This is useful to make operation intuitive in cases where the servo moves opposite of what “feels” like the natural direction for an increasing value.

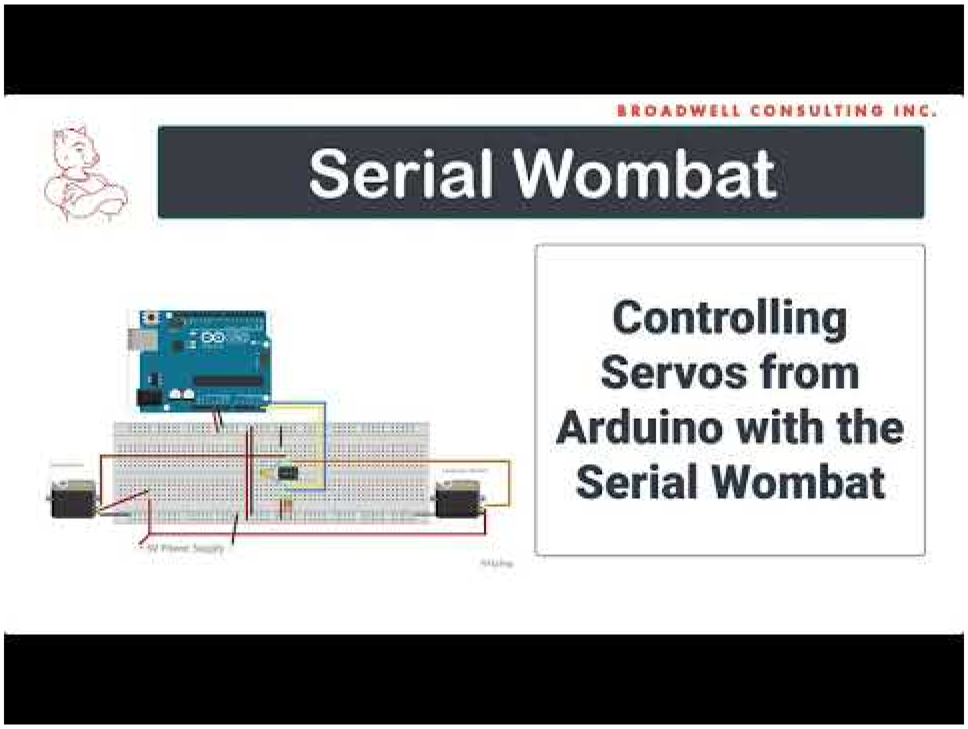
The Arduino class which wraps this functionality also provides an Arduino compatible interface which takes a value of 0 to 180 rather than 0 to 65535 to scale between minimum and maximum pulses.

The Arduino Class is documented here:
https://broadwellconsultinginc.github.io/SerialWombatArdLib/class_serial_wombat_servo.html

Here’s an Arduino example of the Servo pin mode from the Arduino library examples which declares two servos and controls one using the 16 bit interface, and the other using the Arduino compatible interface.

```
#include <SerialWombat.h>

SerialWombatChip sw;    //Declare a Serial Wombat chip
SerialWombatServo ContinuousServo(sw);  // Declare a Servo on pin 2 of Serial Wombat sw
SerialWombatServo StandardServo(sw);   // Declare a Servo on pin 3 of Serial Wombat sw

// A video tutorial is available which explains this example in detail at:
https://youtu.be/WiciAtS1ng0
void setup() {

  {//I2C Initialization
    Wire.begin();
    sw.begin(Wire,0x6C);  //Initialize the Serial Wombat library to use the primary I2C port,
This SerialWombat's address is 6C.
  }
  ContinuousServo.attach(2,500,2500,true); // Initialize a servo on pin 2, 500uS minimum pulse,
2500 us Maximum pulse, reversed
  StandardServo.attach(3);  // Initialize a servo on pin 3 using Arduino equivalent default
values

}

void loop() {

  // put your main code here, to run repeatedly:

  ContinuousServo.write(30);       // Takes a number from 0 to 180
  StandardServo.write16bit(5500);  // Takes a number from 0 to 65535:  Higher resolution
  delay(5000);
  ContinuousServo.write(140);
  StandardServo.write16bit(50000);
  delay(5000);

}
```

A video demonstrating the Servo pin mode is available here:
https://www.youtube.com/watch?v=WiciAtS1ng0


BROADWELL CONSULTING INC.
Serial Wombat
Controlling Servos from Arduino with the Serial Wombat
5V Power Supply

PWM Pin Mode

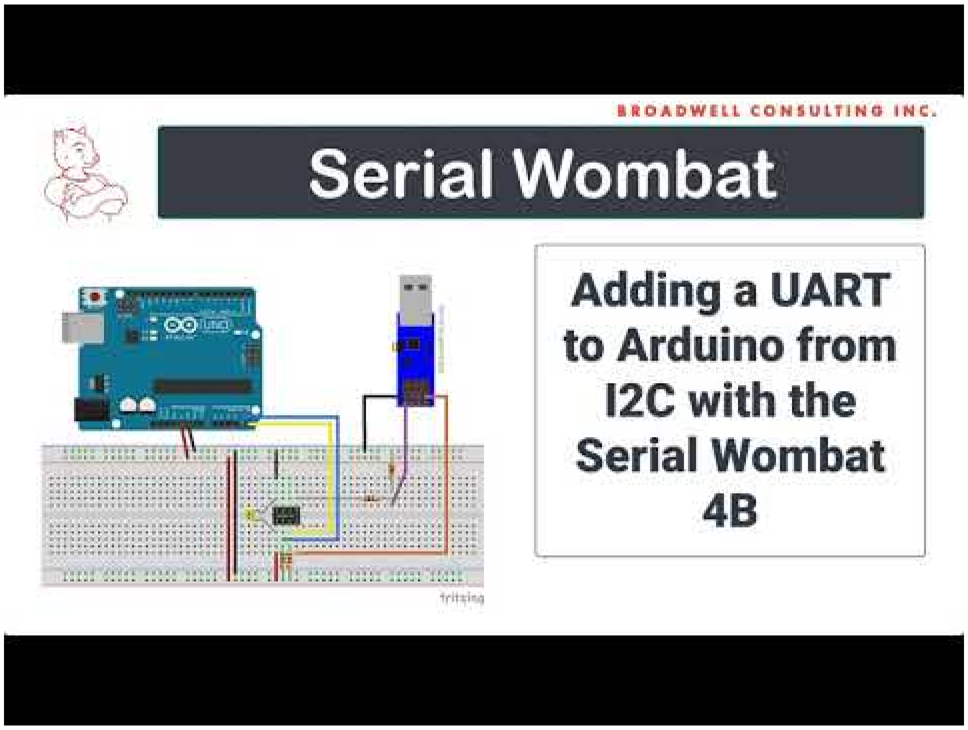
The PWM Pin mode allows the Serial Wombat 4B chip to generate a PWM output at a number of different frequencies. The PWM output hardware can generate PWM duty cycles with 10-bit accuracy (The 16-bit requested duty cycle is rounded to the nearest 10-bit value). The Serial Wombat 4B chip can generate a different duty cycle on each of its 3 outputs. All outputs output at the same frequency.   Available frequencies are:
1 hz
2 hz
4 hz
8 hz
16 hz
32 hz
63 hz
125 hz
244 hz
976 hz
1952 hz
3900 hz
7800 hz
15625 hz
31250 hz

UART Transmit Pin Mode

The UART Transmit Pin mode allows the Serial Wombat 4B chip to send UART data over I2C from the host which can then be converted to UART output. The Serial Wombat 4B chip can support 1 UART output pin at a time. The UART Transmit and Receive pin modes must communicate at the same baud rate.

UART Format of 8 bits, no parity and 1 stop bit is required. Baud rates of 300,1200,2400,4800,9600,19200,38400,57600 and 115200 bits per second are supported.

A video tutorial of the Serial Wombat 4B chip's UART bridge capabilities is available here:


BROADWELL CONSULTING INC.
Serial Wombat
Adding a UART to Arduino from I2C with the Serial Wombat 4B
fritzing

https://youtu.be/C1FjcaiBYZs

The Serial Wombat 4B chip runs on the PIC16F15214 microcontroller. This microcontroller has a published silicon bug which can cause the same byte to be sent twice when queued once. This issue is believed to be limited to communication at 115200 bps on the Serial Wombat firmware. A host-side workaround is available which prevents this bug from manifesting at the expense of significantly lower total throughput. See this video for details:

https://youtu.be/CxmNPz6fW8E

The SerialWombat 4B Chip has an onboard 64 byte buffer for UART reception to allow the host to queue up to 64 bytes to the chip. This is useful when sending at lower baud rates.

On Arduino the UART Transmit pin mode wrapper inherits from the Stream class allowing a Serial Wombat chip based UART to be interfaced using the same methods as accessing a UART integrated into the host.

A video tutorial of the Serial Wombat 4B chip’s UART bridge capabilities is available here: https://youtu.be/l-XiUQ6RXU0

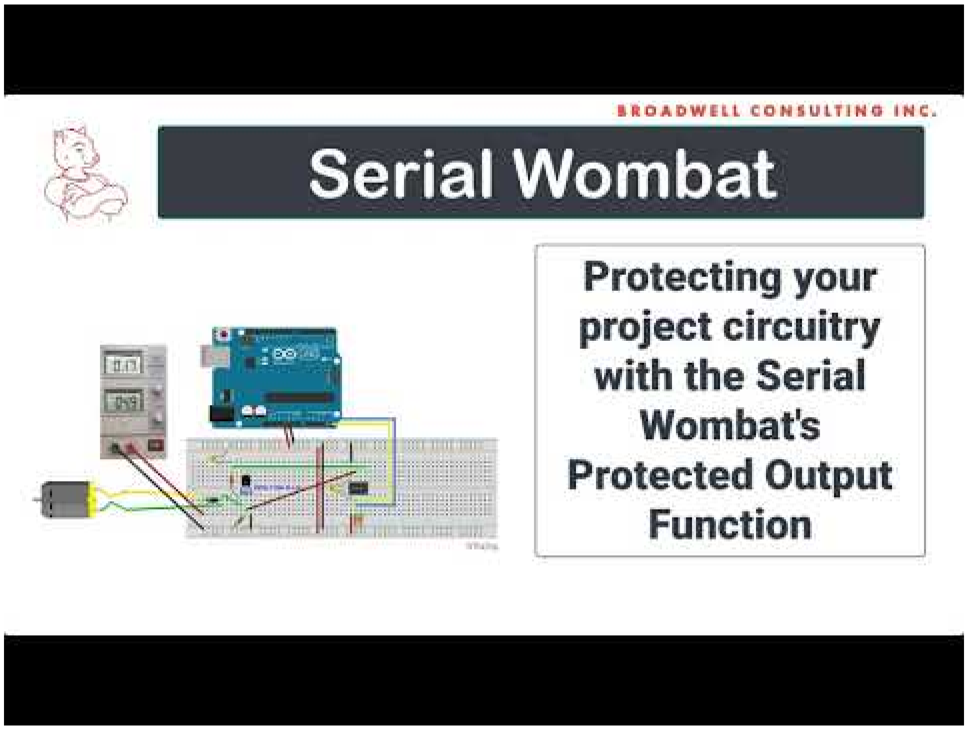
Protected Output Pin Mode

The Seria lWombt Protected Output pin mode is assigned to a Serial Wombat output pin. It monitors another previously configured pin's public data, such as a digital I/O value or an Analog input. If the monitored value does not meet expectations, then the protected pin changes values to a configured state. This allows the Serial Wombat chip to constantly verify a condition without the need for constant polling from the host device.

Warning: The Serial Wombat 4B chip's Protected Output Pin Mode is intended to help prevent accidental damage to hobby circuitry. The Serial Wombat chip and its associated libraries are not designed for use in Safety Critical applications. The Serial Wombat chip should not be used in situations where a malfunction or design defect could result in damage to property, economic loss, or harm to living people or creatures.

The period of time that a mismatch must occur before going to the safe state
is configurable.

A video tutorial of the Serial Wombat 4B chip's protected output capabilities is available here:
https://youtu.be/p8CO04C1q_Y


BROADWELL CONSULTING INC.
Serial Wombat
Protecting your project circuitry with the Serial Wombat's Protected Output Function

The Arduino class is documented here:
https://broadwellconsultinginc.github.io/SerialWombatArdLib/class_serial_wombat_protected_output.html

Here's an Arduino example where pin 3 is configured to monitor pin 1, which is configured as an analog input. The protected output pin is configured to be high unless pin one is higher than 8000 for 10 mS, in which case it latches low until reset by the host.

```
#include <SerialWombat.h>

SerialWombatChip sw;    //Declare a Serial Wombat chip

SerialWombatProtectedOutput swpo(sw);
SerialWombatAnalogInput Feedback(sw);

// This example is explained in a video tutorial at: https://youtu.be/p8CO04C1q_Y

void setup() {
  // put your setup code here, to run once:

  Serial.begin(115200);  //Initialize Arduino Serial Port for terminal use

  {//I2C Initialization
    Wire.begin();
    sw.begin(Wire,0x6C);  //Initialize the Serial Wombat library to use the primary I2C port,
SerialWombat is address 6C.
  }

  swpo.begin(3,1);  // Controlling pin 3.   Feedback from pin 1.
  Feedback.begin(1);  // Begin analog reading on pin 1
}


int i;
void loop() {
      if(swpo.isInSafeState())
     {
      Serial.println("Protected Output Fault Detected, Output set to Safe State!");
     }
     if (i & 0x01)
     {
        swpo.configure(PO_FAULT_IF_FEEDBACK_GREATER_THAN_EXPECTED,8000,10,SW_HIGH,SW_LOW);
        Serial.println("On");
     }
     else
     {
        swpo.digitalWrite(LOW);
        Serial.println("Off");
     }

     delay(100);
     Serial.print ("counts at drain: ");
     Serial.println(Feedback.readCounts());
     Serial.print(Feedback.readVoltage_mV());
     Serial.println(" mV");
     Serial.println();

     delay(3000);
     ++i;
}
```

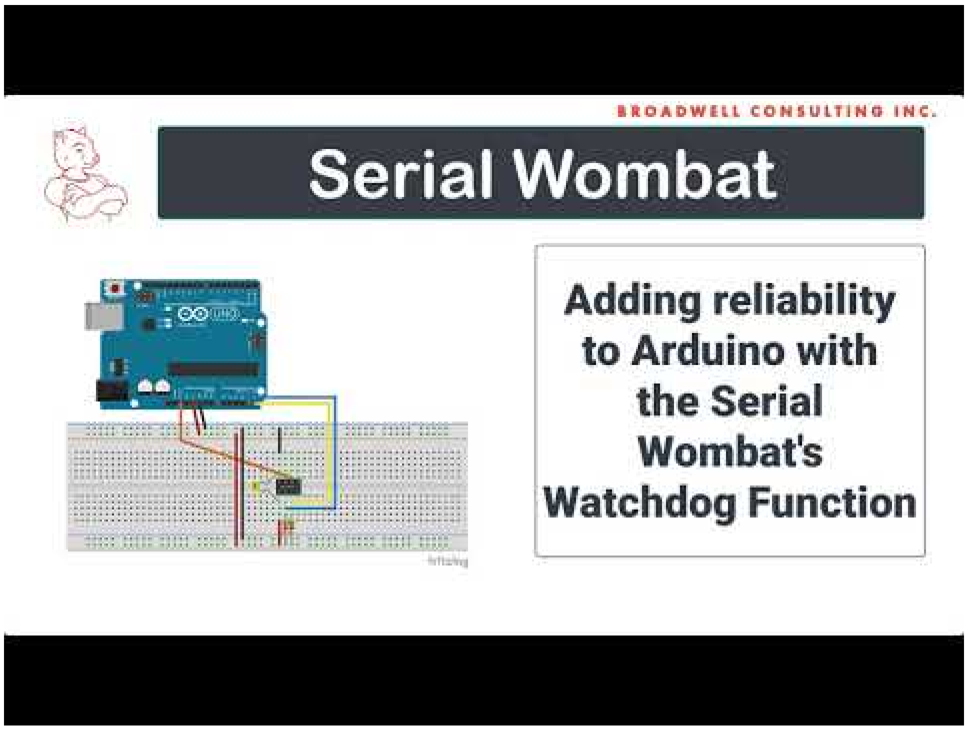
Watchdog Pin Mode

The Serial Wombat Watchdog Pin Mode is designed to improve system reliability in case of communications loss with the host device. This may be because the communications lines are no longer functional (e.g. I2C bus locked up) or the host ceases to communicate (Such as when an Arduino malfunctions due to issues allocating string memory).

Once enabled, the Serial Wombat Watchdog will change its output and optionally other Serial Wombat outputs to predefined states and optionally reset the Serial Wombat itself if a new Watchdog feeding message isn't received within a period of time specified in the initialization.

The output can be used to reset the host, for instance when connected to an Arduino reset pin, or used to shut off an output. For instance, a motor controlled by a SerialWombatWatchdog pin could be configured to turn off if the host doesn't periodically feed the watchdog.

A video tutorial is available:


BROADWELL CONSULTING INC.
Serial Wombat
Adding reliability to Arduino with the Serial Wombat's Watchdog Function

https://youtu.be/fIObjmHmprY

Here’s an example where pin 2 of a Serial Wombat 4B chip is tied to the reset pin of an Arduino. The Arduino chip is designed to malfunction, leaving it stuck in a loop. Eventually, the Serial Wombat 4B pin will pull the Arduino’s reset pin due to not being serviced.

```
#include <SerialWombat.h>

SerialWombatChip sw;     //Declare a Serial Wombat chip
```

```
SerialWombatWatchdog Watchdog(sw);  // Declare a Watchdog pin

// A video tutorial for this example is available at: https://youtu.be/fIObjmHmprY
void setup() {

  {//I2C Initialization
    Wire.begin();
    sw.begin(Wire,0x6C);  //Initialize the Serial Wombat library to use the primary I2C port,
This SerialWombat's address is 6C.
  }
  Watchdog.begin(2,                                // Start the watchdog on pin 2.
                 SW_INPUT,                         // Make the pin Input for normal operation
                 SW_LOW,                           // Make the pin go low on timeout
                 10000,                            // Timeout is 10 seconds
                 false);                           // The Serial Wombat won't self-reset on
timeout


  Serial.begin(115200);
  Serial.println();
  Serial.println("Setup Complete.");
}



// This flawed routine works well if A is a multiple of B, but
// acts badly otherwise because quotient is unsigned and rolls
// back to a big number if the subtraction goes negative.
// Some values, such as 60 / 7 eventually end up returning a
// (wrong) result as the rollover(s) end up eventually
// giving a number that is a multiple of B.
// others such as 60 / 8 stay trapped in the loop forever.
uint8_t DivideAByB( uint8_t A, uint8_t B)
{
  uint8_t C = 0;

  while(A > 0)
  {
    A = A - B;
    ++C;
  }
  return C;
}


int x = 1;
void loop() {

  // put your main code here, to run repeatedly:

   Serial.println();
   Serial.print("60 / ");
   Serial.print(x) ;
   Serial.print(" = ");
   Serial.println(DivideAByB(60,x));
   ++x;

   Watchdog.updateResetCountdown(10000); // Reset the watchdog clock to 10 seconds
   delay(1000);

}
```

Analog Input Pin Mode

The Serial Wombat 4B chip can measure up to 3 separate analog inputs, plus its own source voltage.

The Serial Wombat 4B firmware makes a new 10-bit measurement every 1mS. The pin state machine also provides averaged and filtered values, and keeps track of the minimum and maximum measurements.

Serial Wombat analog measurements are ratiometric, ranging from 0 to 65535 where 0 means that the incoming voltage was equal (within the 10-bit resolution) to ground, and 65535 means that the incoming voltage was equal (within the 10-bit resolution) to the Serial Wombat chip's source voltage. Because 0-65535 represents a 16 bits of resolution but the Serial Wombat 4B chip's A/D unit is only 10 bits, values will exhibit quantization in 64 count increments (e.g. the raw results can be 0, 64, 128, 32768, 51200 … but not 7, 71, 132, 32784, 51213, etc). In the case that the A/D reports a maximum value of 65472, this value is reported as 65535 (to be consistent with using that value to represent positive maximum scale).

Averaged value is computed by adding 64 samples together. For some signals this may effectively increase resolution depending on the amplitude and distribution of noise on the signal. The average output updates every 64 samples, so it some latency is introduced by using this value instead of the raw or filtered values.

Filtered value is computed by taking the previous filtered value times the filter constant + the new value times 65536 minus the filter constant, then dividing the sum by 65536. Given the 1kHz sampling frequency, the following cut-off (3dB down) frequencies can be achieved with constant values:

- 0.5 Hz 65417
- 1 Hz 65298
- 2 Hz 65062
- 5 Hz 64358
- 10 Hz 63202

Filtering adds lag. The higher the filter constant value, the longer it takes for the filter to settle when given a steady input.

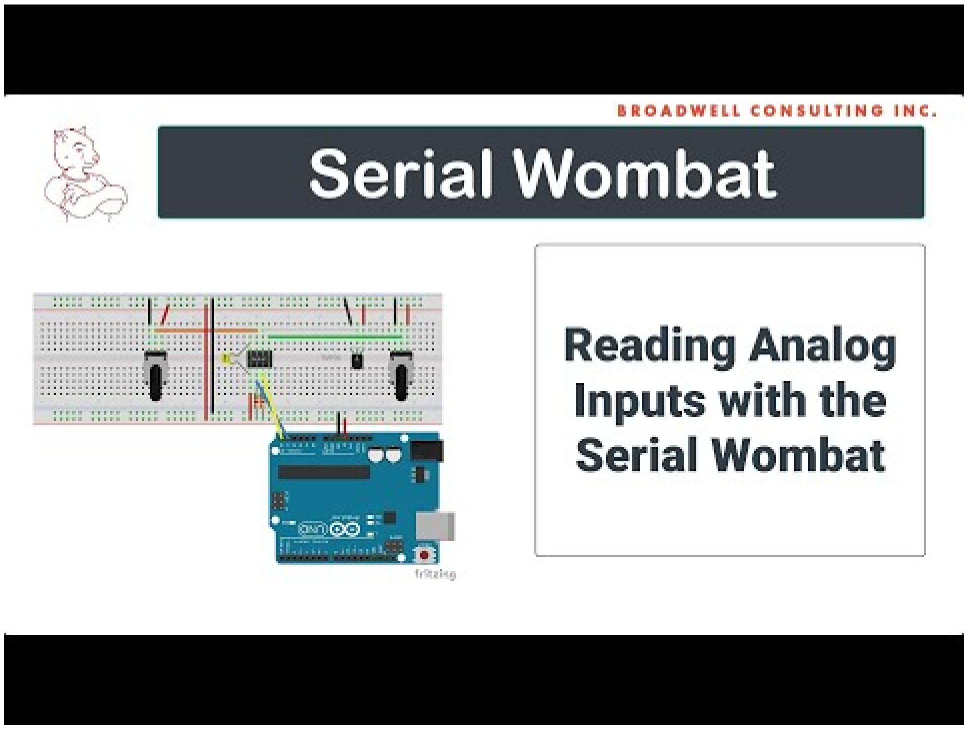
Averaging and filtering are particularly useful on an expansion ADC device such as the Serial Wombat 4B because they allow a higher effective sampling rate than what can be achieved over I2C when also communicating with other devices.

Minimum and maximum tracking record the lowest and highest seen raw A/D conversion value for that pin. These values can optionally be cleared when read in the same communication request. Note that noisy signals can generate minimum or maximum values that are not qualitatively representative of the signal.

The Serial Wombat 4B chip is capable of measuring an internal reference voltage which in turn can be used to infer the system voltage. The Arduino library has functions to retrieve the system voltage, as well as output A/D conversions in mV rather than counts using the measured system voltage as the high value for the ratiometric conversion.

Input impedance for A/D inputs should be 5 kOhm or less. See the PIC16F15214 datasheet for additional performance and electrical characteristics of the A/D converter circuit.

A video tutorial of the Serial Wombat 4B chip's analog capabilities is available here:
https://www.youtube.com/watch?v=_EKIrEVaEhg


BROADWELL CONSULTING INC.
Serial Wombat
Reading Analog Inputs with the Serial Wombat
fritzing

The Arduino class is documented here:
https://broadwellconsultinginc.github.io/SerialWombatArdLib/class_serial_wombat_analog_input.html

Here's an Arduino example of the using 3 pins to monitor two potentiometers and a TMP32 temperature sensor using the Analog Input pin mode:

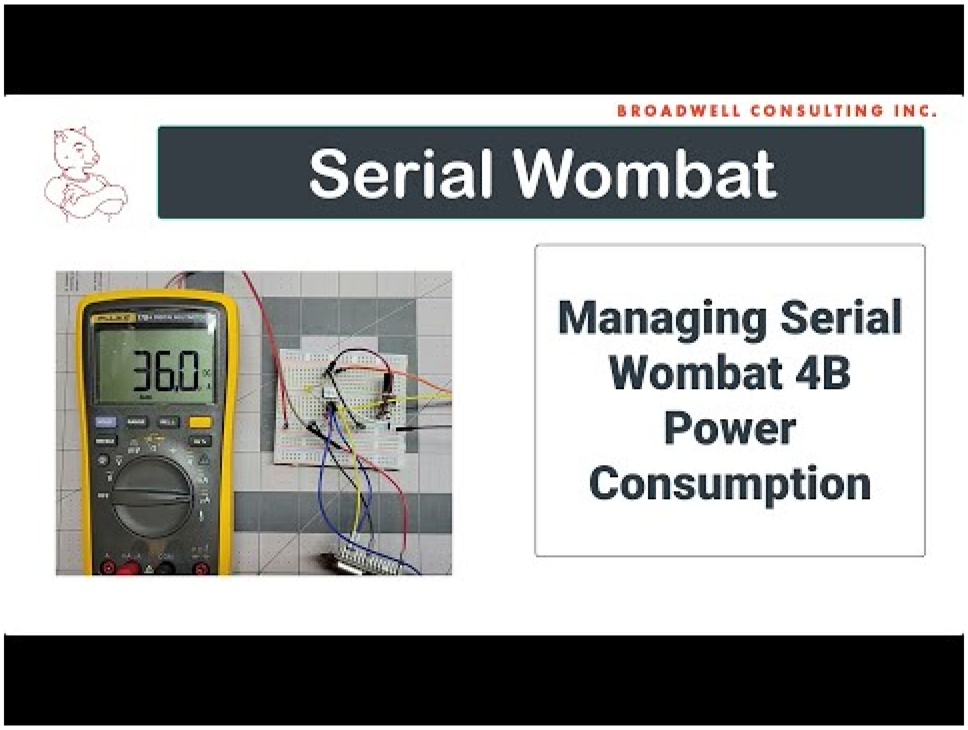
```
#include <SerialWombat.h>
```

```
SerialWombatChip sw6C;    //Declare a Serial Wombat
SerialWombatAnalogInput leftPot(sw6C);  //5k linear Pot
SerialWombatAnalogInput rightPot(sw6C); //5k linear Pot
SerialWombatAnalogInput temperatureSensor(sw6C);

// This example is explained in a video tutorial at: https://youtu.be/_EKlrEVaEhg


void setup() {
  // put your setup code here, to run once:

  { //I2C Initialization
    Wire.begin();
    digitalWrite(A5,LOW);  //Arduino Uno Specific.  Turn off I2C pull up
    digitalWrite(A4,LOW);  //Arduino Uno Specific.  Turn off I2C pull up
    sw6C.begin(Wire, 0x6C); //Initialize the Serial Wombat library to use the primary I2C port,
SerialWombat is address 6C
  }
  leftPot.begin(3);
  rightPot.begin(1);
  temperatureSensor.begin(2,64,65417); // Wombat pin 2, average 64 samples, .5 Hz Low Pass filter
  Serial.begin(115200);
}

void loop() {

  Serial.print("Source V: ");
  uint16_t supplyVoltage = sw6C.readSupplyVoltage_mV();
  Serial.print(supplyVoltage );
  Serial.print("mV      Left Pot: ");
  Serial.print(leftPot.readCounts());
  Serial.print(" ");

  uint16_t leftVoltage = leftPot.readVoltage_mV();
  Serial.print(leftVoltage);
  Serial.print("mV      Right Pot:");

  Serial.print(rightPot.readCounts());
  Serial.print(" ");
   uint16_t rightVoltage = rightPot.readVoltage_mV();
   Serial.print(rightVoltage);

  Serial.print("mV      T:");

  Serial.print(temperatureSensor.readCounts());
  Serial.print(" ");
  Serial.print(temperatureSensor.readVoltage_mV());
  Serial.print("mV ");


  float tempSensor_mV = temperatureSensor.readAveraged_mV();

  //See datasheet for TMP36 Temperature sensor for conversion
   float temperature = (tempSensor_mV - 750) / 10.0 + 25;

  Serial.print(temperature);
  Serial.print(" deg C ");


  Serial.println();
  delay(200);
}
```

Sleep Mode

The Serial Wombat 4B chip is typically running firmware at 32MHz internally, leading to a constant consumption of a few mA of current. The Serial Wombat 4B chip can be configured to enter Sleep mode in which execution of the executive stops, and the Serial Wombat 4B chip waits for an I2C command to wake and resume operation. In Sleep mode the current is typically in the low microamps range, dependent on the configuration of the chip when Sleep was commanded. Note that the chip stops execution of all functions, including protected output and PWM generation, when Sleep is commanded. The state of PWM and UART outputs (high or low) is not specified at this point. Any outputs should be put in a known state by the host prior to entering Sleep mode.

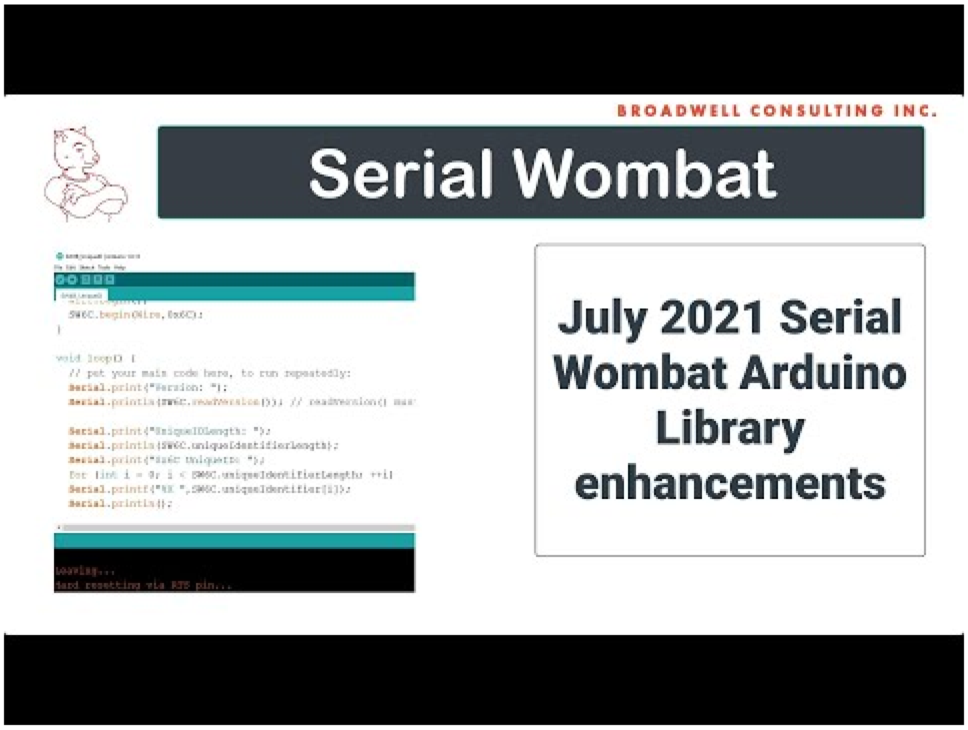
A video on using Sleep Mode is available here:


BROADWELL CONSULTING INC.
Serial Wombat
36.0
Managing Serial Wombat 4B Power Consumption

https://youtu.be/jVkQ1YoqcpI

Unique Identifier

Each Serial Wombat 4B chip has a unique, unchangeable identifier programmed into the microcontroller when it is manufactured by Microchip. A demonstration of this capability is shown in this video:


BROADWELL CONSULTING INC.
Serial Wombat
July 2021 Serial Wombat Arduino Library enhancements

```
  SW6C.begin(Wire,0x6C);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print("Version: ");
  Serial.println(SW6C.readVersion()); // readVersion() mus
  
  Serial.print("UniqueIDLength: ");
  Serial.println(SW6C.uniqueIdentifierLength);
  Serial.print("Sw6C UniqueID: ");
  for (int i = 0; i < SW6C.uniqueIdentifierLength; ++i)
  Serial.printf("%X ",SW6C.uniqueIdentifier[i]);
  Serial.println();
```

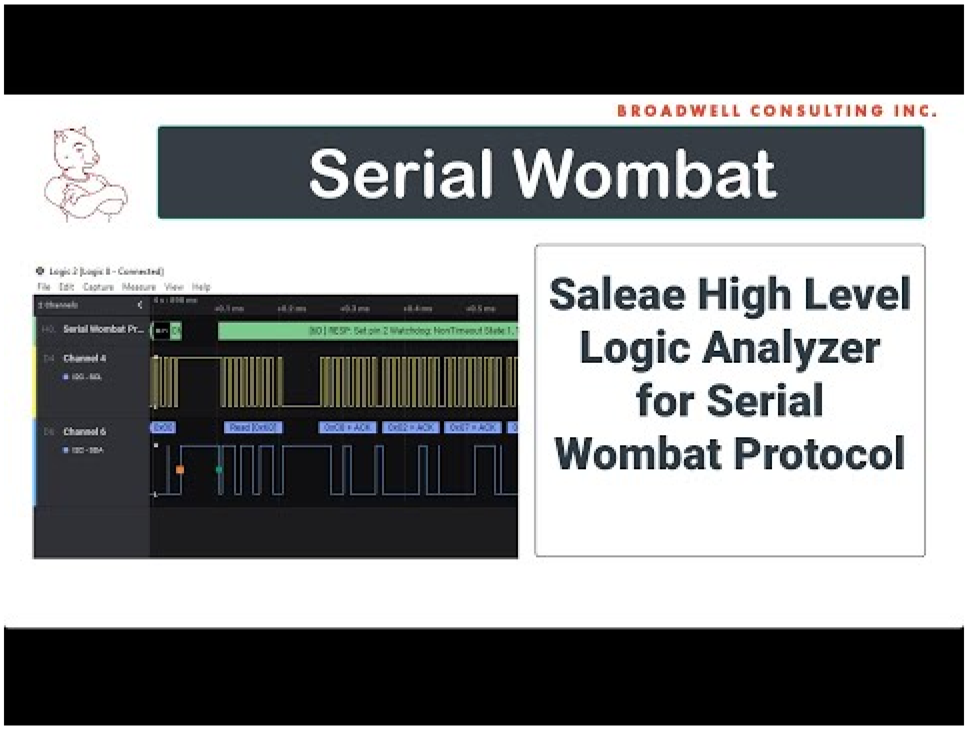
```
Loading...
Hard resetting via RTS pin...
```

https://youtu.be/lHTcKyXT_2Q

Protocol Analyzer

A protocol analyzer is available to monitor and decode Serial Wombat commands sent over an I2C or UART bus. This analyzer runs on top of the Saleae Logic software package and is available for download through that application. This tool can be useful in debugging projects that include the Serial Wombat 4B chip.

A video is available here:


BROADWELL CONSULTING INC.
Serial Wombat
Saleae High Level Logic Analyzer for Serial Wombat Protocol

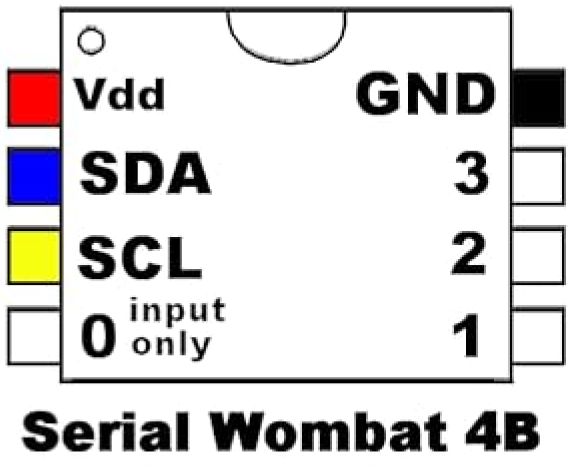
https://youtu.be/cL7kUm9qjvU

Troubleshooting

Step 1: Check the basics

General Stuff:

- Make sure your chip has a stable, in-range power supply and that the included capacitors are attached across the power and ground pins. If you can, verify power voltage using a multimeter
- Make sure your chip is connected properly. Ensure that the chip is in the proper orientation (power pins are near the notch) and that a 100uF capacitor is connected across Vdd and GND.



- Disconnect any loads (such as motors, servos, or relays) from your circuit. Frequently these devices can sufficiently disrupt the power supply such that the Serial Wombat 4B chip's internal low-voltage-reset circuit triggers.   It is suggested that inductive loads not be driven directly from the same supply driving logic portions of the circuit.

- If you're using a decal on the Serial Wombat chip, is it oriented in the correct direction? Make sure the black notch at one end of the decal matches up with the black dot on the chip

- Make sure you're using the latest Arduino / C# / Python library. It's possible your issue is fixed in a newer version than you have

- In Arduino, consider registering the default error handler. This provides helpful information (particularly on the 18AB) about configuration errors.

I2C related Stuff

- If connected using I2C, make sure you have pull up resistors on SCL and SDA (don't rely on internal pull ups on your chip, they're typically out of spec for I2C).

- Make sure your SDA line from the Arduino or other host is attached to the SDA line on the Serial Wombat Chip.   Same for SCL. Did you accidentally cross them?

- Does the Serial Wombat chip finder sketch in the Arduino Examples / Serial Wombat directory find the Serial Wombat chip? This should always work if your hardware is setup correctly

- In your sketch, do you call begin on Wire then on the Serial Wombat chip?

- Can you verify proper I2C traffic operation using a Logic analyzer? See this video for a cheap way to do this. https://youtu.be/cL7kUm9qjvU

Step 2: Check the YouTube video and comments

Go to the Broadwell Consulting Inc. YouTube channel and take a look at the Serial Wombat playlist. Watch the video for the task you're trying to achieve, and check the comments to see if any other users have asked a question about your issue.
If not, then leave a comment with your question on the video that best matches what you're trying to do.

Additional Resources:

YouTube

The Broadwell Consulting Inc. YouTube Channel has many helpful tutorial Videos which walk through how to use the various Serial Wombat chip pin modes and features.

Arduino Library

The Serial Wombat Arduino Library supports the Serial Wombat 4B and Serial Wombat 18AB chips.

The library documentation is available on github.io . Click on the classes tab to see documentation and interfaces for individual pin modes.

The Serial Wombat Arduino Library is available on GitHub . This is a good place to log an issue if you find a bug in the Arduino library or want to request new features. Please don't use the issue system for support requests.

Serial Wombat 4B firmware

The Serial Wombat 4B source code documentation and protocol definition are available on github.io .

The Serial Wombat 4B firmware source code is available on GitHub.

Support and Technical Assistance

If the above troubleshooting and guides don't solve your problem, contact Broadwell Consulting at help@serialwombat.com for support. Support requests sent in over email may take a couple of days to respond. Priority is given to questions asked in public forums such as on the YouTube channel so that others can benefit from the answers.

Revision History

Version	Changes
V2.0.3_A	Initial Version
V2.0.3_B	Added Quadrature Encoder Pin Mode, Updated Title page