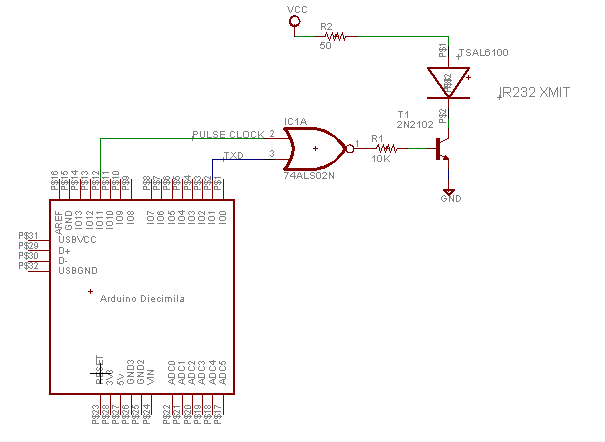
http://forum.arduino.cc/index.php?topic=10555.0

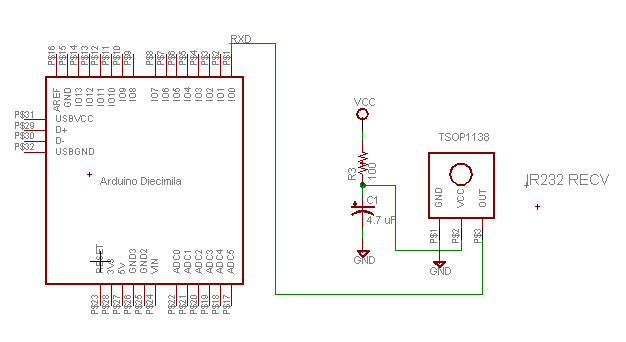
[**Serial comm using IR**](http://forum.arduino.cc/index.php?topic=10555.msg82828#msg82828)

[**May 21, 2008, 02:02 pm**](http://forum.arduino.cc/index.php?topic=10555.msg82828#msg82828)Last Edit: May 21, 2008, 02:03 pm by jmknapp Reason: 1

Here's a method to allow two Arduinos to communicate serially over an IR link, using a few dollars’ worth of parts at the transmit and receive ends. Currenlty this solution is restricted to one-way communication, with transmit hardware at one Arduino and receive hardware at the other. From the software point of view, the arrangement operates transparently as normal Serial communication, with data transmitted and received over TXD and RXD (pins 0 and 1).

Schematics:





The transmit side uses a high-power IR LED from Vishay (TSAL6100 <http://www.rentron.com/Files/tsal6100.pdf>) which allows up to 200ma peak forward current. The circuit above drives the LED at about 70ma, set by the 50-ohm resistor. The IR LED is pulsed by NORing TXD with a pulse clock from pin 11 (OC2A), which provides the 38KHz carrier used by the receiver.

The receiver side uses a 38KHz IR receiver (Vishay TSOP1138 <http://www.rentron.com/Files/TSOP11xx.pdf>) which drives the RXD pin. This device is specced at up to 4000 bits/sec, so a baud rate of 2400 is supported.

The only software required is on the transmit side, to set up the 38KHz square wave on pin 11. If an external oscillator were to be used, this would be an all-hardware solution, with the IR link acting totally transparently as a normal serial link.

I've tested this in a few different circumstances. I can get reliable communication at a distance of 50 feet in the shade, somewhat less in full sunlight outdoors.

Here's the code for the transmit side, which includes the pulse clock setup. The main loop in this case merely writes out "Hello, world!" every 5 seconds:

// defines for setting and clearing register bits

#ifndef cbi

#define cbi(sfr, bit) (\_SFR\_BYTE(sfr) &= ~\_BV(bit))

#endif

#ifndef sbi

#define sbi(sfr, bit) (\_SFR\_BYTE(sfr) |= \_BV(bit))

#endif

#define SYSCLOCK 16000000 // main system clock (Hz)

#define PULSECLOCK 38000 // Hz

#define IROUT 11

uint8\_t timer2top(unsigned long freq);

void setup() {

Serial.begin(2400);

cbi(TCCR2A,COM2A1); // connect OC2A (COM2A0 = 1)

sbi(TCCR2A,COM2A0);

cbi(TCCR2B,WGM22); // CTC mode for TIMER2

sbi(TCCR2A,WGM21);

cbi(TCCR2A,WGM20);

TCNT2 = 0;

cbi(ASSR,AS2); // use system clock for timer 2

OCR2A = 255; // set TOP to 255 for now

cbi(TCCR2B,CS22); // TIMER2 prescale = 1

cbi(TCCR2B,CS21);

sbi(TCCR2B,CS20);

cbi(TCCR2B,FOC2A); // clear forced output compare bits

cbi(TCCR2B,FOC2B);

pinMode(IROUT, OUTPUT); // set OC2A to OUPUT

OCR2A = timer2top(PULSECLOCK);

sei();

}

// main loop

void loop() {

Serial.println("Hello, world!");

delay(5000);

}

// return TIMER2 TOP value per given desired frequency (Hz)

uint8\_t timer2top(unsigned long freq) {

return((byte)((unsigned long)SYSCLOCK/2/freq) - 1);

}

On the receive side, no special software is needed--just use the usual Serial.available() and Serial.read() calls. This code receives characters and prints them:

void setup() {

Serial.begin(2400);

}

void loop() {

int n, i, ch;

n = Serial.available();

if (n > 0) {

i = n;

while (i--) {

ch = Serial.read();

Serial.print((char)ch);

}

}

delay(1000);

}