IBM 1403-N1 Printer Interface

Project Report

Architectural Design, Detailed Design, Test Readiness, Interim, Final Version

Watson Capstone Project <WCP22>
Client: Center for Technology & Innovation (CT&I)
Sponsor: IEEE Binghamton Chapter

9rd December 2013

Revision: - Draft 9 December 2013

Submitted by:
Ryan Kulesza, Lead, EE
Peter Haviland, CoE
Mohammad Imran, CoE
Yu Chao Wang, EE

Faculty Advisor: Professor Maynard
External Advisor: Don Manning
Client Advisor: Susan Sherwood
Program Manager: Professor Jack Maynard
IEEE Chair: William Tracz

Approved for public release; distribution is unlimited.
Submitted in partial fulfillment of EECE 487-488 / ME 493-494 requirements.

Thomas J. Watson School of Engineering and Applied Science
Binghamton University
Binghamton, NY

Abstract

The IBM 1403-N1 printer was introduced in October 1959. Being able to print 1100 lines per minute, the IBM 1403-N1 paved the way for a new era of high-speed and high-volume printing which was only surpassed by laser printers in the 1970s. It was a revolutionary printer in that it was the first of its kind to deploy what is known as on-the-fly printing. With the characters embedded within a fast moving chain, hammers pressed against the paper when right characters were aligned. Key to this kind of printing was to know when to activate which hammer.

In our project, attention was mainly focused on the driver cards within the printer. These drivers; hammer driver and carriage driver, are responsible for hammer firing and forms movement. Using modern day technology, a reconstruction of these drivers was carried out under the guided supervision of CT&I experts. Also essential to printing are feedback signals from the printer itself telling which character is in which position so that the right hammers get fired. Our project utilized a microcontroller which will be programmed to make logical decisions by processing these feedback signals.

This report discusses our project goal and our analysis and reconstruction of the driver circuits which has reached its completion and is ready for full scale fabrication. Also discussed in this report is a detailed analysis of print mechanism and the corresponding algorithm that will be implemented on the microcontroller.

Table of Contents

List of Figures

List of Tables

1 Scope

1.1 Identification

This is the interim project report for Watson Capstone Project WCP22, Interface for IBM 1403 N-1 Printer of 2013-2014.

1.2 System Overview

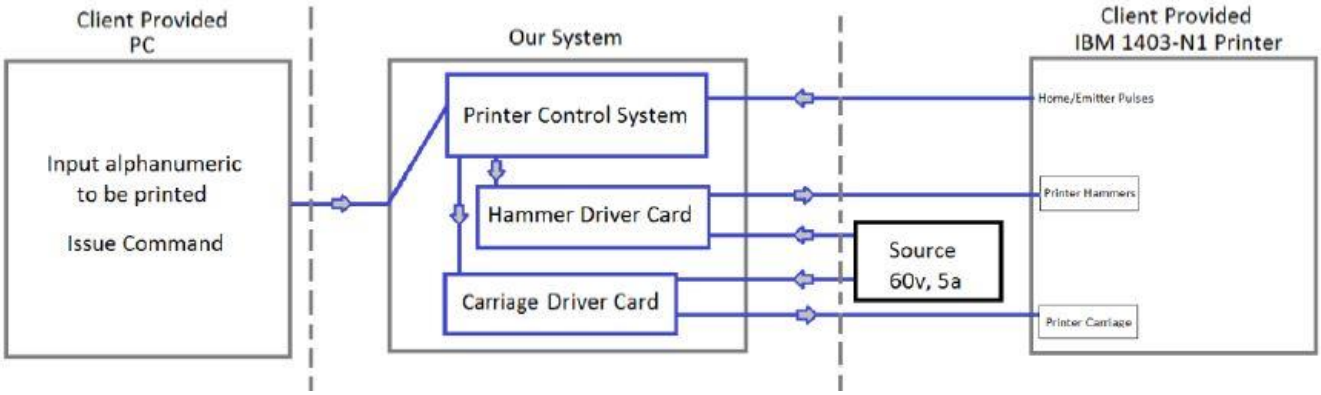


Figure 1 Contextual Flow Chart

In this project IBM's 1403-N1 Printer will be used to print data coming from a modern day PC. The project utilizes a Printer Robot Driver, compatible with the 1403, for limited print capabilities. The robot driver will drive 12 printer hammers and will utilize timing and feedback to synchronize with the printer.

The 1403-N1 printer, developed in 1959, has not been in function for a number of years and hence this project is an attempt to revive the printer with a modern interface. Using readily available, modern components, hammer drivers will be redesigned and produced utilizing the existing IBM schematics and advise from CT&I technical experts.

The project is sponsored by IEEE's Binghamton Chapter, and will be used by CT&I for demonstration purposes at TechWorks located at 321 Water Street, Binghamton, NY.

1.3 Document Overview

This document will define the design features of the IBM1403-N1 Printer Interface project. The document has been created by WCP22, and it applies to the Architectural Design, Detailed Design, and Interim design phases. There are no security or privacy considerations or restrictions pertaining to this document.

2 Referenced Documents

The following documents of the exact issue shown form a part of this document to the extent specified herein.

No.	Title	Date created	Latest Revision
1.	1403 printer models N1 and 3 Maintenance Manual	December 1971	-
2.	1403 printer Maintenance Manual	November 1964	-
3.	Project Requirement Specification (PRS) for IBM1403-N1 Printer Interface	November 2013	November 6, 2013
4.	Project Development Plan (PDP) for IBM1403-N1 Printer Interface	December 2013	December 6, 2013

3 Project Overview

3.1 Project Definition

The IBM1403-N1 Printer Interface project as defined in the Project Requirements Specification (PRS) document will provide hardware and software necessary to operate 12 hammers and the carriage control of the IBM1403-N1 Printer. The input to the system will be meaningful text and line spacing arguments. The output will be the printing of the text and execution of line spacing using the IBM1403-N1 Printer. Delivered hardware will include a Project PCB, for dissemination and collection of signals and the routing of these signals to the microcontroller, Hammer Driver PCBs (of which there will be two), or Carriage Driver PCB. The Hammer Driver PCBs will communicate timed hammer pulses in order to print meaningful text, and the Carriage Driver PCB will actuate line spacing depending on the line spacing argument provided, or if necessary- due to input type- when all 12 print positions have be printed.

3.2 System Design

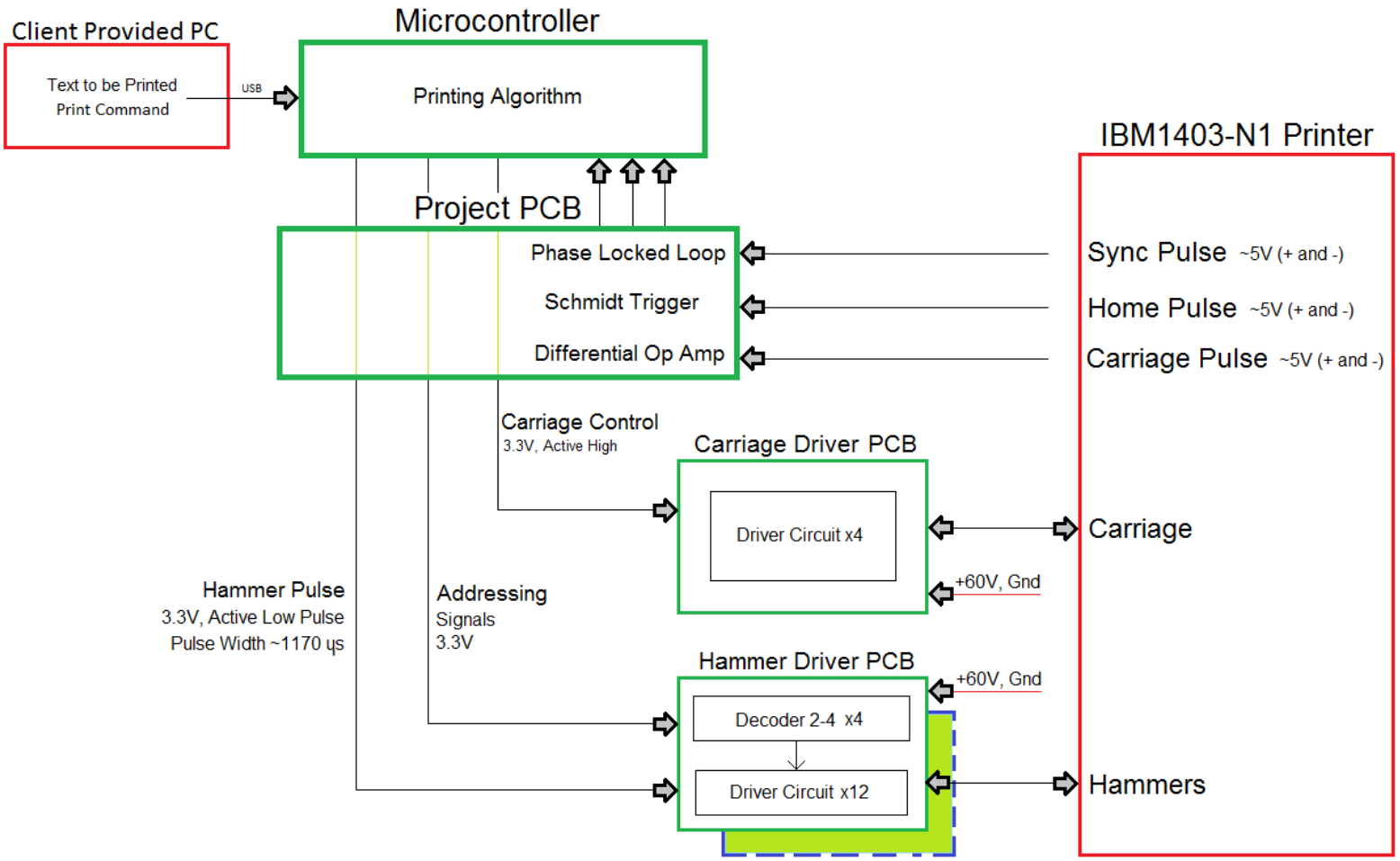


Figure 2 System Flow Chart

The above flow chart describes the major components of our design, as well as, the interface between our design and client provided systems.

Client provided systems appear in red, including a PC, a 60V 5A source, and an IBM1403-N1 Printer. The PC is required for acquisition of text data to be printed and the issuance of the print command. The 60V 5A source is required supply power to twelve hammer driver circuits and four carriage driver circuits. A Single Pull Signal Throw (SPST) switch will be provided by WCP22 in order to turn on and off the 60V 5A source where it interfaces with the Project PCB. This switch eliminates the possibility of unintended hammer strikes or carriage movement due to unknown states in the microcontroller prior to initialization.

The IBM1403-N1 Printer is required to perform the printing function utilizing internal hammer solenoids which will propel the print hammers toward the paper form at specific text locations on the current line. The IBM1403-N1 printer is also required to perform the line spacing function utilizing internal carriage solenoids which will control four hydraulic valves, one valve to start a line space, one valve to stop line spacing, one valve to start line skipping, and one valve to stop line skipping. The IBM1403-N1 is also required to provide feedback to the WCP22 IBM1403-N1 Printer Interface.

There are three types of feedback that will allow our system to synchronize with the IBM1403-N1 Printer. The first type of feedback is a carriage emitter pulse ("E1"), as the carriage position changes the gear connected to the carriage hydraulics generates a signal that will be used to indicate the number of spaces the carriage has performed. The second type of feedback is the cartridge emitter pulse. As the characters revolve on the cartridge a gear connected to the cartridge will generate a signal that will be used to indicate when a new character has aligned with printer hammer one, two, or three. This emitter signal will be known as the "sync" pulse. The third type of feedback is another emitter pulse that is generated by a second gear connected to the printer cartridge. This pulse is coincident with the sync pulse when the characters on the cartridge have returned to their original position. This emitter signal will be known as the "home" pulse.

The Hammer Driver PCB will be delivered by WCP22. The function of this printed circuit board is to provide a 60V 5A pulse of approximately 1170 microseconds in width to any one of the twelve IBM1403-N1 internal hammer solenoids. In order to drive twelve hammers, two identical Hammer Driver Cards will be created. Each of the Hammer Driver Cards will contain six driver circuits, as well as, two 2-4 decoders. The decoder will be supplied with the hammer actuation pulse originating from the microcontroller, as well as, two addressing signals which also originate from the microcontroller. The addressing signals will indicate the circuit path pertaining to a specific hammer driver circuit. The hammer actuation pulse is active low due to the characteristics of the selected decoder. The decoder outputs a logic high signal to the addressed path when the inverted-enable input receives a logic low signal.

The Carriage Driver PCB will be delivered by WCP22. The function of this printed circuit board is to provide a low current signal of less than one amp to any one of the four IBM1403-N1 internal carriage solenoids. The carriage driver circuits will be supplied with four active high signals. These carriage control signals will originate from the microcontroller and will be used to control the IBM1403-N1 internal space start solenoid, space stop solenoid, skip start solenoid and skip stop solenoid. Either the space start or space stop signal will be active at all times to control the IBM1403-N1 hydraulic space valves. Likewise, either the skip start or skip stop signal will be active at all times to control the IBM1403-N1 hydraulic skip valves.

The Project PCB will be delivered by WCP22. The function of this printed circuit board is to condition the three feedback signals from the IBM1403-N1 printer, to provide regulated 3.3V power to the microcontroller, and to disseminate incoming and outgoing logic signals to the Hammer Driver PCB, the Carriage Driver PCB, and the Microcontroller. All three feedback signals will be conditioned for use in the microcontroller by means of a differential operational amplifier stage and a one-sided Schmidt trigger, the output of which will be provided to the microcontroller. The differential stage is necessary to collect the signal data from each of the IBM1403-N1 internal emitters without the presence of a common ground. The one sided Schmidt trigger will generate a logic pulse when the output of the differential stage exceeds the Schmidt trigger threshold. Additionally the sync pulse and the home pulse generated by the Schmidt trigger will be applied to an AND gate, since the "home" condition is satisfied when both the

home pulse and the sync pulse is high. The output of this AND gate will be provided to the microcontroller, in addition to the "sync" pulse, and the carriage emitter pulse to allow for synchronization between the microcontroller algorithm and the IBM1403-N1 printer. To aid in synchronization a phase locked loop circuit will accept as an input the "sync" pulse and perform a divide-by-three function on this pulse effectively creating a clock divider. This clock-divided signal will also be provided to the microcontroller to indicate the timing of three sub-scans. The phase locked loop circuit will be capable of adjusting the timing of the sub-scan pulses as the printer hammers may change the velocity of the cartridge rotation as they make contact with the paper and characters.

The +5V -5V 1A supply will be delivered by WCP22. The purpose of this supply is to provide power to the 3.3V regulator, and all integrated circuits. The supply provides both +5V and -5V to accommodate the rail voltages of our operational amplifiers, without the introduction of an offset between the 60V 5A supply ground and the +5V – 5V 1A supply ground.

The microcontroller of the system is essentially the hub where signals from the PC and printer are received and processed. Its outputs are based on logical decisions which will be captured in our code and outputted to the driver cards.

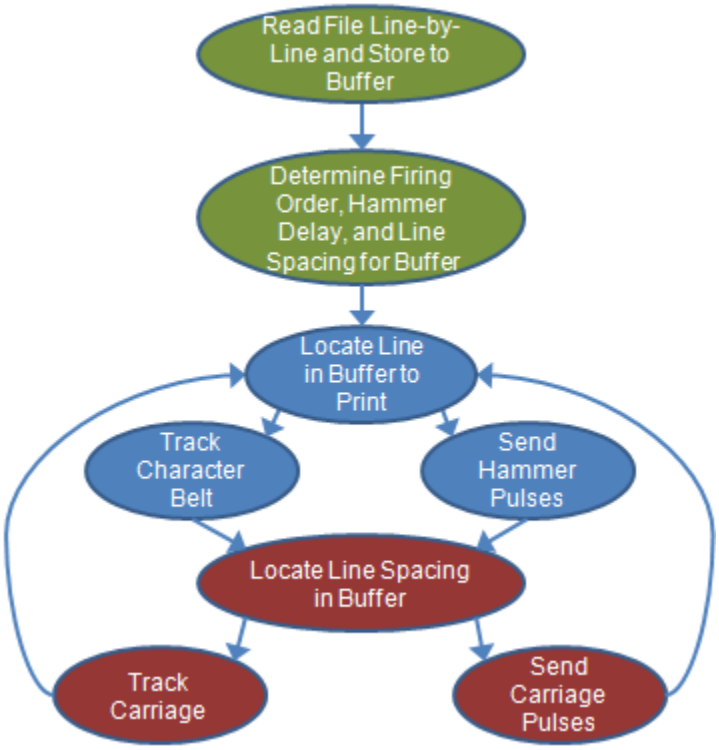


Figure 3 Printing Algorithm Flow Chart

The above flow chart demonstrates the basic flow of operation for the printing algorithm.

Once the user has finished entering the print job in the form of a text file, the data is sent to the microcontroller via USB cable. The microcontroller will accept the data line-by-line and store it to a buffer before printing occurs. This buffer will store the entire page. When the page is completely buffered in the microcontroller, an algorithm will precisely determine which hammer needs to be fired and at which specific times utilizing feedback signals, discussed in detail in section 4, from the printer. The algorithm will also determine the individual line spacing for the

page along with the printer hammer delays. To print the page, the microcontroller will move through the buffer line-by-line. To print a line of characters, the microcontroller will track the print cartridge while simultaneously sending hammer pulses as determined by the print algorithm. When a blank line is met, the microcontroller will utilize carriage signals as described in section 4 to move the page based on the determined line spacing for that individual line. The microcontroller will continue to move through the buffer until the entire page has been printed.

MPIDE will be used to program the microcontroller in the C programming language. This program will control the printing algorithm, the receiving of the text file from the PC, and the interaction with the IBM 1403-N1 printer. MPIDE is free software provided by Digilent and is cross-platform. For further debugging functionality, MPLAB will be utilized. MPLAB is free software and works with the XC32 C/C++ Compiler, available for download online. The ChipKIT Pro Mx4 contains all the built-in hardware for debugging with MPLAB.

The Processing IDE will be used to run the program required to send the text file from the PC to the microcontroller. This program will read the text file and send the data to the microcontroller through the USB wire. Processing IDE is open source and cross-platform software that is programmed using the open source programming language called processing.

The ChipKIT Pro Mx4 contains 74 I/O pins which will allow for future operable expansion to all 132 hammers. The implementation of the 2x4 decoders will allow for the control of all 132 hammers. The ChipKIT Pro Mx4 microcontroller has 512 KB of on board memory and 32 KB of SRAM memory. If the file to be printed is larger than the workable SRAM, then expandable memory will be implemented with the microcontroller to store the buffer for the file.

3.3 Project Schedule

ID	Task Name	Start	Finish
1	**Fall Semester**	**Wed 9/18/13**	**Fri 12/13/13**
2	**Requirements, Analysis, Definition**	**Wed 9/18/13**	**Mon 11/25/13**
3	1403-N1 Printer Analysis	Wed 9/18/13	Wed 10/2/13
4	Data/Signaling Analysis	Thu 10/3/13	Wed 10/23/13
5	Project Requirements Specification	**Wed 10/2/13**	**Sun 11/24/13**
9	*SRR (SystemRequirements Review)*	Mon 11/25/13	Mon 11/25/13
10	**Planning Phase**	**Mon 10/14/13**	**Thu 12/5/13**
11	Project Development Plan	**Mon 10/14/13**	**Wed 12/4/13**
17	*SDR (SystemDesign Review)*	Thu 12/5/13	Thu 12/5/13
18	**Architectural Design**	**Wed 11/6/13**	**Fri 12/13/13**
19	Circuit Component Selection	**Wed 11/6/13**	**Mon 11/18/13**
23	Software Selection	**Wed 11/13/13**	**Mon 11/18/13**
26	Hardware Selection	**Wed 11/13/13**	**Mon 11/18/13**
29	PreliminaryDesign Project Report	Tue 11/19/13	Thu 12/12/13
30	*PDR (Preliminary Design Review)*	Fri 12/13/13	Fri 12/13/13
31	**Detailed Design**	**Fri 11/8/13**	**Fri 12/13/13**
32	Software Design	**Fri 11/8/13**	**Fri 12/6/13**
35	Hardware Design	**Tue 11/19/13**	**Fri 12/6/13**
40	Critical Design Project Report	Tue 11/19/13	Thu 12/12/13
41	*CDR (Critical Design Review)*	Fri 12/13/13	Fri 12/13/13
42	**Spring Semester**	**Mon 1/27/14**	**Fri 5/9/14**
43	**Test Procedures Definition**	**Mon 1/27/14**	**Tue 2/11/14**
44	Hardware Testing Procedure	**Mon 1/27/14**	**Mon 2/10/14**
48	Software Testing Procedure	Mon 1/27/14	Mon 2/10/14
49	*Test Procedures Review*	Tue 2/11/14	Tue 2/11/14
50	**Building, Integration**	**Mon 2/3/14**	**Tue 2/11/14**
51	Hardware Building	**Mon 2/3/14**	**Mon 2/10/14**
54	Software Programming	Mon 2/3/14	Mon 2/10/14
55	*TRR (Built, Ready For Testing)*	Tue 2/11/14	Tue 2/11/14
56	**Testing**	**Wed 2/12/14**	**Wed 3/12/14**
57	Hardware Testing	**Wed 2/12/14**	**Wed 3/12/14**
61	Software Testing	**Wed 2/12/14**	**Wed 3/12/14**
64	**1403-N1 Integration**	**Fri 3/14/14**	**Mon 3/17/14**
65	Wiring	Fri 3/14/14	Mon 3/17/14
66	**1403-N1 Testing**	**Tue 3/18/14**	**Thu 5/8/14**
67	Hardware Testing	**Tue 3/18/14**	**Fri 3/28/14**
71	Software Testing	**Sat 3/29/14**	**Thu 5/8/14**
75	*Tested, Compliant, Delivered*	Fri 5/9/14	Fri 5/9/14

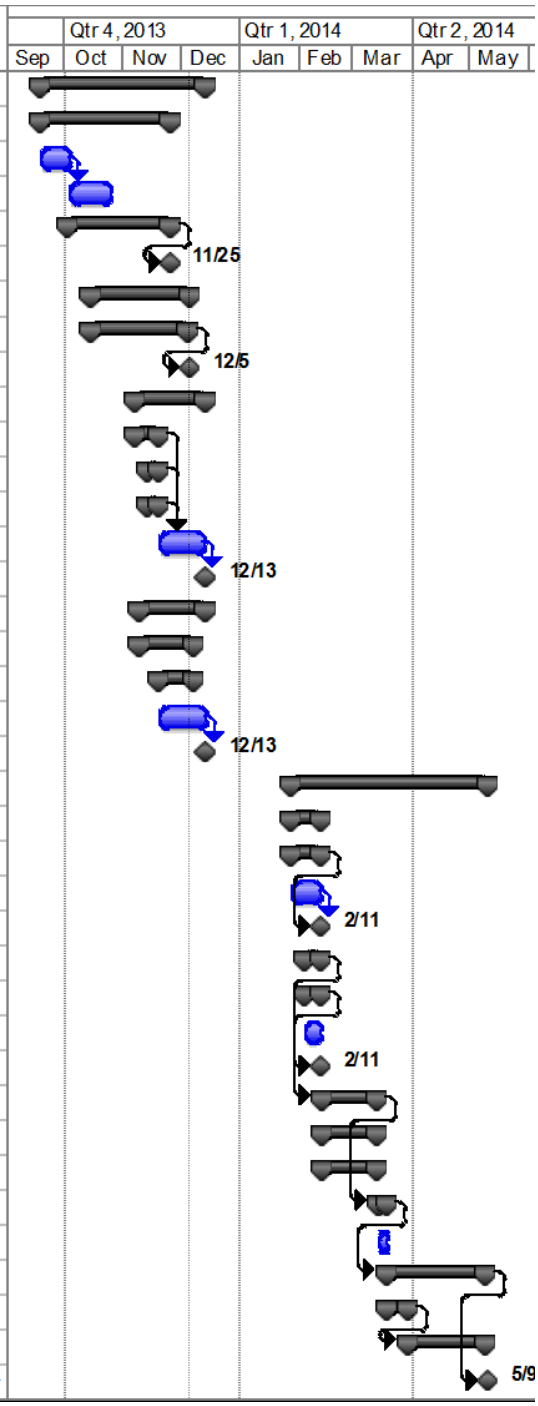


Table 1 Project Schedule

3.4 Project Finances

Item	Original Estimate	Expended	Estimate-to-Completion	Estimate-at-Completion
Electrical Components	75	0	175	175
Printed Circuit Boards	75	0	175	175
Micro Controller	50	0	80	80
Expandable Memory	30	0	30	30
Cables	20	0	20	20
				$480.00

Table 2 Top-Level Financial Summary

The above table discusses the financial plans for the system with an estimated cost at completion.

The manageable budget for this system is $1600 dollars. The estimate-at-completion is currently under half the manageable budget in the range of $480 dollars.

3.5 Conclusion

This section shall summarize the current status of the project, noting the significant challenges, accomplishments, lessons learned, and if applicable, the work yet to be done.

In this semester systematic analysis of the project was carried out, where we determined what the project will accomplish and how it would work. We researched all the necessary parts and components need for making circuitry, laid out the schematic for hammer driver cards as well as our project PCB, made pseudo code that replicated printing functionality and carried out all the necessary trade studies to pick the most suitable microcontroller. Significant challenges were faced in understanding print mechanism of the 1403-N1 printer. The work yet to be done includes ordering of components, population of printed circuit boards (PCBs), thorough testing of the hammer driver card, developing code, implementing the print process using a microcontroller, as well as, interfacing with the IBM1403-N1 Printer. These project milestones are scheduled for the spring 2014 semester.

4 Technical Details

4.1 Design Decisions

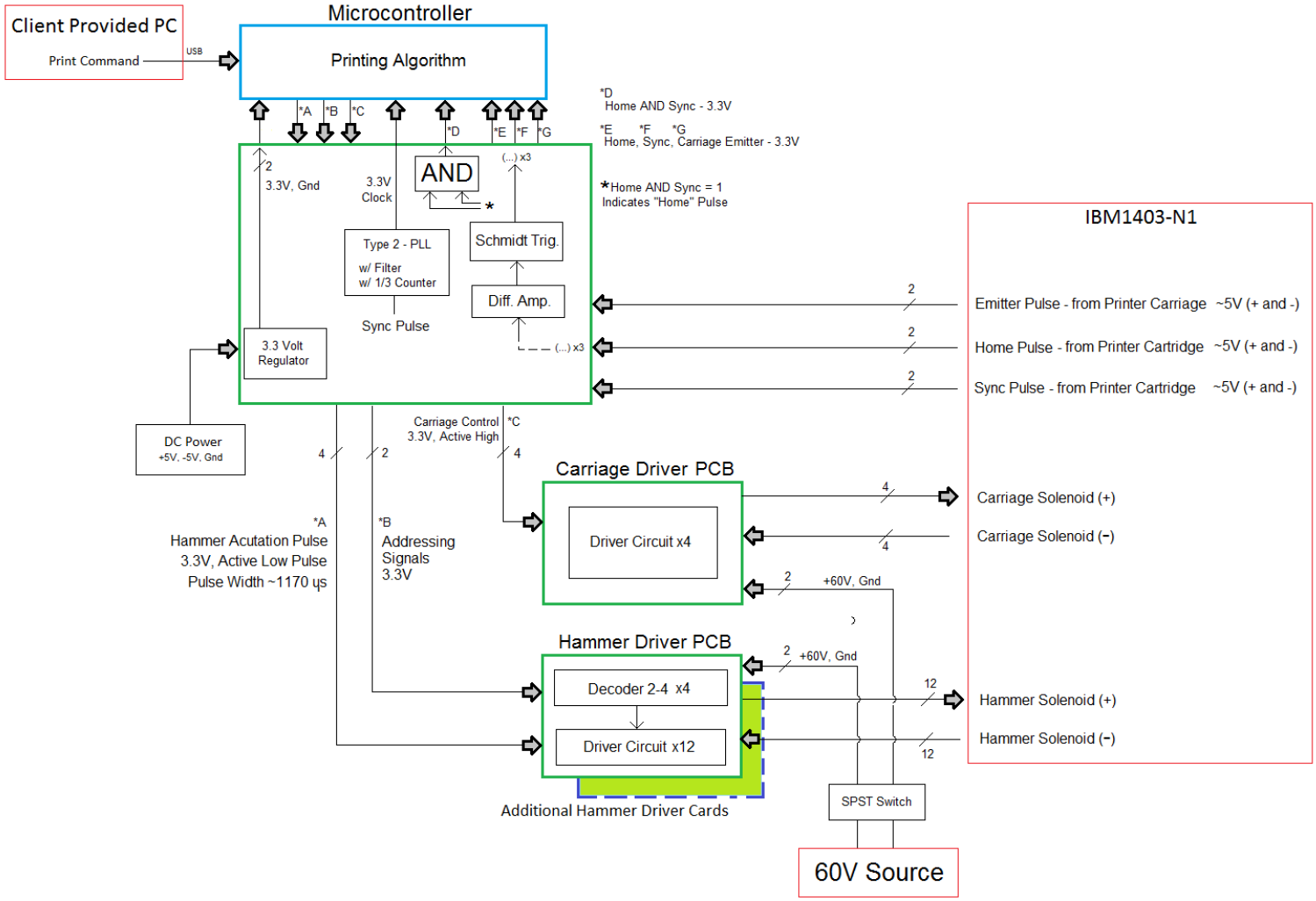


Figure 4 Detailed System Flow Chart

As shown in the Detailed System Flow Chart, figure 4, there are three inputs to the system; data to be printed delivered through a USB from the PC (utilizing a keyboard), feedback signals from the printer and a 60V 5A power supply for the hammer and carriage drivers. There are two major outputs; signals to be sent from the microcontroller to the hammer driver producing a 60V 5A pulse to control hammer solenoids, and signal from the microcontroller to the carriage driver producing a 60V <1A signal to control carriage solenoids. Printing begins by user typing in the desired data that needs to be printed using a modern keyboard and PC. This data is then sent to

the microcontroller via USB. The microcontroller then sends out signals to the hammer driver and printer at correct timings to fire each hammer, and to control the carriage for line spacing.

The input from the keyboard will be buffered and printing will start after synchronization. Once the buffer is full a prompt will be sent to the user that buffer is full. At this point the job of the keyboard is done and printing can begin. To determine correct characters are printed (hammered) onto the paper, correct logic will be implemented to determine rotational speed of the printer cartridge and pulse timings of the hammer driver. This logic will be carefully captured in the code algorithm (C language). If implemented correctly high performance can be achieved as the clock Rate of the microcontroller is quite high for the type of signals we are dealing with in this project. On the user's side, un-allowed input includes typing in data once buffer is full or inputting characters that are not in the printer like tab. These will be dealt by prompting users to re-enter input data. Un-allowed inputs to the hammer driver can be high voltage spikes. These will be dealt by including fuses in the circuitry at appropriate places.

The user will input data using a text file editor on any operating system.

WCP22 will utilize advice of CT&I technical experts in order to produce a prototype with a level of safety appropriate for demonstration at the TechWorks facility.

WCP22 will produce printed circuit boards, via a third party (Advanced Circuits), with markings that may be traced back to the circuit schematic for ease in population of the boards.

WCP22 has designed the IBM1403-N1 Printer Interface with provisions included for expandability. The hardware has been designed using decoders in order to reduce the necessary number of I/O pins so the IBM1403-N1 Printer Interface may be expanded to control all 132 print hammers using the same microprocessor with the addition of twenty Hammer Driver PCBs.

WCP22 has also included in the code a method of receiving input line-by-line. This provision allows the IBM1403-N1 Printer Interface to be expanded to achieve the future goal of interfacing with an IBM1441 mainframe which delivers input line-by-line.

		Arduino Due			chipKIT Pro MX4		
Criterion	Weight	Value	Score	Result	Value	Score	Result
# of I/O Pins	.2	54	2	0.4	74	5	1
Pin Voltage	.1	3.3V	4	0.4	3.3V	4	0.4
I/O Total Current	0.05	130 mA	4	0.2	200 mA	5	0.25
Input Voltage	0.05	7-12V	5	0.25	3.6-12V	5	0.25
Clock Speed	0.2	84 MHz	5	1	80 MHz	5	1
Flash Memory	0.2	512 kB	5	1	512 kB	5	1
SRAM	0.05	96 kB	5	0.25	32 kB	3	0.15
EEPROM	0.05	N/A	0	0	N/A	0	0
Cost	0.1	$58	3	0.3	$80	2	0.2
Sum	1			**3.8**			**4.25**

Table 3 Microcontroller Trade Study

The Microcontroller selected best fit for the system was the chipKIT Pro MX4. This microcontroller, which is provided by Digilent, is compatible with the majority of Arduino code while offering the large I/O required for the system. Although there is less available SRAM when compared to the Arduino Due, this will be combated with expandable memory to hold the potentially large files. The Pro MX4 is also more expensive than the Arduino; however the sheer number of I/O pins supplied by the microcontroller is one of the most important factors for future development of the system.

4.2 Sub-System Design

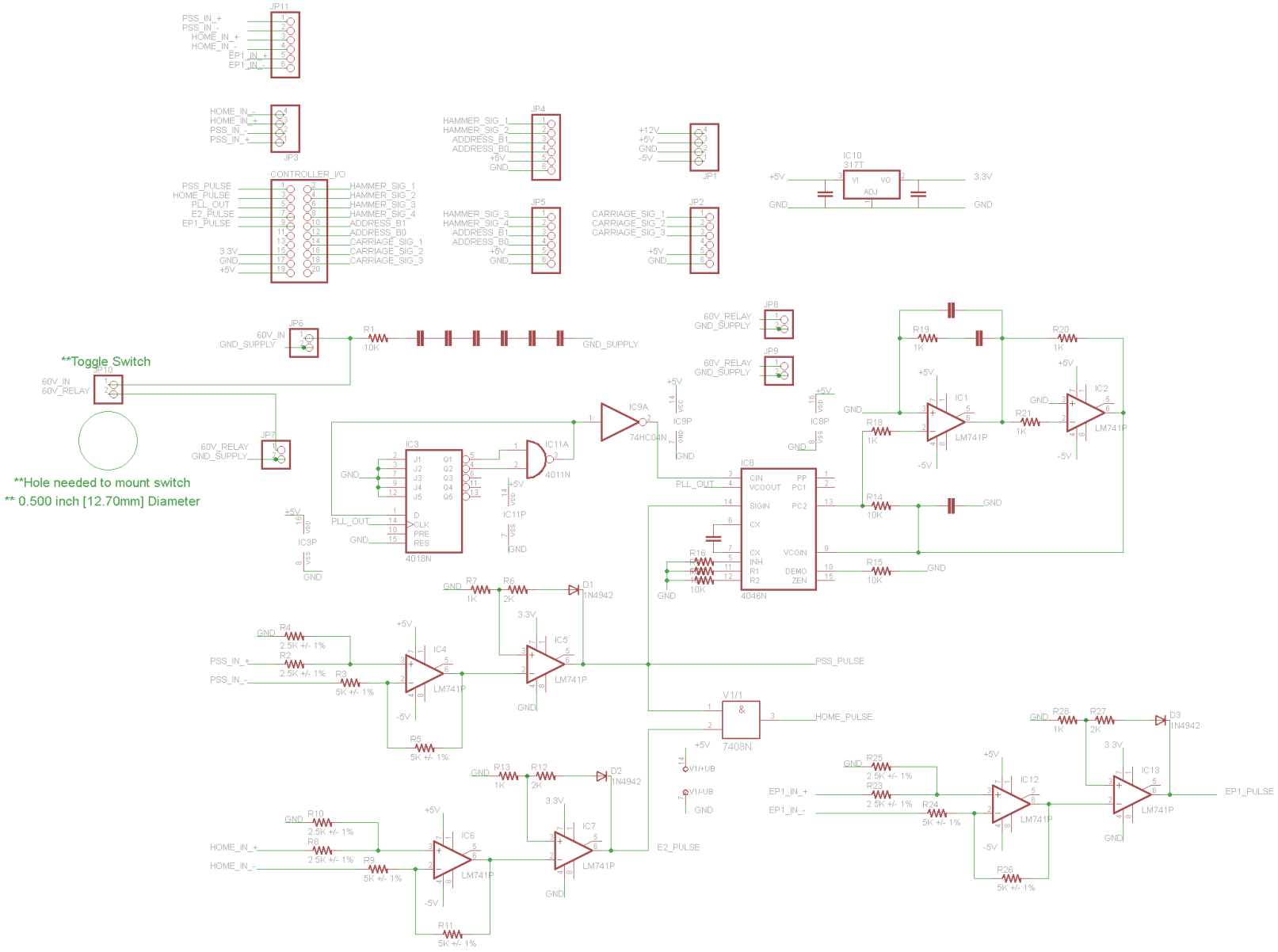


Figure 5 The Project PCB

The Project PCB is responsible for the dissemination of signals to the microcontroller, the Hammer Driver PCBs and the Carriage Driver PCB. The Project PCB will also condition the emitter pulses received from the IBM1403-N1 Printer for use in the microcontroller. Our project will utilize a +12V +5V -5V power supply for the operation of integrated circuits. The project PCB also contains a 3.3V regulator to supply 3.3V power to the microcontroller from the +5V channel of our +12V +5V -5V power supply.

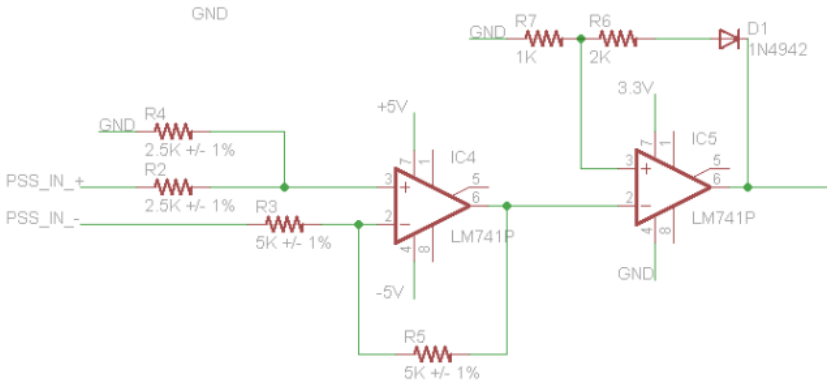


Figure 6 Emitter Input Conversion

The incoming emitter pulses are generated by the revolution of the cartridge gear in the case of the "sync" pulse, an additional cartridge gear in the case of the "home" pulse, and the carriage gear in the case of the carriage pulse "E1." These signals must be converted from periodic analog signals to periodic digital signals that can be used by the microcontroller. In order to accomplish this, the signal is first passed through a differential operational amplifier. This stage collects the potential difference created by the rotating gear with a net gain of 1. This stage is necessary because there is to common ground between the emitter and the IBM1403-N1 Printer Interface. The signal is then passed through a Schmidt trigger. The Schmidt trigger has a diode in the feedback loop to prevent negative feedback from entering the non-inverting input of the operational amplifier. This creates a one-sided Schmidt trigger necessary to provide a signal pulse which will represent the signal digitally. The output of the Schmidt trigger will then be passed to the microcontroller to represent the "sync" pulse and the "carriage pulse."

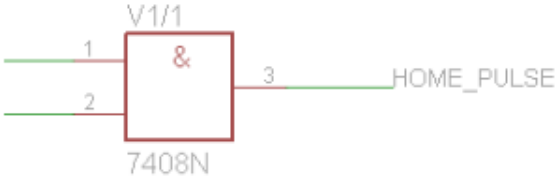


Figure 7 AND Gate

The home pulse and the sync pulse are then passed through an AND gate, shown in figure 7. When the home and sync pulses satisfy the AND condition this indicates the cartridge has returned to its starting position. This signal is of great importance to our algorithm because it will be used to indicate when the algorithm may begin to queue print hammers.

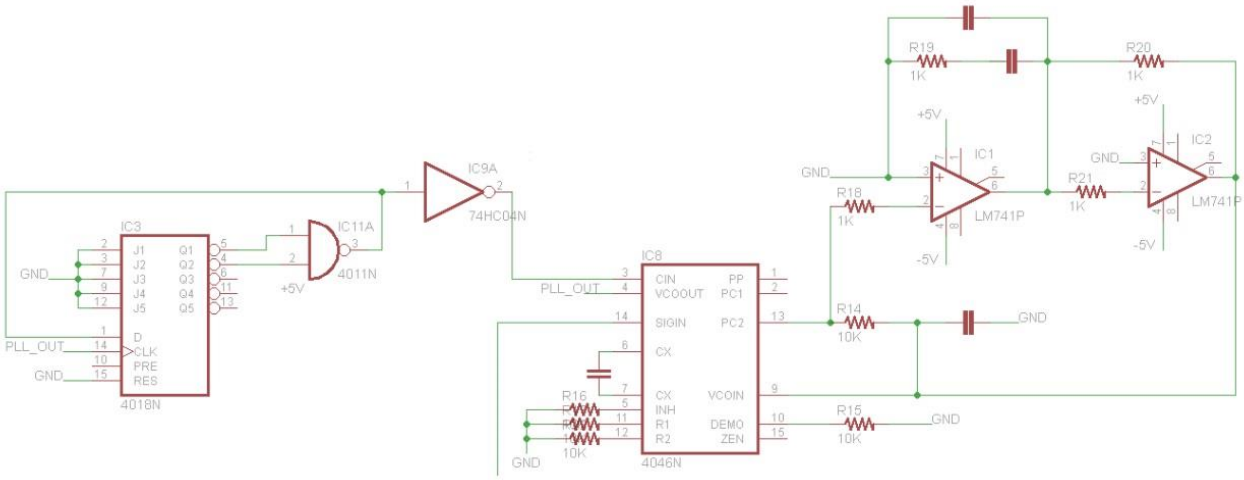


Figure 8 Type 2 Phase Locked Loop

In order to provide a synchronizing pulse for each of the three equally spaced sub-scans, the "sync" pulse is also passed through a type 2 phase locked loop, configured to create a clock divider that performs a divide-by-three function, shown in figure 8. This type 2 phase locked loop may use two different varieties of filters in the forward control path. Both of these filters have been added to our circuit as advised by CT&I expert Bob Arnold. This design decision allows WCP22 flexibility by providing the option of using either filter depending on which of the components are populated on the Project PCB.

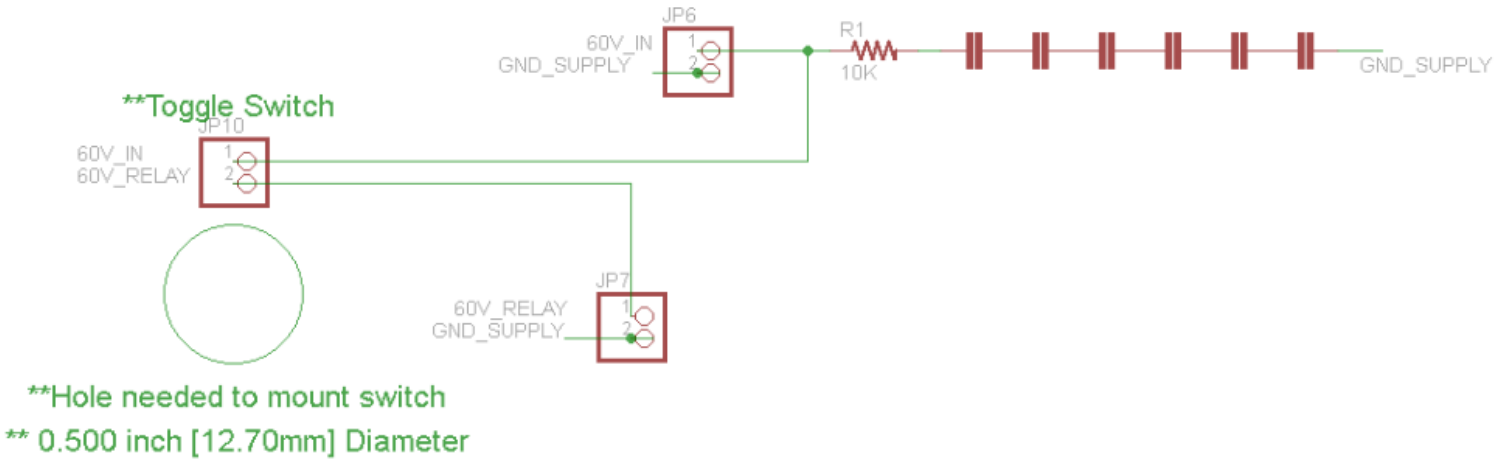


Figure 9 Interface with 60V 5A Source

The Project PCB is also responsible for accepting input from the 60V 5A source. The interface with the source is shown in figure 9. When the toggle switch is in the "off" position, 6 capacitors will be charged through a 10K ohm resistor. The capacitors are in series so each would charge to 10V. This will help stabilize the voltage at 60V as hammers are being fired and the carriage is being controlled. When operating the IBM1403-N1 Printer Interface the microcontroller will be given time to initialize before the 60V 5A source switch is moved to the "on" position. The switch will be moved to the "off" position before the microcontroller is powered down. This will prevent unintended firing of printer hammer or movement of the carriage due to unknown states in the microcontroller.

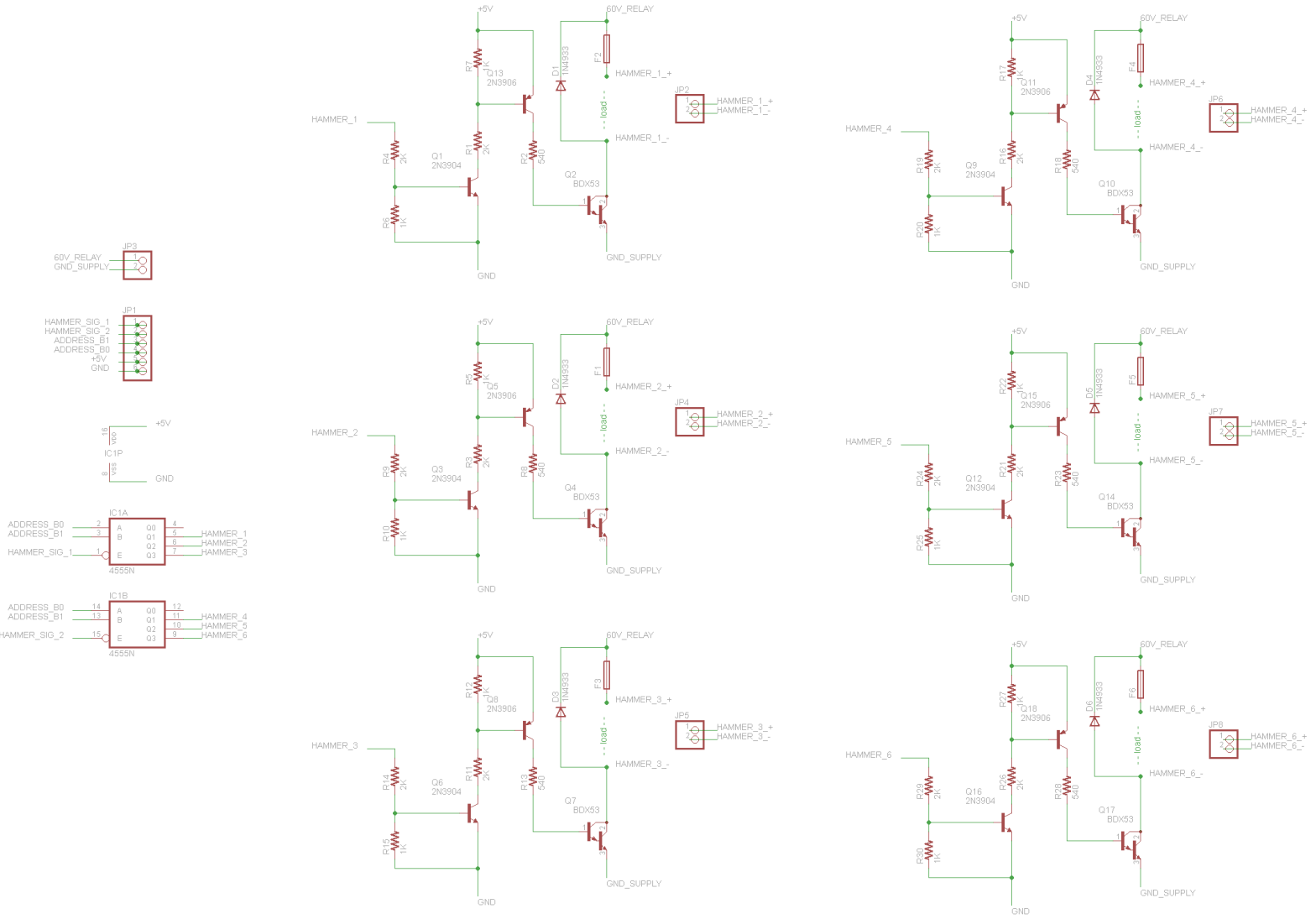


Figure 10 The Hammer Driver PCB

The Hammer Driver PCB, imaged above, will accept an active low pulse of approximately 1170 microseconds, originating from the microcontroller, via the Project PCB. This pulse will be sent to the correct hammer driver circuit by utilizing an inverted-enable decoder that outputs a logic high value on the addressed output when the enable pin receives a low signal.

The decoder used in the design is a dual 2-4 decoder, shown in figure 11. Due to nature of the printer upon receiving the home pulse the address 01, may be sent to the addressing pins of all decoder. This will select hammers 1, 4, 7, and 10 in our project, and may be expanded to select all hammers satisfying the equation 3n-2 (where n is a positive integer) if the project is expanded to include all IBM1403-N1 Printer hammers. Upon receiving the first sync pulse

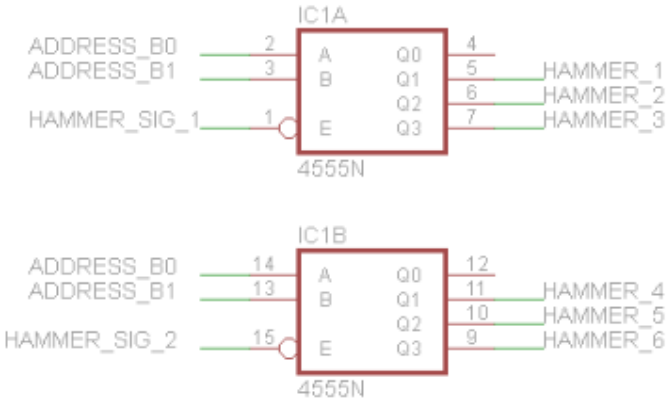


Figure 11 Decoder

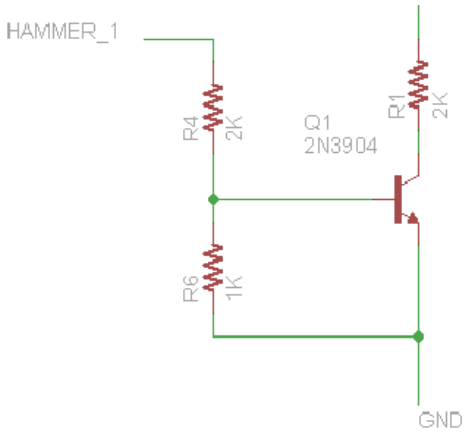


Figure 12 NPN Stage One

the address 10 will be sent to all decoder addressing pins, selecting hammers 2, 5, 8, and 11 in our project, and may be expanded to all hammers satisfying the equation 3n-1 (where n is a positive integer). When the third sync pulse is received the address 11 will be sent to the addressing pins of all decoders. This will select hammers 3, 6, 9, and 12 in our project, and may be expanded to select all hammers satisfying the equation 3n (where n is a positive integer).

The active high output of the decoder will allow current to pass in the initial NPN stage of the driver circuit, shown in figure 12. This configuration allows for the use of a 3.3V logic signal to activate the driver circuit without shifting the logic level provided by the microcontroller as many methods of logic level conversion introduce undesirable delay.

The Hammer Driver PCB also contains protective measures, shown in figure 13, meant to disable a driver if the current signal becomes fixed in the "on" state. This may occur if the hammer is unable to return to its neutral position after contact. A 1.5A slow-blow fuse is included in the current path that will deliver the 60V 5A pulse to the IBM1403-N1 Printer hammer solenoids. This fuse was included in the original IBM1403-N1 hammer driver card, and is rated to 1.5A due to the very low duty cycle that will be imparted on the printer hammer. If this duty cycle becomes large enough due to any error the fuse will then blow disconnecting the affected hammer. The fuse will be mounted on fuse clips to allow for solder-less replacement of a blown fuse. Another protective measure, found in each driver circuit, is a reverse bias diode which is placed in parallel with the fuse and solenoid load. When the current is applied to the hammer solenoid the diode is reverse biased and no current will flow in the diode path. However when the current signal is removed from the hammer solenoid a negative voltage spike will be generated due to the inductive properties of a solenoid coil. The diode will then clamp the reverse voltage spike to approximately 0.7V (the built-in potential of the diode).

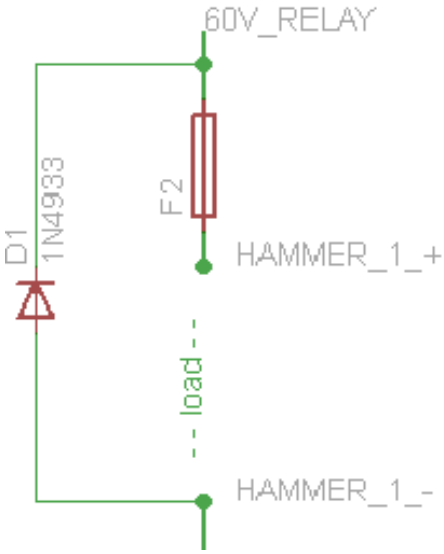


Figure 13 Diode and Fuse

Another protective measure is the PNP transistor, shown in figure 14, which will allow the circuit to remain in the "off" state if the input is floating. Since a floating signal tends to float in the positive voltage direction. This circuit prefference was requested by technical advisor Bob Arnold and is in keeping with IBM circuit design practices. Our decoder provides a similar protective quality although its active high input is not in keeping with the same preference. The decoder outputs are tied to ground when not selected, and additonally the address 00 does not correspond to a signal path. These design decisions are intended to prevent the unintentional firing of hammers if the microcontroller

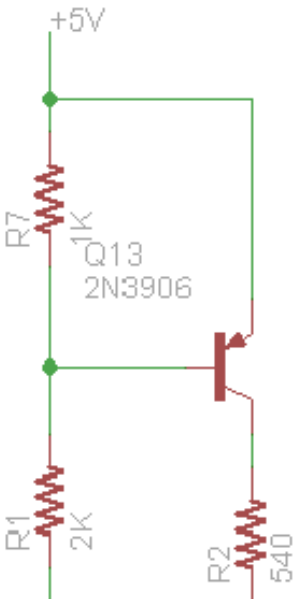
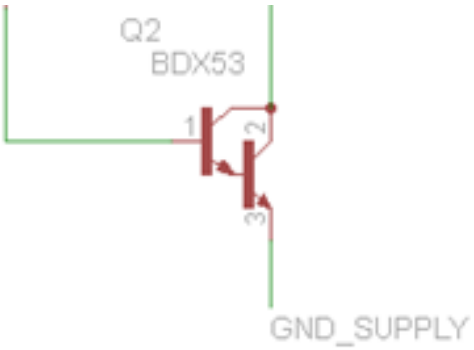


Figure 14 PNP Stage

outputs are in an undefined or floating state prior to initialization. The SPST switch located on the Project PCB serves a similar purpose. This switch will remain in the "off" position until sufficient time has passed, allowing the microcontroller to initialize the states of the output pins prior to the application of the 60V 5A source.



The final NPN stage, shown in figure 15, of the driver circuit is where the current signal is amplified, a darlington transistor was selected in order to ensure a large enough gain and a large enough maximum collector current to provide 5A draw to the hammer solenoid. A TO-220 package was selected for this transistor to allow the necessary thermal dissipation. The footprint of a TO-220 provides exposed copper on the surface of the PCB that may be used to increase the amount of thermal dissipation.

Figure 15 NPN Stage Two

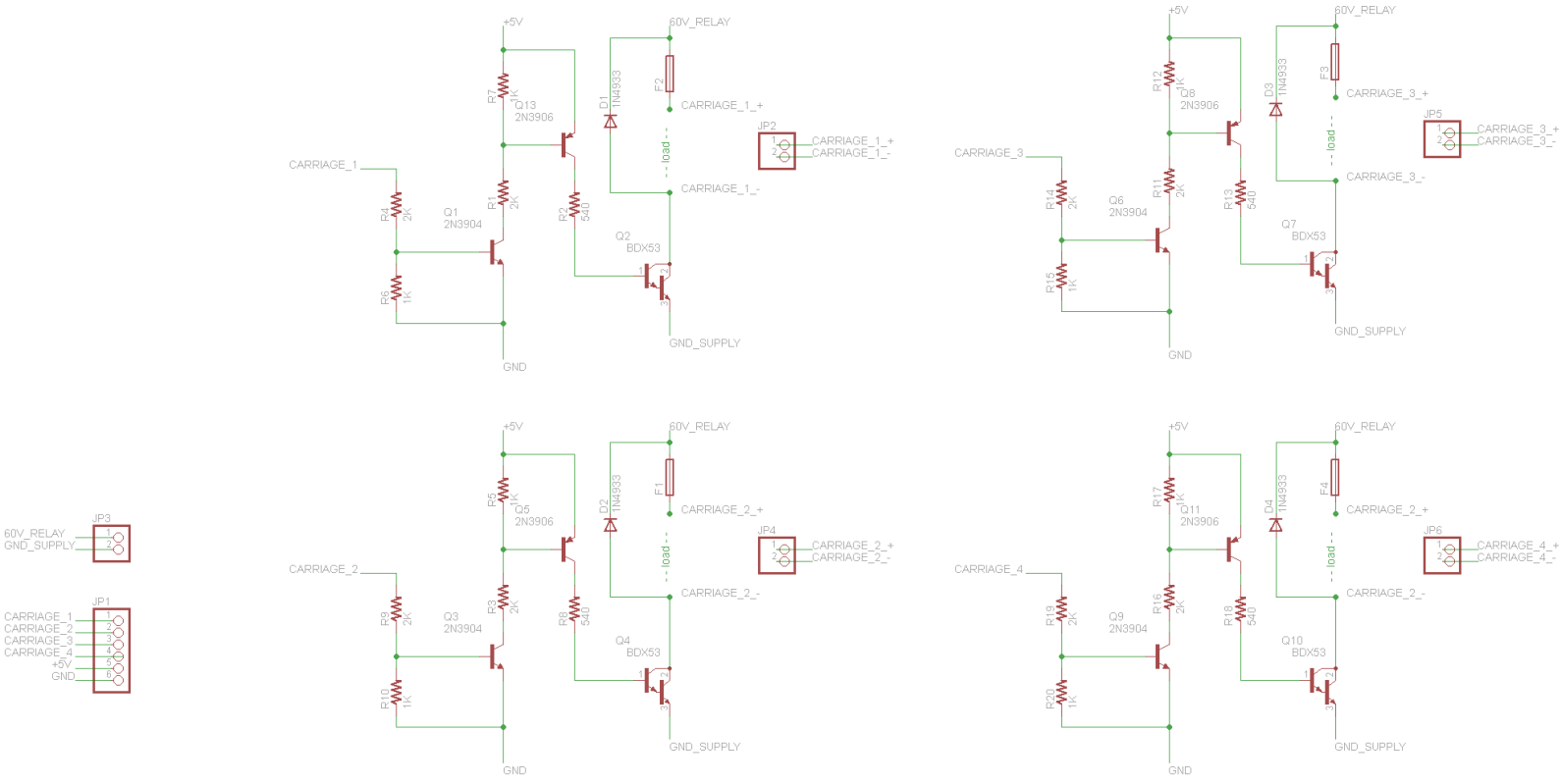


Figure 16 The Carriage Driver PCB

The Carriage Driver PCB receives an active high signal, originating from the microcontroller, via the Project PCB. The signal "Carriage 1" will control the "space start" magnet within the IBM1403-N1 Printer, the signal "Carriage 2" will control the "space stop" magnet, the signal "Carriage 3" will control the "skip start" magnet, and the signal "Carriage 4"

will control the “skip stop” magnet. These four solenoids each control a hydraulic valve which per forms the functions “space start,” “space stop,” “skip start,” and “skip stop.” A connection to the SPST switch controlled 60V 5A source is made via the Project PCB. The Carriage Driver PCB outputs will interface with the IBM1403-N1 printer by wire connection to the corresponding wire in the boot cable which was originally used to connect the IBM1403-N1 printer to a mainframe computer.

The various stages of transistors perform the same functions as in the Hammer Driver PCB. However, the gain of the transistors will be reduced in order to provide a low current signal to the solenoids controlling the “space start,” “space stop,” “skip start,” and “skip stop” valves. The signals produced by the Carriage Driver PCB will provide an “on” state to either the start or stop valves at all times for both the “space” valves and the “skip” valves.

The PC connected to the system will be loaded with a program with the operation of sending the text file data to the microcontroller. An example for the simple operations required for this transmission of data is included in the Appendices. The transmission of data will occur through the serial port. After initialization of the serial port, the program will create a ‘BufferedReader’ type variable to read the file character-by-character and store it to a string. After the end of the file is reached, the program will write each character to the port with a 100 millisecond delay between each write.

In synchronization with this program will be a program loaded onto the microcontroller with the operation of receiving the text file data. An example for this program is provided in the Appendices. In order to store the potentially large file sizes, the microcontroller will use a microSD card adapter in addition to the Arduino SD library. The adapter will interact with the microcontroller through SPI compatibility. Initially, the program will open a file on the microSD card with the write operation. Afterwards, the program will wait for the data from the PC by monitoring the serial port. When the text file data is sent from the PC, the program will read the serial data and write it to the text file on the microSD card.

```
Order of hammer firings and line spacing:

0 New Line(s)

On PSS pulse 13, hammer 3 will be fired with a 4.85us delay, to print T
On PSS pulse 18, hammer 2 will be fired with a 0.00us delay, to print S
On PSS pulse 19, hammer 0 will be fired with a 0.00us delay, to print T
On PSS pulse 101, hammer 1 will be fired with a 0.00us delay, to print E

1 New Line(s)

On PSS pulse 10, hammer 0 will be fired with a 0.00us delay, to print W
On PSS pulse 39, hammer 2 will be fired with a 0.00us delay, to print P
On PSS pulse 68, hammer 4 will be fired with a 4.85us delay, to print 2
On PSS pulse 70, hammer 3 will be fired with a 4.85us delay, to print 2
On PSS pulse 107, hammer 1 will be fired with a 0.00us delay, to print C
```

Figure 17 Printing Algorithm Simulation

The above screenshot shows the print operation for a text file with “TEST” on the first line and “WCP22” on the second line.

Upon reading the entire file, the microcontroller will perform a printing algorithm as shown in the simulation in Figure 17. The printing algorithm will determine for each line when to fire the printer hammers based on the PSS pulse. This is accomplished in software by creating an array that mimics the character belt of the IBM 1403-N1 printer. The program also mimics the operation of the printer by moving through the character belt in the same fashion that the physical

belt rotates in the printer. The printing algorithm also monitors the offset delay for the hammers, as well as the size of the blank lines for carriage control. The data accumulated from the printing algorithm will be utilized when the printing has started. By monitoring the PSS pulses during the actual printing, the program can use the data gained from the printing algorithm to determine when to fire the printer hammers without having to do much on the spot computation. The example for this code, along with a visual simulation for verification of the printing is included in the Appendices.

4.3 Concept of execution

Print Control (Cartridge) Mechanism:

The 1403 N-1 printer follows what is known as on the fly printing. The type, consisting of the characters, moves continuously behind a paper. Refer figure 18 below. The hammers press the paper against the moving type when the required character is in position. The key to this type of functionality is the knowledge of when to fire the right hammer.

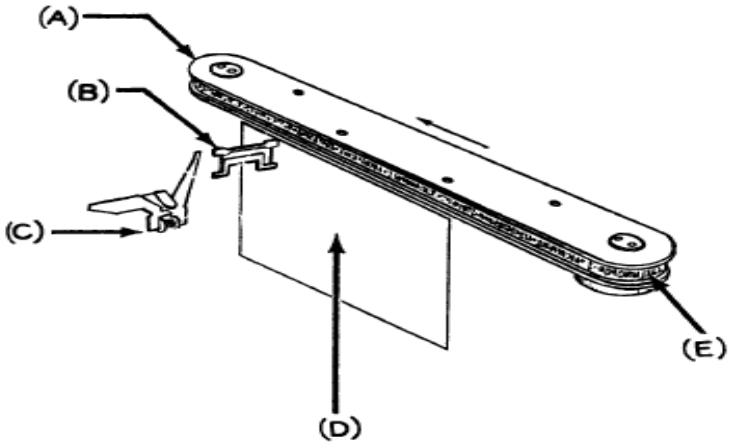


Figure 18 The Printer Cartridge

A – Type array
B – Hammer unit
C – Armature hammer magnet
D – Paper form – stationary during printing
E – Chain

The chain consists of 240 characters. The characters are divided into sets of 3 called slugs, so 3 characters per slug. The type moves at a speed of 206.4 inches per second. A print hammer is available at each print position. The distance between each print position or hammer unit is 0.100 inches. There are 132 hammer units and hence each line consists of 132 characters.

Important signals that need to be monitored are: emitter pulses, home pulses and PSS pulses. These signals come from the printer. In our project these signals are fed to a microcontoller that processes it and based on the timings of these pulse it makes decisions to fire which hammer and at what intervals.

Printing is carried out serially, one character at a time. At any given time one third of the hammers are lined up to some character on the type face. Hence 3 subscans, each containing 44

print positions, are needed to expose all characters to each print position. Figure 19 below outlines the printing operation.

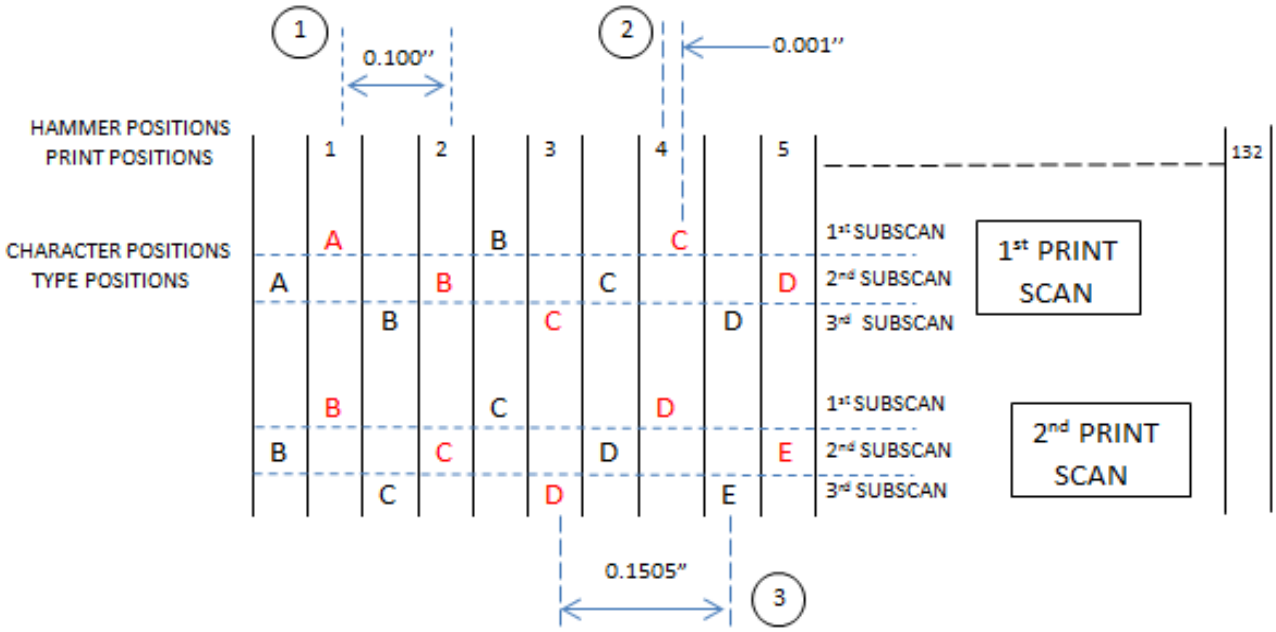


Figure 19 Scan and Subscan States

Characters in red are aligned with hammer.

1 – distance between two hammer/print postions
2 – offset of 2nd aligned character within a subscan
3 – distance between two characters on the type.

One complete scan brings character B to print position 1. Taking into account type speed and distance between two adjacent characters this takes 729 microseconds. Hence time for each subscan will be one third that value or 243 microseconds.

Emitter pulses are sent out at the beginning of each subscan. At the first emitter pulse hammers 1,4,7..,130 are aligned for printing. If character A is to be printed, hammer one can be fired and then hammer 4 is optioned for printing character C, 4.85 microseconds later. Print position 7 is then optioned for printing until one third or 44 hammer positions have been printed. Since type face is moving at a rate of 206.4 inches per second and offset is 0.001 inches it takes 4.85 microseconds for C to align with hammer 4. Offset is included for serial printing. In this fashion 44 hammer positions accumulate a total time of 44x4.85 = 213.4 microseconds of 243 microseconds available per subscan.
At the end of this time, second subscans begins with character B aligned with hammer two. In third subscan character C is aligned with hammer 3. At the end of the third subscan, one scan is complete and 44 characters have been exposed to 44 print posionts.
A home/sync pulse is received once the type has taken one whole revolution that is A is back at print positions 1.

Carriage mechanism:

In order to print multiple lines, the 1403-N1 sends out carriage signals which control line spacing as well as line skipping. The 1403-N1 is capable of printing 1100 lines per minute that is it takes 54 milliseconds per line. Line spacing is carried out by two magnets, space start magnet and space stop magnet. Either one is energized at all times with 60 V supply. At the start of a new line feed space start pulse is turned on. It is then turned off and space stop pulse is turned on. The start of the space stop pulse also signals a carriage-settling single shot signal to be pulsed. The carriage settling single shot allows time for the carriage to settle down after spacing or skipping and to control the start of the next print operation. Single line spacing takes 20 milliseconds. Hence time allowed for printing a single line is 34.55 milliseconds. Our microcontroller is fast enough to pick up these timings and be able to print on a new line after it receives the next PSS pulse.

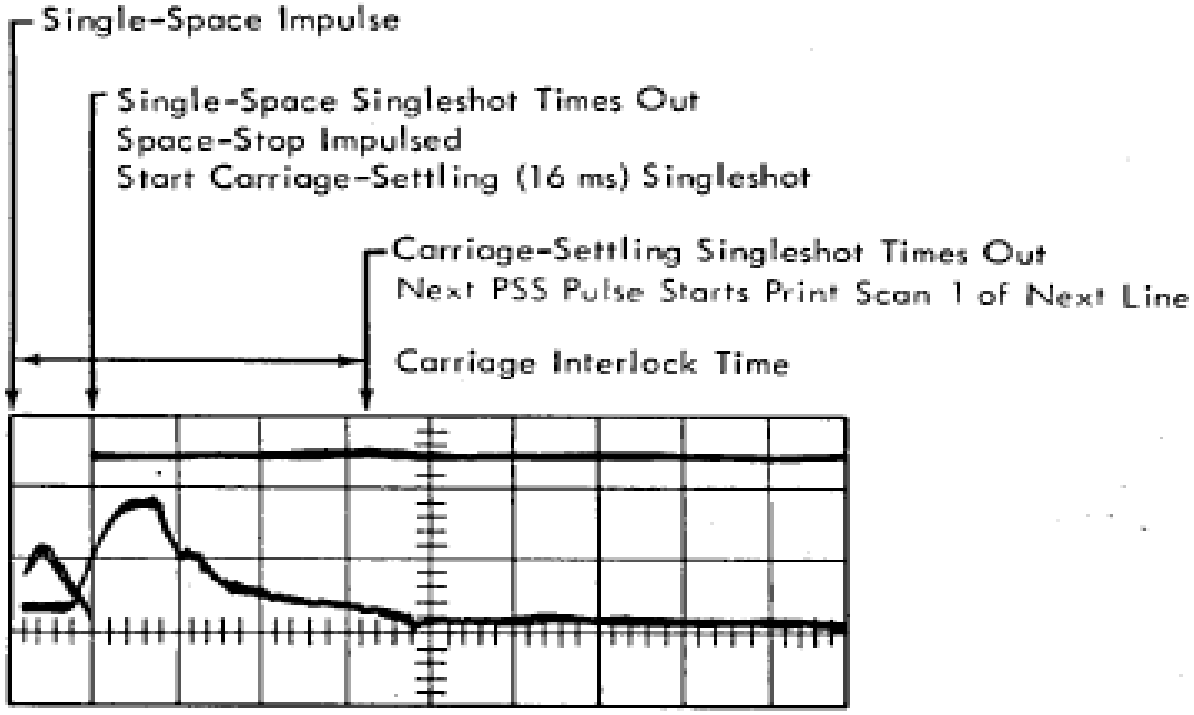


Figure 20 Space/Stop Magnet Operation

Item	1403 Printer Models 3 and N1	
1	Cartridge Type	Train
2	Number of Print Positions	132
3	Maximum Printing Speed (LPM)	1100
4	Train Motor Speed (RPM)	3600
5	Train Velocity (IPS)	206
6	Time Required for Type to Move .001" (Microseconds)	5.0
7	Setting for Calibration of Print-Timing Dial with Print Density Lever Set at C	Has no dial
8	Timing Disk Speed (RFM)	1714
9	Time of Carriage Start Impulse	Print SS-1 of Print Scan 46
10	Time Required to Print One Line with Single Space (milliseconds)	54.54
11	Carriage Interlock Time (ms)	20.8
12	Carriage Type (Speed)	Dual

Figure 21 Timings for 1403-N1 Printer

An emitter pulse called “E1,” also known in this document as the “carriage emitter pulse,” is used to verify the forms movement. For each line skipped it sends out one pulse. These emitter pulses are used to count how many lines have been skipped. They are primarily used for high speed skipping of more than 3 line spaces. For this purpose another set of magnets called skip start and skip stop magnets are used in conjecture with the space start and stop magnets. Both space and skip magnets are turned on at the same time. E1 pulses are counted and with 4 to 10 (adjustable) E1 pulses left, the skip magnet is turned off and with 3 to 6 (adjustable) E1 pulses to go space stop magnet is turned off.

Figure 21 above summarizes the timings discussed in this document regarding the print and carriage functionality.

4.4 Interface design

Internal interfacing of the microcontroller, Project PCB, Carriage Driver PCB, and two Hammer Driver PCBs will be accomplished with pin-head to pin-head connectors; these have been included in the budget and are estimated to cost 20 USD. The interface between the Project PCB and the IBM1403-N1 Printer will be accomplished by connection to the original boot-cable of the IBM1403-N1 Printer. WCP22 will provide wires or wire terminals depending on the needs of CT&I. The Hammer Driver PCB and the Carriage Driver PCB interface to the IBM1403-N1 Printer will be accomplished by connection to the original boot-cable, or through the original card slots on the IBM1403-N1 Printer depending on whether the boot-cable contains a wire connected to the driven signals or not.

5 Traceability and Testing

This project's Test Procedures and Results Report will provide traceability from the project-level requirements to the project's subsystems and major components after the Test Procedures review has been completed in the spring semester.

6 Notes

6.1 Acronyms and Abbreviations

CDR	Critical Design Review
CT&I	Center for Technology and Innovation
IBM	International Business Machines
PCB	Printed Circuit Board
PDP	Project Development Plan
PDR	Preliminary Design Review
PRS	Project Requirements Specification
SAR	System Acceptance Review
SDR	System Design Review
SPST	Single Pull Single Throw
SRR	System Requirements Review
TRR	Test Readiness Review
WCP	Watson Capstone Projects

6.2 Bibliography

Figure 6 - ***ieeexplore****.ieee.org A Development Study of the Print Mechanism on the IBM 1403 Chain Printer,B. J. GREENBLOTT January 1963.*

Figure 8 - SY24-3395-3, Printer Models N1 and 3Maintenance Manual, section 4.16

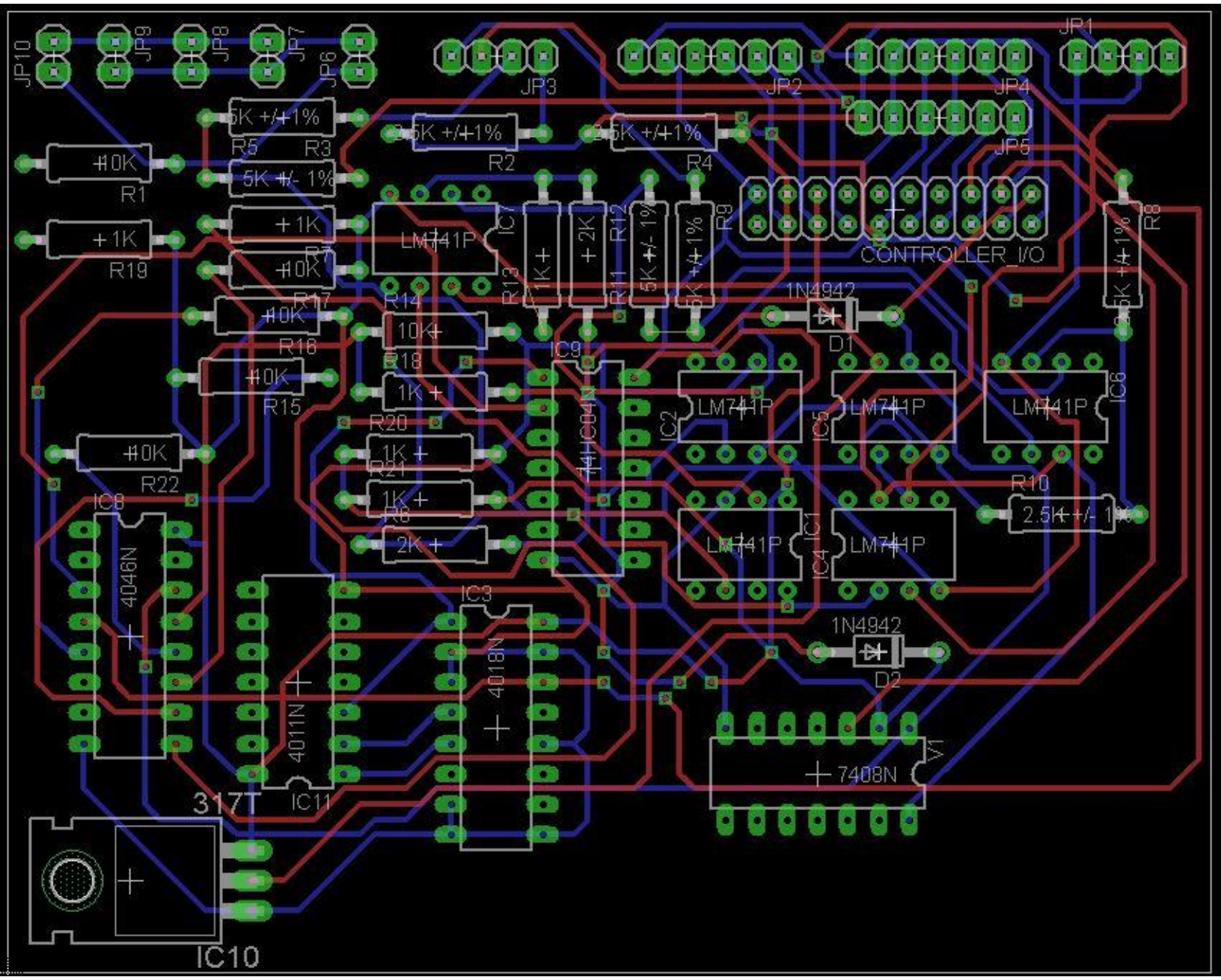
Figure 9 - SY24-3395-3, Printer Models N1 and 3Maintenance Manual, section 4.16.

6.3 Appreciation

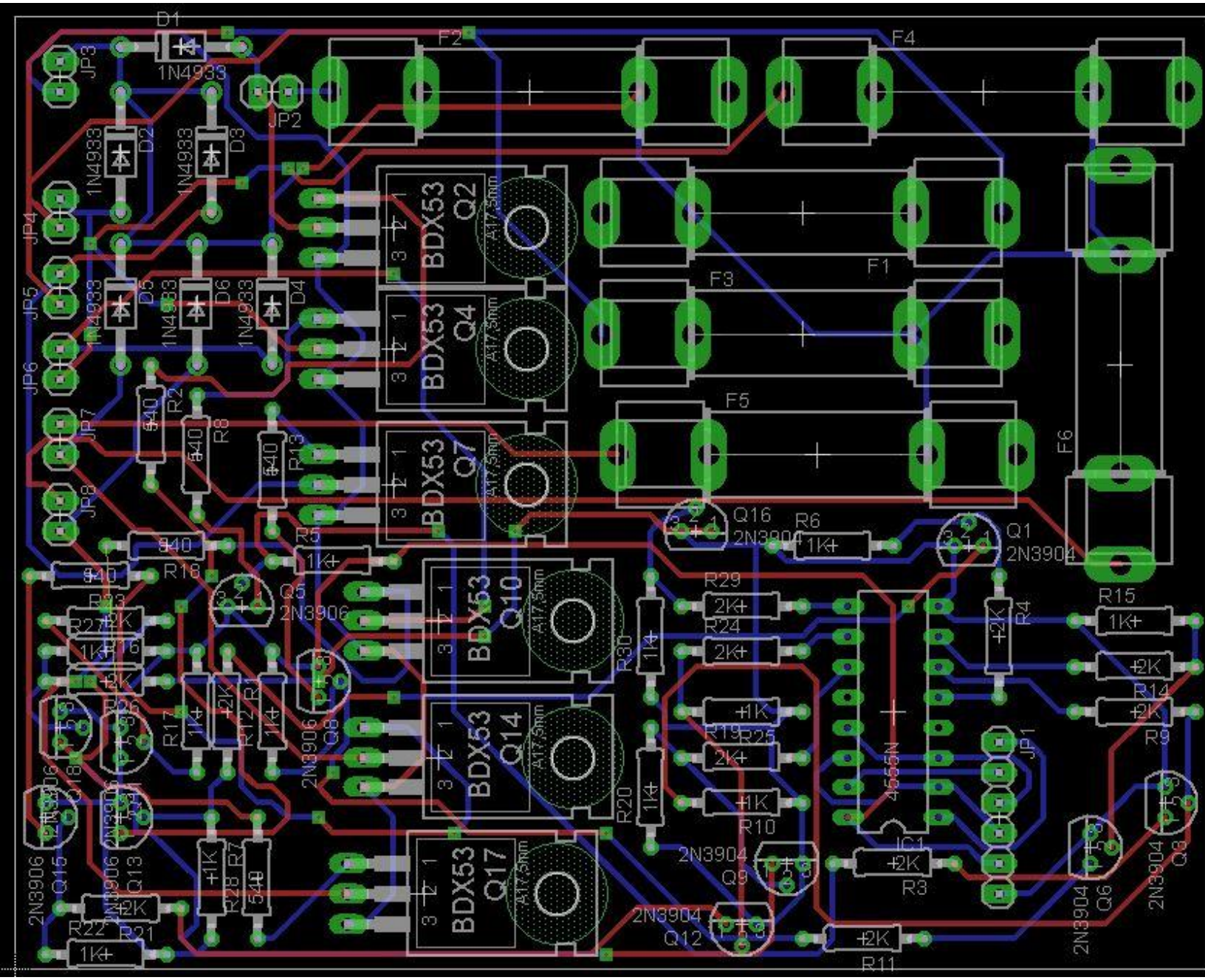
WCP22 would like to extend thanks to the IEEE Binghamton Chapter, the Center for Technology and Innovation, as well as, Professor Jack Maynard of Binghamton University, for invaluable contribution to the success of the IBM1403-N1 Printer Interface project.

A Appendices

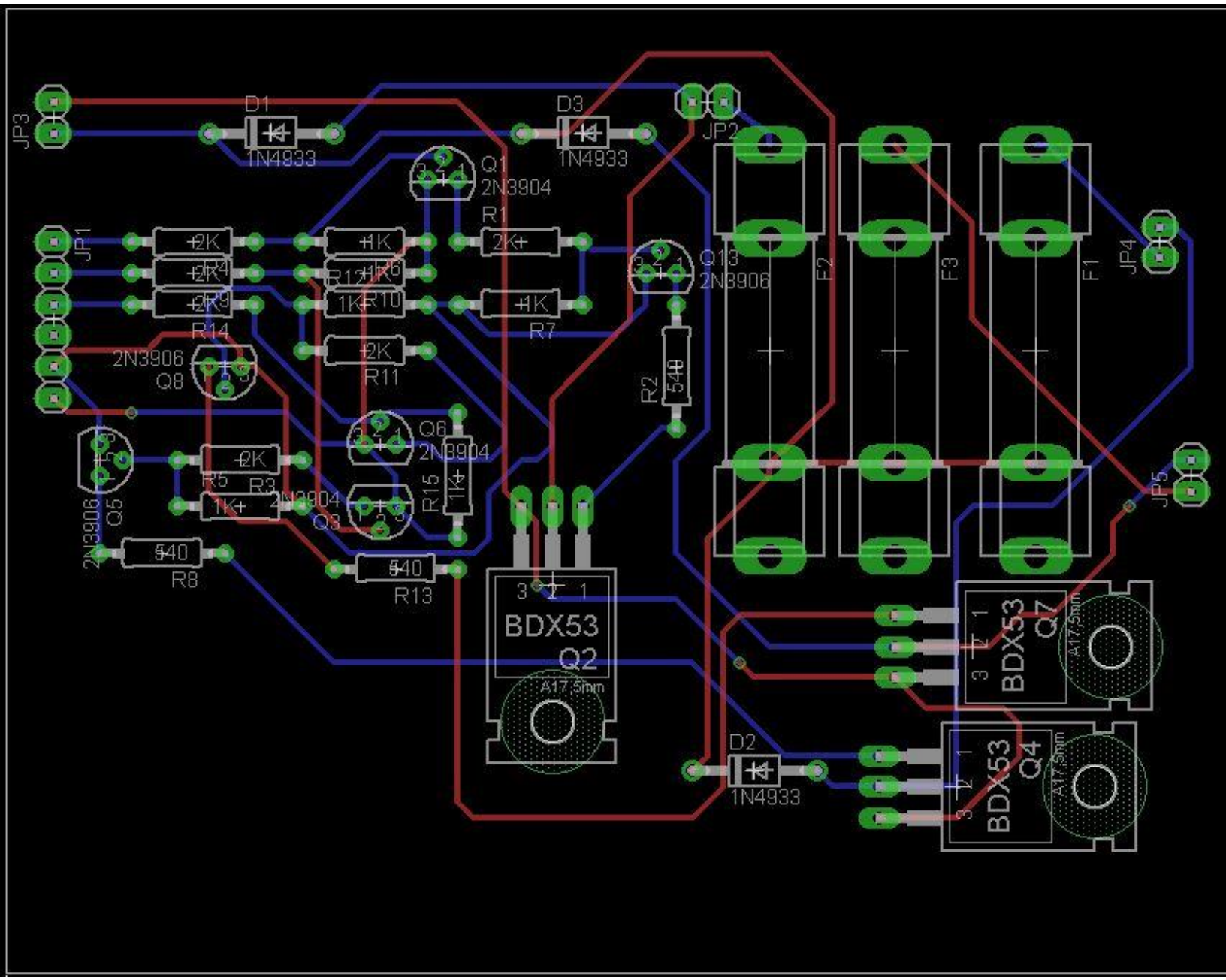
Project PCB Layout



Hammer Driver PCB Layout



Carriage Driver PCB Layout



Electronic Components – Bill of Materials

Index	Quantity	Part Number	Description	Customer Reference	Available Quantity	Backorder Quantity	Unit Price	Extended Price
1	2	296-21340-5-ND	IC PLL CMOS LOGIC W/VCO HS 16DIP	PHASE LOCK LOOP 1:3	2 Immediate	0	1.61000	$3.22
2	8	LM741CNNS/NOPB-ND	IC OPAMP GP 1.5MHZ 8DIP	LINEAR AMPLIFIER	8 Immediate	0	0.70000	$5.60
3	2	296-1570-5-ND	IC QUAD 2-INPUT AND GATE 14-DIP	AND LOGIC GATE	2 Immediate	0	0.36000	$0.72
4	4	296-8390-5-ND	IC DUAL 2TO4 DECOD/DEMUX 16-DIP	DECODER (DEMULTIPLEXER)	4 Immediate	0	0.43000	$1.72
5	25	BDX53CTU-ND	TRANSISTOR NPN 100V 8A TO-220	NPN TRANSISTOR HFE=1000	25 Immediate	0	0.55320	$13.83
6	30	KSA733GTAFSCT-ND	TRANSISTOR PNP 50V 150MA TO-92	PNP TRANSISTOR - HFE=100	30 Immediate	0	0.14920	$4.48
7	30	F3781-ND	FUSE CLIP 3AG/3AB EAR TIN PCB	FUSE CLIP	30 Immediate	0	0.17720	$5.32
8	1	454-1202-ND	PWR SUPPLY 60W 5/12/-5VOUT TRPL	+/-5V POWER SUPPLY	1 Immediate	0	64.92000	$64.92
9	2	296-14236-ND	IC DIV-BY-N COUNTR PRESET 16-DIP	DIVIDE-BY-N COUNTER	2 Immediate	0	0.57000	$1.14
10	1	432-1216-ND	SWITCH TOGGLE SPST 6A 125V	SPST TOGGLE SWITCH	1 Immediate	0	4.56000	$4.56
11	3	497-1485-5-ND	IC REG LDO 3.3V 1.2A TO220AB	3.3V REGULATOR	3 Immediate	0	0.87000	$2.61
12	2	296-9820-5-ND	IC HEX INVERTER 14-DIP	INVERTER LOGIC GATE	2 Immediate	0	0.36000	$0.72
13	2	296-1563-5-ND	IC QUAD 2-INPUT NAND GATE 14-DIP	NAND LOGIC GATE	2 Immediate	0	0.36000	$0.72
14	25	F4812-ND	FUSE 250V SLO-BLO 3AG 1.5A	1.5A FUSE	25 Immediate	0	0.82880	$20.72
15	50	S10KHCT-ND	RES 10K OHM 1/2W 5% CF MINI	RESISTOR - 10K	50 Immediate	0	0.04260	$2.13
16	10	PPC2.55KXCT-ND	RES 2.55K OHM 1/2W 1% AXIAL	RESISTOR - 2.55K 1%	10 Immediate	0	0.19500	$1.95
17	10	PPC5.10KYCT-ND	RES 5.1K OHM 0.4W 1% AXIAL	RESISTOR - 5.1K 1%	10 Immediate	0	0.17900	$1.79
18	50	CF12JT1K50CT-ND	RES 1.5K OHM 1/2W 5% CARBON FILM	RESISTOR - 1.5K	50 Immediate	0	0.05180	$2.59
19	100	CF12JT1K00CT-ND	RES 1K OHM 1/2W 5% CARBON FILM	RESISTOR - 1K	100 Immediate	0	0.03940	$3.94
20	50	CF12JT2K00CT-ND	RES 2K OHM 1/2W 5% CARBON FILM	RESISTOR - 2K	50 Immediate	0	0.05180	$2.59
21	50	100KWCT-ND	RES 100K OHM 1W 5% AXIAL	RESISTOR - 100K	50 Immediate	0	0.10480	$5.24
22	10	P3.6KBACT-ND	RES 3.6K OHM 1/4W 5% AXIAL	RESISTOR - 3.6K	10 Immediate	0	0.09000	$0.90
23	4	296-14286-5-ND	IC DECODER/DEMUX DUAL 16-DIP	DECODER (ACTIVE HIGH)	4 Immediate	0	0.57000	$2.28
24	4	497-1380-5-ND	IC DECODER/DEMUX DUAL 16-DIP	DECODER (ACTIVE HIGH)	4 Immediate	0	2.80000	$11.20
25	30	2N3904D26ZCT-ND	IC TRANS NPN SS GP 200MA TO-92	NPN - PRE-STAGE TRANSISTOR	30 Immediate	0	0.16760	$5.03
26	30	2N3906D26ZCT-ND	IC TRANS PNP SS GP 200MA TO-92	PNP - PRE-STAGE TRANSISTOR	30 Immediate	0	0.16760	$5.03
							Subtotal	$174.95
							Shipping	Estimate
							Sales Tax	unknown
							Total	unknown

Serial Writing Example

```
/*
This program is a simple implementation of the
program to send the text file to the microcontroller.

Operation:
This program will read from a text file (predetermined path).
The program will send one character at a time.
The program will send the entire file to the microcontroller.
*/

import processing.serial.*;
import java.io.*;
Serial port;

void setup()
{
    /*placeholder, will be initialized during implementation*/
    port = new Serial(this, "COM3", 9600);
    port.bufferUntil('\n');

}
void draw()
{
    int position = 0;
    /*get the path/filename of text file to read*/
    File readFile = new File("Filepath:/filename.txt");
    BufferedReader reader = null;
    String fileText = null;

    /*open the file for reading*/
    reader = new BufferedReader(new FileReader(readFile));

    /*read the file and store to fileText
    repeat until end of file*/
    while((fileText=reader.readChar())!=null);

    /*close the file*/
    reader.close();

    /*loop through the string by character*/
    while(position < fileText.length)
    {
      /*send the character through the USB*/
      port.write(fileText[position]);
      /*delay for 100 milliseconds*/
      delay(100);
      /*go to the next position in string*/
      position++;
    }


}
```

SD Operation and Serial Reading Example

```
This code is a simple implementation
demonstrating the microcontroller reading in the
text file data from the PC. The code also demonstrates
the utilization of the SD library for interaction with
the microsd card.

Operation:
Open the text file on the microsd card.
Read in the characters from the PC.
Store the characters to the text file.
Close the text file after completion.
*/

/*necessary library*/
#include <SD.h>
FILE bufferFile;

void setup()
{
	/*open the buffer.txt file on the microsd card*/
	bufferFile = SD.open("buffer.txt", FILE_WRITE);
}

void loop()
{
	byte character;

	/*wait for file from PC*/
	if(Serial.available())
	{
		/*read the character*/
		character = Serial.read();
		/*write the character to buffer.txt
		on the microsd card*/
		bufferFile.print(character);
	}
	else
	{
		/*after all data has been sent, close the file
		and will then be opened with read rights for the
		printing algorithm*/
		bufferFile.close();
	}
}
```

IBM 1403-N1 Printer Simulation

```
/*This program simulates the IBM 1403-N1 printer
for the printing of one 12 character maximum line.
This program includes updating graphics to follow
the movement of the character belt. When alignment
occurs, straight bars appear around the printer
hammer.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>

char printerCartridge[] = "ZYXWVUTS/'$=RQP-,0987654321*.+IHGFEDCBAONM%$&-
,0987654321LKJZYXWVUTS/($&RQP-,0987654321*.+IHGFEDCBAONMLKJ-
,0987654321*.)ZYXWVUTS/@$#RQP-,0987654321*.+IHGFEDCBAONMLKJ-
,0987654321*.)ZYXWVUTS/($&RQP-,0987654321*.+IHGFEDCBAONMLKJ-,0987654321*.)";
char stringToPrint[] = "HELLO WORLD.";
int printed[12] = {0};

int main()
{
	int counter;
	char dummy;
	int startingPosition=0;
	int iterator=0;
	int PSS=1;
	int stringCounter;

	/*simulates only the first rotation of the belt*/
	while(startingPosition<240)
	{
		/*determines starting position of hammers*/
		stringCounter=iterator;
		if(iterator==2)
			counter=startingPosition+1;
		else
			counter=startingPosition;

		/*loops for first line*/
		while(counter<(startingPosition+8))
		{
			/*compares to see if proper alignment occurs*/
			if(printerCartridge[counter] == stringToPrint[stringCounter])
			{
				/*simulates firing hammer*/
				printed[stringCounter] = 1;
			}

			/*increment the character and hammer counters*/
			stringCounter+=3;
			counter+=2;
		}

		/*below builds the graphical layout for the simulation*/
		printf("Printed:  ");
		for(counter=0; counter<12; counter++)
		{
			if(counter != 0)
```

```
                        printf("     ");

                    if(printed[counter] == 0)
                        printf("_");
                    else
                        printf("%c", stringToPrint[counter]);
                }
                printf("\n\n");
                if(iterator == 0)
                {
                    printf("Hammers:
|1|____2_____3____|4|____5_____6____|7|____8_____9____|10|___11____12\n");
                    printf("PSS %3d:  ", PSS);
                }
                if(iterator == 1)
                {
                    printf("Hammers:
1____|2|____3_____4____|5|____6_____7____|8|____9_____10___|11|___12\n");
                    printf("PSS %3d:  ", PSS);
                    printf("      ");
                }
                if(iterator == 2)
                {
                    printf("Hammers:
1_____2____|3|____4_____5____|6|____7_____8____|9|____10____11___|12|\n");
                    printf("PSS %3d:  ", PSS);
                    printf("   ");
                }

                /*determines spacing of character belt print out*/
                for(counter=startingPosition; counter<(startingPosition+8);
counter++)
                {
                    if(iterator == 0)
                        printf("%c        ", printerCartridge[counter]);
                    else if(iterator == 1)
                        printf("%c        ", printerCartridge[counter]);
                    else if(iterator == 2)
                        printf("%c        ", printerCartridge[counter]);
                }

                PSS++;

                if(iterator==0)
                    startingPosition++;

                iterator++;
                if(iterator==3)
                    iterator=0;

                /*pauses for user input*/
                dummy = getchar();
                printf("\n\n");
        }

        printf("\n");
        return 0;
}
```

Printing Algorithm with Descriptions

```
/*This program is the printer algorithm
with complete descriptions of the
print mechanism for corresponding
PSS pulses*/
#include <stdio.h>
#include <stdlib.h>

/*character belt*/
char cartridge[] = "ZYXWVUTS/'$=RQP-,0987654321*.+IHGFEDCBAONM%$&-
,0987654321LKJZYXWVUTS/($&RQP-,0987654321*.+IHGFEDCBAONMLKJ-
,0987654321*.)ZYXWVUTS/@$#RQP-,0987654321*.+IHGFEDCBAONMLKJ-
,0987654321*.)ZYXWVUTS/($&RQP-,0987654321*.+IHGFEDCBAONMLKJ-,0987654321*.)";
/*buffer of line to print*/
char *buffer;
/*tracks progress of printing*/
char *isPrinted;

int charCounter = 0;

int simulateRead(FILE *file);
FILE *simulateOpenFile();
void simulateCloseFile(FILE *file);
void determineLineRange();
int linePrinted(int start, int end);

int main()
{
       /*subscan pulses from printer*/
    int emitterPulse = 1;

       /*character belt trackers*/
    int charBeltCounter = 0;
    int charBeltPosition = 0;

       /*printer hammer trackers*/
       int offsetCounter = 0;
       int bufferCounterPosition = 0;
    int bufferCounter = 0;

       /*lines for carriage control*/
       int newLineCounter = 0;

       /*start and end of line in buffer*/
    int lineStart = 0;
    int lineEnd = 0;

    int isEOF = 0;
    int flag = 0;

       /*begin of simulate reading from PC*/
    FILE *simulateFile = simulateOpenFile();

    if(simulateFile==NULL)
    {
        printf("Simulation Error. This file could not be opened.\n");
        return 0;
    }
```

```
int stillReading = 1;
while(stillReading == 1){
stillReading = simulateRead(simulateFile);
}
   /*end of simulate reading from PC*/


   printf("Order of hammer firings and line spacing:\n");

/*simulate microcontroller loop*/
while(isEOF==0)
{
          newLineCounter = 0;

          /*start finding start and end location inside buffer*/
          if(flag==1 || buffer[lineStart]=='\n')
          {
        flag = 0;
                /*looking for start location*/
        while(true)
        {
            if(buffer[lineStart] == '\n' || (int)buffer[lineStart] == -1)
            {
                              newLineCounter++;
                              if(flag == 0)
                                     flag = 1;
            }

            if(buffer[lineStart] != '\n' && flag == 1)
            {
                break;
            }
            lineStart++;
        }
          }

    flag = 1;
    lineEnd = lineStart;
          /*looking for end location*/
    while(true)
    {
                if(buffer[lineEnd] == '\n')
        {
                        lineEnd--;
            break;
        }
        if(lineEnd == charCounter)
        {
                        isEOF = 1;
            lineEnd--;
            break;
        }

        lineEnd++;
          }
          /*finished finding start and end location in buffer*/
```

```
		printf("\n%d New Line(s)\n\n", newLineCounter);

		/*reset variables for new line*/
		bufferCounter = lineStart;
		charBeltCounter=0;
		bufferCounterPosition=0;
		charBeltPosition=0;
		emitterPulse=1;

		/*printing algorithm performed here*/

		/*loops until line fully printed*/
		while(linePrinted(lineStart,lineEnd)==0)
		{
			bufferCounterPosition = bufferCounter;
	charBeltPosition = charBeltCounter;

			/*loops until reaches end of line*/
			while(bufferCounter<=lineEnd)
	{
				/*checks if correct hammer aligned to correct
character*/
				if(buffer[bufferCounter] == cartridge[charBeltCounter]
&& isPrinted[bufferCounter] == 'x')
		{
					printf("On PSS pulse %d, hammer %d will be fired
with a %.2fus delay, to print %c\n", emitterPulse, bufferCounter-lineStart,
offsetCounter*4.85,buffer[bufferCounter]);
				/*signifies character printed*/
					isPrinted[bufferCounter] = 'o';
		}

				/*used for timing delay of hammers*/
				offsetCounter++;

				/*go to the next aligned character and hammer*/
			bufferCounter=bufferCounter+3;
			charBeltCounter=charBeltCounter+2;

				/*implements circular array for character belt*/
				if(charBeltCounter>=240)
				{
					charBeltCounter=charBeltCounter-240;
				}
			}

			/*reset timing delay for hammers*/
			offsetCounter = 0;

			/*reset hammer back to left most aligned position*/
			bufferCounter = bufferCounterPosition;
			/*move to the next aligned hammer for next pulse*/
			bufferCounter++;

			/*reset hammer alignment for next 3 PSS pulses*/
	if(bufferCounter==lineStart+3)
				bufferCounter=lineStart;
```

```
                        /*reset character back to left most aligned position*/
                        charBeltCounter = charBeltPosition;

                        /*move to the next aligned character for next pulse*/
                        if(emitterPulse%3==0)
                               charBeltCounter=charBeltCounter-1;
                        else
                               charBeltCounter++;

                        /*implements circular array for character belt*/
                        if(charBeltCounter >= 240)
                               charBeltCounter = charBeltCounter-240;

                        /*goes to next PSS pulse*/
                        emitterPulse++;
                }
        }

        simulateCloseFile(simulateFile);
        return 0;
}



/*for simulation purposes only*/
/*will simulate reading a file from PC*/
int simulateRead(FILE *file)
{

    /*string to write*/
    char readChar;

    /*loops through file*/
    while((int)((readChar = getc(file)) != -1))
    {
        charCounter++;

              /*dynamically allocates memory for character*/
        buffer = (char*)realloc(buffer,charCounter);
        isPrinted = (char*)realloc(isPrinted,charCounter);

        isPrinted[charCounter-1] = 'x';
        buffer[charCounter-1] = readChar;

        if(readChar == '\n')
            return 1;
    }

    return 0;
}

/*for simulation purposes only*/
/*will simulate opening a file on PC*/
FILE *simulateOpenFile()
{
    char *fileReadName = "toPrint.txt";
    /*file to write to*/
    FILE *file;
```

```
    /*opens file pointer to write*/
    file = fopen(fileReadName, "r");

    return file;
}

/*for simulation purposes only*/
/*will simulate closing a file on PC*/
void simulateCloseFile(FILE *file)
{
    fclose(file);
}

/*tracks progress of printed line*/
/*0 means not fully printed, 1 fully printed*/
int linePrinted(int start, int end)
{
    int counter;
    for(counter=start; counter<=end; counter++)
    {
        if(isPrinted[counter] == 'x')
            return 0;
    }
    return 1;
}
```