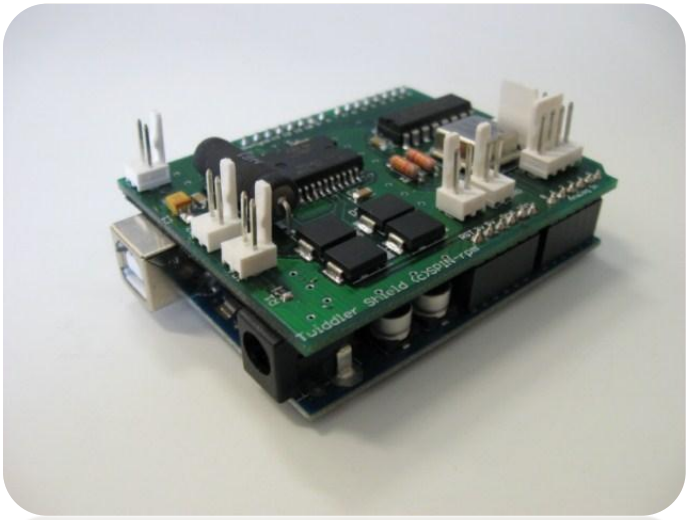
# Twiddlerino.
# Hardware Documentation



**SPIN Research Group**
**Department of Computer Science**
**The University of British Columbia**
**December 2010**

# Contents

## WARNING!

**The motor in the Haptic Display module can generate very strong forces. If an unstable condition arises, the best way to stop the force rendering in the Haptic Display module is to use the power switch in the Interface Box to cut the power to the motor. Trying to stop the motor by holding the knob can be a cause of injuries and/or can cause serious damage to the hardware in the Interface Box.**

# 1. Twiddlerino: the reasons behind a name.

The Twiddler is a single degree of freedom rotary haptic device designed as a low cost development interface platform for use in a classroom environment. In the original design, an Interface Box collects the position information sent by an optical encoder attached to a motor in the Haptic Display module and transmits it using the parallel port to a host computer. An algorithm running in the host computer calculates a force command as a function of the motor position and sends it back to the Interface Box where it is converted to a motor driving signal fed to the Haptic Module.

As the parallel port is no longer a standard feature in modern computers, the decision was made to redesign the system to use instead an USB connection. Overall, the new design is similar to the original Twiddler. The general approach was to maintain the haptic loop running in a computer and have an external Interface Box as in the original Twiddler dealing with feeding the motor with the appropriate voltage and reading the position sent by the encoder. The USB communication loop will then transmit position information to the computer and motor speed/force to the interface box.

The new version of the Twiddler was nicknamed Twiddlerino, since the hardware design of the new Interface Box is based on the Arduino development platform [1]. Arduino is an open-source electronic prototyping platform mainly designed to facilitate the use of data acquisition and actuation technologies to artists, designers and hobbyists. This approach is by no means a limitation for this platform to be used in very demanding applications. An AVR microcontroller is at the heart of all boards in the Arduino family. Software development for this platform is simplified by the use of the Arduino programming language (based on Wiring [5]) and the Arduino Development Environment (based on Processing [4]).

The main purpose of the Arduino boards is to provide an interface bus where additional electronic modules can be connected to perform any desired function. These additional electronic modules (known as "shields" in the Arduino community) are designed as stackable boards that could be attached to the interface bus of the main Arduino board to form compact hardware modules. Developers have at their disposal a wide assortment of pre-designed boards to fulfill an impressive variety of functions ranging from simple input and output to LCD displays and wireless communications. Libraries and Software Development Kits are also available to complement many of the more complex shields.

For the new version of the Twiddler we chose the Arduino Duemilanove [2]. This is the basic Arduino board containing an AVR 8-bit microcontroller (ATMega328p) and a full USB interface based on the FTDI FT232RL chip. The Duemilanove interface bus provides up to 14 digital inputs/outputs lines (D0-D13) and up to 6 analog input lines (AI0-AI5). Up to six of the digital IO lines can be configured to perform pulse width modulation (PWM) and two of them are used in serial communication duties if the platform is to be connected via USB to a host computer[1].

Using a microcontroller based platform simplifies a lot the design of the Interface Module. For instance, the generation of the PWM signal that controls the motor is made by software instead of by hardware components as in the original Twiddler. Also, some of the low level data acquisition tasks running in the

[1] At the time of writing, a new version of the basic Arduino board was released, substituting the *Duemilanove* model. The *Arduino Uno* board [3] is 100% compatible in functionality and 100% pin out compatible with the *Duemilanove*, meaning they can safely be exchanged if the need arises. The only functional change in the new model was the use of a different USB-to-serial (the FTDI FT232RL was substituted by an ATMega8U2 chip), but this do not have any effect on the shield design. It is expected that future versions of this board will be backward compatible with earlier outdated versions no longer in production as it has been the case so far.

host computer in the previous versions are now done by the firmware running in the Duemilanove board, increasing the efficiency of the overall process.

Still some additional circuitry was needed to address the needs of the project. Due to the specifics of this project, none of the commercially available shields fulfilled the requirements to drive the Haptic Display. A custom shield was designed in order to drive the motor and to collect and decode the position of the knob in the Haptic Display. The full electronic diagram of the shield, the PCB design and the connection diagrams between the shield and the external features on the chassis as well as between the Interface box and the Haptic Module are presented in the Appendix A.

## 2. System Overview

The Twiddlerino inherited the Power Supply and the Haptic Display modules from its predecessor. Only the Interface Box needed to be redesigned in order to accommodate the new communication method. Besides the change in functionality, the new design approached some other issues present in the older version. For instance the ambiguity of the connection between the Haptic Display and the Interface Box was eliminated by using a single polarized connector instead of the two needed in earlier versions. From an aesthetic point of view the interface box also received a makeover. The components of the Twiddlerino are shown in Figure 1.

Figure 1. Components of the Twiddlerino. From left to right: Haptic Display module, Interface Box and Power module.

The back panel of the Interface Box holds a DB9 connector to connect to the Haptic Display module, accepting the data from the rotation encoder and sending away the motor drive signal. The front panel of the Interface Box holds a power switch, a power indicator, the external power input and the USB connector (Figure 2).

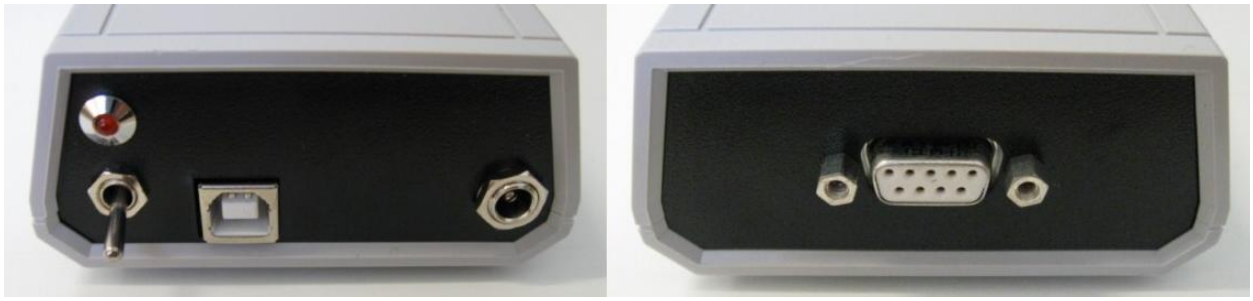


Figure 2. Interface Box front and back panels.

The Interface Box can be powered directly from the USB cable when it is plugged to a host computer. This feature comes in handy when downloading firmwares to the Duemilanove board. However, an external power supply is needed to feed the power necessary to run the motor on the Haptic Display. The power switch in the Interface Board only controls the use of the external power supply and does not have any effect on powering the Duemilanove board through the USB cable. The power indicator in the front panel will light dimly when the Interface Box is using power from the USB interface and will fully light when the external power supply is connected and the power switch is ON. The Duemilanove board will switch automatically to the external power supply whenever one is plugged in and the power switch is ON.

The following section will discuss the design of the shield. A more detailed explanation of the hardware related to the Arduino platform will not be included in this document since it is easily available from the platform website [1]

## 3. Shield design considerations

The shield contains the circuitry needed to perform two main functions: decoding the quadrature signal received from the encoder in the Haptic Display module and providing the power signal to drive the motor. The full electric diagram and the PCB design for the shield are shown in the Appendix A.

The quadrature decoder unit needs a total of 11 digital lines (8 inputs and 3 outputs) and the motor controller needs another 3 digital output lines, including one carrying a PWM signal. This adds up to a total of 14 digital lines needed for the shield to perform its functions. Since the lines D0 and D1 from the Duemilanove interface bus are used in the serial communication with the host computer, there are only 12 bidirectional digital lines available. However, as the analog input lines are not used, they can be configured as digital outputs to reach the required number of lines.

The Duemilanove interface lines used in the shield and their functions are listed in Table 1.

Table 1. Duemilanove Interface lines used in the shield.

| IO Line | Configured as | Function |
|---|---|---|
| D2 to D9 | Digital input | Quadrature Encoder Data Lines |
| AI0 | Digital Output | Quadrature Encoder Output Enable |
| AI1 | Digital Output | Quadrature Encoder High/Low Byte Selector |
| AI2 | Digital Output | Quadrature Encoder Reset |
| D10 | Digital Output (PWM) | Motor Power |
| D11 | Digital Output | Motor Direction |
| D12 | Digital Output | Motor Direction |

The unused lines DIO13, AI3, AI4 and AI5 are available in an additional connector as extra lines to be used in projects requiring additional inputs. This is an internal connector and is only accessible if the chassis cover is removed.

## Note.

**The standard firmware provided with the Twiddlerino does not enable any functionality related to the unused lines. It is up to the user to modify the software components in order to access these extra capabilities to fit any particular application and to substitute the standard rear panel with another carrying an additional external DB9 connector. Extra panels are available from the class instructor on demand.**

An extra internal connector is provided to supply 5v and up to 400mA to power a fan if the heat dissipated by the motor control unit becomes excessive. However, tests performed on the system rendered this situation very unlikely to happen, but the connector was left in the design as a cautionary measure.

All the internal connections of the Interface Box are shown in the connection diagram included in the Appendix A..

# 4. The quadrature decoder unit

The Interface Box receives a quadrature signal from the encoder in the haptic module. This signal consists of two square waves with 90 degrees phase difference providing information on both the amount of movement (in steps limited by the resolution of the encoder) and the direction of this movement, as described in Figure 3.

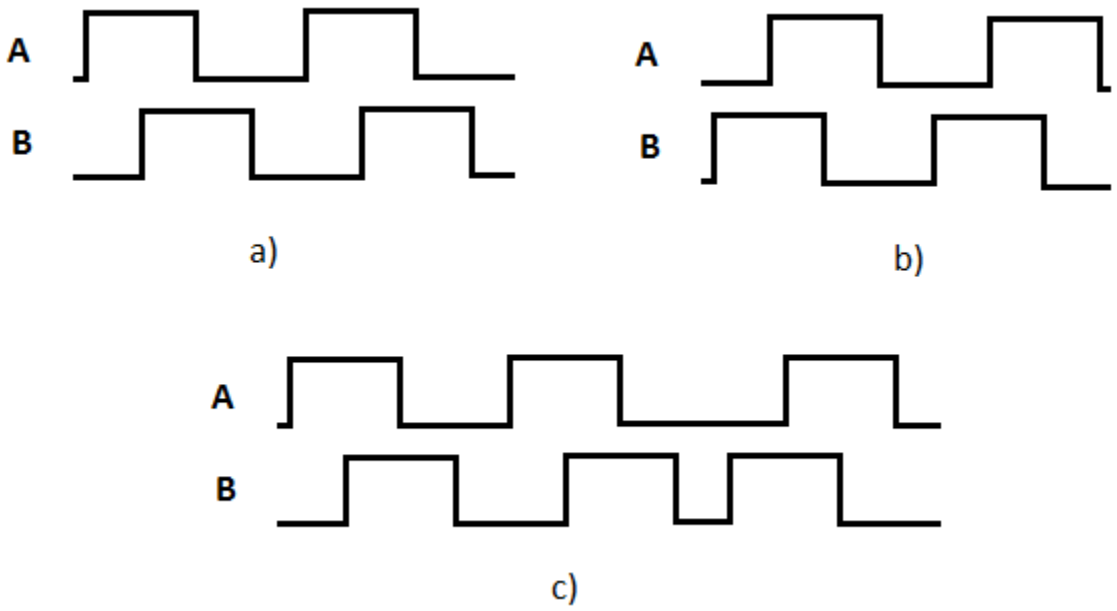



Figure 3. Quadrature signals as offered by encoders representing: a) axis rotating in one direction, b) axis rotating in the opposite direction, c) axis switching the direction of the rotation.

Figure 3 a) shows the signal A leading the signal B, representing that the axis rotated 8 steps. Steps are counted as the edges of both waveforms rise and fall. Figure 3 b) shows that the signal B is leading the signal A, indicating that the axis rotated 8 steps in the opposite direction as the one depicted in a). Figure 3 c) shows the signals resulting from a rotation of 8 steps in one direction followed by a rotation of 4 steps in the opposite direction. The role of a quadrature decoder circuit is to take both A and B signals and produce a number that increments as the shaft connected to the encoder rotates in one direction and decrements as it rotates in the opposite direction.

In the shield, the quadrature decoder unit is built around the HCTL-2017 integrated circuit (Figure 4). The 2017 contains the quadrature decoding logic, a binary up/down state counter and an 8-bit bus interface. This is a synchronous circuit, which means it needs a clock input to perform its functions. This clock signal is provided by a 32 MHz oscillator. The 2017 returns a 16-bit binary word corresponding to the value reported by the counter. However, as its data bus is only 8-bits wide some extra control lines are needed in order to properly read the correct position.

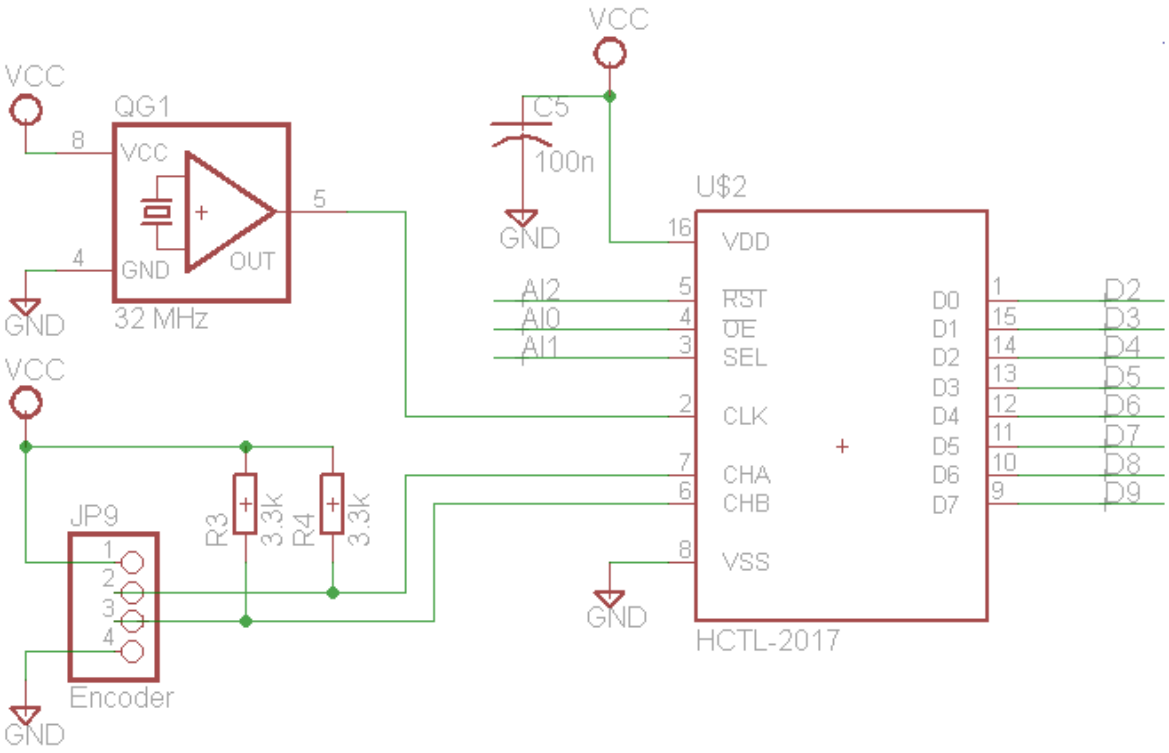



Figure 4. The quadrature decoder unit.

The position information is sent over the lines D2-D9 (the Data Bus) to the main Arduino board. The OE (Output Enable, AI 0) line controls when the decoder outputs are available at the Data Bus. A low voltage level ("0") at OE enables the outputs.

The SEL (AI 1) line selects which byte will be presented to the Data Bus. A "0" selects the High byte and a "1" (a high voltage level) selects the Low byte.

When active, both SEL and OE trigger an internal inhibit signal that doesn't allow the 2017's output data latch to be updated by new position values. This way the value presented to the Data Bus is stable during the time it takes to be read by the microprocessor in the Arduino board. The process of reading the word containing the count value can then be achieved by the following sequence of actions:

1. Send a "0" to OE (Enables the data output, inhibits data latch from being updated)
2. Send a "0" to SEL (Selects the High Byte)
3. Read the High Byte,
4. Send a "1" to SEL (Selects the Low Byte)
5. Read the Low Byte
6. Send a "1" to OE (Disables the data output, allows data latch to be updated)

As described earlier, the 2017's data output won't change while the OE signal is kept at a low level, but its internal counter keeps counting every position change and whenever OE goes to 1, the position data latch starts to be updated with the new values until the next time that OE goes low to request another reading cycle.

OE and SEL are synchronous signals, meaning that they will make an effect the next time the clock signal goes high after they are changed. This should not be a reason of concern since the clock signal used in the

circuit is considerable faster than the quadrature signals, so no position change will be lost. However, the Reset signal (AI 2) that is used to return the counter to a zero value is asynchronous. This means that whenever this line receives a "0", the internal counter will go to zero no matter what the other signals are.

## 5. The motor controller unit

The motor control circuit is built around an L298 integrated circuit: a dual full-bridge driver. Even when each one of the bridges contained in this IC is able to handle the current needed by the motor in the Haptic Module, both bridges were connected in parallel to minimize heat dissipation and to extend the life expectancy of the system. The motor control circuit is shown in Figure 5.

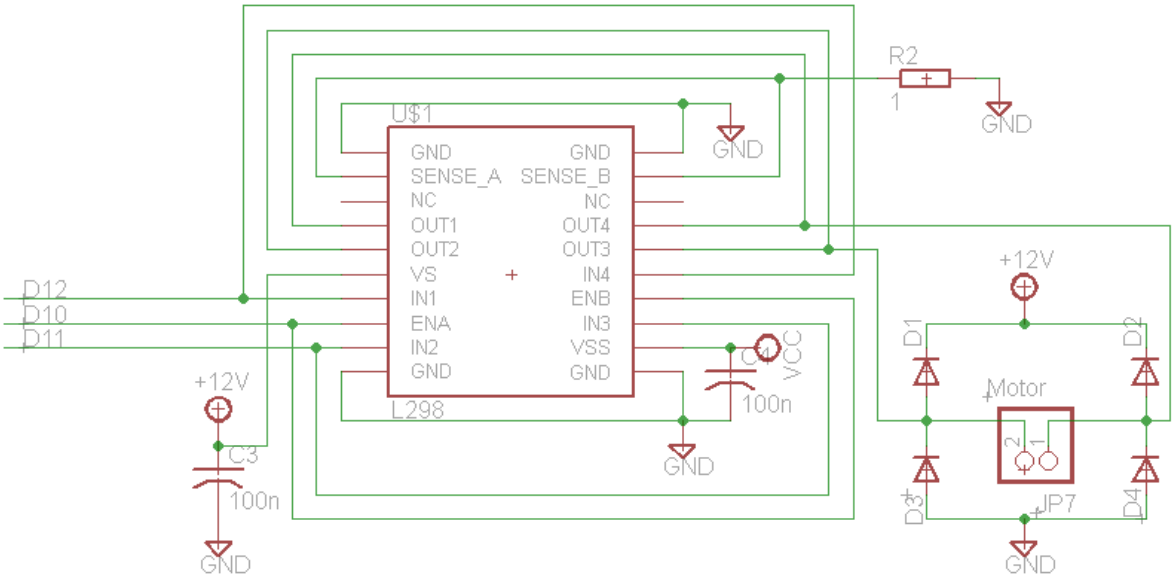



Figure 5. The motor control unit.

The operation of this unit is rather simple. The direction of the motor rotation is controlled by the lines D11 and D12. These lines are connected to the bridge inputs (D11 drives inputs 2 and 3 and D12 drives inputs 1 and 4). A high on any input will drive high the correspondent output. Therefore, to guarantee a current flow through the motor, the lines D11 and D12 should never be driven high at the same time. The bridge IC receives the power level to be delivered to the motor as a PWM signal sent over the D10 line. This line is connected to the bridges' "enable" inputs.

Pulse width modulation is a very valuable technique to control analog devices using digital lines. The motor used in the Haptic Display module is an analog device and requires different voltage levels in its inputs in order to move faster or slower. A voltage closer to 0 will move the motor slowly, meanwhile the motor will move at maximum speed when it is fed with the full value of the power supply. The digital lines can only transmit two voltage levels: fully off or fully on. Using a pulse-width-modulated signal is a way to digitally encode a specific level of an analog signal by means of modifying the duty cycle of a square wave signal as shown in figure 6.

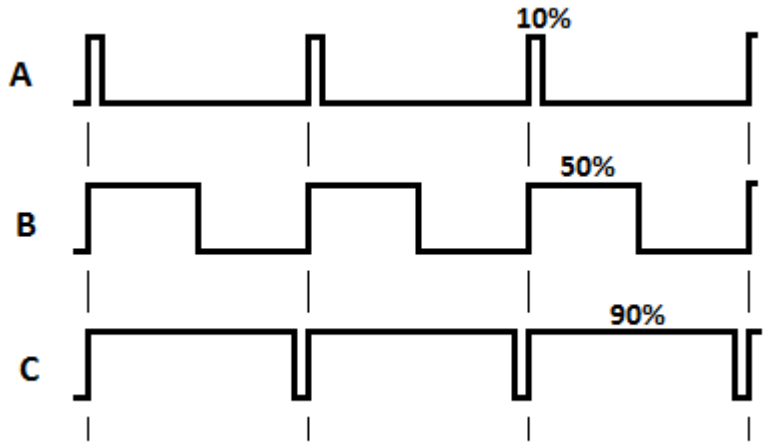



Figure 6. PWM signals of varying duty cycles

Signals in figure 6 are encoding different analog signals, even when they are still digital signals. Signal A is ON only for 10% of the period. If the power supply is 12V, it means that it is encoding a 1.2V analog signal. In the same manner, signal B encodes a 6V signal and signal C encodes a 10.8V. Power is supplied to the load only when the PWM signal is ON.

By using a PWM signal to drive the L298's "enable" inputs, the power supply to the motor is guaranteed to be provided only when the PWM signal is high. This way, as the PWM signal stays ON longer, the power transferred to the load will be larger, effectively acting as a larger analog signal.

The motor is connected to the center of the bridge formed by the diodes D1 to D4. This is the standard way to provide the necessary current protection needed when driving inductive loads.

The L298 has an internal protection that triggers when the current delivered to the motor exceeds its maximum operational level. As explained above, the L298 only controls the voltage presented to the motor. The current flowing through the circuit is offered by the power supply and depends on the demand placed by the motor itself. A situation where the current limit of the L298 is exceeded could occur if the motion of the knob in the Haptic Display module is opposed by the user for a large amount of time. For instance, when rendering a wall, the motor will oppose the movement of the knob past the position at which the wall is rendered. If the user forces the knob past this limit, the system will provide a larger signal to the motor in order to return the knob to a position "outside" the wall. The farther past the limit the user takes the knob, the higher will be the current demanded by the motor in order to counter the effect of the user's actions. If this situation is maintained for too long, the L298's internal protection will trigger, turning off the power supplied to the motor and thus stopping the force rendering.

We strongly recommend not forcing the motor past the limits established by the active rendering, or at least not to use excessive force for long periods of time to move the motor past an established limit. Similarly, the best way to handle a situation where the motor spins out of control is to use the switch in the Interface Box to cut the power to the motor. In these cases, trying to stop the motor by hand might be a cause of injuries.

## 6. References

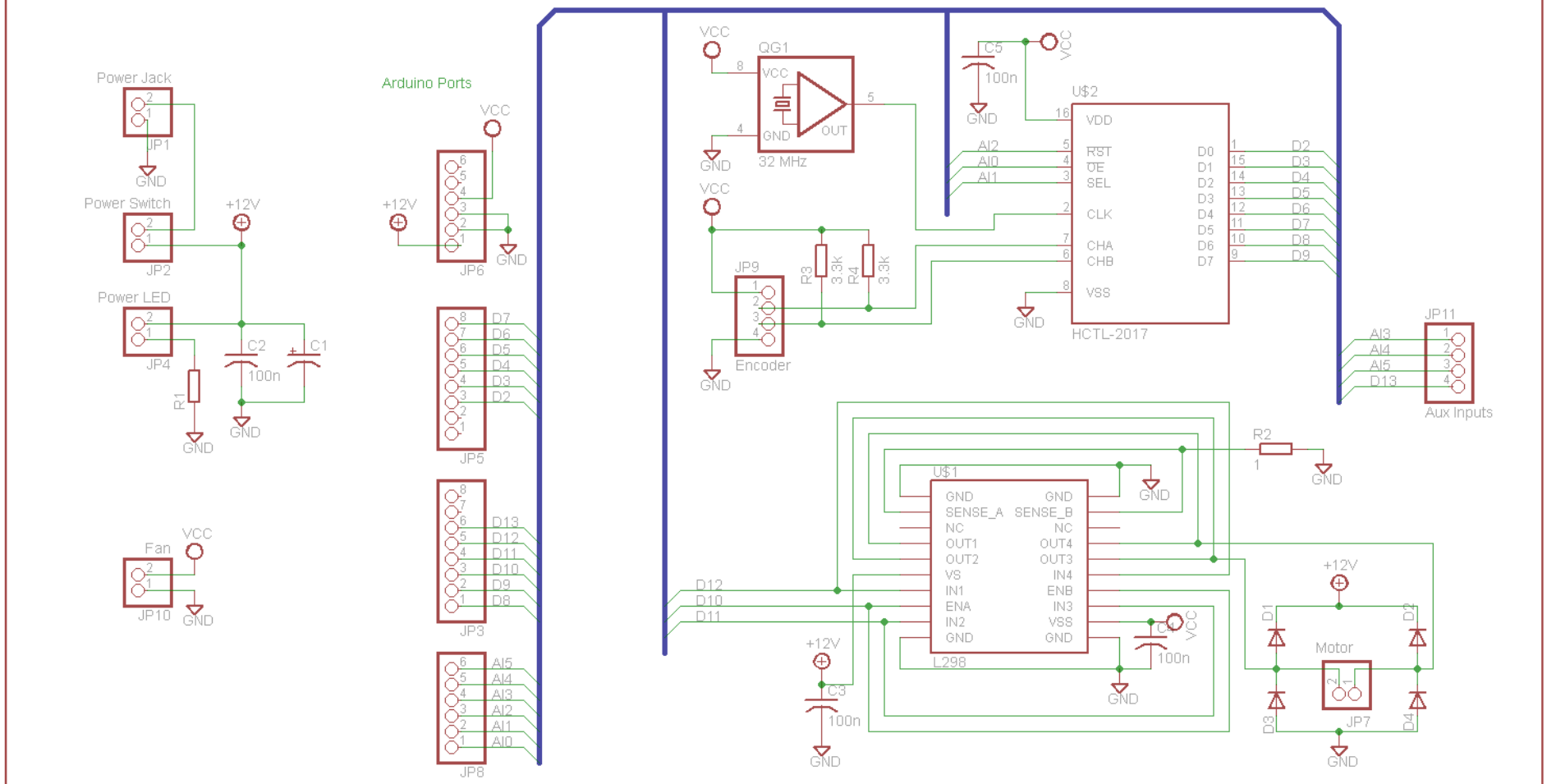

1. Arduino platform website (http://arduino.cc)
2. Arduino Duemilanove (http://arduino.cc/en/Main/ArduinoBoardDuemilanove)
3. Arduino Uno (http://arduino.cc/en/Main/ArduinoBoardUno)
4. Processing: Open source programming language and environment (http://www.processing.org)
5. Wiring: Open source programming environment and I/O board (http://wiring.org.co)

# Appendix A

Twiddlerino Shield. CAD files

## A1. Twiddlerino. Bill of Materials

| Description | Qtty | DigiKey Part Number | Part Number (PCB) |
|---|---|---|---|
| DIODE FAST RECOVERY 50V 3A SMC | 4 | RS3A-FDICT-ND | D1, D2, D3, D4 |
| IC DRIVER FULL DUAL 20-PWRSOIC | 1 | 497-3624-1-ND | U$1 |
| IC DECODER/COUNTER 16BIT 16-DIP | 1 | 516-1881-5-ND | U$2 |
| OSC 32.000 MHZ 5.0V 1/2 SZ | 1 | XC254-ND | QG1 |
| CAP .10UF 50V CERAMIC X7R 1206 | 4 | 311-1179-1-ND | C2, C3, C4, C5 |
| CAP TANT 22UF 25V 10% SMD | 1 | 718-1517-1-ND | C1 |
| RES 5.10K OHM 1/8W 1% 0805 SMD | 1 | 541-5.10KCCT-ND | R1 |
| RESISTOR SILICONE 1.0 OHM 5W | 1 | ALSR5F-1.0-ND | R2 |
| RES 3.3K OHM CARBON FILM 1/4W 5% | 2 | P3.3KBACT-ND | R3, R4 |
| CONN HEADER 36POS .100 VERT TIN | 1 | WM6436-ND | JP3, JP5, JP6, JP8 |
| CONN HEADER 2POS .100 VERT TIN | 5 | WM4111-ND | JP1, JP2, JP4, JP7, JP10 |
| CONN HEADER 4POS .100 VERT TIN | 2 | WM4113-ND | JP9, JP11 |




Arduino Ports

JP6_1: Vin
JP6_2: GND
JP6_3: GND
JP6_4: 5V
JP6_5: 3V
JP6_6: Reset

JP5_1: 0 / TX
JP5_2: 1 / RX
JP5_3: 2 / Ext IT ---------- Data0
JP5_4: 3 / Ext IT / PWM -- Data1
JP5_5: 4 ---------- Data2
JP5_6: 5 / PWM ---------- Data3
JP5_7: 6 / PWM ---------- Data4
JP5_8: 7 ---------- Data5

JP3_1: 8 ---------- Data6
JP3_2: 9 / PWM ----- Data7
JP3_3: 10 / PWM --- Motor Power
JP3_4: 11 / PWM --- Motor Dir
JP3_5: 12 ---------- Motor Dir
JP3_6: 13 ---------- Auxiliary I/O
JP3_7: GND
JP3_8: ARef

JP8_1: AI 0 -- (configd as DigOut) - $\overline{OE}$
JP8_2: AI 1 -- (configd as DigOut) - SEL
JP8_3: AI 2 -- (configd as DigOut) - $\overline{RST}$
JP8_4: AI 3 -- Auxiliary I/O
JP8_5: AI 4 -- Auxiliary I/O
JP8_6: AI 5 -- Auxiliary I/O

TITLE: Twiddler Shield (SMD - Single)V2

Document Number:

REV:

Date: 07/12/2010 10:02:13 AM

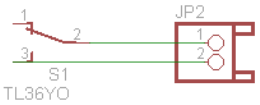
Sheet: 1/1

**A2. Twiddlerino. Shield Electric Diagram**

## Front Panel

Power Jack

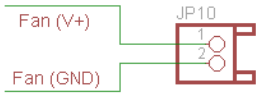

Power Switch




Power LED


## Internal connector

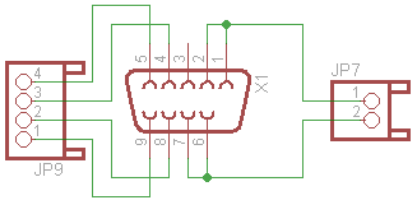
Fan Connector




Not used in the standard version
Provides power for a 5V fan
Current limited to 400mA

## Back Panel

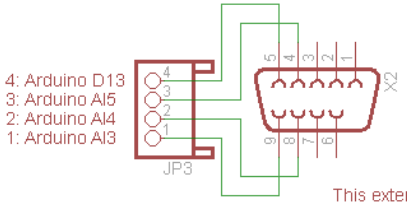
Internal wiring. Standard back panel DB9 socket.




Matching DB9 plug pinout.
Haptic Display Module connector.

1,2: Motor RED
6,7: Motor BLACK
4: Encoder YELLOW
5: Encoder BLACK
8: Encoder BLUE
9: Encoder RED
3: Unused

Internal wiring. Auxiliar back panel DB9 socket.



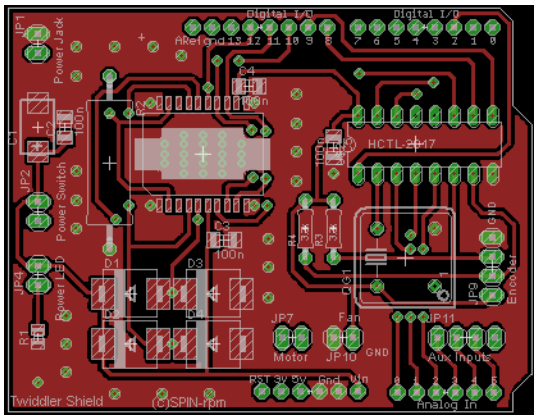

Matching DB9 plug pinout.

4: Arduino AI5
5: Arduino D13
8: Arduino AI4
9: Arduino AI3
1,2,3,6,7: Unused

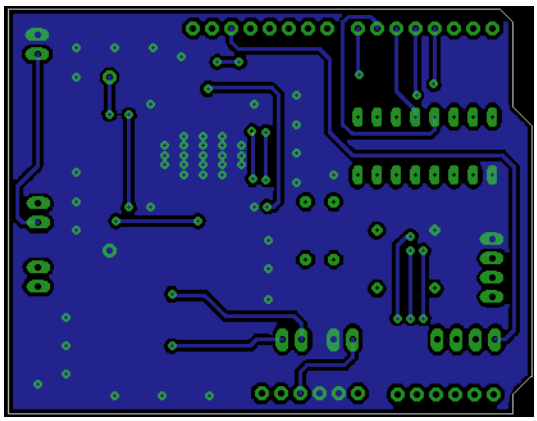
This external connector is not provided in the standard package
Ask the class instructor for a back panel containing this extra connector
if you need to use the functionality of the unused interface lines

| | |
|---|---|
| TITLE: Twiddler Shield (Connection Diagram) | |
| Document Number: | REV: |
| Date: 07/12/2010 10:03:16 AM | Sheet: 1/1 |

**A3. Twiddlerino. Interface Box Connection Diagram**




a)



b)

## A4. Twiddlerino. Printed Circuit Board Layout

a) Top copper layer and silk screen
b) Bottom copper layer