DDF Controller Board Documentation v2.00

Grant Elliott and Scott Torborg*
Dropout Design

(Dated: January, 2006)

1. INTRODUCTION

Note: There are now two revisions of the DDF hardware in use: the original DDF 1.17 and the updated DDF 2.05. Most orders placed in or after January 2006 will use the revised hardware 2.05. Please take note of which revision you are using. The number is printed on the right side of the board, above the Molex connector footprint. Periodically, notes relating to DDF117 or DDF205 will appear in bold. Please read these, as they are extremely important.

The DDF Controller Board was developed for the 1E Disco Floor installed at MIT's East Campus in January of 2005 and has since become Dropout Design's flagship product. The board offers USB control of 192 LEDs with 16 level intensity control as well as 64 binary switches. See `web.mit.edu/storborg/ddf/` for more information about the floor or `www.dropoutdesign.com` to purchase PCBs.

Documentation, schematics, and source code are made available under the Creative Commons license. You are free to use this information for nonprofit use and to produce derivative work provided, in both cases, that Grant Elliott and Scott Torborg be credited as the original creators. Derivative work must also be released under this license and be used for exclusively non-profit use. Any exceptions to these conditions must be approved by Grant Elliott and Scott Torborg, who may be contacted at ddf@mit.edu. For more information, please see the Creative Commons website.

While these files may not be used commercially, purchasers of boards are free to use them for any purpose, including commercial applications. Grant Elliott and Scott Torborg reserve all rights on board layout.

The DDF Controller Board and associated documentation and code are provided as is with no warranty. Dropout Design is not responsible for any damage caused to the board or other equipment through use or misuse of the board or documentation. When properly assembled, the board is known to function as claimed.

1.1. What You Need

In addition to a PCB, available from Dropout Design, you will need to obtain the parts listed in Appendix A. The total cost of parts, neglecting header, will be approximately $70 in unit quantity, in addition to the board. Alternatively, boards are now available in kit form. Additionally, you will need to purchase tricolor LEDs and cabling to connect them to the board, a 5V power supply capable of sourcing 4A (less if not all LEDs are populated), an AVR programmer (available from Atmel, or you can build your own with a parallel port), and a voltmeter. An oscilloscope may also be useful for debugging, but is not necessary.

2. HARDWARE

The DDF Controller Board measures 4.25" by 5". The bottom of the board contains the USB interface on the left and debugging test points on the right. The remainder of the board is divided into four quadrants, each of which maps to a row of the dance floor and consequently will be referred to simply as a row. Row one is located directly above the USB interface, with row two above it, and rows three and four to their right. Each row contains 16 cable connectors, each intended for a cell of three LEDs and a sensor. Cable connectors are numbered counter-clockwise beginning at the lower left of each quadrant.

2.1. Technical Description

The FTDI232BM located in the controller region provides a USB to serial interface and passes data into the USART of the Atmega8. The Atmega8 interprets this data and issues commands over the I2C bus to the 16 MAX7313 LED drivers. Each LED driver corresponds to one color (or the sensors) of one row and is assigned an address in hardware as given by Table I. The LED drivers then control intensity through 4 bit pulse width modulation (PWM). Please see the data sheets on these three components to learn more about their functionality as well as about the USB, serial, and I2C communications standards.

2.2. Assembling the Board

If you are not familiar with surface mount soldering, we recommend you begin by reading Mike Anderson's guide (Appendix C) which also includes helpful advice unique to this board.

---

*Electronic address: `ddf@mit.edu`

Red	XXXX0XX0
Green	XXXX0XX1
Blue	XXXX1XX0
Sensor	XXXX1XX1
Row 1	XX10XX0X
Row 2	XX10XX1X
Row 3	XX01XX0X
Row 4	XX01XX1X

TABLE I: I2C Address Masks for colors and rows.

A bill of materials may be found in Appendix A. You must populate the USB interface at the bottom of the board and may then populate between one and four of the quadrants. Each quadrant requires four LED drivers.

Begin by soldering the QSOP24 MAX7313 LED drivers in the quadrants, taking care to not short adjacent pins. Now is a good time to populate the resistor packs and decoupling capacitors located around each driver as well. (**DDF205:** There is a slight misprint on the board; the decoupling capacitors C1-C16 are not labeled. The 0603 footprints between adjacent Max7313s are C1-C16, the $.1\mu F$ decoupling capacitors.) The part numbers of resistor packs included in kits and assembled boards are given in the bill of materials; you should choose your own for the LEDs you have. Experiment to normalize apparent brightness between colors while not drawing more than 20mA with any LED. For matched LEDs, such as tricolor LEDs, this will most likely entail choosing resistors so that 20mA flows through each LED. Also note that you will most likely need to populate the resistor packs with 2 quad-packs (as shown in the bill of materials) instead of a single 8-pack, due to the small set of values available in an 8-pack. You should not populate header or wires at this time.

Next, populate the USB interface starting with the FTDI232BM and ATMEGA8, each of which is a TQFP32. (**DDF205:** There is a small misprint on the DDF205 silkscreen; the dot indicating pin 1 of U18, the FTDI232BM, is not visible. Pin 1 is at the upper left corner if the board is held so that the USB and power ports are at the bottom. That is, pin 1 is the farthest from the power connector. The silkscreen for the U19 dot is clearly visible at the bottom left corner.) Now solder the crystal on top of the FTDI232BM and 0603 resistors and capacitors around them both chips. Finally, solder the power and communications LEDS, the power connector, and the USB jack. Note that you may chose between the circular power plug or the Molex connector. There is no need to populate both. (**DDF117:** There is a misprint on the board. If you are using DDF117 and wish to use the Molex connector, you must put the connector on the underside of the board or swap the red and yellow lines on your cable. This problem has been resolved in DDF205; the Molex connector may safely be populated on top of to board.) To the right of the USB interface are several test points. These may be populated with .1" header, or left vacant.

DDF117: If you are populating more than one quadrant of LED drivers, you will need to use external pull up resistors on the I2C lines (If you chose to use internal pull-ups, you must modify the firmware appropriately). The external pullups are located along the SDA and SCL LEDs. If you do not populate these LEDs, you must bridge the pads on the LED footprint, so that the resistor connects to +5V. Some LEDs may also interfere with the I2C interface. It is therefore recommended that the SDA and SCL LEDs not be populated during normal use, though the footprint is present for testing purposes. None of this applies to the RXD and TXD LEDs, which may be populated or left vacant. DDF205 users may disregard this note, as the SDA and SCL LEDs have been removed and replaced with pullups.

DDF205: The latest hardware revision supports two modes of operation. The USB interface portion of the board may draw its power from either the 5V barrel connector or through USB. The former mode is equivalant to the original DDF117 hardware. The latter mode offers improved reliability when the 5V power supply is of low quality or connected by a large length of wire. (USB power will only operate the logic circuits. Power to the LEDs must still be provided using the barrel or Molex connector.) As such, if the board is to be used while connected to USB (as the vast majority of applications will require), it should be configured for USB power. There are two solder jumpers: SJ1, located to the right of the USB connector, and SJ2, located above it. For USB powered operation, use a ball of solder to bridge the center and bottom portions of SJ1 and the center and right portions of SJ2. For compatibility mode, bridge the center and top of SJ1 and the center and left of SJ2.

Finally, connect wire or header to the pads located around each quadrant. Pay attention to the numbering and orientation of these pads. Orientation is labeled on the left and right sides of the board and each connector footprint is clearly numbered.

2.3. Connecting LEDs and Sensors

Each cable connector has five lines - +5, R, G, B, and S. The power line is connected internally to +5V and should be wired to the anode of the three LEDs as well as one side of the sensor switch. The lines marked R, G, B should be connected to the cathodes of the three LEDs controlled by this cable. The line marked S should be connected to the other side of the sensor. Note that this line is internally pulled down by resistor packs R7, R8, R15, R16, R23, R24, R31, R32, which should be chosen accordingly.

3. FIRMWARE

Firmware for the Atmega8 is available online in the file firmware.c, which uses the libraries usart.c (a modification of a standard Atmel library for serial communication) and TWI_Master.c (a standard Atmel library for I2C communication). We recommend using AVR-GCC to compile this code. A compiled form is also available and should be preferable for most users.

You must also program the fuses on the Atmega8 to use an appropriate clock source. Ordinarily, the Atmega8 should use its internal oscillator set to 8MHz. (**DDF205:** Alternatively, you may wish to use an external clock source populated on X1.)

This firmware implements a base set of commands completely compatable with the firmware used on the 1E Dance Floor and a base set of error codes. For compatibility with existing implementations, it is recommended that the base commands and error codes be preserved.

A command consists of a single byte specifying the action, followed by the appropriate number of data bytes. Upon completion, the Atmega will return an appropriate number of data bytes, followed by a status byte. Each bit of the status byte corresponds to an error code, any number of which may be set. As such, a status byte of 0x00 corresponds to success. The base set of commands is given in Appendix B.

The firmware included uses a serial baudrate of 56Kbps to communicate with the USB interface chip. The baudrate may be increased to as much as 1Mbps. Please see the Atmega8 datasheet for information on supported baudrates.

4. SOFTWARE

Example host software is included for some POSIX platforms. Drivers for the FTDI FT232BM USB interface chip are available on the FTDI website for Windows (including CE), Linux, Mac OS X, FreeBSD, and OpenBSD. The included host software is most thoroughly tested on Mac OS X, although it should be fairly easily portable to other platforms.

The host software is written in C, and depends on several important libraries and header files:

- **fcntl.h**: The fcntl constants are used to open the POSIX device files corresponding to the module serial interfaces.
- **stdio.h**: The stdio functions are used to read and write bytes to the serial interface. These bytes give appropriate commands and data to the firmware running on the microcontroller in order to control the LEDs and other functions.
- **termios.h**: The termios structure and constants are used to initialize the serial port when opening it for reading and writing. This is what sets the baud rate, parity, bit format, etc.
- **stdlib.h**: The stdlib contains the malloc() function, which is used to allocate memory to the structures and buffers used to send data and commands to the floor.
- **libpthreads, pthread.h**: The pthreads library is a basic threading library. This allows I/O functions (reads and writes) to happen in parallel, so the control application doesn't block and wait on each read and write. This is critical for reasonable performance.

The host software included is in the form of a simple API that other applications can use (there are some example applications included as well). The functions available are prototyped in ddf.h, and the full functions are in ddf.c. Also, you will need to define appropriate constants in ddf.h for the size of your floor (number of modules). A different layout would also require adjusting the code.

The ddf library is threaded with pthreads, so it may not be usable within other thread libraries (for example, SDL threads). This may be of concern if you are developing a plug-in for another application that will control the dance floor. If this is the case, you should use pthreads when developing your plug-in, to maintain compatibility. If you are not interfacing with other threaded software, the threading should be completely transparent, and does not concern the application programmer.

Here's an example application to write random colors to the floor.

```
/* include the ddf library */
#include "ddf.h"

void main() {
  /* declare the dancefloor structure */
  dancefloor *fl;
  /* declare a buffer to write colors from */
  unsigned char *buf;
  /* just an iterator */
  int i;

  /* seed the random numbers with time */
  srand(time(NULL));

  /* initialize the dancefloor */
  fl = init();

  /* reset the floor to black */
  powerdown_dancefloor(fl);

  /* loop forever */
  while(1) {
    /* fill the buffer with random bytes */
    for(i = 0; i < ROWS*COLS*3; i++)
      fl->buf[i] = rand() % 256;
    /* write the buffer to the floor */
    write_dancefloor_buf(fl);
  }
}
```

When the floor is initialized, the process spawns a new pthread for each module. That module is then responsible for sending commands over the serial interface to that module. When a command is not being sent, the process simply waits for the master process to send a signal. It does this with a pthread condition.

More complex examples are included with the software. Also, refer to the ddf.h header file for specifications of other useful functions.

4.1. Third-Party Software

Since our code base is open source under the GPL, we encourage third parties to develop custom software to suit their needs. The Washington University IEEE Society produced an extension of the original software available at `washufloor.blogspot.com`. A Windows port produced by Clint Rutkas is also available at `www.betterthaneveryone.com`. We provide this information as a service, but of course cannot support third party software.

5. DEBUGGING

Debugging can sometimes be a painful process. To help you through it, here are a few suggestions if you run into problems.

5.1. General problems

If you are having any sorts of problems with the board not working, the first thing to do is check your soldering. Regardless of whether or not the board was working prior to the problem, a bad solder joint is the most likely culprit. It's particularly difficult to get good soldering joints on the 0402 resistor packs and MAX7313 LED drivers. You can check continuinity between pins on different ICs by measuring resistance with a multimeter. Another good way of identifying poor solder joints or bridges is to hold the board up to a bright light. This emphasizes the traces and pins on the ICs.

Another good thing to check is the board's power supply. The board draws a lot of current, so even if you are using a 5V power supply, the voltage may be dropping under heavy load and resetting the components. If the board stops responding sometimes when the many LEDs are turned on, you should consider using heavier power cables or a better power supply.

5.2. USB interface is not working

Surprisingly, many USB problems are caused by poor solder connections on the USB jack. If the jack is hit or leaned on, it may break the solder joints. Try touching up these connections before looking into more complex issues.

If the USB interface is not working and you have thoroughly checked all of your soldering and component placement, there are a few other possibilities. Many OSs offer USB drivers with extra debug capability and probing applications. This can give you a great deal of information about what is going on with the USB bus, and may point to the problem.

One possibility is that the EEPROM on the board is corrupt, and that is causing FT232BM to be configured improperly. You can try desoldering the 93C46 EEPROM to see if this corrects the problem.

5.3. Board is returning nothing or errors

The most likely case here, if all other possibilities have been checked, is that the ATMega8 is not properly programmed. In addition to reprogramming the microcontroller with the firmware, you should carefully check the programming fuses to ensure that the microcontroller is running at the right speed (8MHz with the provided firmware) and the right power supply voltage.

A painful mistake is to set the programming fuses such that the microcontroller uses an external oscillator (there is none on the board). Recovering from this is difficult–it is possible to use a function generator to provide a clock on the correct pins. (**DDF205:** As a solution to this problem, the new hardware revision contains a footprint for an optional resonator X1 in a Murata CSTCE package. While we typically do not recommend using this resonator for operation, it provides an excellent means by which to recover from incorrectly set fuses).

5.4. LEDs are not working

If only some LEDs are not working, the most likely problem is the soldering and connection of those LEDs (or those LEDs are broken). If an entire row of LEDs of one color is not working, the problem is probably with a MAX7313 driver chip. You should check to make sure that the chip is getting the right power supply (the MAX7313s are powered off of 3.3V provided by a small regulator on the USB driver chip) and they are properly connected to the I2C bus.

Another possible problem is that the I2C bus is not working at all. You should carefully check the pullup resistors used on the I2C bus and make sure that they match appropriately to the number of LED drivers you have populated.

6. THANK YOU

We hope the DDF Controller Board can serve your needs. We also hope you will find our documentation to be of the highest quality. This document will be updated periodically as we receive feedback. If you have comments or suggestions, feel free to contact us at ddf@dropoutdesign.com.

APPENDIX A: BILL OF MATERIALS

Ref	**Description**	**Mfg**	**Mfg Part**	**Supplier**	**Supplier Part**	Q
Board	DDF Controller Board	Dropout Design	DDF205	Dropout Design	DDF205	1
R1, R2, R9, R10, R17, R18, R25, R26	4x0402 Res. for Red LED	CTS	742C083151JTR	Digikey	742C083151JCT-ND	16
R3, R4, R11, R12, R19, R20, R27, R28	4x0402 Res. for Green LED	CTS	742C083820JTR	Digikey	742C083820JCT-ND	16
R5, R6, R13, R14, R21, R22, R29, R30	4x0402 Res. for Blue LED	CTS	742C083820JTR	Digikey	742C083820JCT-ND	16
R7, R8, R15, R16, R23, R24, R31, R32	4x0402 Res. for Sensor Pullup	CTS	742C083102JTR	Digikey	742C083102JCT-ND	16
R33, R34	27Ω 0603 Res.	Rohm	MCR03EZPJ270	Digikey	RHM27GCT-ND	2
R35	470Ω 0603 Res.	Rohm	MCR03EZPJ471	Digikey	RHM470GCT-ND	1
R36	4.7kΩ 0603 Res.	Rohm	MCR03EZPJ472	Digikey	RHM4.7KGCT-ND	1
R37, R39	10kΩ 0603 Res.	Rohm	MCR03EZPJ103	Digikey	RHM10KGCT-ND	2
R38, R45, R46	1.5kΩ 0603 Res.	Rohm	MCR03EZPJ152	Digikey	RHM1.5KGCT-ND	3
R40	2.2kΩ 0603 Res.	Rohm	MCR03EZPJ222	Digikey	RHM2.2KGCT-ND	1
R42, R43, R44	220Ω 0603 Res.	Rohm	MCR03EZPJ221	Digikey	RHM220GCT-ND	3
C1-C16, C18-C22, C24	.1μF 0603 Cap.	Rohm	MCH185CN104KK	Digikey	511-1175-1-ND	22
C17	4.7μF 0603 Cap.	Panasonic	ECJ-1VB1E475K	Digikey	PCC2318CT-ND	1
C23	33nF 0603 Cap.	Panasonic	ECJ-1VB1E333K	Digikey	PCC1769CT-ND	1
C25	33μF SMD Electrolytic B Cap.	Panasonic	EEV-HA1A330R	Digikey	PCE3003CT-ND	1
X2	6MHz Crystal	Murata	CSTCR6M00G53Z-R0	Digikey	490-1218-1-ND	1
U1-U16	Led Driver	Maxim	MAX7313AEG+	Maxim	MAX7313AEG+	16
U17	EEPROM	Atmel	AT93C46A-10SI-2.7	Digikey	AT93C46A-10SI-2.7-ND	1
U18	USB→Serial Interface	FTDI	FTDI232BM	Parallax	604-00031G	1
U19	Microcontroller	Atmel	AtMega8-16AI	Digikey	ATMEGA8-16AI-ND	1
CIRCULAR	Power Jack	CUI	PJ-102BH	Digikey	CP-102BH-ND	1
USB	USB B Jack	Mill-Max	897-30-004-90-000000	Digikey	ED90003-ND	1
TX, RX, PWR	Indicator LEDs	Lite-On	LTL-10223W	Digikey	160-1087-ND	3
ISP	2x3 Male Programming Header	Jameco	7000-2X3SG	Jameco	11503	1
T1-T64	Female Round Header	Jameco	6100-1X10	Jameco	102200	64
PLUG	Power Plug	CUI	PP-002B	Digikey	CP-004B-ND	1

APPENDIX B: PROTOCOL SPECIFICATIONS

The upper nib of a command gives its class. Currently, only classes 1-8 are in use. Class 0 is reserved for bootloader configuration and should not be used. Classes 9-F currently have no indicated use, though this is subject to change.

1. Read and Write Commands

Classes 1, 2, and 3 are reserved for write commands, read commands, and combination write/read commands, respectively. Additionally, lower nibs of 0 or 1 refer to commands that affect the entire board or a row, respectively. Lower nibs of 2, 3, and 4 are reserved for reads and writes of chips (that is, a single color of a single row), cells (three colors of a single cable connector), and individual LEDs, though these are not currently implemented. 2 commands are unimplemented because the current application makes their use unlikely to be efficient (though the code is in place, it is used as a helper function of a 1 command and not intended for direct calling). 3 and 4 commands are unimplemented because the DDF Controller hardware makes these commands more costly than a 1 command. They are reserved for cross compatibility with future hardware revisions.

0x10	**Write Module**
Input	96 Bytes Intensity Data
Output	Status Byte
Description	Sets intensities of all 192 LEDs (0xF is full off). Each set of 24 byte refers to a row (board quadrant). Within those, the first 8 bytes are red data, the next 8 are green data, and the last 8 are blue data. The first byte of color data contains the intensity of the first LED in the lower nib and the intensity of the second LED in the upper nib. Similarly, the second byte contains the third LED's intensity in the lower nib and the fourth's in the upper nib, and so forth.

0x11	**Write Row**
Input	1 Byte Row, 24 Bytes Intensity Data
Output	Status Byte
Description	Sets intensities of 48 LEDs in a single row (a number between 0 and 3 corresponding to a board quadrant). The first 8 bytes are red data, the next 8 are green data, and the last 8 are blue data. The first byte of color data contains the intensity of the first LED in the lower nib and the intensity of the second LED in the upper nib. Similarly, the second byte contains the third LED's intensity in the lower nib and the fourth's in the upper nib, and so forth.

0x20	**Read Module**
Input	None
Output	8 Bytes Sensor Data, Status Byte
Description	Reads sensor data for the entire board. Each pair of two bytes refers to a row (board quadrant). The lowest bit of the first byte is the first led of the row and the lowest bit of the second byte is the ninth.

0x21	**Read Row**
Input	1 Byte Row (0-3)
Output	2 Bytes Sensor Data, Status Byte
Description	Reads sensor data for the indicated row (a number between 0 and 3). The lowest bit of the first byte is the first led of the row and the lowest bit of the second byte is the ninth.

0x30	**Read and Write Module**
Input	96 Bytes Intensity Data
Output	8 Bytes Sensor Data, Status Byte
Description	Performs both a module write and a module read in a single command.

0x31	**Read and Write Row**
Input	1 Byte Row, 24 Bytes Intensity Data
Output	2 Bytes Sensor Data, Status Byte
Description	Performs both a row write and row read in a single command.

2. Preprogrammed Modes

Class 4 is reserved for preprogrammed (stand alone) behaviors. Currently only an effective power down, which turns all LEDs off, is implemented. Possible uses of this class include preloaded images or animations (though the latter requires some modification of the firmware structuring).

0x40	**Power Down**
Input	None
Output	Status Byte
Description	Turns all LEDs off.

3. Communications

Class 5 is reserved for communications related commands. Currently only ping is implemented.

0x50	**Ping**
Input	None
Output	Status Byte
Description	Returns a status byte of 0x00 (success).

4. Maintenance

Class 6 is reserved for maintenance commands. Currently only an online reset is implemented.

0x60	**Online Reset**
Input	None
Output	Status Byte
Description	Reinitializes LED driver hardware. Useful in the event of a brownout.

5. Diagnostics

Classes 7 and 8 are reserved for firmware and hardware diagnostics. Command 0x70 is reserved for version number; the remaining commands in this class are intended to be used as a pathway for obtaining debugging information, such as variable values. Since it is not intended to exercise hardware outside of the AVR, commands of class 7 need not return a standard status byte. Class 8 may include test patterns, sensor feedback testing, automatic diagnostic of LED drivers, etc. Currently, only a test pattern is implemented.

0x70	**Firmware Version Number**
Input	None
Output	1 Byte Version Number
Description	Returns the firmware version number. Historically, this is done with an implied decimal between the upper and lower nibs. Note that this command does not return a status byte.

0x80	**Test Pattern**
Input	None
Output	Status Byte
Description	Displays a test pattern. Most likely, this consists of flashing each LED in turn, though the implementation is left up to the user to suit their application.

APPENDIX C: SURFACE MOUNT SOLDERING

Courtesy of Mike Anderson.

1. Preparation

Here I will briefly go over soldering the different components involved in assembling a DDF controller board. Before you start you will absolutely need the following: a fine pointed soldering iron with temperature control, a flux pen, solder wick, and fine solder (I also carry tweezers and a small dental pick in my arsenal of tools, but these aren't necessary). In preparation, wet the sponge that comes with your soldering iron, or if there is none get an old sponge and keep it on a dish next to your iron. Turn the iron to greater than 550 degrees F, but less than 650. Tin the end of the soldering iron. You are ready to start.

2. QSOP Packages

Begin with the QSOP packages (MAX7313), as they are the hardest and you will be thankful when they're finished. To solder a surface mount QSOP, start by melting a dab of solder onto one of the corner pads. There is already solder on all of the pads, but this is important to get the positioning of the IC correct. Use your fingers to pick up the chip and place it over the pads (make sure it is in the correct orientation!). While holding it with one finger in the right place, use the soldering iron with your other hand to press down the lead above your solder dab such that the IC is attached to the board at that one location. Use this one location as a pivot to make any additional adjustments and solder down one more lead. Don't worry if these solder joints are clumsy or don't look great, their purpose is to hold down the IC while you solder the rest of the leads. This part is usually the hardest, once you can consistently position and attach qsops you will be set.

To solder the rest of the leads use the same methodology as soldering through-hole components, you want to heat up the lead and the pad at the same time and allow the solder to flow onto both. I use the following method: hold the tip of the iron at the point on the pad where the lead and pad meet, wait for it to heat up (can take a few seconds if it's the first lead on the side), apply solder at the top of the lead (where it goes into the IC), the solder should be melted by the heat of the lead, you should be able to watch the solder flow down to the edge of the pad. Do this for all leads.

You should be able to do this without to much mess, but you if you get bridges don't worry, that's what the flux pen and solder wick are for. To remove solder bridges place the solder wick across the top of the bridge and lay the tip of the iron atop the wick such that a maximal area is covered. Wait until you can feel the solder melt under the wick, then move the wick around so that it will suck up the molten solder. This process can be tricky at first, but once you get a feel for it, it will go quickly. Now all of the leads are soldered and there are no bridges, but it might not look so great. This is where the flux pen comes in. The purpose of the flux pen is that adding the flux will allow the solder to melt and reflow, but not at the places where the flux is. Push down on the flux pen to release the flux and apply to both sides. Carefully go over each pin allowing the solder to reflow, the joints should look smooth and have minimal amounts of solder, they should be shiny (not dull), and cover the entire lead and pad. If you don't like the residue the flux pen leaves behind on the PCB you can use alcohol wipes to get rid of it.

3. 0402 Resistor Arrays

The next component I soldered were the resistor arrays. You may notice that these packages have no leads, but rather half-moon indents on both sides. Start by places a dab of solder on one pad in the corner of the package. Use a toothpick or dental pick to hold the resistor pack steady while you melt the solder dab into the half moon. The package should be stationary now, go ahead and solder down the rest of it. There's really no trick to it, you just need to get some practice. Start with the ground sides of the switch resistors, those don't matter if you bridge them. (Resistor arrays are orientation independent)

4. 0603 Passives

To finish off a quadrant of led drivers don't forget the 1μF decoupling capacitors. More 0603 passives can be found in the controller region. Put solder down on one pad and use tweezers to hold the passive above the pad. Use your toothpick or dental pick to hold down the other end while you apply the soldering iron to the other. It should be attached now, just apply solder to the other end as usual. (All passives used in this project are orientation independent)

5. TQFP Packages

TQFP's are soldered in the same manner as the QSOP's. However, be careful of two things: do not switch the positions (the Atmega is U19 and the FTDI is U18) and take care to line up the small white circle on the chip (denoting pin 1) with the small white circle on the board. As with the QSOP's the hardest and most crucial part to your success is making sure the leads are lined up with the pads. Try getting one lead down and use it as a pivot to line up the other leads and then solder a second lead (in an adjacent corner). This will be frustrating at first, but stick with it and you will quickly improve.

6. Crystal

The crystal has 3 grooves underneath, each lined with metal that will suck the solder underneath of it. Put a dab of solder on one of the pads, and place the crystal over it. Heat up that dab of solder from the edge of the pad until you can see it flow under the crystal. Then apply solder and heat to one end of a channel; once the solder starts melting wait until you can see it come out the other side, before removing the iron.

7. Through-hole Components

Now that all of the surface mount components are installed, you will want to install all of the connectors, header, LED's, etc. in order of height, from shortest to tallest. For header in large sections (i.e connectors for the tiles), you may want to try taping down a row of header from the top, then carefully flipping it over and soldering the joints on the bottom. For individual header pieces (i.e atmega interface, debug interface), I would suggest getting a lot of solder on the tip of your iron and holding the header in place with one hand while you get one pin attached. The joint may be sloppy, but the header should be attached and not tilted. Solder the rest of the leads and go back and clean up the first joint.

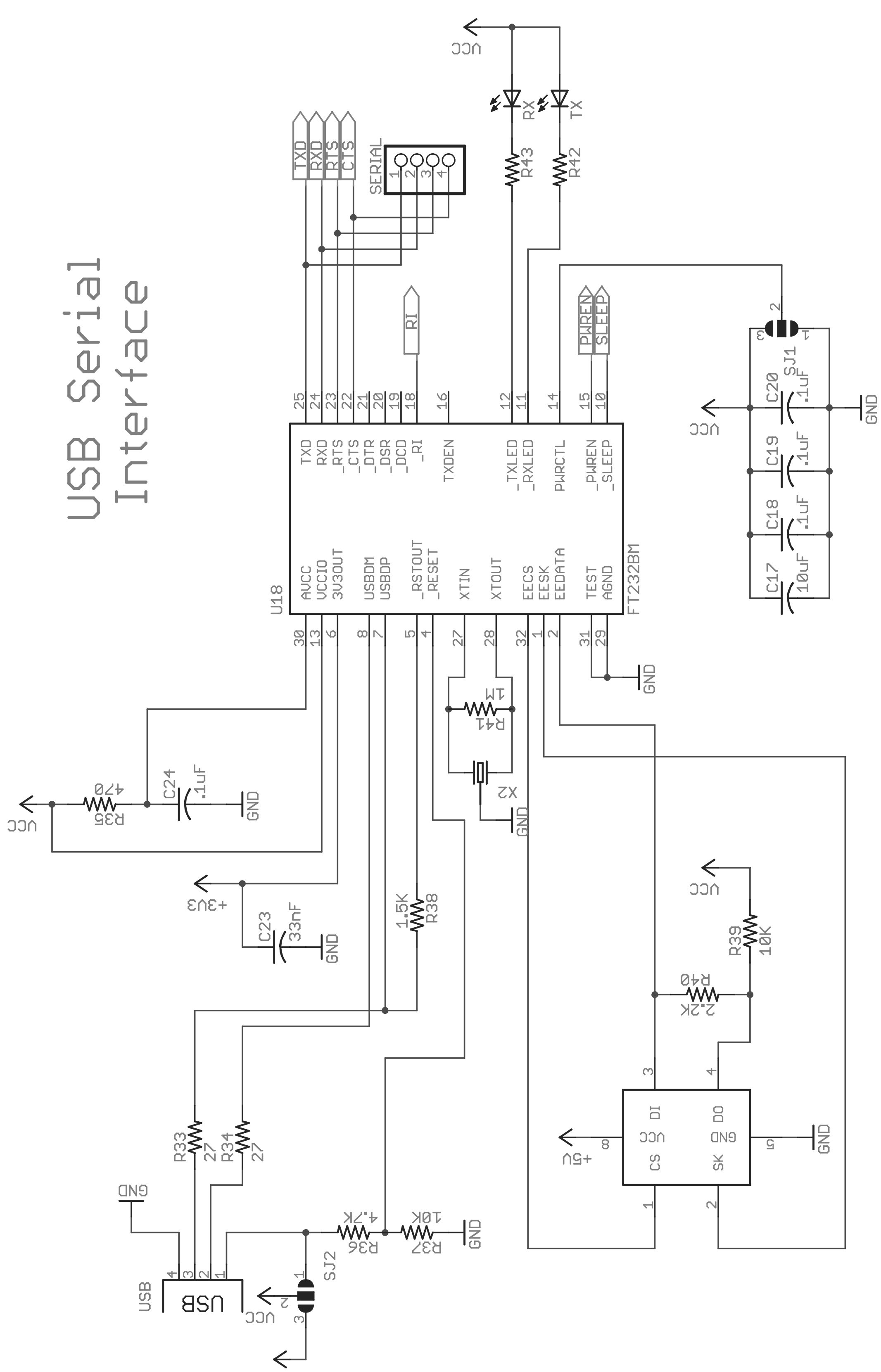
The USB connector is relatively simple. However, I suggest you load up on the solder for these joints, and make sure the joints are good. We found that a lot of communication issues arose from poor solder joints on the USB connectors.

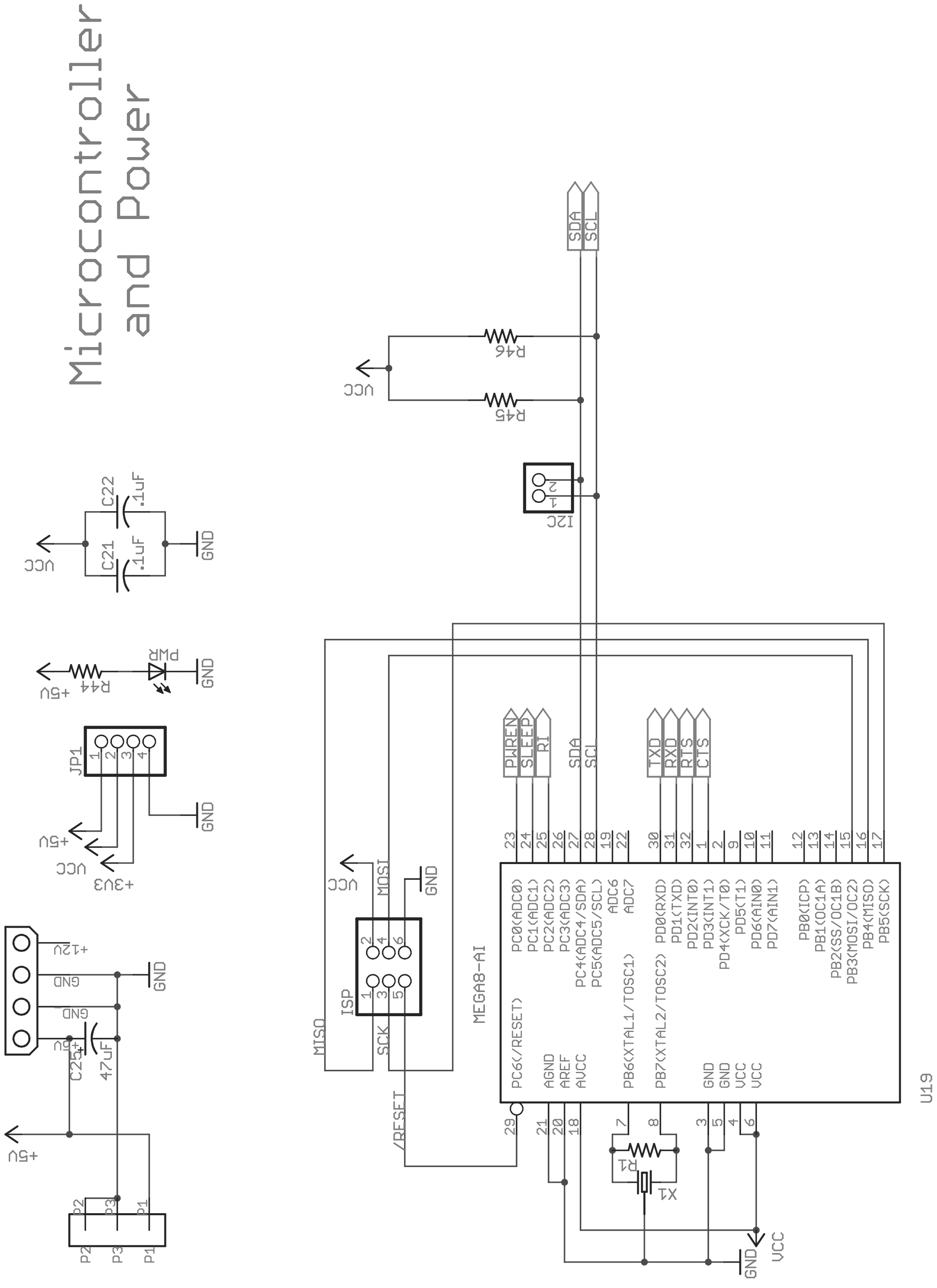
8. Aesthetics

If you like, you can (carefully) go over the board with a toothbrush (or rag) and rubbing alcohol to get rid of the shiny rosin spots that are left when the flux evaporates.

APPENDIX D: SCHEMATICS

The schematics are divided up into three sheets. The first sheet (Figure 1) shows the USB interface, which provides a serial interface to the microcontroller and a low-current 3.3V for the LED controllers. The second sheet (Figure 2) shows the microcontroller, which interprets the serial commands and interfaces to the LED controllers via I2C. The third sheet (Figure 3) shows LED controllers for a single row. This schematic is then replicated four times on the board.


USB Serial Interface
U18
FT232BM
AVCC
VCCIO
3V3OUT
USBDM
USBDP
_RSTOUT
_RESET
XTIN
XTOUT
EECS
EESK
EEDATA
TEST
AGND
TXD
RXD
_RTS
_CTS
_DTR
_DSR
_DCD
_RI
TXDEN
_TXLED
_RXLED
PWRCTL
_PWREN
_SLEEP
SERIAL
TXD
RXD
RTS
CTS
RI
PWREN
SLEEP
R43
R42
RX
TX
VCC
SJ1
C20
.1uF
C19
.1uF
C18
.1uF
C17
10uF
GND
R41
1M
X2
R35
470
C24
.1uF
+3V3
C23
33nF
R38
1.5K
R33
27
R34
27
USB
SJ2
+5V
R36
4.7K
R37
10K
R39
10K
R40
2.2K
DI
DO
VCC
GND
CS
SK


Microcontroller
and Power
SDA
SCL
R45
R46
VCC
I2C
1
2
C21
.1uF
C22
.1uF
GND
R44
PWR
+5V
JP1
1
2
3
4
+3V3
+12V
C25
47uF
P1
P2
P3
ISP
MISO
MOSI
SCK
/RESET
MEGA8-AI
U19
PC6(/RESET)
AGND
AREF
AVCC
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
ADC6
ADC7
PD0(RXD)
PD1(TXD)
PD2(INT0)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
PWREN
SLEEP
RI
TXD
RXD
RTS
CTS
R1
X1

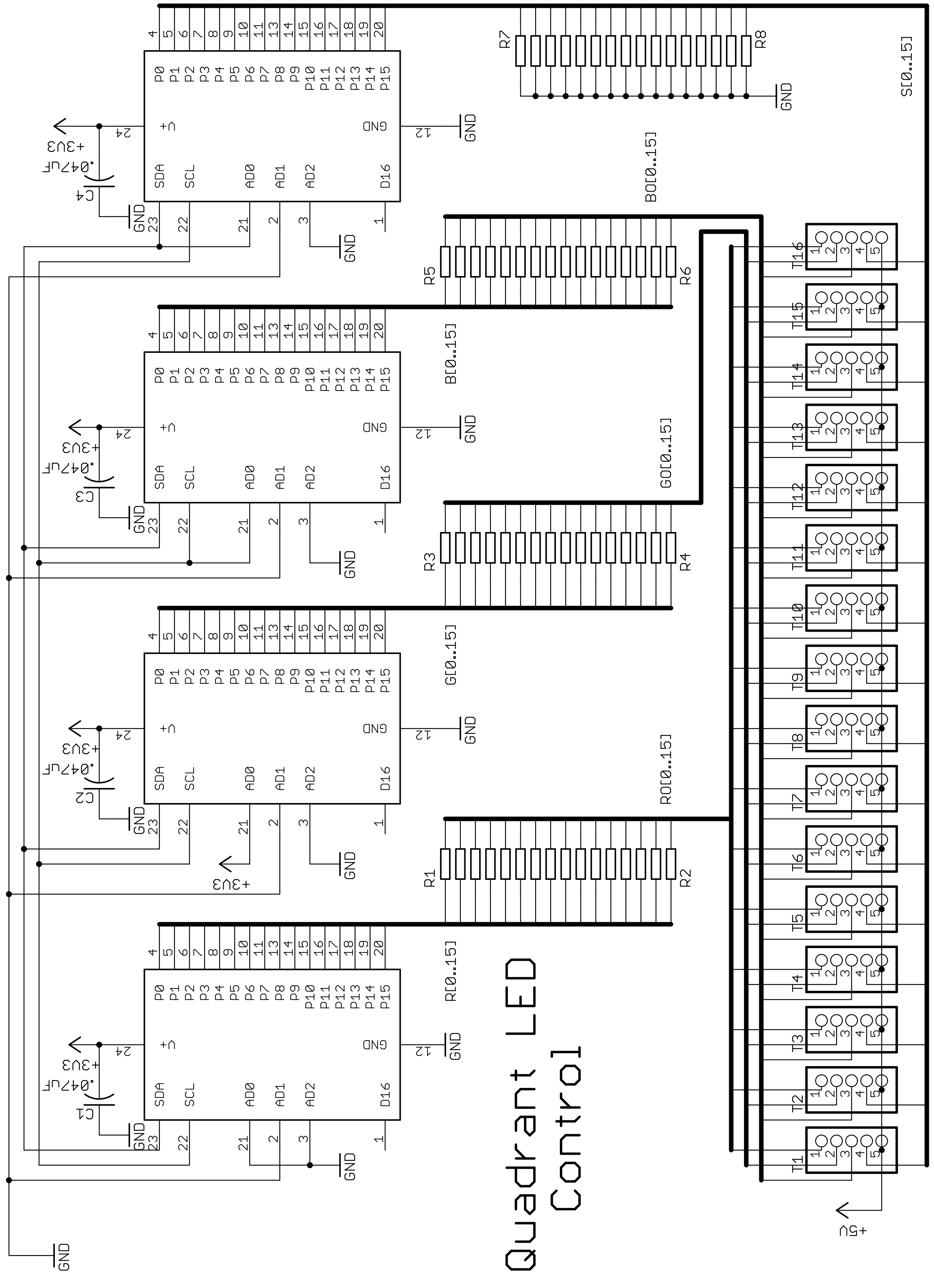

Quadrant LED Control
S[0..15]
R7
R8
GND
B0[0..15]
B[0..15]
G0[0..15]
G[0..15]
R0[0..15]
R[0..15]
R1
R2
R3
R4
R5
R6
C1
C2
C3
C4
.047uF
+3V3
+5V
SDA
SCL
AD0
AD1
AD2
D16
V+
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
T14
T15
T16

FIG. 3: Quadrant Schematic