INKJET PRINTER FOR THE REPRAP

By AMBERISH JAIPURIA

Supervisor: Adrian Bowyer

Assessor: Jafar Jamshidi

Summary

The aim of this project is to build an inkjet printer for the RepRap.

The RepRap is a fused deposition modelling (FDM) rapid prototyper (RP machine). The unique thing about RepRap is that most of the parts for it will be made using another RepRap machine (although the machine is still being developed).

The print cartridge to design the printer around is the *HP 51604A*, a thermal inkjet. The printing is driven by short electrical pulses that cause some of the ink to vaporise and spray onto the print medium.

To control these pulses, the microcontroller to be used is the *arduino*, a microcontroller with open-source software.

The electronics used to drive the circuit involved using two transistor arrays (Darlington arrays) to drive the printing, which drove the print cartridge using a 20V (regulated down from 24V) supply. This meant that the pulses to print had to be about 6.5 µs.

The transistors were driven by the *arduino* microcontroller, which pulsed the resistors that drive the printing. To make the *arduino* do this, various software programs had to be written in the *arduino*'s programming language, which is similar to *C*.

A push button controller was also added, which allows the user to turn the printer on and off using a button.

The mechanical design to hold the print cartridge and connect the contacts on the cartridge to the electronic circuit proved to be fairly challenging, but was solved by using slightly bent springy wire guided using slots. The cartridge is held in place by using a top and bottom section, the top located by the conical shape of the top of the print cartridge, and the bottom by two locating pins on the cartridge. These two sections are held together using 4 M3 bolts.

There was a major issue, which was solved by making a minor addition to the circuitry, and turning the circuit off when uploading programs to the *arduino* board.

To avoid this switching on and off of the circuit easier, a transistor needs to added, blocking out the arduino's ground when the circuit needs to be switched off.

Another future addition to this design is to build a simple holder that allows the printer to be connected to the RepRap's moving head.

It would also be a good idea to try our different inks in the printer, hopefully resulting in complex electronic circuits being printed out using this printer.

Contents

Introductory Sections

Aim

The primary aim of this project is to build an inkjet printer for the RepRap machine. The aim of this project is not to produce a perfectly designed printer, just one that works. Future versions of the printer (which will be built in future versions of RepRap) are likely to be designed more efficiently, and in a more suitable manner for the capabilities of what the RepRap will be capable of.

The print cartridge to design around is the HP 51604A. There is extremely detailed information available about this particular print cartridge [1], and it is very versatile.

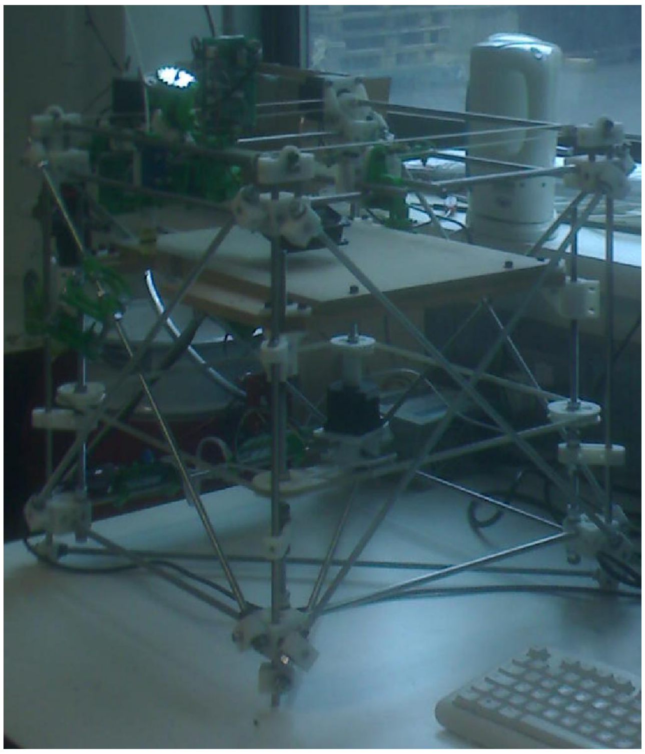
The printer requires a device to hold the cartridge, and needs some means of connecting to the electrical contacts on the cartridge. The mechanical holding parts must be manufacturable by the rapid prototyper (RP machine) used to make many of the RepRap's parts. It is thought that the RepRap should be able to produce similar parts fairly soon.

The electronics which connect to and drive the printer (to cause it squirt ink, rather than actually move the print head) must also be designed and produced. To aid this, an open source microcontroller, *Arduino*, has been provided.
The *arduino* uses its own coding language, based on C++, which must be used to programme the microcontroller; writing the code is also part of this project.

RepRap

The fundamental aim of the RepRap project is to build a self replicating rapid prototyper [2].

Figure 1: The RepRap machine



A rapid prototyper (RP machine) is basically a 3D printer, which uses a computer to control the building of a part; layer by layer. The specific type of rapid prototyping involved in the RepRap project is fused deposition modelling (FDM).

FDM is where the building material is melted and squeezed out of a nozzle in tiny amounts to make cross sections of a shape (once the material has cooled); after which the nozzle moves up (or, more often, base moves down) and the nozzle squirts out the next layer. Using this technique it is possible to construct extremely complex 3D shapes.

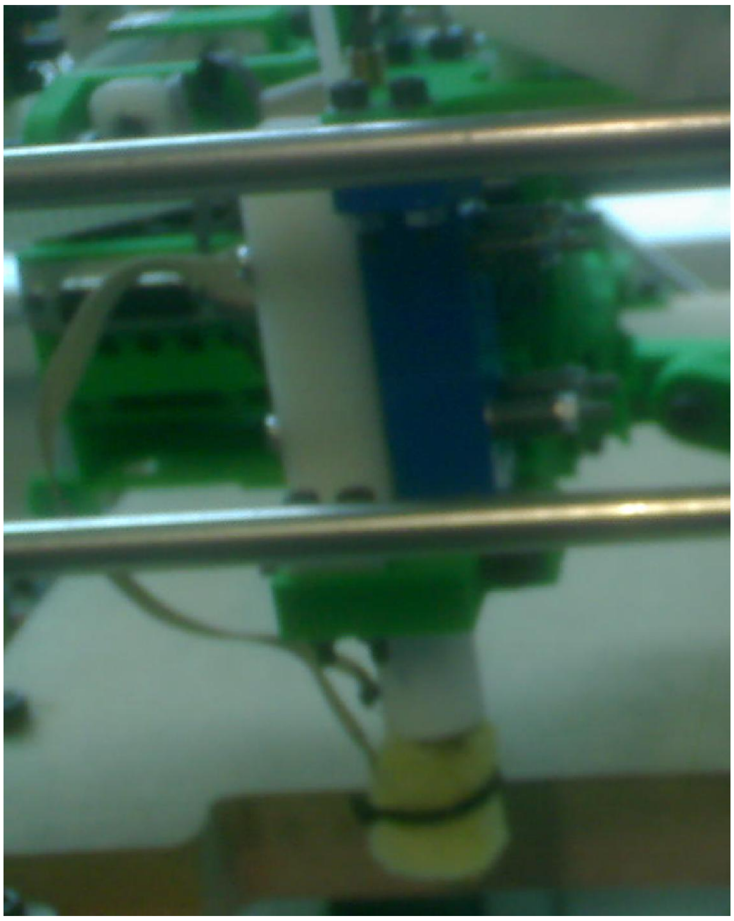


Figure 2: The RepRap's FDM head

This technique, is, however, limited by the fact that it is not possible to construct holes in the z axis, as there is nothing to build the material on. Commercial FDM machines typically use some form of support material, which allows the building of holes. The support material is broken (or even dissolved) away after the whole part is built; although it is still not possible to make cavities that have no open faces (are entirely contained within the part) without leaving some support material in the hole.

RepRap tries to use this technique of RPing to build the majority of the parts for the RepRap machine, which is in itself a FDM RP machine (which mostly builds using polycaprolactone or polylactic acid). The idea is that this method can be used by the RepRap machine to make most of the parts for another RepRap machine [2]; thus making the machine capable of self-replicating (where the ‘rep’ in RepRap comes from; ‘rap’ is from rapid prototyper).

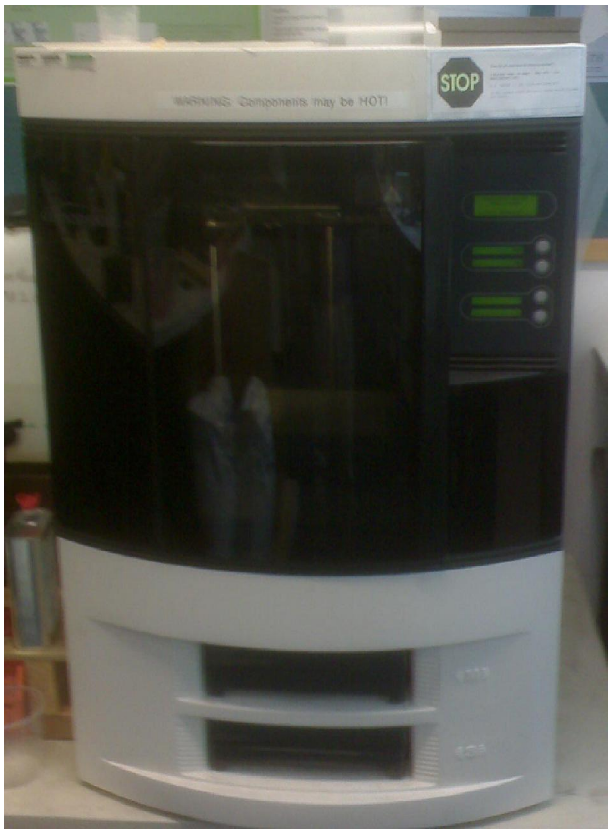
Generally, when people are informed about a machine which can self-replicate, they instantly think of a machine that can construct all the parts necessary to build an identical machine, *and assemble it*. This is where RepRap differs from many other attempts at 'artificial' self-replication. Rather than trying to build extremely large and complex machines to assemble the machine, the machine is assembled by the user. This means that the RepRap tries to make the best of both worlds, using the dexterity of humans (fingers in particular), along with the precision of a computer controlled machine.

Along a similar vein, the current aim of the RepRap project is to only build most of its own parts; not all. The parts that it can't build are things like motors, microchips and power supplies (and most electronics in general). It was also decided that it needn't manufacture its own mechanical fasteners, as they are ubiquitous, and would take a lot of effort and time to construct to an acceptable standard; time and effort which could be better spent improving the general capability of the machine instead[3]. It is hoped that future generations of the machine will be able to manufacture all the parts required to build a RepRap machine, but that seems unlikely in the immediate future.

Stratasys dimension BST

Since the RepRap is not yet at the stage where it can build objects with the accuracy that is required for this project, the parts for this project will be produced on an RP machine that has been used to make many of the parts of the current version of RepRap (and has similar capabilities to what the next generation(s) of RepRap will probably have; capabilities like print resolution).

Figure 3: The Stratasys Dimension BST



The commercial RP machine used is the Stratasys Dimension BST (*Strat*); an FDM (as described in the RepRap section) 3D printer. The *Strat* uses ABS plastic [4] as its building material, and uses a support material (material that is placed where there are gaps in the model, so that there is something to build the higher layers on) that must be broken away manually. The *Strat* uses 0.245 mm or 0.33 mm thick layers or ABS and support material [4].

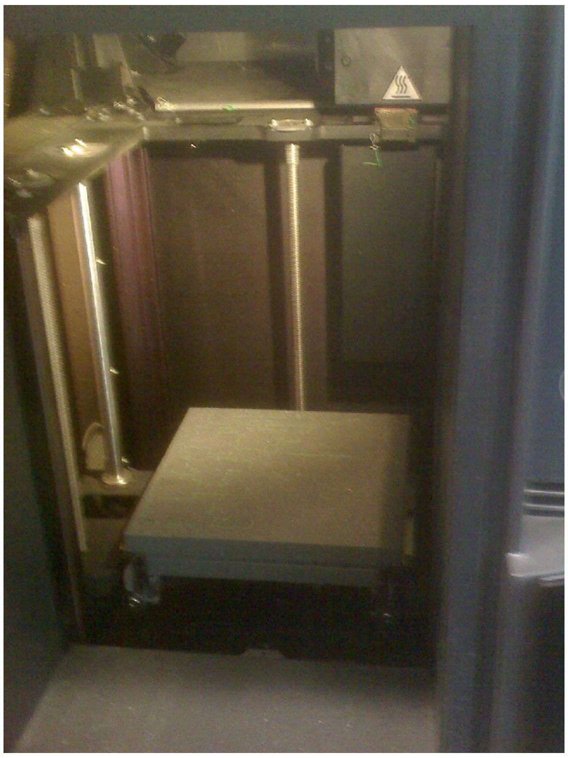


Figure 4: The inside of the Strat

Thermal inkjets and the HP 51604A

The inkjet printer in this project is designed around the Hewlett-Packard 51604A print cartridge. This particular cartridge was chosen because there is a lot of information available about it (including a book on how to control it [Gilliland, M. *Inkjet Applications*]), and that it is described as being a fairly robust print cartridge.

The printer is a Thermal Ink Jet (TIJ), which is commonly used in most domestic applications. The way it works is:
A small amount of ink is allowed to fill a chamber near the print head. A small resistor is located near the chamber. When a small pulse is sent through this resistor, it heats up, vaporising the ink. This gaseous ink then escapes, via a nozzle, condensing on the surface of the material being printed on. Another droplet of ink replaces the droplet that's just been fired, re-filling the printing chamber [1].

When TIJ printers were first developed, the ink was contained within a bladder; so as the ink was used, the bladder slowly collapsed, creating a small amount of suction to prevent the ink from escaping when it wasn't meant to (or *dribbling* out). The 51604A uses a sponge within the cartridge to perform the same task [1].

The HP 51604A has 12 nozzles, adjacent to twelve 65 Ω resistors [1]. They are in a single line, and span 3.2mm (which is the maximum width of print without making additional adjacent passes) [1]. The layout of the bottom of the print cartridge is shown here:

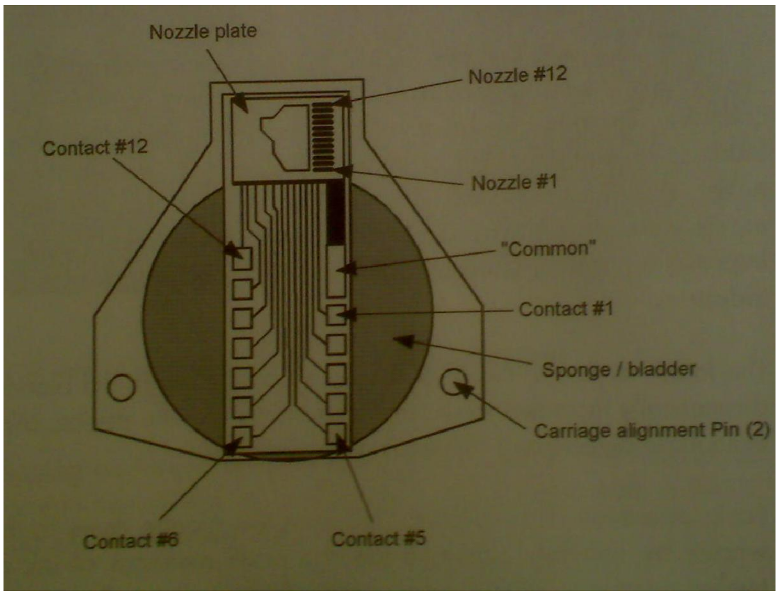


Figure 5 [4]**:** A diagram of the underside of the HP 51604A print cartridge

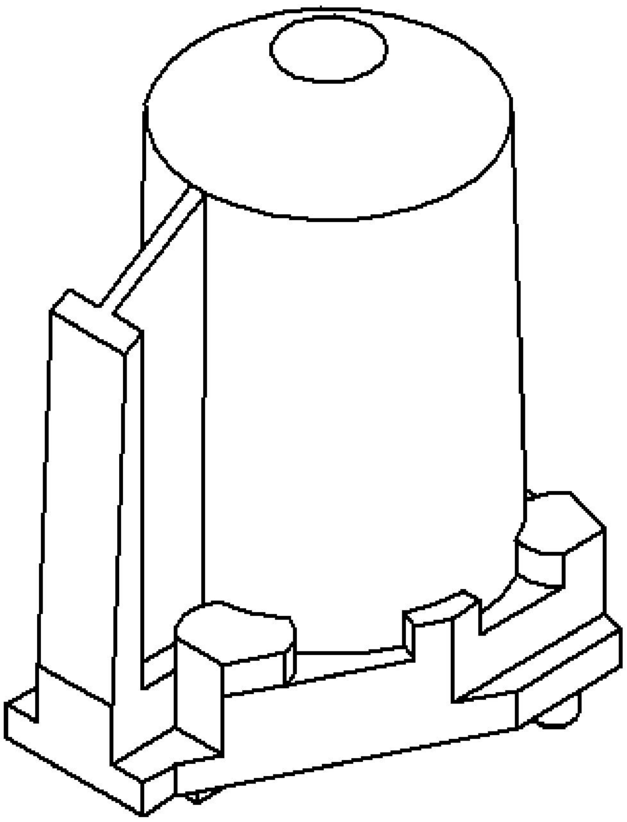


Figure 6: A model of the print cartridge

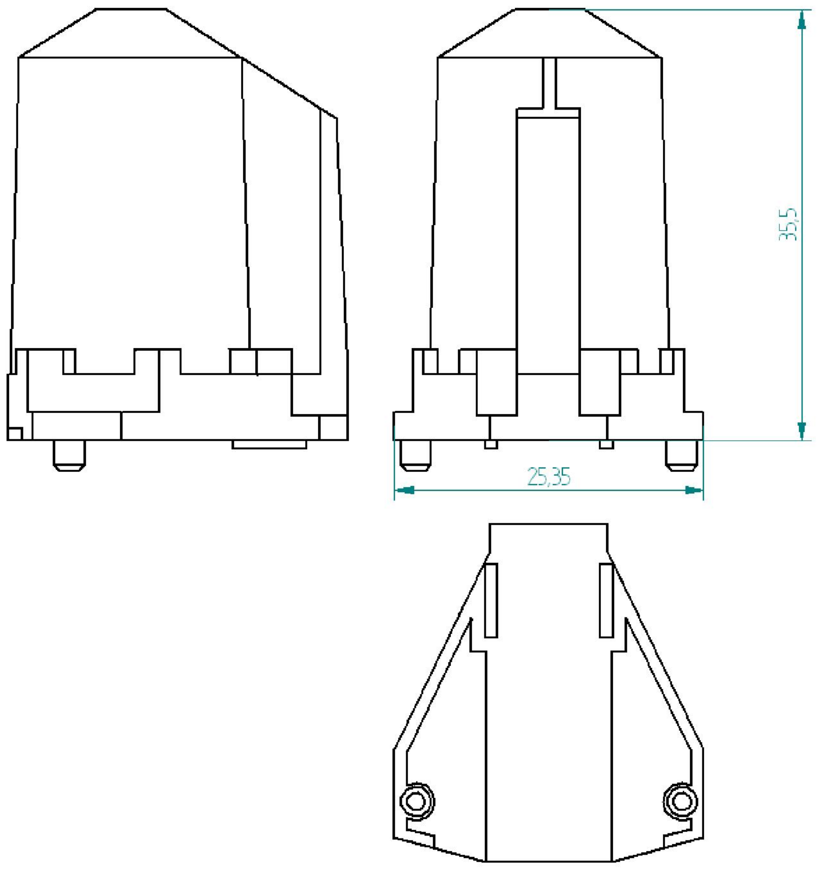


Figure 7: Orthographic views of the print cartridge

It may be worth noting that the contacts are in the same plane as the print head; meaning that the connecting and support mechanisms must be thin, so as to not degrade the print quality. Although the cartridge specification indicates that the print cartridge to paper spacing should be no greater than 1.5 mm [5], Gilliland suggests that the specification is a little under optimistic, and that anything up to 7 mm should be ok[1]. It does, however, state that print quality does start to deteriorate at this range; stating that in the end it is down to the designer to find an acceptable compromise between cartridge holder thickness and print quality.

The spec also states that the minimum distance between cartridge and print medium is 0.5 mm [5], but this is unlikely to be an issue for this design.

As mentioned, the inkjet fires when a small pulse is put across a resistor. The characteristics of the pulses, and many other limitations regarding the triggering pulses, are mentioned in Gilliland's book [1].

The optimum firing energy of the print cartridge is about 40 microjoules. This means that the pulse width can be equated to the voltage using the following equation:

$$pulsewidth = \frac{resista\eta ce \times energy}{voltage^2}$$

$$pulsewidth = \frac{65 \times 0.00004}{voltage^2}$$

Gilliland has calculated this for the optimum values for this, and provides this table (the pulse widths have been rounded to the nearest half microsecond) [1]:

Table 1: The firing times at different voltages

Voltage	**Pulse width**
(V) DC	**(µs)**
20	6.5
21	6
22	5.5
23	5
24	4.5

Since these are described as the optimum pulse widths, it may be possible to change the pulse widths so they do not exactly match up with data in the table; Gilliland states that changing the pulse times may be an option, but that performance may drop [1]. He goes on to elaborate, saying that reducing the pulse time could result in poor droplet quality and clogged nozzles, while too long a time (and energy) could burn out the resistors [1].

Gilliland also notes that with the extremely short pulse times, it might be better to use a lower voltage with a longer pulse time, increasing the amount of control the programmer has on the pulse width. The reason for this is that at such short timespans, the microcontrollers processing time becomes a significant factor; for example, the *arduino* board takes approximately 3 to 4 µs to turn an output pin on and off with no commands between the on and off commands. This means that, since the microcontroller being used is the arduino, the voltage cannot be over about 26 volts (and may run into some difficulties at 24 volts) [1].

Gilliland recommends that the time between the same nozzle being fired twice should be at least 800 µs [1]. This criterion is based on the time taken for the ink to refill in the chamber, and any firing at a faster frequency will result in poor quality printing.

The HP 51604A specification sheet states that up to two nozzles, and no more, can be fired simultaneously. Gilliland suggests that this is because of the limit on the energy that the conductors within the print cartridge can effectively deliver. Gilliland also says that two adjacent nozzles cannot be fired simultaneously, as this will give unpredictable results. This is probably because the droplets would interact, and could end up some distance from the intended target based on the relative velocities of the droplets. Gilliland goes on to suggest that maximising distance between nozzles being fired at the same time will give the best and most reliable performance [1].

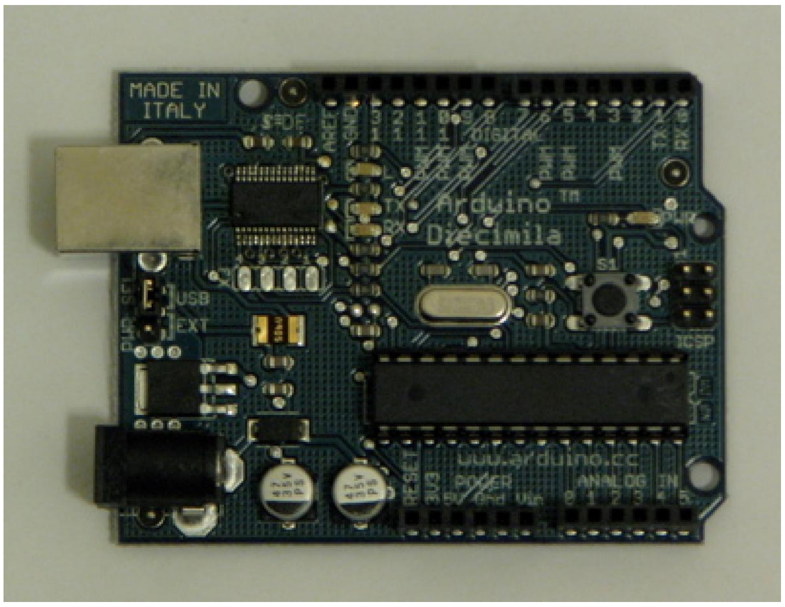
The dead time, or the time between firing two pairs of nozzles, according to Gilliland, is 0.5 µs [1]. Although Gilliland does not explicitly state a reason, it is likely to be because this is the minimum gap required to prevent the aforementioned issues of energy limits on the circuitry and interference between droplets.

A lot of these specifications will not cause the printer to fail, but will adversely affect the print quality. Gilliland makes a significant effort to emphasise that print quality is entirely subjective, and that the designer must decide whether the quality of the printing is good enough for the given application [1].

Arduino

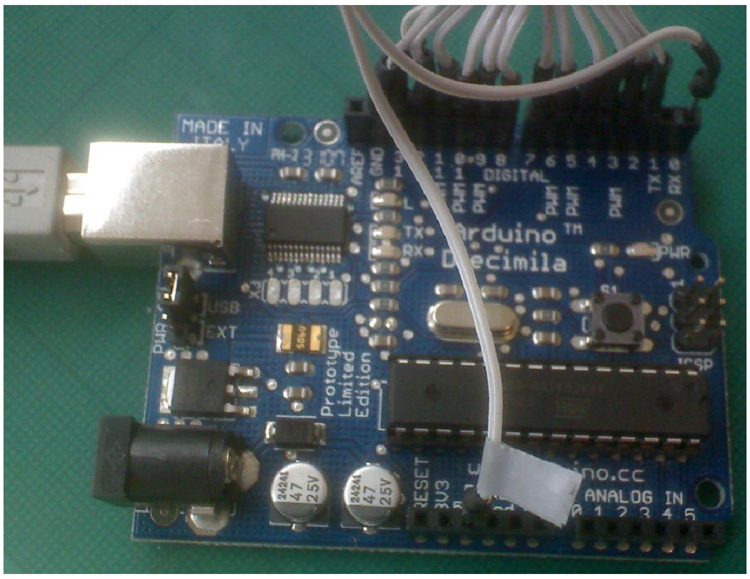
Arduino is an open-source device with a microcontroller [6]. It comes with its own software that, along with allowing the computer to communicate with arduino board (uploading programs to the board, and downloading information from it via the boards 'serial' output), also provides space to write code for the microcontroller, and a compiler to test it. The *arduino* uses its own programming language; based on the programming language *C++* with added functions to aid the programming of the microcontroller.

Figure 8: The *arduino* board

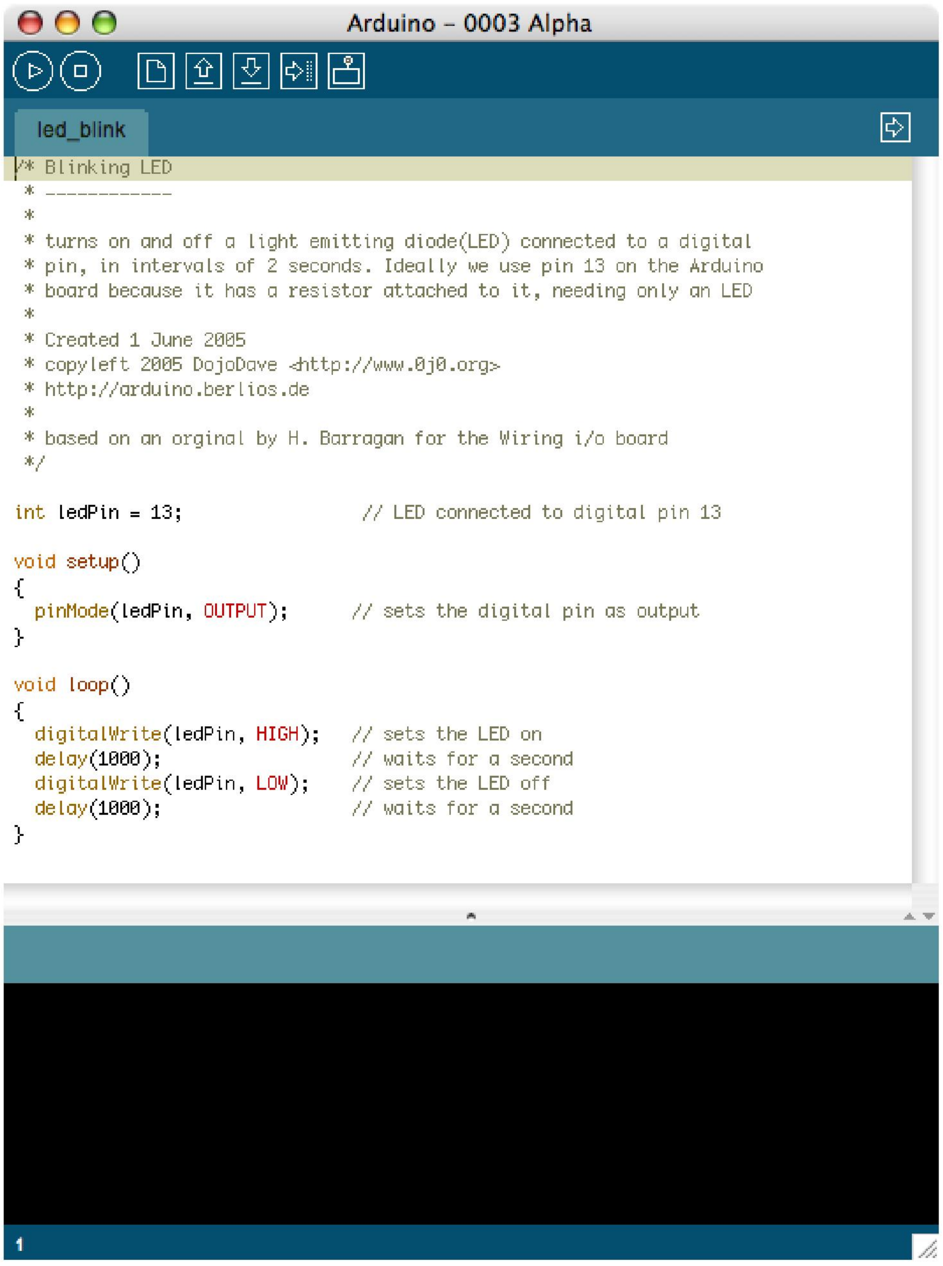


The specific model being used is the *Arduino Diecimila*. This model contains an ATmega168 microcontroller and 14 digital input/output pins, along with 6 analogue input pins (the analogue pins can be used as digital extra pins, addressed from 14 to 19, with the analogue 0 being used as digital 14). The board has a 5V output, and can produce 40mA per pin. The board can be powered by a power supply within 6 – 20 V, although it does recommend that the power supply be between 7 – 12 V. Alternatively, the board can be powered using a USB connection from a PC, which can also be used to communicate with the device and download programs from the PC onto the *arduino* board. There is also a serial connection available via digital pins 0 and 1 which can be used to communicate with the PC (or a host of other devices). The board has 16 kb of flash memory, of which 2kb is used by the boot loader (which is particularly useful because of the frequent reprogramming of the microcontroller). The board has a clock speed of 16 MHz [6].

Figure 9: The arduino board, with wires to drive the printer (with pushbutton)



The software used to program in and communicate with the board is open-source, which means that everyone can download, use and alter all the arduino codes (it may also be worth noting that the microcontroller is also open-source, which means that one with significant knowledge of microcontrollers can modify the chip). A screenshot of the program used to write and upload the codes is shown here:



```
/* Blinking LED
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED
 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

int ledPin = 13;                 // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Figure 10 [6]**:** A screenshot of the *arduino* software interface

The code is written into the main part of the window. The sketch is then verified (or compiled), and then uploaded to the board. During the verification and uploading, the return informs the user about what the computer is doing/has done and notifies him/her of any errors. This window is also used for serial communication (where the arduino sends signals back to the PC. This is done using functions in the 'serial' library, like the code '*Serial.println("Hello World!");*' which prints the words '*Hello World!*' in the return box).

There are two basic parts to an arduino *program*; the *setup* and the *loop* parts. Both *setup* and *loop* functions are required for all programs [6] (even if there is no information in one of them, they have to be present; just without any code in the curly brackets). The *setup* function is used for declarations and non-recurring events. Therefore it is typically used to initialize variables and things like *pin modes*. This function runs just once after each reset of the board. The *loop* function, on the other hand, repeats itself. It loops continuously, executing the program again and again, and varying as the program changes (and responds). The functions can be described by saying that *setup* is preparation, and *loop* execution, although this is a significant oversimplification (and isn't always true).

Some of the functions that will be used (excluding the ones already present in C++) include '*pinMode(pin, mode)*', '*digitalWrite(pin, value)*', and '*digitalRead(pin)*'. As the first part of the last two functions suggests, these functions are used for the digital pins.

The first function, '*pinMode(pin, mode)*', is used to tell the arduino whether a specific pin is an input or output pin; the contents of the brackets is changed so '*pin*' is replaced with the number of the pin, and '*mode*' replaced with either '*INPUT*' or '*OUTPUT*', depending on whether the pin is an input to or output from the board. So to make pin 4 an output pin, the code to enter would be:

```
pinMode(4, OUTPUT);
```

Before going through the next two functions, certain constants that are not normally found in C++ must be mentioned. The constants in question are 'HIGH' and 'LOW'. When talking about outputs, setting a pin high will set it to 5V, while a low setting connects the pin to ground. When referring to inputs, an input of 3V or more will result in a high reading, while 2V or lower will be read as low.

The '*digitalWrite(pin, value)*' function is used to set output pins to either high or low. Once again the information is entered in the brackets, with the pin number replacing '*pin*', and either '*HIGH*' or '*LOW*' replacing '*value*' based on what's desired (it may be worth noting that '*HIGH*' doesn't necessarily turn things on and '*LOW*' off; these are often inverted). To make pin 2 to go high and then low, immediately after, the code entered would be:

```
digitalWrite(2, HIGH);
digitalWrite(2, LOW);
```

The third function mentioned, is the '*digitalRead(pin)*' function. As its name suggests, it also relates to a digital pin, but instead of writing, the function reads it. This function is used for inputs, and reads whether a pin is high or low based on the criteria for input constants outlined earlier. With this function, '*pin*' is replaced with the pin number of the input to be read. This function does, however, vary from the previous functions, as this is function is not a simple command that can just be executed. If this function is simply run, the microcontroller will find the state of the pin, and promptly ignore it. The program either needs to use the information immediately, or requires a place to store this information. To use the information immediately, a code similar to this can be used:

```
if (digitalRead(11) == HIGH)
//do something exciting;
```

But it is considered bad practise to have functions embedded within other functions. Therefore, the output should be stored instead, by equating it to a variable:

```
int val;
val = digitalRead(11);
```

This means that the reading that was taken will be available as a variable, and can be called upon by other functions.

Another useful pair of functions are the ‘*delay(ms)*’ and ‘*delayMicroseconds(μs)*’ functions. These are used, as their names suggest, to make the microcontroller wait a set amount of time between two other commands. For the regular delay function, the ‘*ms’* in the brackets is replaced with the time, in milliseconds, that the microcontroller needs to pause for. Similarly, when using the ‘*delayMicroseconds(μs)*’ function, the ‘*μs’* is simply replaced with the time, in microseconds, that the microcontroller needs to wait for. So if the controller needs to wait 2.341533 seconds, the code to use would be:

```
delay(2341);
delayMicroseconds(533);
```

It may also be worth noting that both the variables to be entered must be of the integer (or ‘*int*’) class, although it is still possible to *push* a double through the regular ‘*delay(ms)*’ function (but for such instances the microsecond delay function could just be used), although this does not work for the ‘*delayMicroseconds(μs)*’ function.

Another useful function is the *‘millis()’* function. This function returns the time, in milliseconds, since the *arduino* started the current program. This function, like the digital reading function needs to be used or stored to be useful. When storing this function, the variable used has to be of the ‘*long’* class, as this number gets very large very quickly. The function itself is also a ‘*long’*, and therefore returns to 0 when it ‘overflows’, which happens after about 9 hours and 32 min. In this code, a variable called time is created and stores the time since the program started (at a particular instant):

```
long time = millis();
```

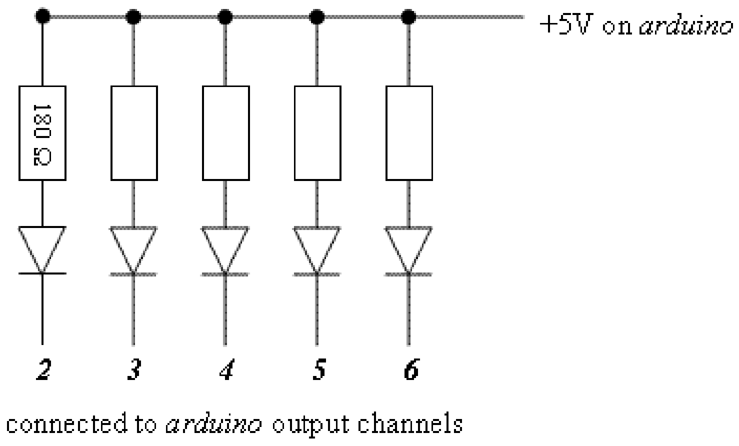
Discussion

Electronics:

The first thing done involved making a circuit to visualise the output of the arduino.

The circuit used was as follows:

Figure 11: A plan for a circuit with 5 LEDs



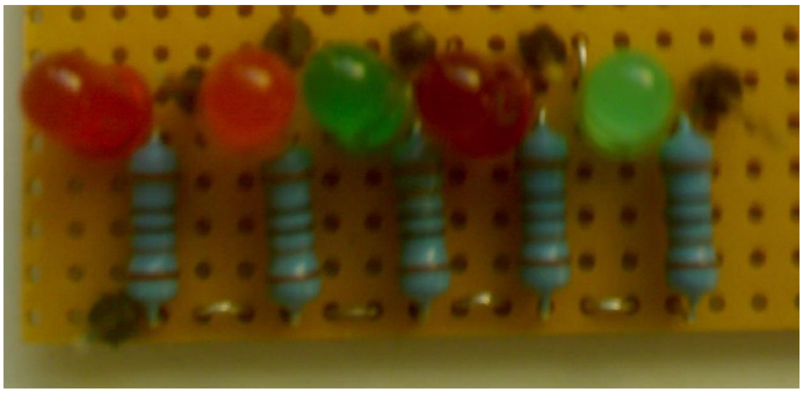


Figure 12: The 5 LED circuit

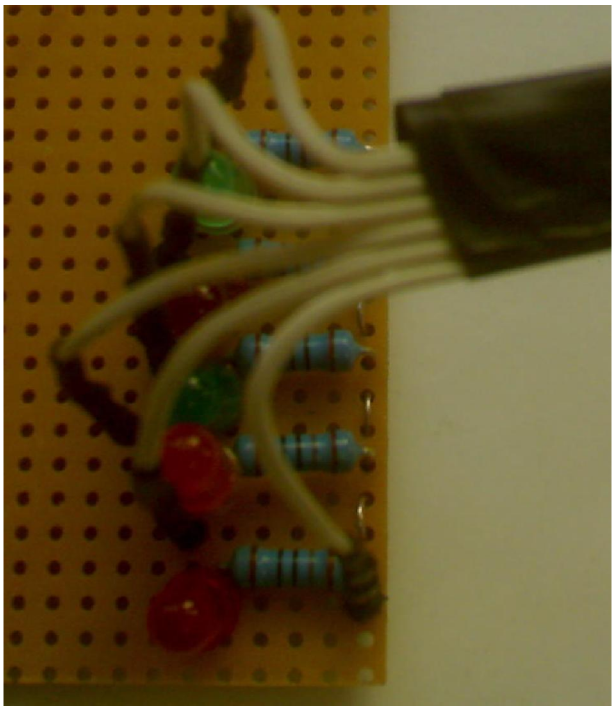


Figure 13: The 5 LED circuit, with wires

This circuit consists of 5 LEDs (each connected to a 180 Ω resistor) connected in parallel to the 5 V output on the arduino board. The other end of the LEDs are each connected to an output pin on the arduino. By setting the output to each pin using the arduino software to either *high* or *low*, the LEDs can individually be turned on and off.

The resistors are required to prevent the current in the LEDs from getting to high; the resistance can be found by the using the equation:

$$\Delta V = i \times R$$

$$R = \frac{V_{\sup} - V_{loss}}{i}$$

Where

V_{sup}	= supply voltage
V_{loss}	= voltage lost across diode
i	= desired current
R	= resistor required

$$V_{\sup} = 5V$$
$$V_{loss} \approx 1.5V$$
$$i \approx 20mA$$

$$\therefore R = \frac{5-1.5}{20\times10^{-3}}$$
$$R = 175\Omega$$

180 Ω is the first resistor up, when rounding up.

The preceding circuit was used to test some of the programs on the arduino; although the length and frequency of the pulses used to actually run the printer are too small to see with the human eye.

This model also gave the author of the this report an opportunity to learn more about circuit boards and various techniques involved in soldering.

The next circuit built was designed to interface between the arduino and the print head.

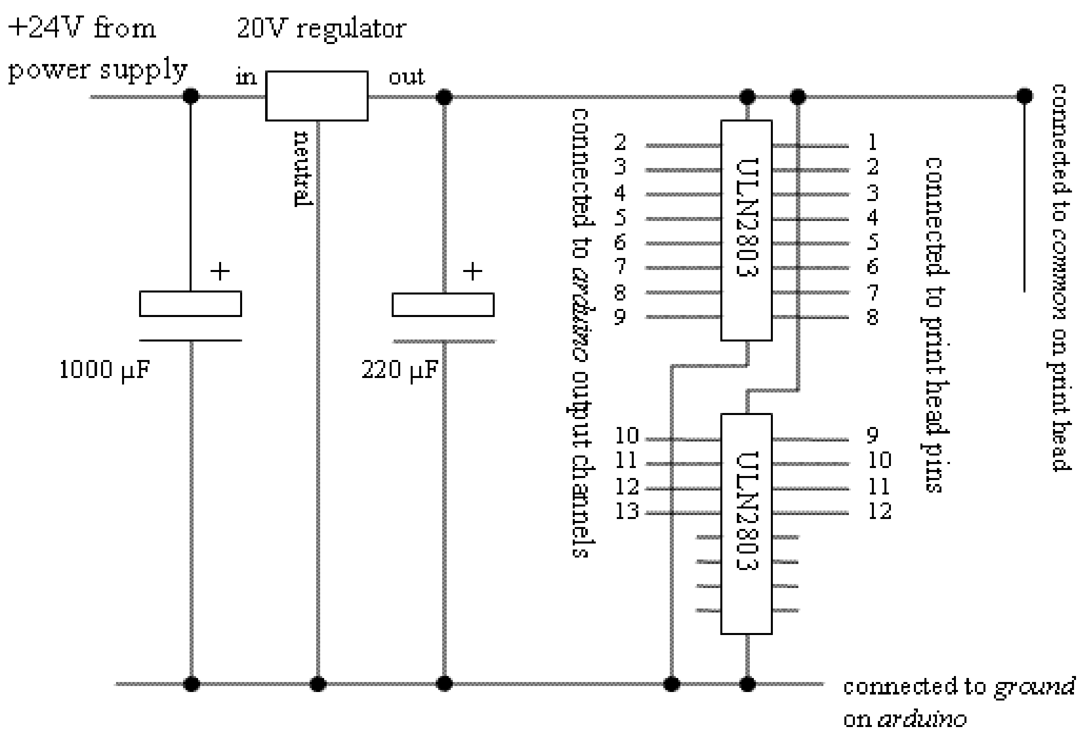

+24V from power supply
20V regulator
in
out
neutral
1000 µF
220 µF
connected to arduino output channels
2 3 4 5 6 7 8 9
10 11 12 13
ULN2803
ULN2803
1 2 3 4 5 6 7 8
9 10 11 12
connected to print head pins
connected to common on print head
connected to ground on arduino

Figure 14: The circuit used to control the printer

This circuit is powered by a 24 volt DC power supply (which powered by mains electricity), which is connected to a 20 volt regulator. The 20 volt output is connected to the common on the print head. The neutral pin of the regulator is connected to ground rail, which is also connected to the ground pin on the arduino. Either side of the regulator, polarised capacitors protect the transistor arrays from sudden changes in voltage.

Two transistor arrays also run ‘between’ the two rails. The transistor arrays are ULN2803 chips; eight Darlington arrays. They are used to allow the small change in voltage created by the arduino board when it varies the state of an output pin to control the output of a signal of a much greater voltage. When the channel needs to be off, the chip connects the output of that particular channel to the +20V common; when the channel needs to be on, the chip connects the output to the ground.

(the pins are numbered differently because the arduino website suggests that the digital 0 and 1 pins should not be used unless entirely necessary, as they are used for serial communication. Therefore the connections from the arduino go from 2 to 13, while the nozzles are numbered 1 to 12).

Due to issues with the microcontroller (discussed in the **whole design** section), the circuit was changed with the addition of pull down resistors (1kΩ) between the inputs from the *arduino* board and the *ground*. This is manly to prevent the signals from the arduino from 'floating'; which is where the output doesn't have a defined state and therefore flickers randomly between the states (*high* and *low*), resulting in random, undesired signals.

Most of the circuit was unchanged; this diagram shows where the resistors were added:

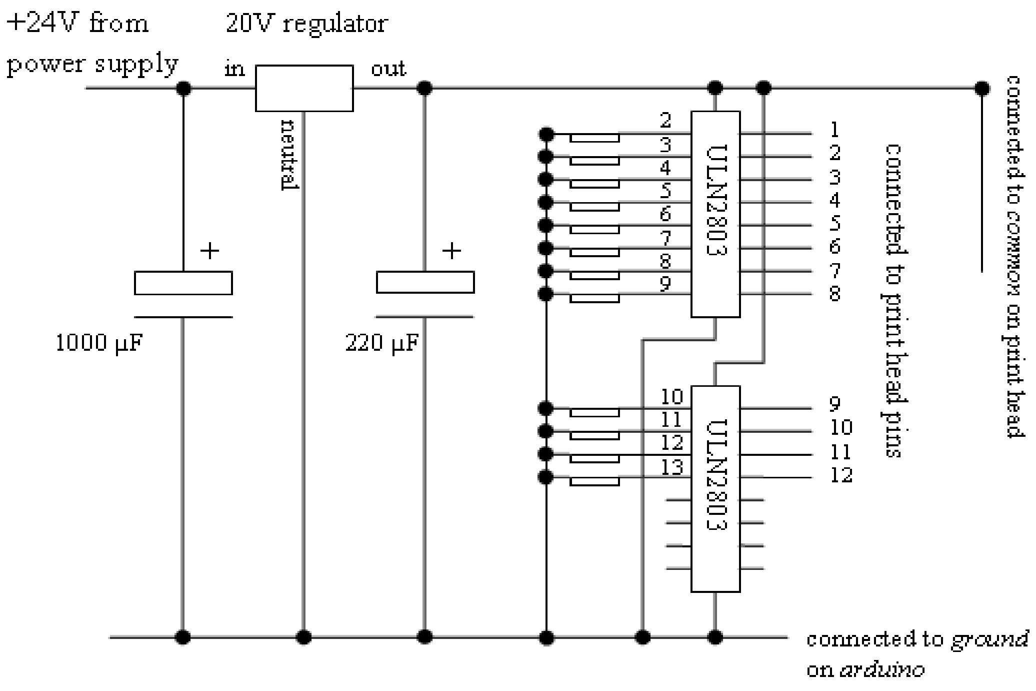


Figure 15: The modified circuit used to drive the printer

Another addition made to the circuit was the addition of a push button controller:

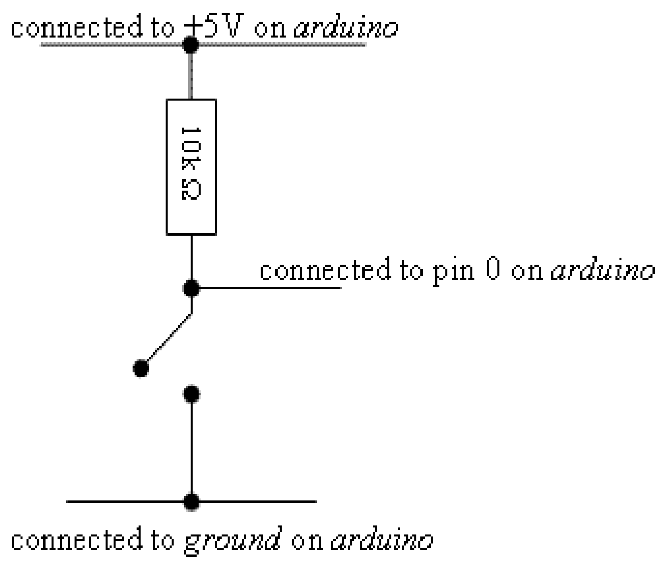

connected to +5V on *arduino*
10k Ω
connected to pin 0 on *arduino*
connected to *ground* on *arduino*

Figure 16: A diagram of the pushbutton controller

The way that this circuit works is that the resistor acts as a ‘pull-up resistor’; when the switch is open (like in the illustration), the arduino reads ‘*high’* at pin 0, and when the switch is closed, the much greater resistance between the output to the arduino and the 5 volt rail compared to the resistance between the output and ground means that the output almost goes to ground, and arduino reads a ‘*low’* at pin 0. Using this information, a program can be written to control the printer using the pushbutton.

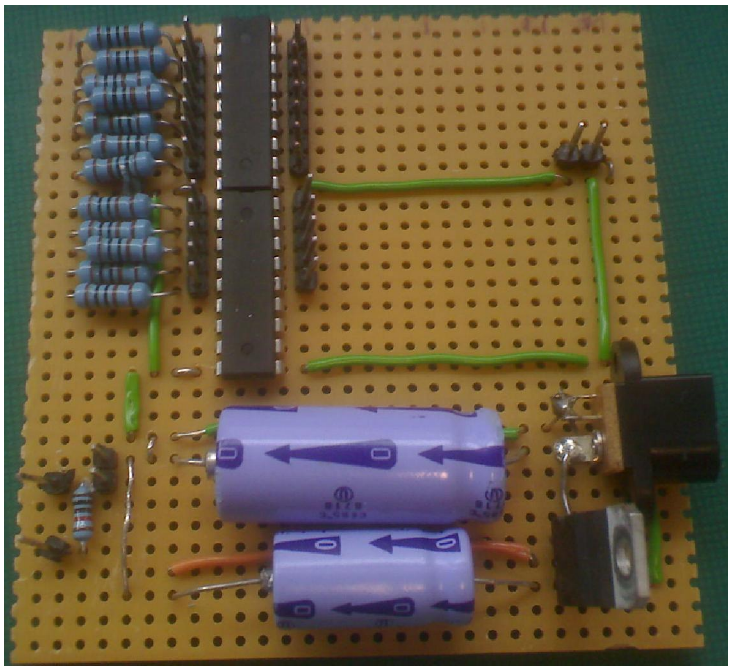


Figure 17: The final circuit

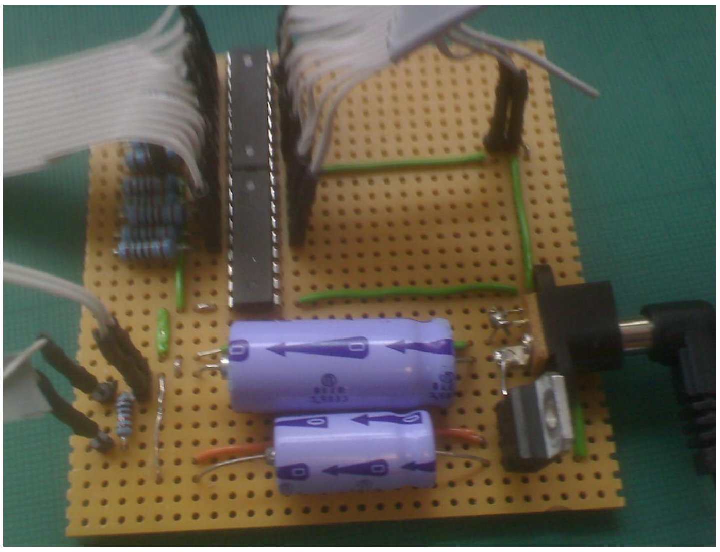


Figure 18: The final circuit with all the wires

Software

The *arduino* website contains a lot of examples of *sketches* (files of code) that describe how to complete certain tasks with the *arduino*. Along with aiding understanding of the code, some of these examples are particularly useful for this particular project. In particular, the '*Blink'* program describes how to make an LED turn on and off at given intervals[6].

```
/* Blinking LED
 * ------------
 *
 * turns on and off a light emitting diode(LED) connected to a digital
 * pin, in intervals of 2 seconds. Ideally we use pin 13 on the Arduino
 * board because it has a resistor attached to it, needing only an LED

 *
 * Created 1 June 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 * based on an orginal by H. Barragan for the Wiring i/o board
 */

int ledPin = 13;                 // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

There is a modification of this program that is also very useful; the '*Loop'* sketch[6]. This sketch flashes a series of LEDs one after the other (it turns an LED on and then off, then turns the next LED on and off, and so on; rather than turning them on one after the other, and then off one after the other), and then flashes them again in the reverse order.

```
int timer = 100;                   // The higher the number,
                                   //the slower the timing.
int pins[] = { 2, 3, 4, 5, 6, 7 }; // an array of pin numbers
int num_pins = 6;                  // the number of pins
                                   //(i.e. the length of the array)

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   //from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}

void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);   // turning it on,
    delay(timer);                  // pausing,
    digitalWrite(pins[i], LOW);    // and turning it off.
  }
  for (i = num_pins - 1; i >= 0; i--) {
    digitalWrite(pins[i], HIGH);
    delay(timer);
    digitalWrite(pins[i], LOW);
  }
}
```

This sketch is written for a circuit with six LEDs, so it can be easily modified for the initial circuit which is described in the **Electronics** section of this report, which has five LEDs. It is useful for this project to have the LEDs just flash sequentially, and pause before flashing again in the same sequence (rather than flashing in the opposite sequence, as the '*loop*' sketch does). To change the sketch so the LEDs only flash 'one way', a small amount of the code simply needs to be deleted.

It is also possible to modify the sketch so that it's not only the time that the LEDs are on for that is controlled, but also the time between two LEDs coming on and the time to pause between each loop (the pause between turning off the last LED and turning the first one on again). In this example the LEDs flash on for 60 ms, with a 10 ms pause between consecutive LEDs flashing, and a 500 ms wait at after flashing the last LED and flashing the first LED again.

```
int pulsetimer =  60;        // The pulse time (in milliseconds)
int pins[] = { 2, 3, 4, 5, 6 }; // an array of pin numbers
int num_pins = 5;               // the number of pins
                                // (i.e. the length of the array)
int deadtimer = 10;              // the dead time between flashes
int waittimer = 500;           // the wait time between the
                                // last and first flashes

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}

void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) {   // loop through each pin...
    digitalWrite(pins[i], HIGH);     // turning it on,
    delay(pulsetimer);               // flash time
    digitalWrite(pins[i], LOW);      // and turning it off.
    delay(deadtimer);                // dead time between LEDs
  }
  delay(waittimer);                // waiting after firing a full set

}
```

The sketch can now be changed fairly easily to work for the printer. Since the pulse required through the resistors is short, the previous sketch can be modified so that the desired times for three variables (the time of the pulse, the dead time between pins, and the pause between the last pin firing and the first firing again) is used instead. Initially, only one side of the printer is being used, so there are only five nozzles to worry about (the other seven are on the other side, since the side to be worked on has to have the two common connectors on it). Based on the information researched (as described in the Arduino section of this report), the pulse time will be set to 6 µs, the dead time to 1 µs, and the pause time to 765 µs. This means that the code will have to be altered so that the microsecond delay function is used instead of the regular delay function (which can't support non-integer values, so the pulse time has been rounded down and the dead time rounded up, erring on the side of caution).

```
/* A sumple program to generate a series of pulses that can fire 5
nozzles on a hp 51604 printer with a 21V DC supply. */

int pulsetimer =  6;              // The pulse time (in milliseconds)
int pins[] = { 2, 3, 4, 5, 6 };   // an array of pin numbers
int num_pins = 5;                 // the number of pins
                                  // (i.e. the length of the array)
int deadtimer = 1;                // the dead time between pulses
int waittimer = 765;         /* the wait time between the last and first
                                pulses, calculated by taking away the time
                                for all the nozzles to fire and the dead
                                time between them taken away from the
                                minimum pulse spacing per nozzle*/

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)  // the array elements are numbered
                                  // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);     // set each pin as an output
}

void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) {  // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);  // firing time
    digitalWrite(pins[i], LOW);     // and turning it off.
    delayMicroseconds(deadtimer);   // dead time between pins
  }
  delayMicroseconds(waittimer);     // waiting after firing a full set

}
```

Upon inspecting the output from the *arduino* using an oscilloscope, it was found that the pulse and dead times are significantly greater than programmed. This is probably due to the fact that the times being used are so short, that the time the microcontroller takes to process each function starts having an effect. After more in-depth analysis of this phenomenon, it was found that the *arduino* takes 3-4 µs to switch a pin high and low immediately after (this was found by turning the pulse delay into a comment; using '//'). It was also found that microsecond delay function was fairly accurate, taking about as long as the sketch suggests it should (suggesting that there is little processing time involved in this function).

Therefore, the code was altered so that the pulse time was about 6.5 µs after adjustment; so the delay function only paused for 3 µs.

This sketch shows the code for making all 12 nozzles on the printer fire consecutively.

```
/* A sumple program to generate a series of pulses that can fire 12
nozzles on a hp 51604 printer with a 21V DC supply. */

int pulsetimer =  3;       // The pulse time (in milliseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
                           // an array of pin numbers
int num_pins = 12;         // the number of pins (i.e. the length of
                           // the array)
int deadtimer = 1;         // the dead time between pulses
int waittimer = 1500;      /* the wait time between the last and first
                              pulses, calculated by taking away the time
                              for all the nozzles to fire and the dead
                              time between them taken away from the
                              minimum pulse spacing per nozzle*/

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)    // the array elements are numbered
                                    // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);       // set each pin as an output
}

void loop()
{
  int i;

  for (i = 0; i < num_pins; i++) {    // loop through each pin...
    digitalWrite(pins[i], HIGH);      // turning it on,
    delayMicroseconds(pulsetimer);    // firing time
    digitalWrite(pins[i], LOW);       // and turning it off.
    delayMicroseconds(deadtimer);     // dead time between pins
  }
  delayMicroseconds(waittimer);       // waiting after firing a full set

}
```

This sketch was used as the primary test for all print cartridges used in this project. It is useful because it is possible to see how many of the cartridges are working, and is based on plenty of fires by each cartridge, rather than just one or a few fires, where a misfire can be extremely misleading. The main reason for using this sketch, however, is because it is quick to show the performance of the print cartridge.

On an aside, an attempt was made to alter this sketch so that the board would pulse two resistors simultaneously, but that did not work (although it is not essential to be able to do this anyway). It is also worth noting that this sketch does not stop, so to stop the printer, the power must be turned off to either the circuit or the arduino board (both are acceptable and wont damage the circuitry).

The previous sketch can also be modified to just fire each nozzle a set amount of times between resets. Here is an example; with all the nozzles firing 4 times (the first nozzle fires 3 times, then the 2nd fires thrice, and so on):

```
/* A sumple program to generate a series of pulses that can fire 12
nozzles on a hp 51604 printer with a 21V DC supply. */

int pulsetimer =  3;       // The pulse time (in milliseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
                           // an array of pin numbers
int num_pins = 12;         // the number of pins (i.e. the length of
                           // the array)
int deadtimer = 1;         // the dead time between pulses
int waittimer = 1500;      /* the wait time between the last and first
                              pulses, calculated by taking away the time
                              for all the nozzles to fire and the dead
                              time between them taken away from the
                              minimum pulse spacing per nozzle*/

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
```

```
        for (i = 0; i < num_pins; i++) {      // loop through each pin...
            digitalWrite(pins[i], HIGH);      // turning it on,
            delayMicroseconds(pulsetimer);    // firing time
            digitalWrite(pins[i], LOW);       // and turning it off.
            delayMicroseconds(waittimer);     // waiting after firing
            digitalWrite(pins[i], HIGH);      // turning it on,
            delayMicroseconds(pulsetimer);    // firing time
            digitalWrite(pins[i], LOW);       // and turning it off.
            delayMicroseconds(waittimer);     // waiting after firing
            digitalWrite(pins[i], HIGH);      // turning it on,
            delayMicroseconds(pulsetimer);    // firing time
            digitalWrite(pins[i], LOW);       // and turning it off.
            delayMicroseconds(waittimer);     // waiting after firing
            digitalWrite(pins[i], HIGH);      // turning it on,
            delayMicroseconds(pulsetimer);    // firing time
            digitalWrite(pins[i], LOW);       // and turning it off.
            delayMicroseconds(waittimer);     // waiting after firing
            delayMicroseconds(deadtimer);     // dead time between pins
          }
      }

      void loop()
      {
      }
```

An extremely similar program can be used to just fire one nozzle at a time, to test which nozzles are blocked up (or don’t work, although that can be tested by measuring the resistance from each pin to common wire; so only the nozzles with the correct resistance need to be tested with this).

```
/* A simple program to generate a series of pulses that can fire 12
nozzles on a hp 51604 printer with a 21V DC supply. */

int pulsetimer =  3;       // The pulse time (in milliseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
                           // an array of pin numbers
int num_pins = 12;         // the number of pins (i.e. the length of
                           // the array)
int deadtimer = 1;         // the dead time between pulses
int waittimer = 1500;      /* the wait time between the last and first
                              pulses, calculated by taking away the time
                              for all the nozzles to fire and the dead
                              time between them taken away from the
                              minimum pulse spacing per nozzle*/

void setup()
{
  int i = 0;            //extablishes the pin to be used. can be 0-11

    pinMode(pins[i], OUTPUT);      // set the pin as an output

      digitalWrite(pins[i], HIGH);     // turning it on,
      delayMicroseconds(pulsetimer);   // firing time
      digitalWrite(pins[i], LOW);      // and turning it off.
      delayMicroseconds(waittimer);    // waiting after firing
      digitalWrite(pins[i], HIGH);     // turning it on,
      delayMicroseconds(pulsetimer);   // firing time
      digitalWrite(pins[i], LOW);      // and turning it off.
      delayMicroseconds(waittimer);    // waiting after firing
      digitalWrite(pins[i], HIGH);     // turning it on,
      delayMicroseconds(pulsetimer);   // firing time
      digitalWrite(pins[i], LOW);      // and turning it off.
      delayMicroseconds(waittimer);    // waiting after firing
      digitalWrite(pins[i], HIGH);     // turning it on,
      delayMicroseconds(pulsetimer);   // firing time
      digitalWrite(pins[i], LOW);      // and turning it off.
      delayMicroseconds(waittimer);    // waiting after firing
}

void loop()
{
}
```

A ‘fun’ sketch can be made to actually write text, although it is not particularly useful, as the sketch is only useful for writing a specific word/phrase, and it would take a while to write anything that can print something really meaningful. This sketch writes the word ‘Hello’:

```
/* A simple program to generate a series of pulses that can print the word
'hello', with a hp 51604 printer with a 21V DC supply. */

int pulsetimer =  3;        // The pulse time (in microseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 }; // an array of pin numbers
int num_pins = 12;              // the number of pins (i.e. the length of the array)
int deadtimer = 1;          // the dead time between pulses
int letterwait = 4000;      // the wait between letters
int waittimer = 2000;       /* the wait time between the last and first pulses,
                               calculated by taking away the time for all the
                               nozzles to fire and the dead time between them
                               taken away from the minimum pulse spacing per nozzle
                            */

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output

//'H'
  //|
  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //-
    digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
   delayMicroseconds(waittimer);         // time waiting after firing a full set
    digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
   delayMicroseconds(waittimer);         // time waiting after firing a full set
    digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
   delayMicroseconds(waittimer);         // time waiting after firing a full set
  //|
  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set

//wait between letters
delayMicroseconds(letterwait);

//'E'
  //|
for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //doing the = bits
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
     digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
    digitalWrite(pins[0], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[0], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
     digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
    digitalWrite(pins[0], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[0], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
     digitalWrite(pins[5], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[5], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
    digitalWrite(pins[0], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[0], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set

//wait between letters
delayMicroseconds(letterwait);
```

```
//'L'
  //|
  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);               // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //_
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set

//wait between letters
delayMicroseconds(letterwait);

//'L'
  //|
  for (i = 0; i < num_pins; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);               // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //_
```

```
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set

//wait between letters
delayMicroseconds(letterwait);

//'O'
  //short |
  for (i = 1; i < 6; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);               // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //=
    digitalWrite(pins[0], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[0], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);               // dead time between pins
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
    digitalWrite(pins[0], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[0], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);               // dead time between pins
    digitalWrite(pins[11], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);              // firing time
    digitalWrite(pins[11], LOW);   // and turning it off.
  delayMicroseconds(waittimer);          // time waiting after firing a full set
  //short |
  for (i = 1; i < 6; i++) { // loop through each pin...
    digitalWrite(pins[i], HIGH);    // turning it on,
    delayMicroseconds(pulsetimer);             // firing time
    digitalWrite(pins[i], LOW);   // and turning it off.
    delayMicroseconds(deadtimer);              // dead time between pins
  }
  delayMicroseconds(waittimer);          // time waiting after firing a full set
}

void loop()
{

}
```

The arduino website also contains some useful sketches regarding the use of a pushbutton as a controller. There are two ways of doing this; the first (which is simpler) is to turn the output on when the button is pushed, and off when it is released, and the second is where pushing (and releasing) the button will change the state of the output (so if it was off, it is turned on, and turned on if it was off).

The program for turning the output on when the button is pushed and off when it is released is described in the sample sketch ‘Button’. The code for it is:

```
int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin);  // read input value
  if (val == HIGH) {         // check if the input is HIGH
                             // (button released)
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH);  // turn LED ON
  }
}
```

This can be modified so it makes all the nozzles fire sequentially in a loop by combining this sketch with the sketch that continually fires all the nozzles in the '*loop*' function:

```
/* A simple program to generate a series of pulses that can fire the
nozzles on a hp 51604 printer with a 21V DC supply and push button. */

int inPin = 0;
int pulsetimer =  3;        // The pulse time (in microseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
                            // an array of pin numbers
int num_pins = 12;          // the number of pins (i.e. the length of the array)
int val = 0;
int deadtimer = 1;          // the dead time between pulses
int waittimer = 800;       /* the wait time between the last and first pulses,
                              calculated by taking away the time for all the
                              nozzles to fire and the dead time between them
                              taken away from the minimum pulse spacing per nozzle
                            */
void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}

void loop()
{
  int i;
  val = digitalRead(inPin);
  if (val == LOW)  {
    for (i = 0; i < num_pins; i++) {   // loop through each pin...
      digitalWrite(pins[i], HIGH);     // turning it on,
      delayMicroseconds(pulsetimer);   // firing time
      digitalWrite(pins[i], LOW);      // and turning it off.
      delayMicroseconds(deadtimer);    // dead time between pins
    }
    delayMicroseconds(waittimer);      // time waiting after firing a full set
  }
}
```

Using the previous function, the following print was produced:

Figure 18: Using a push button controller

The other method of pushbutton control (where pushing the button toggles whether the output is high or low) is shown in the sample sketch, '*Debounce*'. The sketch is named after the method used to prevent the output from flickering because we, as humans, can't change a buttons state from out to pushed-in instantaneously; which can confuse the microcontroller. The sketch for using a button to toggle an LED:

```
int inPin = 7;         // the number of the input pin
int outPin = 13;       // the number of the output pin

int state = HIGH;      // the current state of the output pin
int reading;           // the current reading from the input pin
int previous = LOW;    // the previous reading from the input pin

// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;         // the last time the output pin was toggled
long debounce = 200;   // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  pinMode(outPin, OUTPUT);
}

void loop()
{
  reading = digitalRead(inPin);

  // if we just pressed the button (i.e. the input went from LOW to HIGH),
  // and we've waited long enough since the last press to ignore any noise...
  if (reading == HIGH && previous == LOW && millis() - time > debounce) {
    // ... invert the output
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;

    // ... and remember when the last button press was
    time = millis();
  }

  digitalWrite(outPin, state);

  previous = reading;
}
```

This can be modified to include the loop which causes the all nozzles to fire, so that the button toggles the printing:

```
/* A program to generate a series of pulses that can fire the
nozzles on a hp 51604 printer with a 21V DC supply with a button. */

int inPin = 0;
int pulsetimer =  3;     // The pulse time (in microseconds)
int pins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
                         // an array of pin numbers
int num_pins = 12;       // the number of pins (i.e. the length of the array)
int deadtimer = 1;       // the dead time between pulses
int waittimer = 800;     /* the wait time between the last and first pulses,
                            calculated by taking away the time for all the
                            nozzles to fire and the dead time between them
                            taken away from the minimum pulse spacing per nozzle
                          */
int state = HIGH;
int reading;
int previous = LOW;

long time = 0;
long debounce = 200;

void setup()
{
  int i;

  for (i = 0; i < num_pins; i++)   // the array elements are numbered
                                   // from 0 to num_pins - 1
    pinMode(pins[i], OUTPUT);      // set each pin as an output
}

void loop()
{
  int i;
  reading = digitalRead(inPin);
  if (reading == HIGH && previous == LOW && millis() - time > debounce){
    if (state == HIGH)
      state = LOW;
    else
      state = HIGH;
      time = millis();
  }
  previous = reading;

 if (state == LOW)  {
   for (i = 0; i < num_pins; i++) { // loop through each pin...
     digitalWrite(pins[i], HIGH);    // turning it on,
     delayMicroseconds(pulsetimer);             // firing time
     digitalWrite(pins[i], LOW);   // and turning it off.
     delayMicroseconds(deadtimer);   // dead time between pins
   }
   delayMicroseconds(waittimer);     // time waiting after firing a full set
  }
}
```

Mechanical design

The aim of the mechanical design is to create good electrical contact between the contacts on the printer and the connecters used to send a signal to the printing resistors. The mechanical design must also allow the ink to actually fire onto the print medium unobstructed, and the distance between the print head on the cartridge and print medium should be within the given constraints. The print cartridge holder should also be manufacturable using a fused deposition modelling (FDM) rapid prototyper; ideally within the constraints of the current iteration of RepRap, although it is acceptable to manufacture it to the standard of the Strat which is what future iterations (or perhaps *generations*) of RepRap should be capable of.

The brief of this project also suggests that this is project is to design the part so that the printer works, rather than reaching the best possible solution (suggesting the mechanical part will probably be redesigned as the RepRap develops).

Because the main driver for the design was making reliable contact, some aspects of the design were set extremely quickly. The solution appeared to tend towards a design with two plates connected by screws; the top plate too be located by a tapered hole that fit on to the tapered cylinder on the top of the cartridge, and the bottom plate located by the locating. The bottom section is designed to contain the connectors, while the top section helps locate the cartridge relative to the bottom section (ensuring that the locating pins do not come out of their holes) and providing a force to push the face of the cartridge with the contacts on it against the bottom section and the connectors, ensuring a good contact.

An initial idea was to bend thin flat pieces of metal up, with the flat section inserted into slots (including a top section that holds the flat section down) so that the tips of the bent section would make connections with the contacts on the print cartridge.

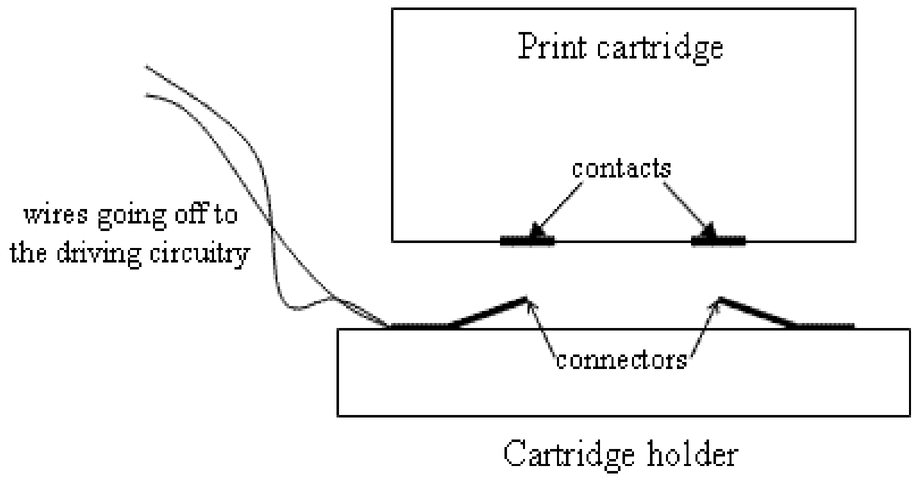


Figure 19: The principle behind the first design

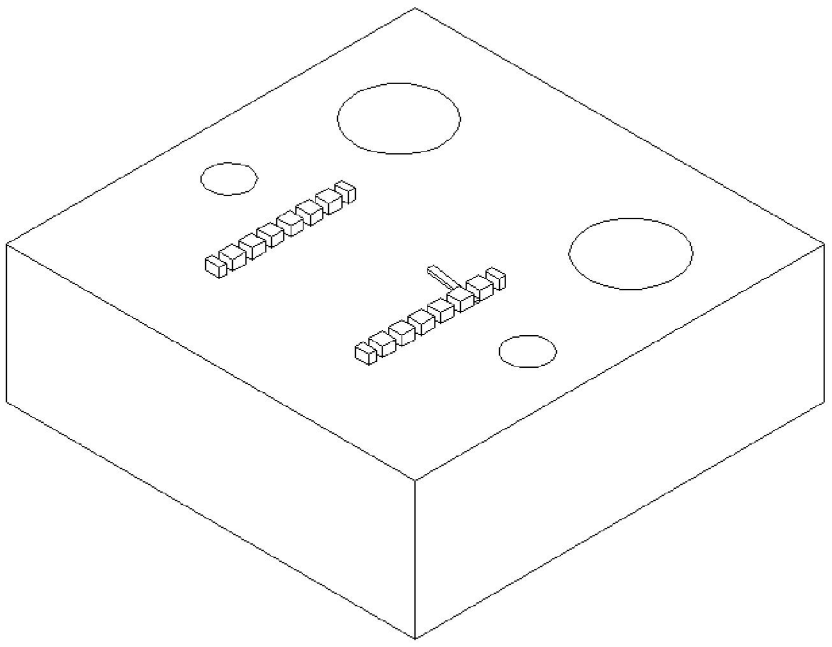


Figure 20: A CAD model of the design

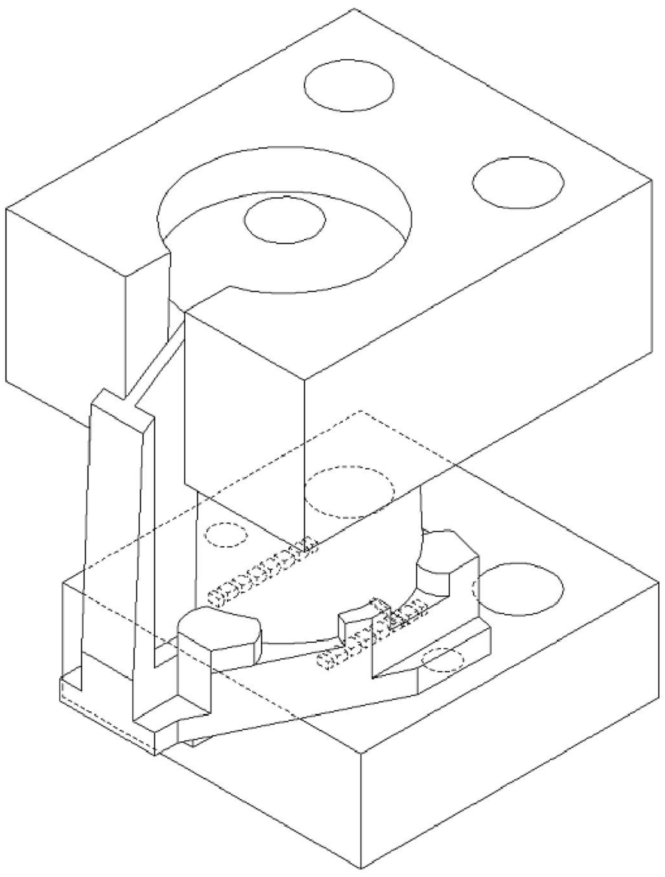
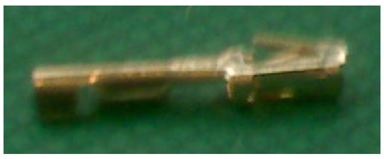


Figure 21: A CAD model of the assembly

This design is likely to be a very reliable method of making the connections, and would allow the cartridge holder to be very thin, as the slots holding the connectors could be inserted into the sunken portion of the base.

Another advantage of this design is that using part of a standard circuit board pin socket provides a connector of almost ideal proportions and material properties. The dimensions of the part are very good and the fact that the material is usually used for a connecter means that it is likely to be very conductive; but the material is a little too malleable (but it is probably just stiff enough to be useful).



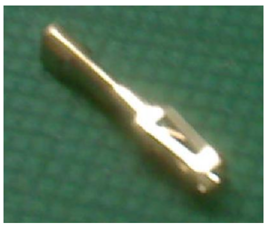


Figure 22 and **Figure 23:** A standard circuit board pin socket (with the connector section highlighted)

Unfortunately, upon examining the limitations of the rapid prototyper, it quickly becomes apparent that the rapid prototyper cannot make the slots, as the gap is too thin and the material on the sides would just merge. This means that the entire design is unviable, and the method of making the connection will have to be fundamentally altered.

Another idea was to use springy wire to make the contacts instead. Springy wire is good at staying largely straight, but is a little flexible (and malleable too, but less so than the circuit board pin socket).

The design would involve uncovered slots for the springy wire to sit in. To ensure an even better connection, the slots will be angled horizontally, so that only the very tip of the springy wire is in contact with the print cartridge (so that the connector does not get bent, by the force of the cartridge pushing down, in such a way that the tip which is meant to make the connection is forced away from the contact on the cartridge).

The ends of the springy wire will have to be connected to a regular ribbon cable, so as to get a reliable flexible connection between the connectors and the driving circuitry.

The springy wire is very thin, and the minimum slot thickness producible by the prototyper is significantly larger than the wires diameter. This means that slot will not be able to accurately direct the connecter, and something will need to be added to ensure the connectors are all directly under the contacts. This was done by cutting another slot, running perpendicular to the other slots , in which a piece of plastic can be inserted, in which thinner cuts can be added manually with a saw. This suggests that for the most reliable connection this slot would be as close to the contacts as possible, to reduce the distance in which the wire can curve.

There is also another issue regarding locating the connectors; they must be held still, and the previous constraint would only line up the connectors, not locate them horizontally. For this, an additional bit of plastic is needed to hold the wires down. This should also force the connector to follow the incline of the slot it's in, rather than being free to vary its angle and not make the connection as reliably.

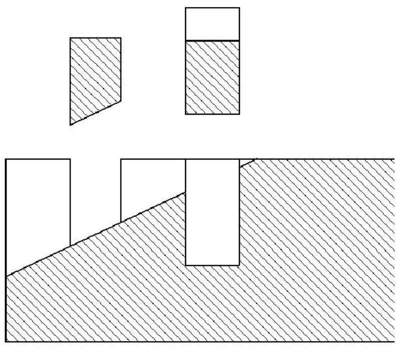


Figure 24: A diagram of how the mechanism would work

It would be significantly easier to manage a single piece that pushes down on all the connectors than lots of separate little pieces. Unfortunately, the holes for the cartridge locating pins get in the way of where the connector locating piece would go, so the piece must be split; giving 2 pieces instead.

Since the contacts on the bottom of the print cartridge are so close together (1.3 mm pitch; from start of contact to the start of the next one), the prototyper will not be able to build the slots with a thin wall of material between the slots of such a narrow thickness (either the walls or slots would fail). To solve this issue, the slots were designed so that they fork out as they move away from the contacts.

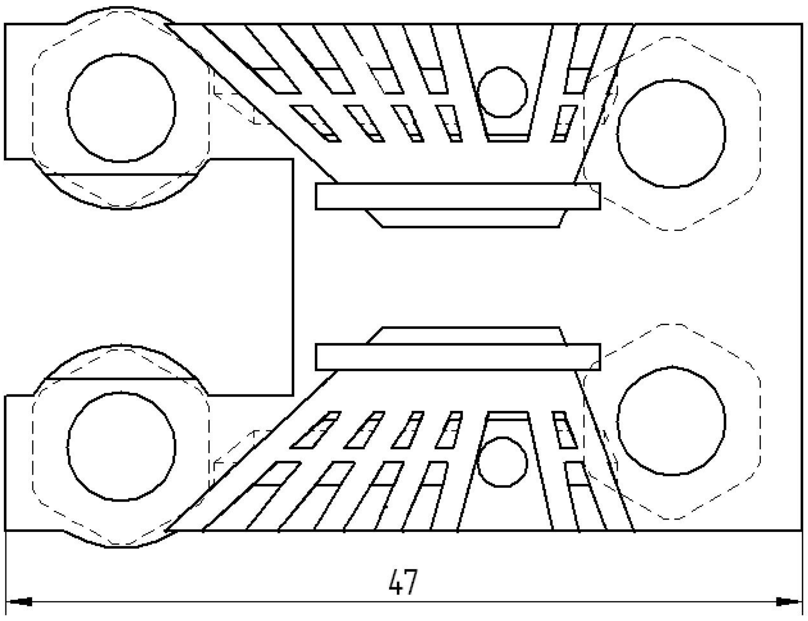
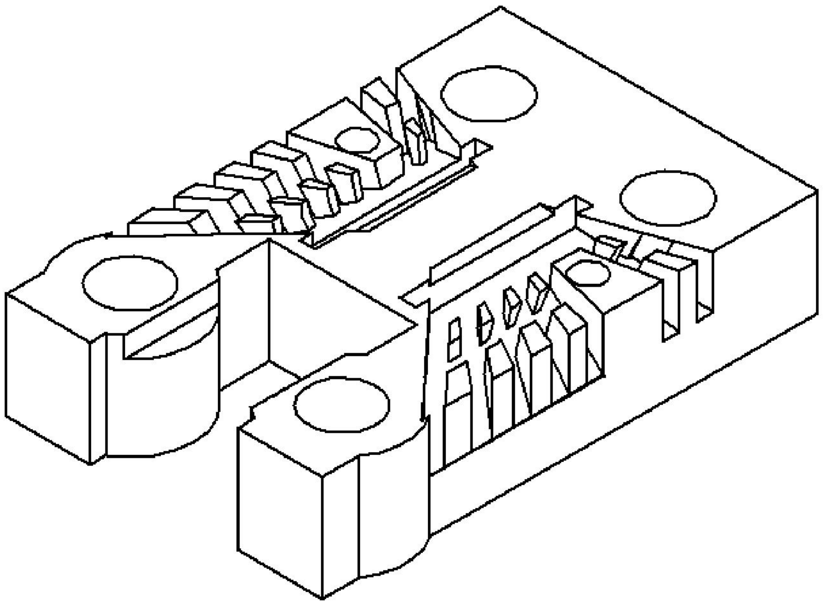


Figure 25: A CAD plan of the 2nd design

Figure 26: An isometric view of the 2nd design



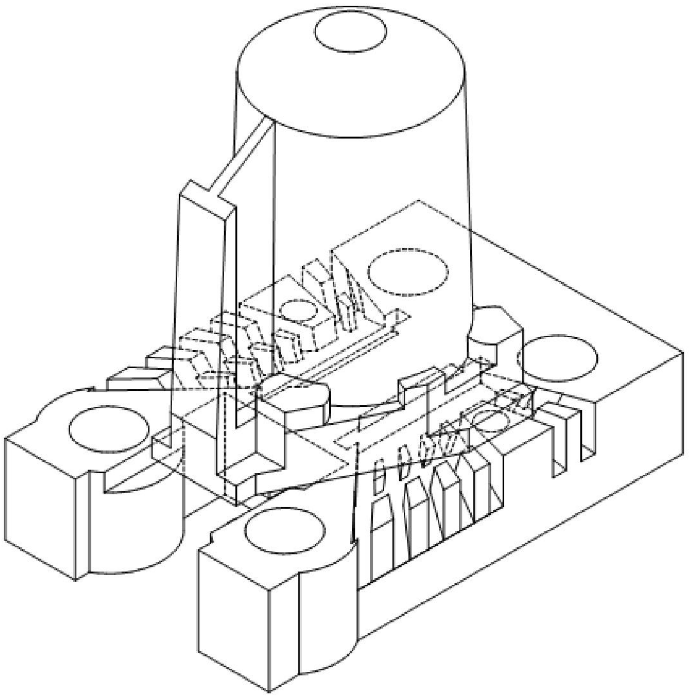
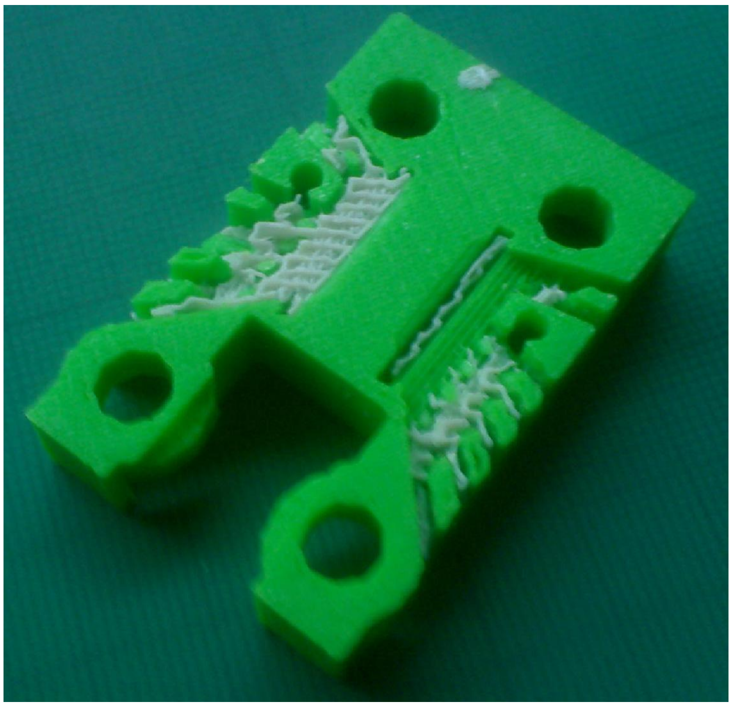


Figure 27: The cartridge holder with the print cartridge

Figure 28: A photograph of the cartridge holder, demonstrating some of the issues with this design

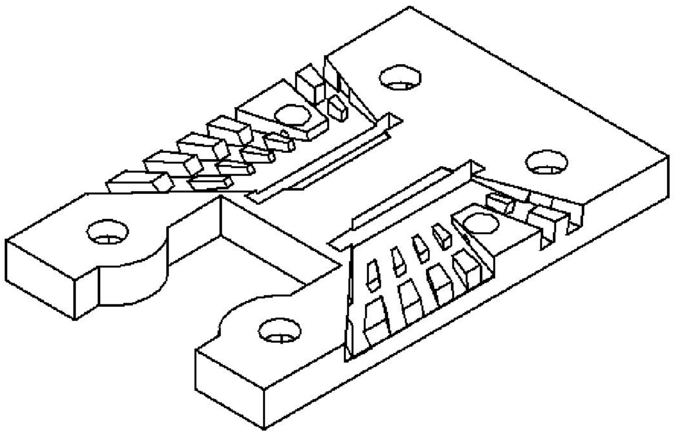


This idea seems potentially quite effective. Unfortunately, there were several issues with this particular base, but all fixable. The first issue is that the base is simply too thick.

To keep the distance between the print head and the print medium as small as possible, the nuts by the base are counter sunk into the base. In this design, the bolts used are M6, which need a 5mm counter sink plus material to support the nut (or head of the bolt), which meant the overall thickness of the base was about 10 mm. This is very thick, and the bolts were downsized to M3. The issue with M3 bolts is that the longest standard M3 bolt is not long enough to attach the bottom and top sections. Therefore, M3 threaded rod has to be cut to sections of the right size. This does, however, mean that the total thickness of the base can be reduced to 4mm, although it also means the connectors will be coming in at a shallower angle, which could affect the performance of the connectors.

Another problem, caused by human error is that the part was manufactured the wrong way up. This means that the rapid prototyper had to fill the slots with the support material to hold up the rest of the body. This is a serious issue, as the small size of the walls between the slots mean that it's easy to break the walls when removing the support material. Therefore, in the next iteration of this design the part will be built with the slots on top. This means that the support material will fill the countersunk portion of the nut holes at the bottom (which should not be a significant issue, as the walls around the nuts will get thicker once the bolt size has been reduced).

Figure 29: The revised cartridge holder



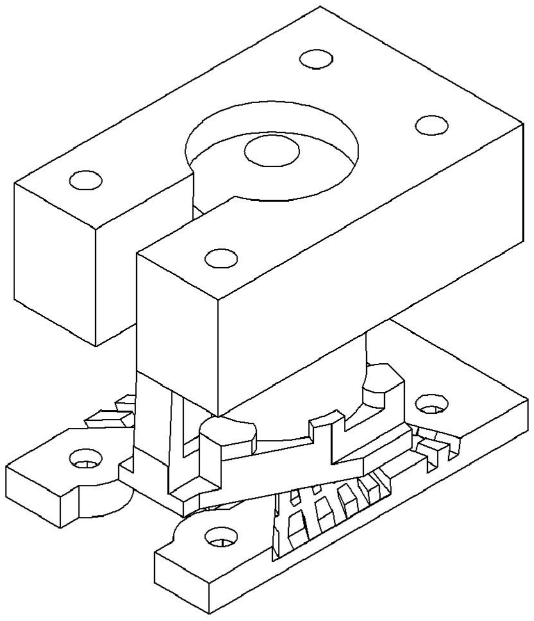


Figure 30: The cartridge assembly with the new holder

The new design was made as described previously. Although this particular iteration of this design is better than the previous one, it still has some major issues. The first issue is that the short section of the inter slot wall is still incredibly fragile, with most of the wall sections shearing off when the contacts were inserted.

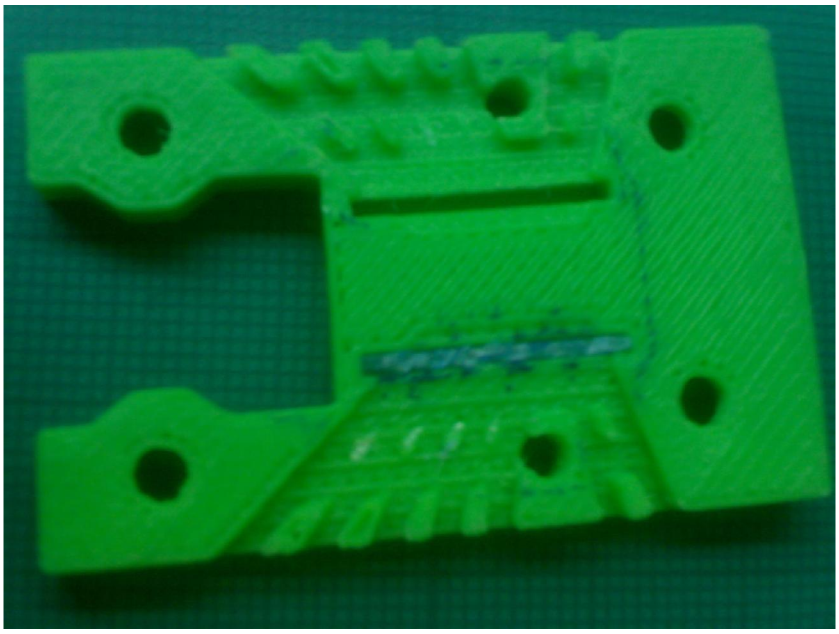
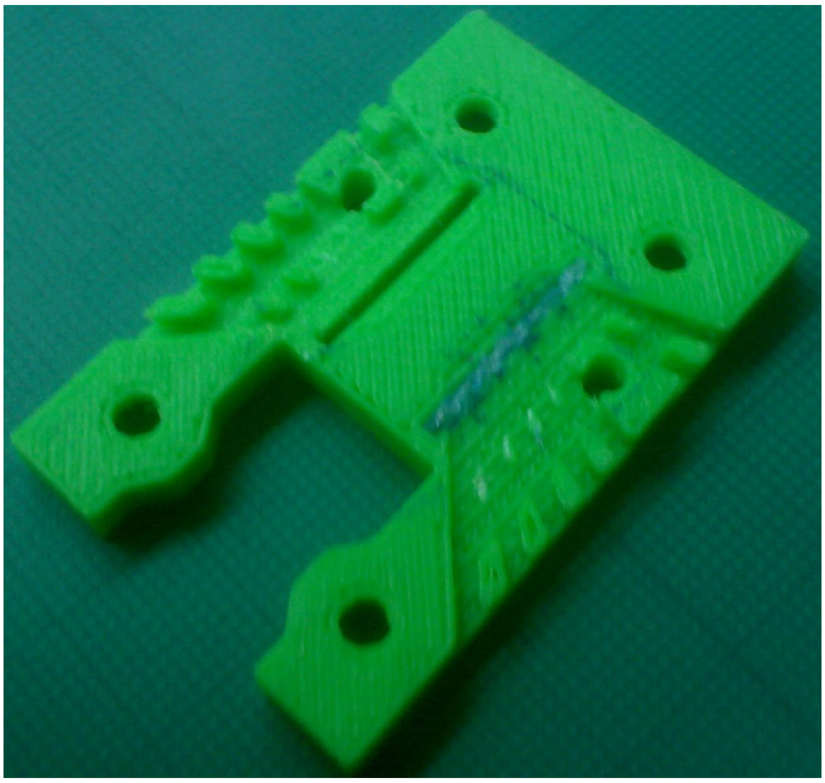


Figure 31: The 3rd design of the cartridge holder in plan view

Figure 32: Another view of the 3rd design. Notice the broken inter slot walls



Another issue was that the short pieces of plastic designed to prevent the connectors moving in and out of the slots were failing completely; there was simply not enough friction (the plastic in question here is acrylic). Fortunately a single solution for both these issues was found.

To prevent the connectors from moving around too much, the springy wire connectors were kept extremely short; approximately the length of the slots they would be placed in. At their ends they were soldered to wires which go together to make a ribbon cable. The connection was wrapped in heat to make it neater and to reduce the risk of stray connection and make everything neater. Fortunately, it was discovered that the diameter of the connection wrapped in (flexible) heat shrink is only slightly greater than width of the slots. This means that by inserting part of the heat shrunk connection into the slot provides an excellent means of locating the connectors. This also means that no pieces are needed to press down on the connectors, and therefore the wall can be made into a single section, which is significantly less likely to break.

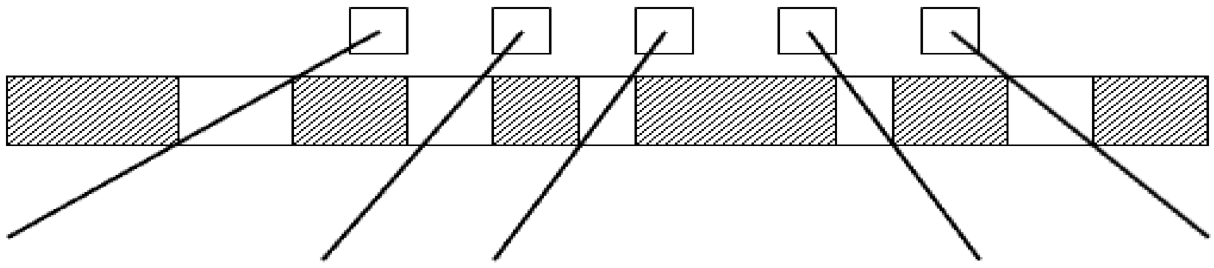
There is another major issue with this particular design, relating to the insert that locates the connectors just by the contacts on the cartridge. As mentioned earlier, this needs to be as close as reasonably possible. Unfortunately, it is too close, and the insert is hitting the lip on the cartridge that surrounds the contacts. As a result of this the protruding part of the slots cut into the locating insert is very short, making the insert less functional. It also means that the insert is extremely difficult to remove.

The other problem with the locating insert is the material used. Sawing this was extremely difficult, with the blade sliding and material chipping, giving extremely poorly defined slots.

To solve these issues, the slot for the insert will be moved further away from the contacts (so that the insert can rise into the sunk portion of the base of the cartridge), and the slot will be cut right through the base, giving more control over the amount that the locating insert protrudes out of the base by.

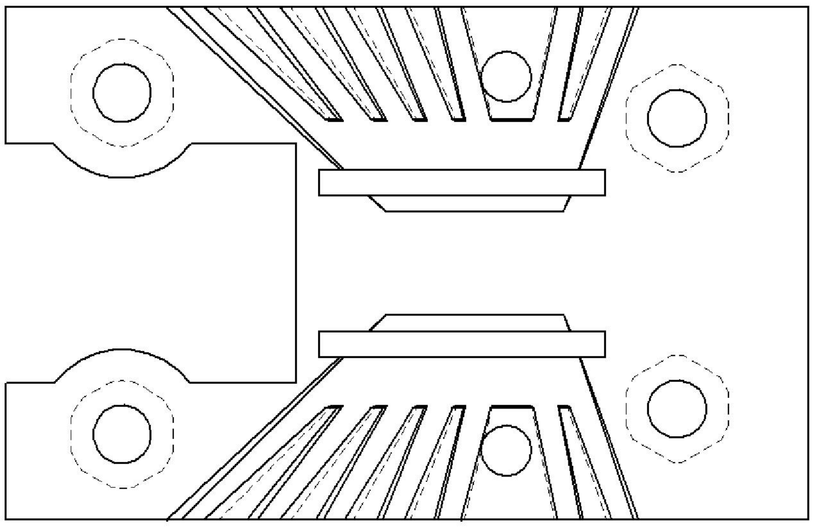
The locating insert will also be manufactured using the rapid prototyper. This once again raises the issue of the fact that the RP machine can’t make a slot thin enough for the required purposes. To work around this, the slots will all be perpendicular to the insert, with the width of each slot determining the angle of the connectors (and the location of the slots still determining the location).

Figure 33: A diagram demonstrating how the locating inserts would work



Although the part would be made most accurately by manufacturing it with the part that is on the bottom at the bottom, and the slots on the top, that would make the walls between the slots more likely to shear off (due to the weakness between laminate layers). Therefore, the insert must be manufactured on its side.

Figure 34: A CAD model of the newest design (4th)



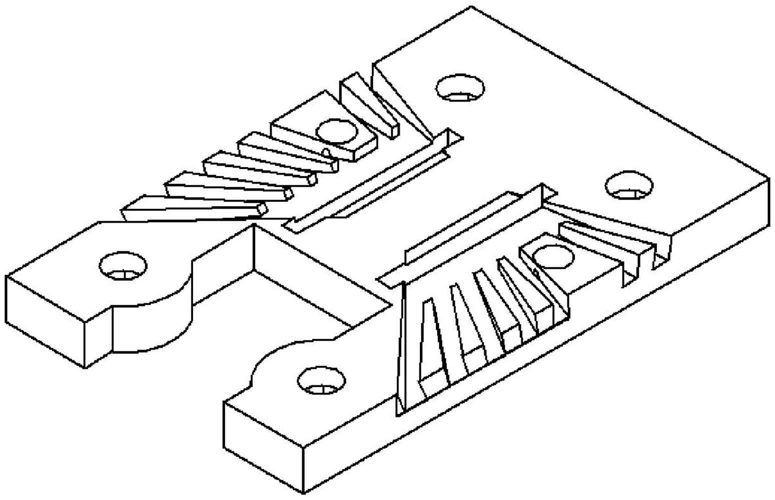
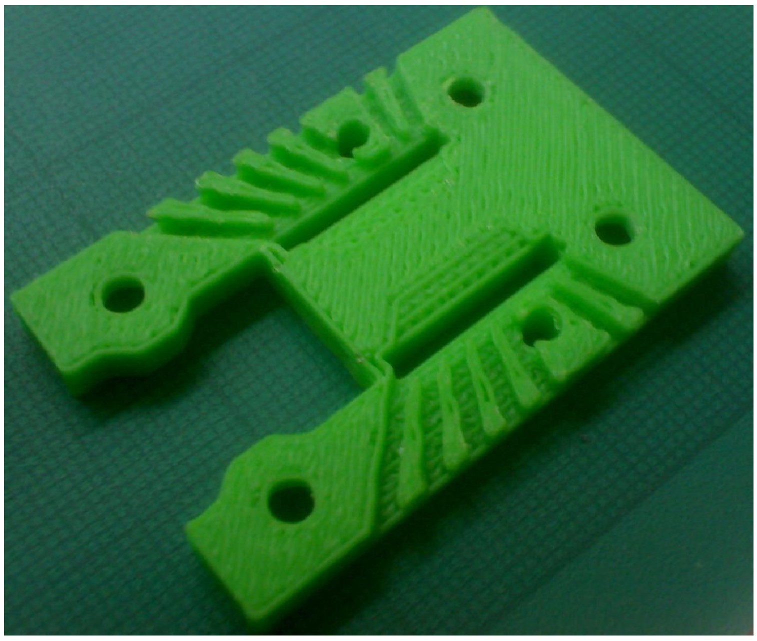


Figure 35: Another CAD image of the 4th design

Figure 36: A photograph of the 4th design base



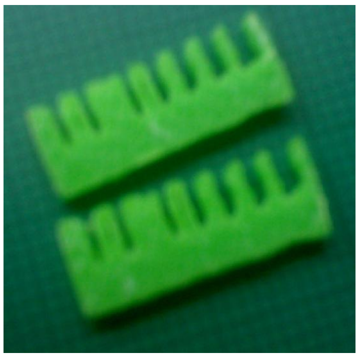


Figure 37: The locating inserts for the 4th design

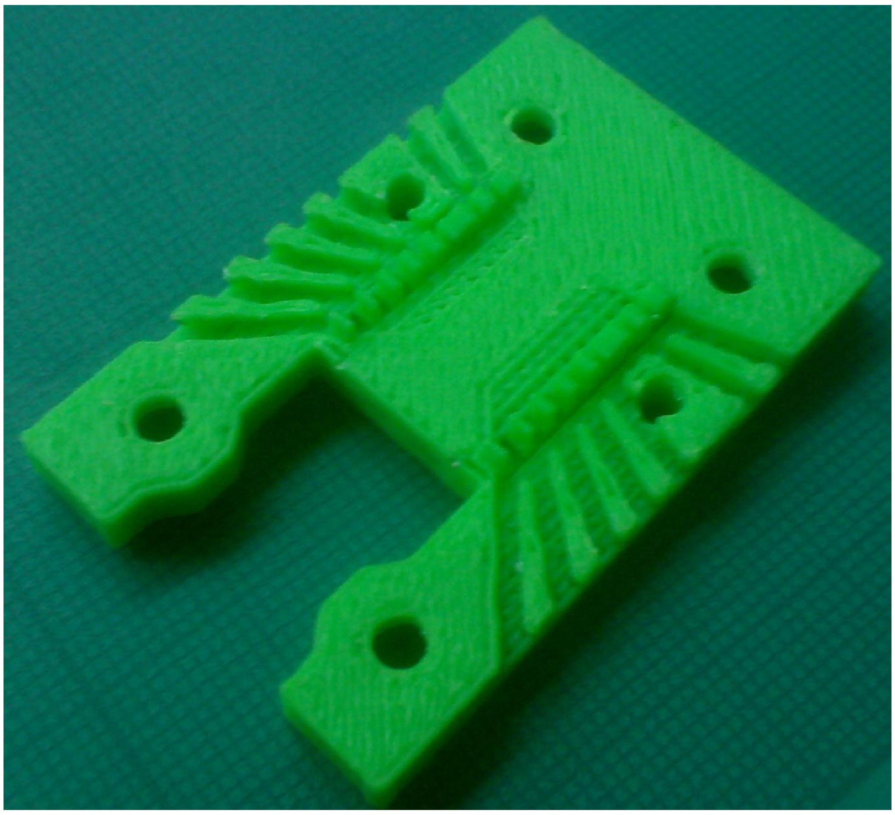


Figure 38: The 4th design; base and inserts

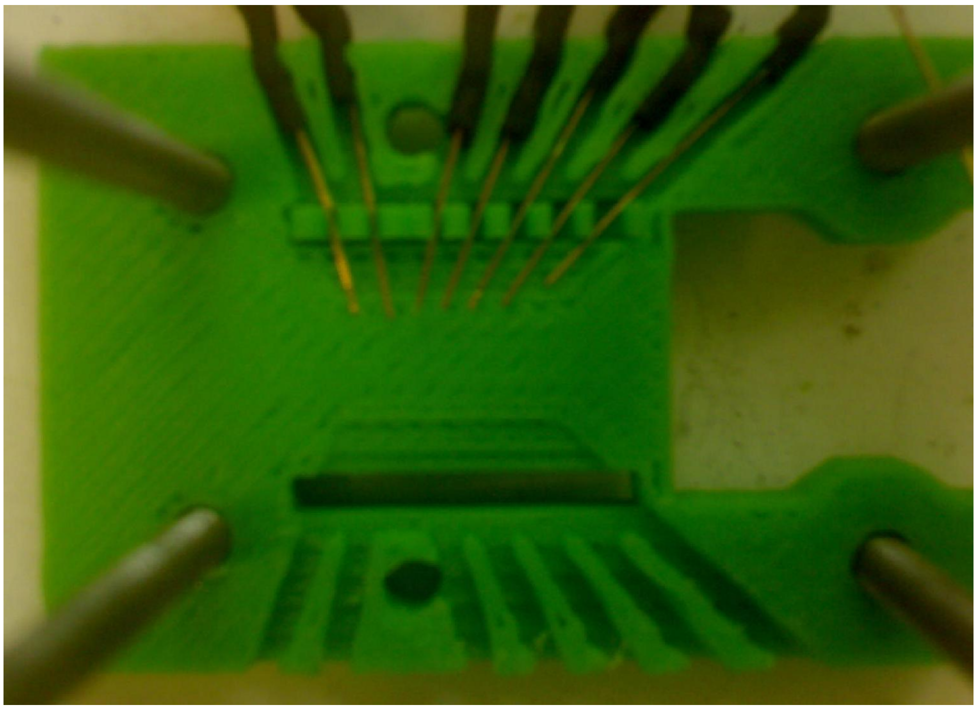


Figure 39: The 4th design with connectors

This iteration of the design was significantly more effective than the previous ones. Although not all the connections were consistently reliable, many were, and using this design it was possible to actually do some simple printing (just things like a thick line).

To test how many of the connections work, the resistance between the wire connected to the common and the wire connected to each contact can be measured, and comparing this figure to the known resistance of the firing resistor, 65 Ω, reveals whether or not the connection worked. (If the connection was bad, the resistance was typically infinite, and sometimes the connector to the contact nearest to the common would short against the common, giving a resistance reading of about 0.03 Ω).

Although this design is good, the connections need to be more reliable. To ensure that the connections had the best chance of being effective, the size and spacings of the contacts were re-measured (as they are extremely small, and proved particularly challenging to measure). It was discovered from this that although the contact dimensions were fairly accurate enough, the distance between the first contact and the cartridge locating pin was inaccurate. Although the inaccuracy was less than 0.5mm, it still has a significant effect given that the contacts are so small (the connectors were just making contact with the edges of the contacts, rather than sitting right in the middle of it).

Since a new bottom section will have to be made, a less essential modification was decided upon; to have a rectangular section immediately under where the contacts sit cut out. This will allow visualisation of how good the connections are, and which way the connectors need to be moved.

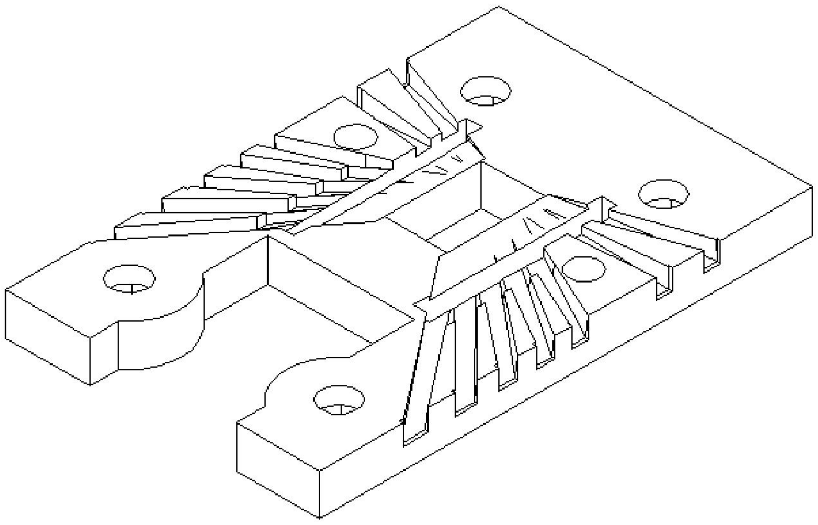


Figure 40: A CAD model of the 5th iteration of the design

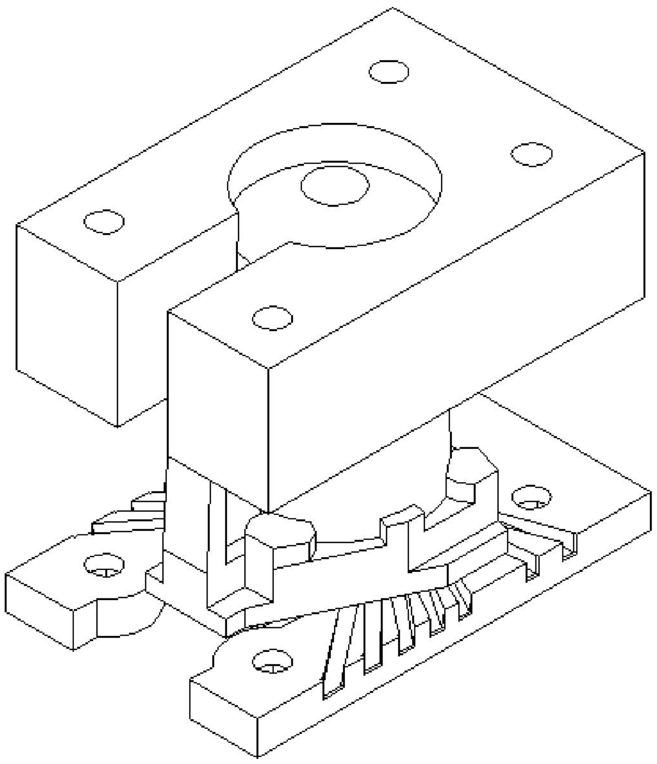


Figure 41: A CAD model of the assemble with the 5th bottom holder design

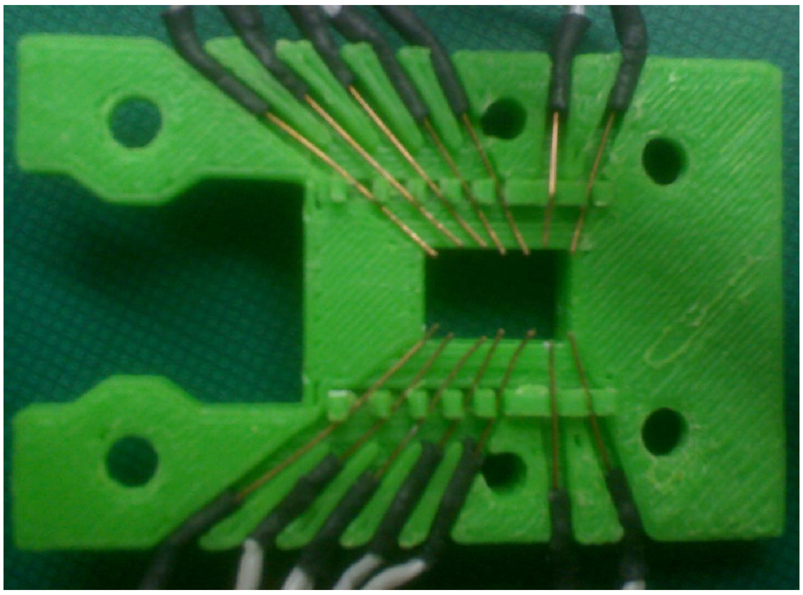


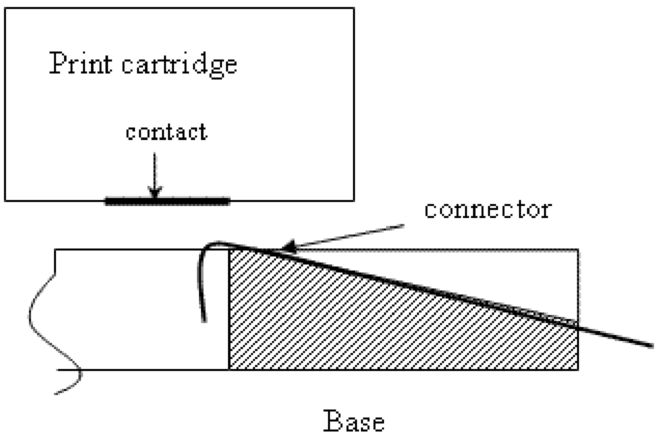
Figure 42: The 5th design with connectors

It may be worth noting that there was an issue while building this bottom section; the depth of the z axis on the Strat was not set properly, so the piece started to separate from the RP base before it was completed, and the part came out a little bent, but still looked useable.

As was expected, this design did have a severe issue due to the fact that the base was not pushing the connectors up directly under the contacts. This meant that although the new design meant that the connectors could easily be accurately located directly below the appropriate contacts, the connections were not made reliably, because the lack of force meant that there was a small vertical gap between the contacts and the connectors.

The first attempt to fix this was done by bending the tips of the connectors down and *away* from the contacts, in the hope that the flatter surface will cause the connectors to pushed up more by the base, and make better contact.

Figure 43: An attempt to improve the connection between the contacts and the connectors



This proved ineffective, as it was extremely difficult to bend the springy wires in the right angle and with the contacts (and therefore ends of the connectors) being so close together, the length of the bent portion must be incredibly small to prevent the wires from shorting against each other.

An alternative idea was to build a small cuboid of exactly the same proportions as the rectangular hole, and, once the connectors have been located and the user is satisfied with their position, the rectangular piece is pushed in to give good connections.

Unfortunately, this did not work either. An initial theory for this failure was that the force pushing the pins down from the ends of the connectors was strong enough to displace the piece, but the piece was an extremely tight fit into the whole, and inspecting the assembly after inserting the piece and then taking off the print cartridge reveals that the top of the piece is level with the top of the base. This suggests that the issues while manufacturing the cartridge holding base may be the root cause of this issue.

Rather than immediately making a new base, a potential alternative was attempted. The idea is to bend the connectors up very slightly (only the ends of the connectors, by a very small angle). This was done by removing the print cartridge, leaving the connectors in place; using an object (in this case a screw, but something flatter, like a screw driver would be more appropriate) to hold down all but the ends of the connectors, and then pushing the cuboid piece through the hole so it just bent the connectors up a little bit. This should mean that all the connectors bend up a little, and therefore should be more likely to make a good connection.

Figure 44: Using the cuboid to bend the connectors up

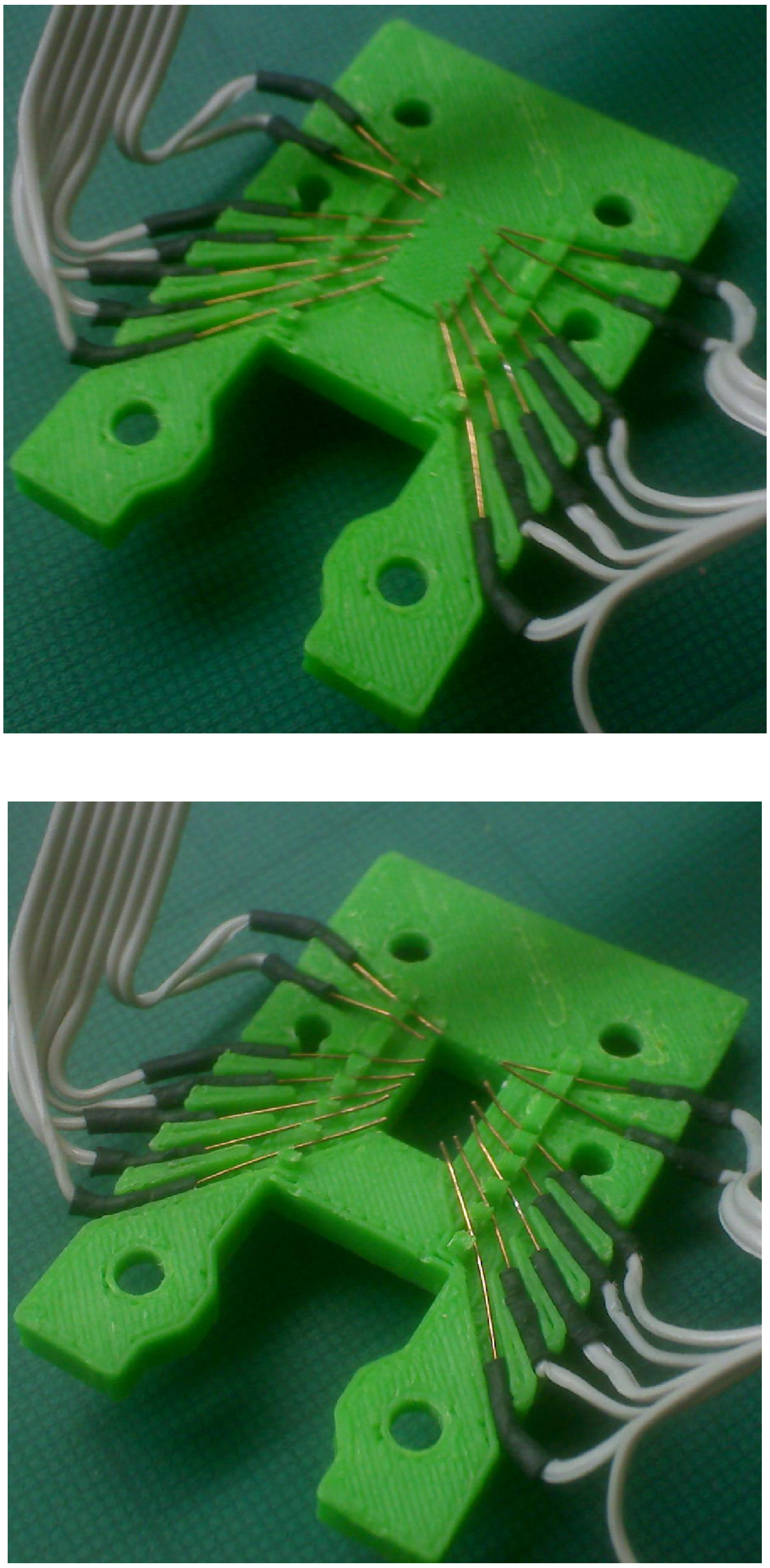


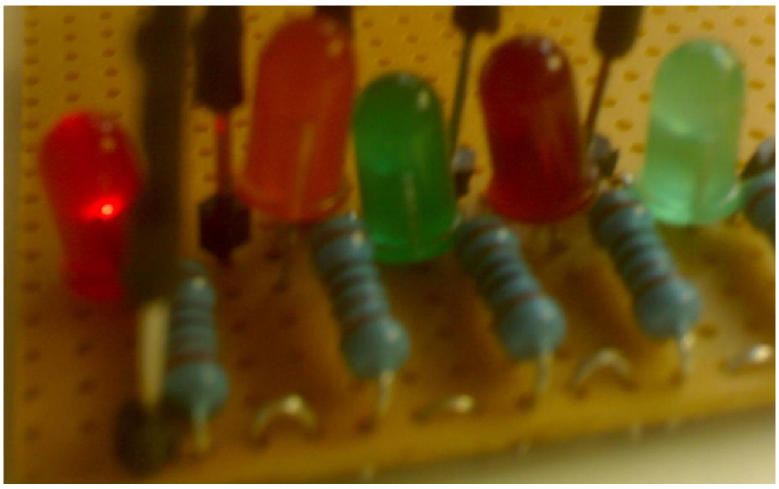
Figure 45: The connectors after they have been bent up

Upon inspecting the design by measuring the resistances, it is found that this design with the previously mentioned modifications gives good, reliable connections for all the connectors. This means that all the nozzles should work (unless they are clogged).

Whole design (uniting the separate sections)

The various elements must all come together when making a system to drive the printer. The first point where the separate elements of this project were linked was after the initial circuit was built. This circuit is the one with five LEDs on it, and was fired using the appropriate code. This caused the LEDs to flash, as shown here:

Figure 46: Flashing LEDs



To gain a better feel for the software, this sketch was altered in various ways, including changing the program so the LEDs normal state was on, and the LEDs blinked instead of flashing (by simply swapping the 'HIGH' and 'LOW'), and changing the timing of, and gap between, the flashes.

The next point where the separate sub systems was brought together was in preparing to test the whole system, by testing the electronics and software together once again. This was done by driving the electronics (figure 3) using the software written for the actual print cartridge (code 3). By using an oscilloscope instead of the print cartridge, it was possible to visualise the currents being sent to the print cartridge, and ensure that the output was what was desired (just to be safe).

Although all three parts were still being developed continually even after initial attempts to test the entire system together, the main area of development was the mechanical design, as there is no way of testing the (already built) electronics and software without a means of connecting it all to the print cartridge. Once this was working well enough (the cartridge holder described in Figures 3-4, which was later replaced by the final iteration of the design, Figures 3-4), the whole system was put together and tested.

One major issue that occurred is that the resistors on the print cartridge kept blowing. This happened with multiple resistors (on most cartridges, 6 resistors blew, but not the ones in the same location). This, along with being frustrating, was a serious issue as the cartridges only have 12 nozzles, so if 6 of them no longer function, the print resolution is severely affected, especially since the resistors that were blowing were often non-sequential, making the failed cartridges more or less useless. Also, identifying the issue that caused the resistors to blow up proved to be extremely challenging to identify.

Initially, it was thought that the connectors were failing, but upon closer inspection of the cartridge, it was found that the common and some of the contacts were no longer connected (the resistance across some of the pins, when measured, was infinite). After discovering that the resistors had blown, the literature regarding the print cartridge was re-examined, and it was found that Gilliland emphasised that the firing energy must not exceed 40 μJ (which the 6.5 μs calculated pulse time at 20 volts generated across the resistor generated). Tests on the cartridges that had already lost half their resistors revealed that increasing the pulse width to 20 μs still didn't cause the resistors to fail, but it was assumed that half the resistors could be better quality than the other half, and further examination of this hypothesis went ahead anyway (partly just to be thorough, and partly because, flawed though it was, it was the best theory at the time).

The pulse widths were re-examined using the oscilloscope, and it was discovered that although pulse width (which was rounded up to 7 μs) itself was as mostly as expected (although the pulses did vary in time, which was unexpected, they were all within 1 μs of each other, and the longest pulse width is the one in discussion here), the time taken for the pulse to die down was non-negligible. To account for this, the pulse width was reduced by 1 μs (so the main pulse was 6 μs, with the decay accounting approximately for the rest of the firing energy). This appeared to have no effect, and the pulse time was reduced right down to minimum possible, 3.5 μs, which still had no effect. At this stage, however, many of the nozzles weren't firing, which, was due to the pulse being too short.

At this stage the possibility of the pulses being too long when everything else was proper was dismissed. Although the excess energy in the pulses was still the most likely culprit, why the pulses were too long must be as a result of an error elsewhere.

The next theory was poor programming. There were certain programs that seemed to result in a greater a percentage of the nozzle failures than others; in particular the one that fires each pin individually, and the 'fun' one that tried to print the word 'hello'. All programs, especially the ones mentioned above were closely examined, checking all the pulse widths, dead times, and pause times, and their locations; but nothing appeared to be awry there. After many repeated inspections and rewritings of most of the sketches, it was determined that this is also unlikely to be the issue.

It was then suggested that perhaps the outputs from the arduino were, at some point, 'floating'. This is where there is no defined signal (often due to disconnected connectors), and the output flicks randomly from high to low. This, should it be the case, would result in extremely long pulses (compared to the length of the required pulses). The way to fix this is to add pull down resistors that connect the arduino outputs, via a resistor, to the ground. This means that the transistor arrays will always have a defined input, eliminating any 'floating'.

As a significant extension of this, basic analysis of the method of uploading programs to the arduino revealed that this was the most likely time for the outputs to float. It was also noted that the pin 13 in particular, which is connected to an LED on the board, flickered at a visible rate (i.e. it was on for much, much, longer than 6 μs), which might explain why, with the last two cartridges before discovering this issue, the resistors that blew were all sequential, with the resistor connected to pin 13 being at one end of this sequence (it was connected to the nozzle at one end); this didn't occur with the first two cartridges, in which the pin connected to pin 13 wasn't connected due to the holder that was used with them being designed using incorrect dimensions. Because of the aforementioned issue, it was also decided to turn the power to the transistor circuit off when uploading programs to the *arduino*.

Both the last two attempts to correct the issue were implemented at the same time; the pull down resistors were added, and the transistor circuit has been switched off whenever uploading to the microcontroller. As a result, it appears that the problem has been fixed; although it is not known which method fixed the issue (and therefore the precise nature of the cause).

Conclusions

(includes recommendations for future work)

The overall design, although challenging, proved solvable. The electronics seemed to work fairly reliably (except for the occasional builder related error in soldering), and the software also seemed to work reasonably early on in the project. The mechanical design took up the most time to design and construct.

One factor that caused significant issues with manufacturing the mechanical cartridge holder was the resolution of the 3D printer. If the resolution was greater, the design could be simplified greatly, probably resulting in a more reliable design than the one produced.

Future work

The first thing that needs to be done is the printer has to actually be connected to the RepRap machine. To do this, a mechanical link must be built to link the print cartridge holder to the moving head of the RepRap machine. This has been considered, but due to constraints on time, was not actually built.

The RepRap head requires a fairly simple design (a rectangle with certain protrusions to hold it in place), and this can easily be linked to the printer by using simple extending the existing bolts and designing an appropriate rectangular piece which fits into the RepRaps head, and has holes in it to allow the bolts through. This design is particularly useful, as it allows the height of the print head relative to the RepRaps moving head to be adjusted, which is required, due to the flexibility of the cartridge holder.

The RepRap head has a 17-pin connector to connect the current printing device (be it 3D or 2D) to the computer that controls the machine. However, it was decided that, initially, the *arduino* sketches would be uploaded to the microcontroller before hand via the USB connecter. An input on the board would be connected to the controlling computer, which would send signals at key points in the code to tell the arduino when to go (most of the code would have to be triggered by inputs from the computer, which means that there may have to be a lot of *if* loops embedded in other *if* loops, ending up almost like a Russian doll).

To do this, however, it would be rather inconvenient to constantly switch the power to the circuit on and off, so a transistor would need to added, so that the power can be switched off/the outputs blocked electronically using a signal from the RepRap controlling computer.

Another possibility is to try to remove the ink in the printer and replace it with a different, ‘more useful’, ink. Alternative inks being considered include ‘solder resist’ and any reasonably conductive ink. These two, used together could print a variety of circuits (either one would ease the construction of circuits), and there is also potential for using special inks to make semiconductors (although, admittedly, the author of this report does not know very much about the possibilities and limitations of this).

Acknowledgements

I would like to thank Adrian Bowyer for significant amounts of help with the electronics, and some with the design, Ed Sells for helping me use the Strat and with RP machines generally (and his wealth of knowledge on the RepRap) and Ian Adkins for helping to solve the major issue using his electronics experience.

References

Reference	Author (s)	Details
1	Gilliland M	Inkjet Applications
2	Wiki	Reprap.org
3	Bowyer, A	Personal conversation
4		Stratasys Dimension BST specification sheet
5		HP 51604A specification sheet
6		Arduino website