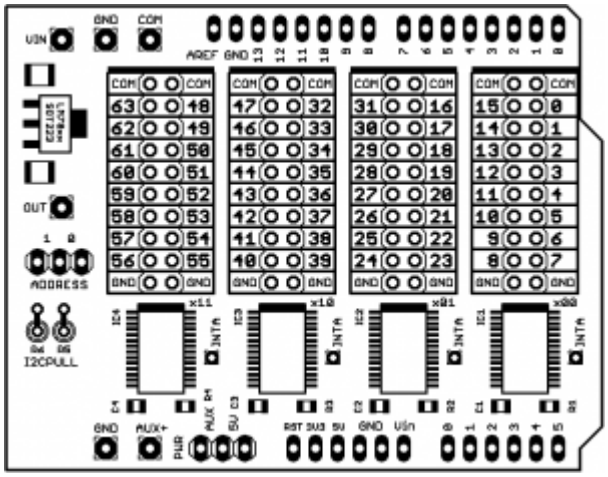
Centipede Shield

The Centipede Shield [http://macetech.com/store/index.php?main_page=product_info&cPath=4&products_id=23] is an add-on PCB for standard layout (Uno, Duemilanove, Diecimila) Arduino microcontroller boards. It uses the Wire I2C interface on analog pins 4 and 5 to provide 64 general purpose I/O pins. Any pin can be configured for input or output. The shield uses four Microchip MCP23017 [http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en023499] 16-pin digital I/O expander chips.

Features



Type:	Arduino Shield
Function:	Add 64 general purpose I/O pins
Works with:	Centipede Breakout [http://macetech.com/store/index.php?main_page=product_info&cPath=9&products_id=26], Centipede Breadboard Breakout [http://macetech.com/store/index.php?main_page=product_info&products_id=27]
Fits:	Arduino Diecimila [http://arduino.cc/en/Main/ArduinoBoardDiecimila], Duemilanove [http://arduino.cc/en/Main/ArduinoBoardDuemilanove], Seeeduino [http://www.seeedstudio.com/depot/seeeduino-v212-fully-assembled-arduino-compatible-p-389.html], etc.
Power Supply:	5V (from Arduino) and/or external power source (5V or use optional regulator)
Pin	Analog 4, 5 (for Wire/I2C; 5V, GND

Usage:	
Inputs:	Up to 64 digital inputs, COM, VIN, AUX+
Outputs:	Up to 64 digital outputs, 4 pin change interrupts
Other features:	Optional passthrough female headers, I2C address select

I/O Ports

Each I/O port corresponds to one MCP23017 IC. The 20-pin 0.1“ headers allow access to 16 I/O, two **GND** pins, and two **COM** pins. On the Centipede Shield, the pins are numbered sequentially from **0** to **63**. The pins on each port correspond to pins **A0-A**7 and **B0-B**7 for each chip as listed in the MCP23017 datasheet [http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf]. The **COM** pad is connected to all the **COM** pins on the I/O ports. This allows more ground wiring or a power source to run in the same cable as the I/O signals if needed.

Power Connections

By default, the power for all chips on the Centipede Shield is drawn from the Arduino 5V bus. This is selected by the **PWR** jumper, shorting the **5V** position. If a large number of loads will be powered by the Centipede Shield output pins, such as a large number of LEDs, then it may be necessary to use an external power source to prevent damage to the Arduino 5V regulator or the computer's USB port.

PWR Jumper

If the **PWR** jumper is shorting the **5V** position, then the Centipede Shield will be drawing its power for operation from the Arduino 5V bus. If the **PWR** jumper is shorting the **AUX** position, then an external power source can be soldered to the **AUX+** pad. Since the Arduino uses 5V logic, a 5V external source is recommended, but it may be possible to use a 3.3V source.

ADDRESS Jumper

The MCP23017 I2C interface can have up to 8 unique bus addresses. The Centipede Shield uses **X00** to **X11**, where **X** is the value of the **ADDRESS** jumper. This allows the Centipede Shield address space to be shifted by four places, either to prevent interference on the I2C bus with another device, or to allow two Centipede Shields to coexist.

I2C Interface

The Centipede Shield uses the official Arduino designation of I2C pins, namely the alternate functions of Analog **4** and **5** on the original Arduino pin configuration. While the Centipede Shield is in use, the analog input functions of Analog **4** and **5** cannot be used. The Centipede Shield can be accessed directly using the Wire library, or using the custom functions in the Centipede Library [http://macetech.com/Centipede.zip].

Optional Regulator

An area for an optional SOT223 LM78Mxx regulator and 1210 SMD capacitors is available if onboard power regulation from an external source is needed. Power can be applied to the regulator on the **GND** and **VIN** pads, and regulated output is available on the **OUT** pad. If desired, the **OUT** pad can be wired to the **AUX+** pad, and the **PWR** jumper set to short the **AUX** position to power the Centipede Shield from the optional regulator. This solution is recommended if a large number of high current outputs is required. For example, 25 LEDs at 20mA each would be 500mA additional load on the Arduino regulator, which likely would fail or cause computer problems if powered from a USB port.

INTA Pads

The **INTA** pads are connected to the MCP23017 interrupt A output pads. With the correct configuration of MCP23017 registers, the interrupts can fire when pin changes from a programmable state are detected. This could be very useful for detecting changes quickly without having to poll all the I/O over I2C frequently. At this time, the Centipede Library [http://macetech.com/Centipede.zip] does not yet provide an API for accessing the interrupt functions, but that feature will soon be available.

Code Example

The Centipede Library has been updated to work with Arduino 1.0

The following code example illustrates using the Centipede Shield with the Centipede Library (download here) [http://macetech.com/Centipede.zip]. The library attempts to encapsulate the Centipede I/O access functions in a way that is familiar to Arduino users. the library supports using two Centipede Shields automatically; pin **64** is equivalent to pin **0** on the Centipede Shield with the **ADDRESS** jumper set to **1**.

- .digitalWrite([0...127], [LOW...HIGH]) - Acts like normal digitalWrite.
- .digitalRead([0...127]) - Acts like normal digitalRead.
- .pinMode([0...127], [INPUT...OUTPUT]) - Acts like normal pinMode.
- .pinPullup([0...127], [LOW...HIGH]) - Activates internal 100K pullups on inputs when HIGH.
- .portWrite([0...7], [0...65535]) - Writes 16-bit value to one chip. Useful for writing 16 outputs at the same time.
- .portRead([0...7]) - Reads 16-bit value from one chip. Useful for reading 16 inputs at the same time.
- .portMode([0...7], [0...65535]) - Write I/O mask to one chip. In binary, a 0 means output and a 1 means input. Easier to use than a long list of pinMode() commands.
- .portPullup([0...7], [0...65535]) - Set pullup mask on one chip. In binary, a 0 means no pullup and a 1 means pullup is active.
- .init() - Sets all registers to initial values.

Extract the Centipede Library files into a Centipede directory in your Arduino Libraries directory. It will be available the next time you start the Arduino environment.

```
// Example code for Centipede Library
// Works with Centipede Shield or MCP23017 on Arduino I2C port

#include <Wire.h>
#include <Centipede.h>


/* Available commands
  .digitalWrite([0...127], [LOW...HIGH]) - Acts like normal digitalWrite
  .digitalRead([0...127]) - Acts like normal digitalRead
  .pinMode([0...127], [INPUT...OUTPUT]) - Acts like normal pinMode
  .portWrite([0...7], [0...65535]) - Writes 16-bit value to one port (chip)
  .portRead([0...7]) - Reads 16-bit value from one port (chip)
  .portMode([0...7], [0...65535]) - Write I/O mask to one port (chip)
  .pinPullup([0...127], [LOW...HIGH]) - Sets pullup on input pin
```

```
  .portPullup([0...7], [0...65535]) - Sets pullups on one port (chip)
  .portInterrupts([0...7],[0...65535],[0...65535],[0...65535]) - Configure interrupts
                 [device number],[use interrupt on pin],[default value],[interrupt when != default]
  .portCaptureRead(0...7) - Reads 16-bit value registers as they were when interrupt occurred
  .init() - Sets all registers to initial values

  Examples
  CS.init();
  CS.pinMode(0,OUTPUT);
  CS.digitalWrite(0, HIGH);
  int recpin = CS.digitalRead(0);
  CS.portMode(0, 0b0111111001111110); // 0 = output, 1 = input
  CS.portWrite(0, 0b1000000110000001); // 0 = LOW, 1 = HIGH
  int recport = CS.portRead(0);
  CS.pinPullup(1,HIGH);
  CS.portPullup(0, 0b0111111001111110); // 0 = no pullup, 1 = pullup
  CS.portInterrupts(0,0b0000000000000001,0b0000000000000000,0b0000000000000001);
*/

Centipede CS; // create Centipede object


void setup()
{
  Wire.begin(); // start I2C

  CS.initialize(); // set all registers to default

  CS.portMode(0, 0b0000000000000000); // set all pins on chip 0 to output (0 to 15)
  //CS.portMode(0, 0b0000000000000000); // set all pins on chip 1 to output (16 to 31)

  //TWBR = 12; // uncomment for 400KHz I2C (on 16MHz Arduinos)

}


void loop()
{
  for (int i = 0; i < 16; i++) {
    CS.digitalWrite(i, HIGH);
    delay(10);
  }

  for (int i = 0; i < 16; i++) {
    CS.digitalWrite(i, LOW);
    delay(10);
  }
}
```