Circuit Monkey

500-0023-01

and

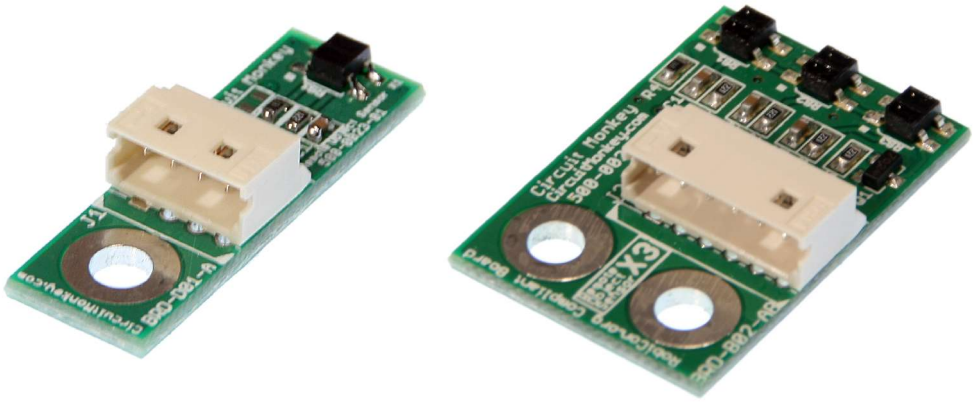
500-0024-01

Reflective Object Sensor

Fairchild QRE1113GR based modules

Rev. A

User Guide



Version: 1.0

2010/01/30

Illustrations and photographs by Mark J Koch
This document was created with *OpenOffice 3*

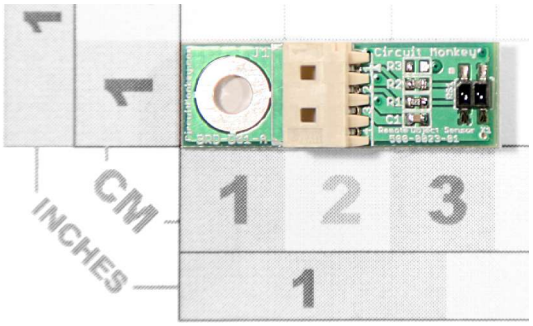
Table of Contents

Overview

Circuit Monkey [http://circuitmonkey.com] presents part numbers ***500-0023-01 and 500-0024-01***, reflective-object-sensor (***ROS***) boards based on the *Fairchild QRE1113GR* Reflective Object Sensor. The *ROS* module provides sensing of near objects and/or surface reflection for a wide variety of applications. The sensor incorporates a infrared LED as the reflected light source. The ROS modules from Circuit Monkey are available in two models, a single sensor and triple sensor. The single sensor is well suited for applications such as near object sensing, and motion limit sensing. The triple sensor is well suited for line-following. velocity and direction measurement.

The Input/Output signals on the ROS module are connected using *Molex 53015 (MicroBlade)* series for reliable and vibration resistant connections. These connectors are *RobiCon.org* compliant *(an open-source interconnect standard that we are proposing)*. The board is 40mm x 20mm and features two mounting holes placed on a 10mm apart at one end of the board. The holes are plated and connected to logic ground and accept screws of size M3 or less *(M3, 6-32, M2.5, 4-40, etc.)*.

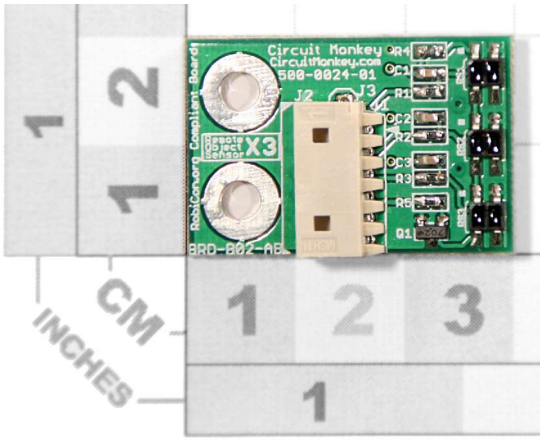
Single Sensor (X1) – *PN: 500-0023-01*



Features

- Power supply voltage; VCC = 5V
- 4mm range
- Perfect for limit detection.
- Operating Current:
  - Up to 30mA with LED operating.
- *RobiCon.org* compliant:
  - *Molex MicroBlade* connector, 4-pin.
  - Board outline and mounting.
- Dimensions:
  - 30mm x 10mm x 6.8mm (WxDxH)
  - 1.18"W x 0.39"D x 0.27”H

Triple Sensor (X3) – *PN: 500-0024-01*



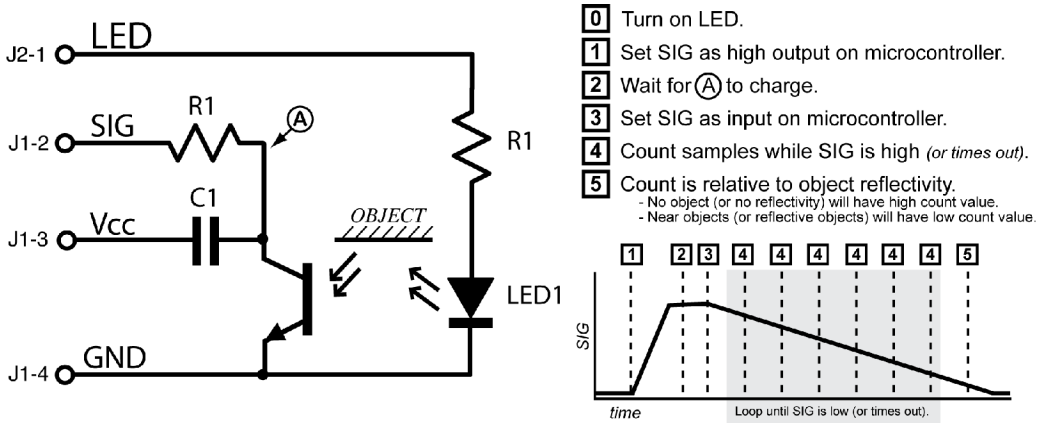
Features

- Power supply voltage; VCC = 5V
- 4mm range
- Perfect for limit detection and line following.
- Operating Current:
  - Up to 30mA with LED operating.
- *RobiCon.org* compliant:
  - *Molex MicroBlade* connector, 4-pin.
  - Board outline and mounting.
- Dimensions:
  - 30mm x 20mm x 6.8mm (WxDxH)
  - 1.18"W x 0.79"D x 0.27”H

Block Diagrams

ROS X1 sensor

The ROS X1 module features a single Fairchild QRE1113GY reflectance sensor. The sensor is capable of measuring distance and/or reflectance of objects within 4mm of the sensor. The built in LED acts as a light source for the built-in photo-transistor. In the presence of infrared light, the sensor will conduct electricity.



A block diagram and timing diagram representing the single (X1) ROS module board is shown below.

As the the ROS module is a digital output, we use a resistor and capacitor in the circuit to convert the analog value at point "A" in the diagram into a digital value. Using a timing loop to measure decay of signal "A" we can measure varying proximity or reflectance of objects near the sensor.

In software, a simple method of getting the reflectance value is accomplished by 'charging' the signal at point "A" up to Vcc (+5V) and then reading back the value at SIG. The program loops and monitors the logic value of SIG while keeping a count of how many loops have been run. When an object is near the sensor, the phototransistor will begin to conduct electricity and quickly drain the charge from point "A" causing the logic value at SIG to quickly reach logic 0. When there is no object or the object is not very reflective, the charge at point "A" will remain.

Q: When waiting to charge node "A", how long do I wait?

A: Node "A" is part of a simple RC (resistor-capacitor) network. When a voltage is applied at SIG it will take five *RC time constants* to charge node "A" to that voltage. Here's the formula:

$$t = 5 \cdot R \cdot C$$

In our case: ***R*** = *220* ; ***C*** = *0.01uF*

$$t = 5 \cdot 220 \cdot 0.00000001$$

$$t = 0.000011\,S = 11\text{uS}$$

See: http://www.tpub.com/neets/book2/3d.htm for a concise explanation.

Analog Output (Custom Configuration)

Alternately, the sensor could be configured for input to an analog pin on your microcontroller. The sensor would put out a voltage inversely proportional to reflected light. That configuration involves replacing the capacitor at C1 with a pull-up resistor (try 10K). Circuit Monkey plans to provide that configuration as a product soon. Please contact us if you need any customization.

LED

The LED is an infrared type and built into the QRE1113 component. It can be turned on and off via J1 pin 1.

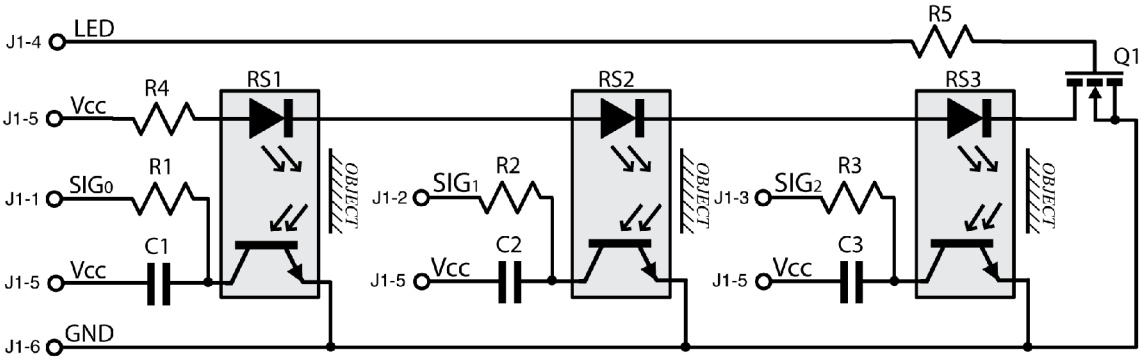
Is it on?

When on, the LED is not visible to the human eye. When debugging your circuit, you can see if the LED is on by viewing it through a video camera (or your cell phone camera). Most video cameras can pick up infrared light.

ROS X3 Sensor

The ROS X3 module features three Fairchild QRE1113GY reflectance sensors placed 8mm apart from each other. The sensor is capable of measuring distance and/or reflectance of objects within 4mm of the sensor. The built in LED (in each sensor) acts as a light source for the built-in photo-transistor. In the presence of infrared light, the sensor will conduct electricity.

A block diagram representing the triple (X3) ROS module board is shown below.



The function of each sensor is identical to the ROS X1 board. The primary difference is that the LED signal controls all three LED emitters (all on or all off). The "SIG" outputs are independent of each other.

See the detailed description of the ROS X1 module in the previous section to learn more about the theory of operation for this device.

Connectors

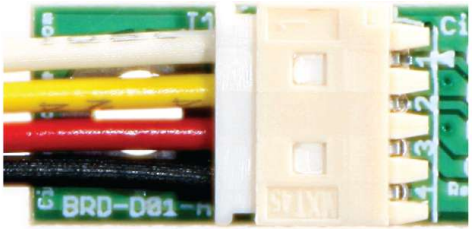
ROS X1 – PN: 500-0023-01

J1 – Main Connector

This connector features four contacts, LED, SIG, +5V and Ground. Provide power to the board by hooking +5VDC to pin 3 and Ground on Pin 4. Connect pin 1 and 2 to your host processor.

Pin	Description
1	LED
2	SIG
3	Vcc (+5VDC)
4	Ground

1 : LED
2 : SIG
3 : +5VDC
4 : Ground



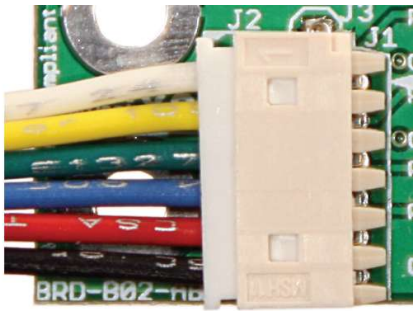
ROS X3 – PN: 500-0024-01

J1- Main Connector

This connector features six contacts, LED, SIG0, SIG1, SIG2, +5V and Ground. Provide power to the board by hooking +5VDC to pin 5 and Ground on Pin 6. Connect pin 1, 2 ,3 and 4 to your host processor.

Pin	Description
1	LED
2	SIG0
3	SIG1
4	SIG2
5	Vcc (+5VDC)
6	Ground

1 : LED
2 : SIG0
3 : SIG1
4 : SIG2
5 : +5VDC
6 : Ground



Applications

ROS X1 (500-0023-01) Sensor Module Test

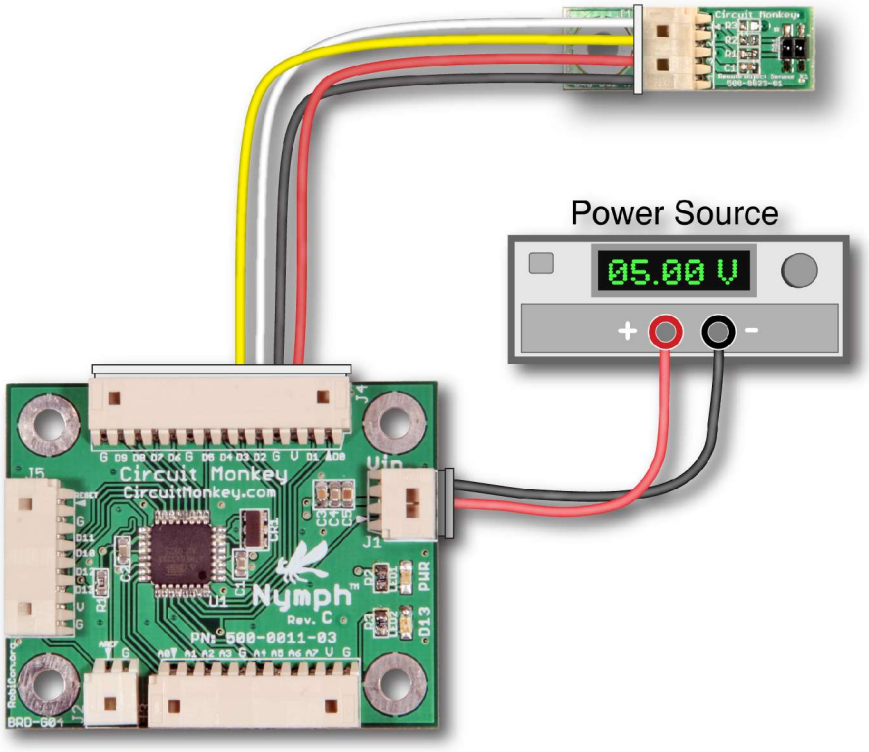
Function

This test is designed to do the following:

- Monitor the sensor
- Near object brightens the *Nymph* D13 LED. Far (or no object) dims the D13 LED.

Hookup

Our actual test hook-up is shown below. Test code follows the hookup diagram.



ROS x1 module J1 Pin	Description	Wire Color	Nymph Controller J4 Pin	Atmega328P Signal	Arduino Signal
1	**LED**	**White**	*5*	*PD2*	*D2*
2	**SIG**	**Yellow**	*6*	*PD3*	*D3*
3	**+5V (Vcc)**	**Red**	*3*		
4	**Ground**	**Black**	*4*		

ROS module Test Code (for Atmega328P – Nymph Compatible)

Using the preceding hook-up diagram, one can test the functionality of the ROS module board with the following code. It was designed to run on a *ATmega168/328* board. We tested with our own *Nymph* product but this code should also run on an *Arduino/Freeduino* or similar board, as they are also *Atmel AVR* based.

```
/**
  Circuit Monkey
  500-0023-01  --  Remote Object Sensor X1 -- Manufacturing Test

  Designed for Nymph (ATmega328) Microcontroller

  Author:  Mark J Koch

  Description:  The 500-0023-01 Remote Object Sensor (X1) board features a
  Fairchild QRE1113GY reflective bject sensor on a RobiCon.org compliant
  printed circuit board (PCB).  The board features a 4-pin RobiCon compliant
  connector which supplies Ground, Power (+5VDC), IR LED signal and signal
  output.

  500-0023-01 Pin-out:
      1 - Signal
      2 - LED
      3 - Vcc (+5VDC)
      4 - GND

  Nymph Hookup:

      J4-3  Vcc
      J4-4  GND
      J4-5  IR LED  -- PD2
      J4-6  Signal  -- PD3

  How this code works:
      1. Initialize display LED at D13 (on Nymph board).
      2. Initialize IR LED.
      3. Main loop runs poor-man's PWM for display LED brighness based on
         polled 'Signal' value.

================================================================================
License:   BSD

Copyright (c) 2009, Circuit Monkey
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions  in binary form   must reproduce the  above copyright
      notice,  this list of conditions and  the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of Circuit Monkey  nor the names of its contributors
      may be used to endorse or promote products derived from this software
      without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,  BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY   DIRECT,  INDIRECT,  INCIDENTAL,  SPECIAL,  EXEMPLARY,  OR
```

```
CONSEQUENTIAL  DAMAGES  (INCLUDING,  BUT  NOT  LIMITED TO,  PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES;  LOSS OF USE, DATA, OR PROFITS;  OR BUSINESS
INTERRUPTION)  HOWEVER CAUSED  AND ON ANY THEORY OF LIABILITY,  WHETHER IN
CONTRACT,  STRICT  LIABILITY, OR TORT  (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
==========================================================================
 **/

#include <avr/io.h>
#define F_CPU 16000000UL  // 16 MHz - Nymph Crystal
#include <util/delay.h>

#define MAX_SAMPLES 3600

// Define LED Pin on Nymph (D13 = PB5)
#define LEDOUT PORTB5
#define LEDPORT PORTB
#define LEDDDR DDRB
#define LEDDDRPIN DDB5

// Define IR LED Pin :  Nymph (D2 = PD2)
#define ROS_IRLED_PORT  PORTD
#define ROS_IRLED_OUT   PORTD2
#define ROS_IRLED_DDR   DDRD
#define ROS_IRLED_DDRPIN DDD2

// Define Sig Pins on Nymph (D3 = PD3)
#define SIG_0_PORT      PORTD
#define SIG_0_IN  PIND3
#define SIG_0_OUT PORTD3
#define SIG_0_DDR       DDRD
#define SIG_0_DDRPIN    DDD3


/* _delay_ms uses a floating point datatype. If you call
   that function many places in your code, the hex binary
   becomes bloated. An integer is enough for us:

   delay x milliseconds
 */
void delay_ms(unsigned int x) {
        while(x){
                _delay_ms(0.96);
                x--;
        }
}

void initLED() {
        LEDDDR |= (1<<LEDDDRPIN);  // enable pin as output
}

void setSigAsOutput(void) {
      SIG_0_DDR |= (1<<SIG_0_DDRPIN);
}

void setSigAsInput(void) {
      SIG_0_DDR &= ~(1<<SIG_0_DDRPIN);
}

void setSig(int v) {
      if ( v == 1 ) SIG_0_PORT |=  (1<<SIG_0_OUT);
      else          SIG_0_PORT &= ~(1<<SIG_0_OUT);
}
```

```
void initIrLed() {
       ROS_IRLED_DDR|= (1<<ROS_IRLED_DDRPIN); // enable pin as output
      irLedOff();
}

void irLedOn() {
      ROS_IRLED_PORT |= (1<<ROS_IRLED_OUT); // STBY Pin High
}

void irLedOff() {
      ROS_IRLED_PORT &= ~(1<<ROS_IRLED_OUT); // STBY Pin Low
}

void setLED( int v ) {
      if ( v == 1 ) LEDPORT |=  (1 << LEDOUT);
      else          LEDPORT &= ~(1 << LEDOUT);
}

int getSig() {
      return (PIND & (1<<PIND4));   // Returns 0 when object near.
}

int measureSig() {
      int sampleCount = 0;

      setSigAsOutput();             // Set SIG as output
      setSig(1);              // Set SIG high
      _delay_us(11.0);        // Charge line for 11uS
      setSigAsInput();        // Make SIG input (high impedance)

      // take measurements until line goes low or times out.
      while ( sampleCount < MAX_SAMPLES && getSig() ) sampleCount++;

      return sampleCount;
}

int main(void) {
      int onTime = 0;
      initLED();
      initIrLed();
      while (1) {
            irLedOn();
            onTime = measureSig(0);
            setLED(0);
            delay_ms(onTime/100);
            setLED(1);
            delay_ms((MAX_SAMPLES-onTime)/100);
        }
        return(0);  // Never gets here.
}
```

ROS X3 (500-0024-01) Sensor Module Test

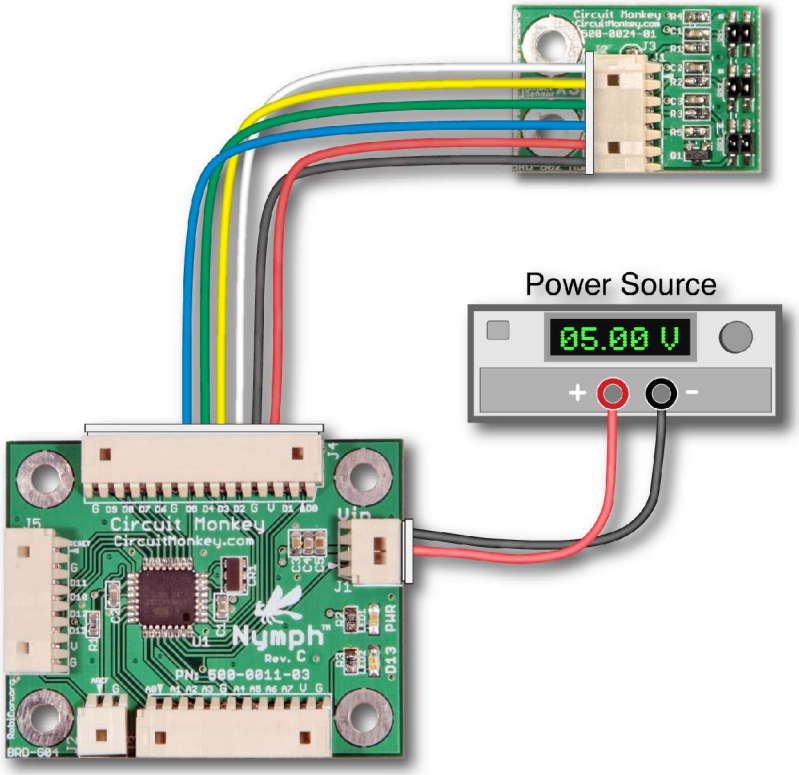
Function

This test is designed to do the following:

- Monitor the sensors
- Print out values to serial connection.

Hookup

Our actual test hook-up is shown below. This hookup requires a serial connection to your computer using a TTL level serial adapter such as the FTDI R232-TTL-5V. Test code follows the hookup diagram.



ROS X3 module J1 Pin	Description	Wire Color	Nymph Controller J4 Pin	Atmega328P Signal	Arduino Signal
1	**LED**	**White**	*5*	*PD2*	*D2*
2	**SIG0**	**Yellow**	*6*	*PD3*	*D3*
3	**SIG1**	**Blue**	*7*	*PD4*	*D4*
4	**SIG2**	**Green**	*8*	*PD5*	*D5*
5	**+5V (Vcc)**	**Red**	*3*		
6	**Ground**	**Black**	*4*		

ROS X3 module Test Code (for Atmega328P – Nymph Compatible)

Using the preceding hook-up diagram, one can test the functionality of the ROS module board with the following code. It was designed to run on a *ATmega168/328* board. We tested with our own *Nymph* product but this code should also run on an *Arduino/Freeduino* or similar board, as they are also *Atmel AVR* based.

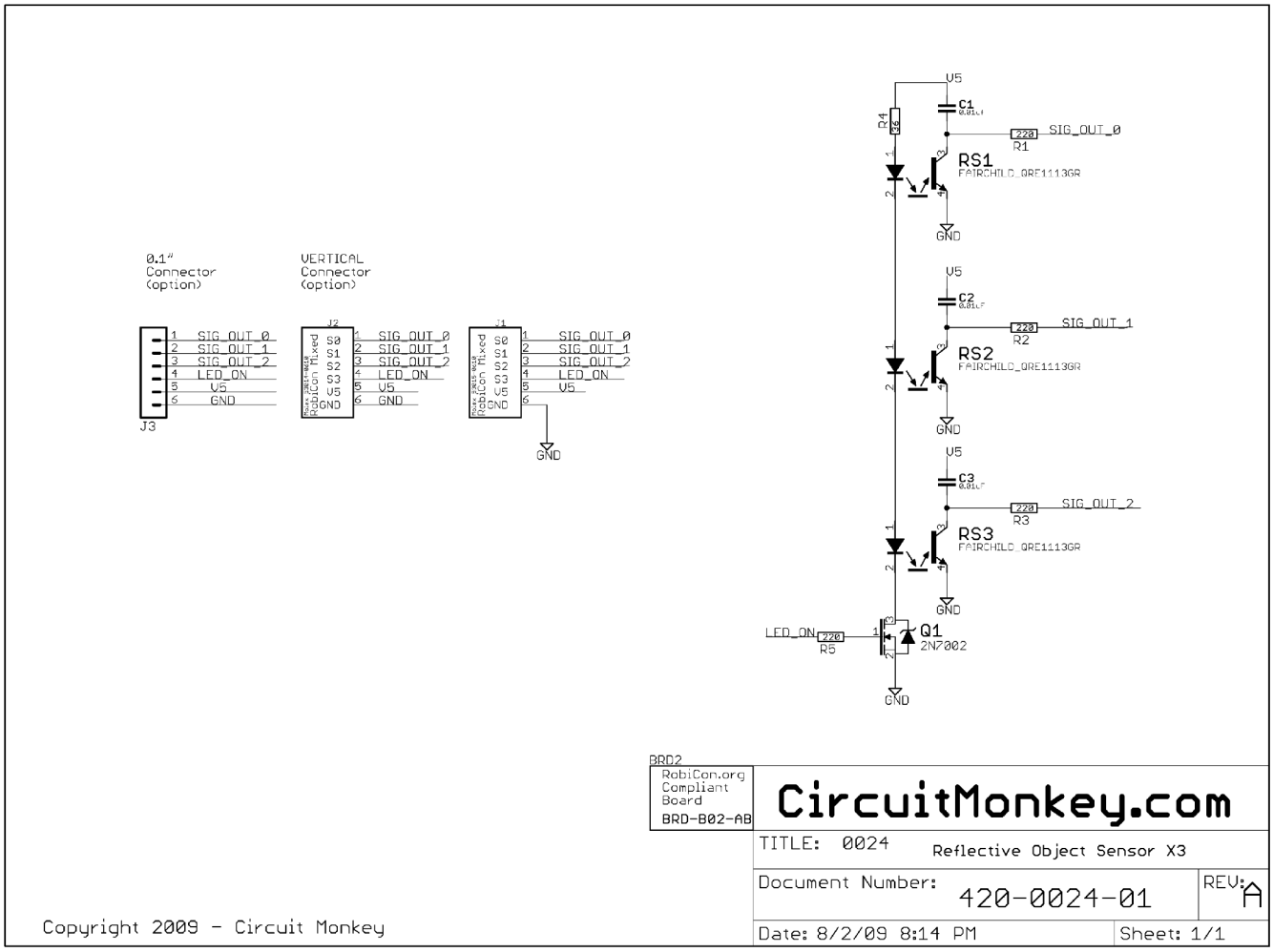
```
/**
  Circuit Monkey
  500-0024-01  --  Reflective Object Sensor (X3) Board Test

  Designed for Nymph (ATmega328) Microcontroller

  Author:  Mark J Koch

  Description:  The 500-0024-01 Remote Object Sensor X3 (ROS-X3) board
features
  three sensors on a RobiCon.org compliant printed circuit board (PCB).  The
  board features a 6-pin RobiCon compliant connector which supplies Ground,
  Power (+5VDC), LED emitter signal and three sensor signals.
  value.

  500-0023-01  J1 Pin-out:
      1 - LED Emitter
      2 - Signal 0
      3 - Signal 1
      4 - Signal 2
      5 - Vcc (+5VDC)
      6 - GND

  Nymph Hookup:

      J4-3  Vcc
      J4-4  GND
      J4-5  Signal 0  -- PD2
      J4-6  Signal 1  -- PD3
      J4-7  Signal 2  -- PD4
      J4-8  LED Emitter -- PD5

  How this code works:
      1. Initialize IR Emitter LED.
      2. Initialize Serial UART on PD0 and PD1.
      3. Initialize Inputs ( Sig 0, Sig 1, Sig 2 )
      4. Main loop polls Sig 0 through 2 and prints values out to serial
         port.

================================================================================
License:   BSD

Copyright (c) 2009, Circuit Monkey
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions  in binary form   must reproduce the  above copyright
      notice,  this list of conditions and  the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of Circuit Monkey  nor the names of its contributors
      may be used to endorse or promote products derived from this software
      without specific prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,  BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY   DIRECT,  INDIRECT,  INCIDENTAL,  SPECIAL,  EXEMPLARY,  OR
CONSEQUENTIAL  DAMAGES  (INCLUDING,  BUT  NOT  LIMITED TO,  PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES;  LOSS OF USE, DATA, OR PROFITS;  OR BUSINESS
INTERRUPTION)  HOWEVER CAUSED  AND ON ANY THEORY OF LIABILITY,  WHETHER IN
CONTRACT,  STRICT  LIABILITY, OR TORT  (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
============================================================================
 **/

#include <avr/io.h>
#define F_CPU 16000000UL  // 16 MHz - Nymph Crystal
#include <util/delay.h>

#define BPS 9600

// Define IR LED Pin :  Nymph (PD5)
#define ROS_IRLED_PORT  PORTD
#define ROS_IRLED_OUT   PORTD5
#define ROS_IRLED_DDR   DDRD
#define ROS_IRLED_DDRPIN DDD5

// Define Sig Pins on Nymph (PD2,PD3,PD4)
#define SIG_PORT  PORTD
#define SIG_DDR   DDRD
#define SIG_IN          PIND

#define SIG_0_IN  PIND2
#define SIG_0_OUT PORTD2
#define SIG_0_DDRPIN    DDD2

#define SIG_1_IN  PIND3
#define SIG_1_OUT PORTD3
#define SIG_1_DDRPIN    DDD3

#define SIG_2_IN  PIND4
#define SIG_2_OUT PORTD4
#define SIG_2_DDRPIN    DDD4

// _delay_us uses a floating point datatype if you call
// that function in many places in your code then the binary becomes
// bloated. An integer is enough for us:
//
// delay x microseconds:
void delay_us(unsigned int us) {
        while(us){
                _delay_us(0.96);
                us--;
        }
}

void initSigs() {
      setSigAsInput( (int)0 );
      setSigAsInput( (int)1 );
      setSigAsInput( (int)2 );
}


void setSigAsOutput( int channel ) {
```

```
      switch ( channel ) {
            case 0: SIG_DDR |= (1<<SIG_0_DDRPIN);
                    break;
            case 1: SIG_DDR |= (1<<SIG_1_DDRPIN);
                    break;
            case 2: SIG_DDR |= (1<<SIG_2_DDRPIN);
                    break;
      }
}

void setSigAsInput( int channel ) {
      switch ( channel ) {
            case 0: SIG_DDR &= ~(1<<SIG_0_DDRPIN);
                    break;
            case 1: SIG_DDR &= ~(1<<SIG_1_DDRPIN);
                    break;
            case 2: SIG_DDR &= ~(1<<SIG_2_DDRPIN);
                    break;
      }
}

void setSig( int channel, int val ) {
      switch( channel ) {
            case 0:
                  if ( val == 1 ) SIG_PORT |= (1<<SIG_0_OUT);
                  else SIG_PORT &= ~(1<<SIG_0_OUT);
                  break;
            case 1:
                  if ( val == 1 ) SIG_PORT |= (1<<SIG_1_OUT);
                  else SIG_PORT &= ~(1<<SIG_1_OUT);
                  break;
            case 2:
                  if ( val == 1 ) SIG_PORT |= (1<<SIG_2_OUT);
                  else SIG_PORT &= ~(1<<SIG_2_OUT);
                  break;
      }
}


void initIrLed() {
        ROS_IRLED_DDR|= (1<<ROS_IRLED_DDRPIN); // enable pin as output
      irLedOff();
}

void irLedOn() {
      ROS_IRLED_PORT |= (1<<ROS_IRLED_OUT);  // Turn it on.
}

void irLedOff() {
      ROS_IRLED_PORT &= ~(1<<ROS_IRLED_OUT); // Turn it off
}

void sigChargeWait() {
      delay_us(11); // Wait 11uS
}

int getSig(int channel ) {
      switch ( channel ) {
            case 0:     return (SIG_IN & (1<<SIG_0_IN) );
            case 1:     return (SIG_IN & (1<<SIG_1_IN) );
            case 2:     return (SIG_IN & (1<<SIG_2_IN) );
      }
      return -1; // Bad channel requested.
}
```

```
int measureSig(int channel) {
      char str[120];
      int maxSamples =  3600;       // How many loops before time-out
      int sampleCount = 0;

      setSigAsOutput(channel);      // Set SIG_x as output
      setSig(channel,1);
      sigChargeWait();        // Charge line for 11uS
      setSigAsInput(channel);       // Make SIG_x input (high impedance)

      // take measurements until line goes low or times out.
      while ( sampleCount < maxSamples && getSig(channel) ) sampleCount++;

      return sampleCount;
}

void serialInit(unsigned int bt) {
      UBRR0H = (unsigned char) (bt >> 8);       // Set the baud rate
      UBRR0L = (unsigned char) bt;
      UCSR0C = (3 << UCSZ00);             // Framing  8N1
      UCSR0B = (1 << RXEN0) | (1 << TXEN0);     // Enable RX and TX
}

void serialWrite(unsigned char c) {
      while ( !(UCSR0A & (1 << UDRE0)) )
            ;
      UDR0 = c;
}

int writeSerialString(char *str) {
      char i = 0;
      int done = 0;
      while (!done) {
            serialWrite(str[i++]);
            if (str[i] == '\0') done = 1;
      }
      return 0;
}

int main(void) {
      int onTime0 = 0;
      int onTime1 = 0;
      int onTime2 = 0;
      char str[50] = "";

      serialInit( ( F_CPU / BPS / 16 ) - 1);
      writeSerialString( "Circuit Monkey -- ROS-X3 Tester\r\n");

      initIrLed();
      initSigs();
      while (1) {
            //str = "";
            irLedOn();
            onTime0 = measureSig(0);
            onTime1 = measureSig(1);
            onTime2 = measureSig(2);
            irLedOff();

            sprintf(str,"Channels: %d\t%d\t%d\n\r", onTime0,onTime1,onTime2 );
            writeSerialString(str);
        }
        return(0);
}
```

Schematic



Parts Customization

J1 Stuffing Options (also see Appendix B)

Description	X1 Part – 4-pin	X3 Part – 6-pin
Vertical *RobiCon*	*Molex 53014-0410*	*Molex 53014-0610*
Right-Angle *RobiCon*	*Molex 53015-0410*	*Molex 53015-0610*
Straight Pin Header	*Molex 22-28-4045*	*Molex 22-28-4065*

SMT Resistors

These resistors are 0603 (0.06" x 0.03") type surface mount. You will need a fine point soldering iron and some skill to change these parts. However, there are many tutorials on the web describing techniques for unsoldering and soldering these small components. *Circuit Monkey* is happy to change the values to your specifications when you order. If you already have a board, we'll do it for free if you pay the return postage (usually first class in the USA). Contact us first to arrange this.

Appendix A: References and Links

Sensor

Fairchild Semiconductor (FairchildSemi.com)

QRE-1113 Reflective Object Sensor

http://www.fairchildsemi.com/pf/QR/QRE1113.GR.html

Links

Circuit Monkey

http://www.circuitmonkey.com

RobiCon.org

A proposed open standard for Robotics Interconnect.

http://www.robicon.org

Atmel

Manufacturers of the *AVR/ATmega* microcontroller chips.

http://atmel.com/products/AVR/

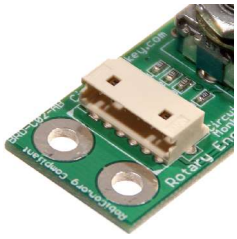
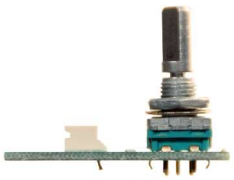
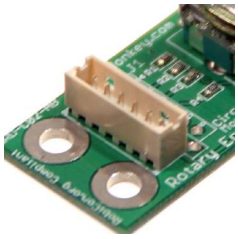
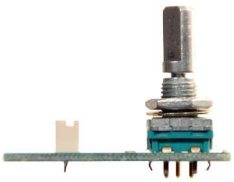
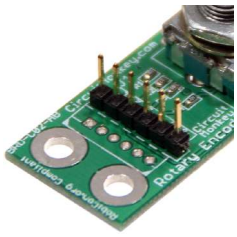
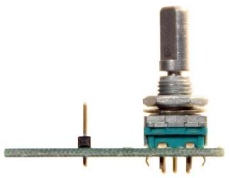
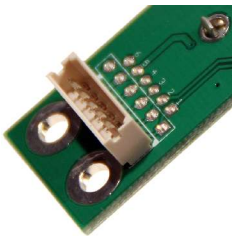
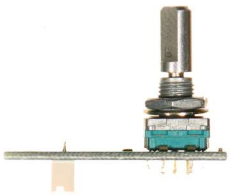
AVRFreaks.net

AVR Freaks -- ROS module Discussion

Appendix B: Alternate Configurations

J1 Connector Types and Orientation

The PCB footprint for J1 features a combination of three connector row holes for various connector options show in this table below. Please note the position of pin 1 (the square-shaped pads) regardless of configuration. These photos show our *Rotary Encoder* product *(PN:500-0026-01)* but the connector options are the same.

1. Right-Angle RobiCon – Top Mount – 2mm		
2. Vertical RobiCon – Top Mount - 2mm		
3. Pin Header – Top Mount – 2.54mm (0.1”) Right-angle pin header also possible.		
4. Vertical RobiCon – Bottom Mount – 2mm		
5. Pin Header – Bottom Mount – 2.54mm (0.1”)