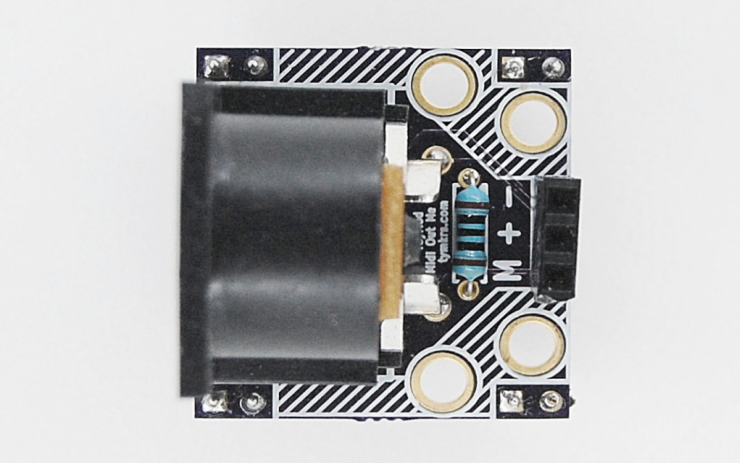
The Toymakers @ tymkrs.com
Questions? Please contact us:
feedback@tymkrs.com

DATASHEET



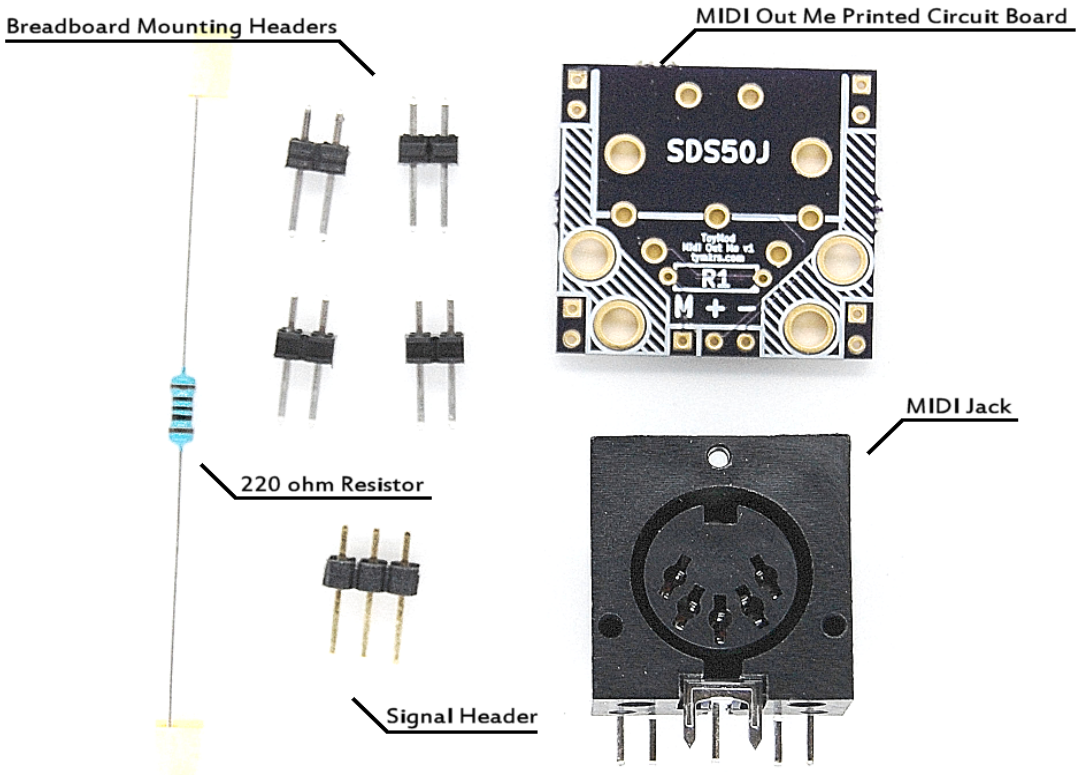
MIDI Out Me

Microcontroller to MIDI Kit

The MIDI Out Me kit allows you to command and control MIDI devices over a standard MIDI cable!

- Kit Type: Through-hole soldering
- Assembly instructions: In datasheet
- Function: Microcontroller to MIDI Device kit
- Uses 3 pins on the MCU to send outgoing MIDI signals
- Designed with MIDI specifications

KIT CONTENTS



Contents of the Midi Out Me Kit:

- Midi Out Me printed circuit board (25.5 x 24.23 x 1.60mm)
- 4 – 1x2 male headers
- 1 – 1x3 female header
- Electrical Components

Electrical Components:

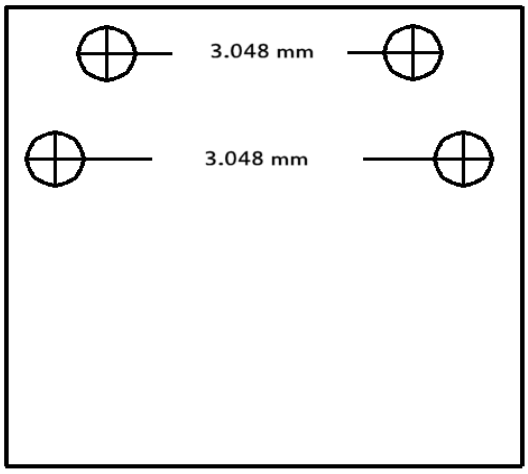
Reference	Quantity	Type	Value
R1	1	Resistor, 1/4W	220 ohm
SDS50J	1	Female MIDI Jack	---

Recommended Operating Conditions

Parameter	Ratings	Unit
Supply Voltage	4.5 – 5.5	V
Operating Temperature	-40 to +85	ºC

The MIDI Out Me circuit is built to MIDI specifications.

Mounting Holes:



Tools and material required for assembly (not included with the kit):

- Soldering iron
- Solder
- Wire clippers

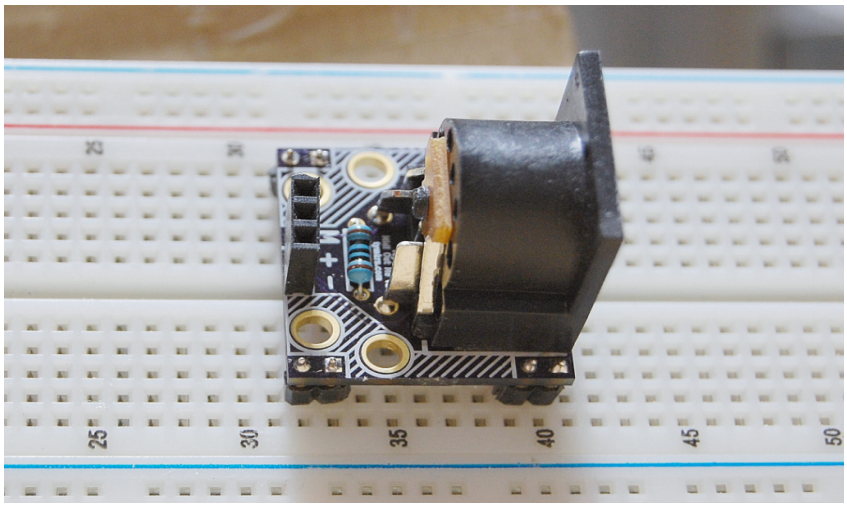
User provided items required for intended function:

- MIDI Cable
- MIDI device to control

Additional physical/electrical specifications:

- Printed Circuit Board size: 1.00 x 0.95 x 0.063" (25.5 x 24.23 x 1.60mm)
- PCB thickness: 0.063" (1.60mm), not including any components
- PCB thickness: 0.945" (24mm), max height with MIDI jack
- Mounting holes: 4 holes provided. See drawings for locations and size.
- Breadboard headers are not connected to the circuit electrically – they are for stability only.

Additional Pictures:



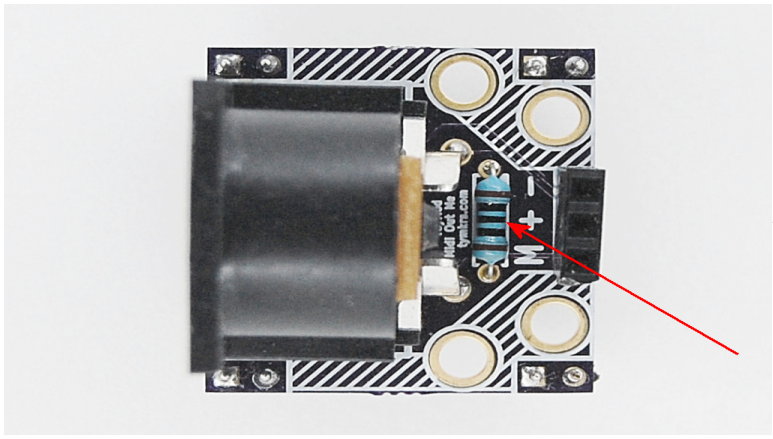
Assembled PCB
on breadboard

Assembly Instructions

Build Notes:

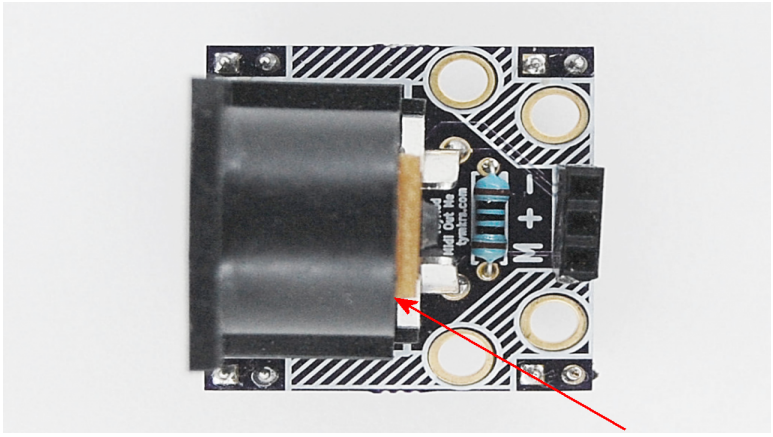
- **Method of use:** The connections run from the microcontroller to the Midi Out Me. The Midi Out Me is connected to power, ground, and a serial data pin on Propeller (or other MCU). MIDI jack is then connected to the MIDI device of choice through a MIDI Cable.

Step 1: Put in the components



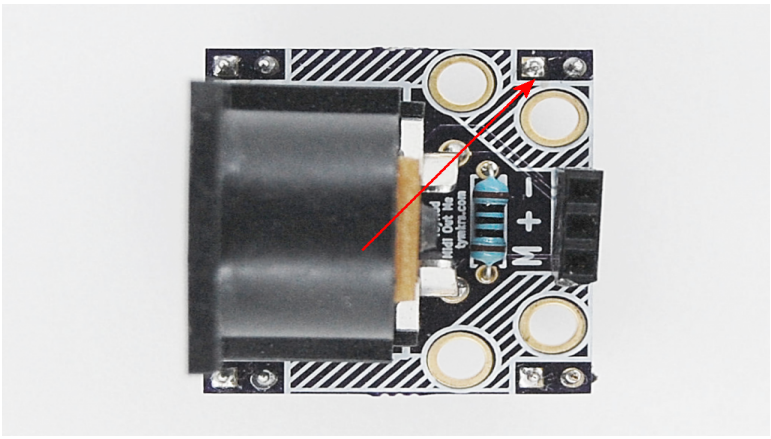
R1: 220 ohm resistor

You can bend the leads before putting it in the PCB. Polarity does not matter.



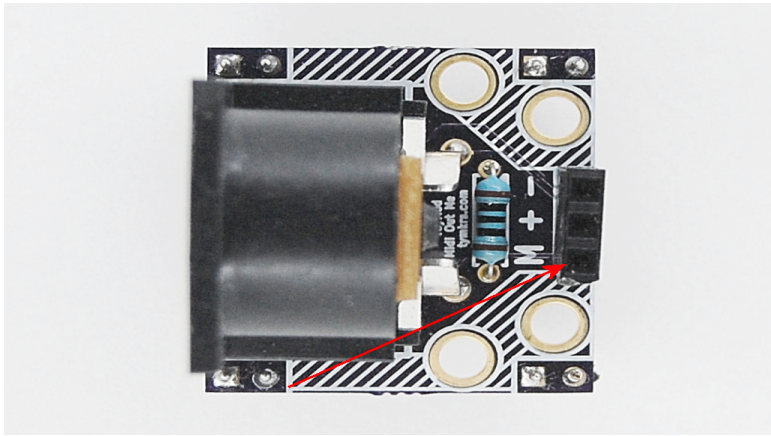
MIDI Jack

There's only one way to put the jack in – it will face outwards so you can plug the MIDI cable in!



Breadboard Headers

These are not electrically connected to the circuit. They are also breadboard friendly!



Microcontroller Header

This is a 1x3 female header that allows for the controlling microcontroller to interface with the MIDI devices it is controlling.

Step 2: Clip the extra leads!

I use 60/40 0.38mm gauge solder for these pads. But also have 1.3mm gauge solder for the larger solder pads. Using nibbers or nail clippers, trim the extra leads off of the electrical components!

Example Code

The following code is thanks to a fellow friend and maker: @chasxmd at http://iradan.com

Can be found: http://pastebin.com/i8v8K3kB This code was written for a 16-bit dsPIC. It is just a bit of sample code which sends the same command over and over. It's a Control Change Channel 1, Controller 17, Value 104.

```
/*
 * File:   main.c
 * Author: Charles Ihler @chaxmd
 * http://iradan.com
 *
 * Device: dsPIC33FJ16GS402
 *
 * Project: Test MIDI Out
 *
 * Blink an LED RB15 for confirmation of OSC speed.
 *
 * Created on March 7, 2015, 8:39 PM
 */

 #define FFRC 7372800ULL         // FRC oscillator frequency, Hz
 #define FOSC 80000000ULL        // Desired system clock frequency, Hz
 #define FCY     (FOSC/2)
 #define BAUDRATE 31250
 #define BRGVAL ((FCY/BAUDRATE)/16)-1

#include <p33Fxxxx.h>
#include <stdio.h>
#include <stdlib.h>
#include <libpic30.h>
#include <pps.h>

  _FOSCSEL(FNOSC_FRCPLL)              //set clock for internal OSC with PLL
  _FOSC(OSCIOFNC_OFF & POSCMD_NONE)   //no clock output, external OSC disabled
  _FWDT(FWDTEN_OFF)                   //disable the watchdog timer
  _FICD(JTAGEN_OFF & ICS_PGD1);       //disable JTAG, enable debugging on PGx1 pins
/*
 *
 */

void init_UART(void) {

    U1MODEbits.STSEL = 0;   // 1-Stop bit
    U1MODEbits.PDSEL = 0;   // No Parity, 8-Data bits
    U1MODEbits.ABAUD = 0;   // Auto-Baud disabled
    U1MODEbits.BRGH = 0;    // Standard-Speed mode
    U1BRG = BRGVAL;         // Baud Rate setting
//    U1STAbits.UTXISEL0 = 0; // Interrupt after one TX character is transmitted
//    U1STAbits.UTXISEL1 = 0;
//    IEC0bits.U1TXIE = 1; // Enable UART TX interrupt
    U1MODEbits.UARTEN = 1; // Enable UART
    U1STAbits.UTXEN = 1; // Enable UART TX


//for interrupt use..
//void __attribute__((__interrupt__)) _U1TXInterrupt(void)
//{
//IFS0bits.U1TXIF = 0; // Clear TX Interrupt flag
//U1TXREG = 'a'; // Transmit one character
//}


}

void blink(void){


    PORTBbits.RB15 = 1;         //on
    __delay_ms(100);
    PORTBbits.RB15 = 0;         //off
    __delay_ms(100);
}

void init_PPS (void) {
```

```
    //Assign PPS Ports (UART)
    PPSUnLock;
    iPPSInput(IN_FN_PPS_U1RX, IN_PIN_PPS_RP8);
    iPPSOutput(OUT_PIN_PPS_RP9, OUT_FN_PPS_U1TX);
    PPSLock;

}


int main(int argc, char** argv) {


    // setup internal clock for 80MHz/40MIPS
    // 7.37/2=3.685*43=158.455/2=79.2275
    CLKDIVbits.PLLPRE=0;        // PLLPRE (N2) 0=/2
    PLLFBD=41;                  // pll multiplier (M) = +2
    CLKDIVbits.PLLPOST=0;       // PLLPOST (N1) 0=/2
    while(!OSCCONbits.LOCK);    // wait for PLL ready

    ADPCFG = 0xFFFF;            //kill all analog Pins, digital only.
    ODCBbits.ODCB15 = 0;        //normal output
    ODCBbits.ODCB12 = 0;
    TRISBbits.TRISB15 = 0;      //output
    TRISBbits.TRISB12 = 0;      //output

    init_PPS();
    init_UART();

    __delay_ms(200);

    U1TXREG = 'a'; // Transmit one character

    while(1) {
        blink();
        //Send a Test MIDI Output
        U1TXREG = 0xB1; //Control Change Channel 1
        U1TXREG = 0x11; //Controller 0x11 (17)
        U1TXREG = 0x68; //Value 0x68 (104)
    }



    return (EXIT_SUCCESS);
}
```