ND PCR – Device Documentation

Frank Kuhny

Howard Research Group

University of Notre Dame

Summer 2013

Overview

ND PCR is a device made by students in the Howard Research Group in the Electrical Engineering department at the University of Notre Dame. The goal of this device is to provide a field-portable means of amplifying DNA for any of a number of applications including testing for invasive species of aquatic animals in the Great Lakes waterways. The following report documents the current state of both the hardware and software that is being used to run ND PCR. It is intended as a reference to anyone who intends to begin work with any part of the device. The code referred to in Section 1 can be found in Appendix A of this document, and the schematics referred to in section two can be found in Appendix B of this document.

***Section 1: Code***

*Declaration of variables/globals:*

The beginning part of the code for ND PCR declares all of the variables and globals that are used throughout the program. This part of the code takes place from line 1 to line 123. The latter parts of the code are the functions used in the code as well as the operational part of the code.

Lines 1 through 26 are introductory comments to the program stating contributors to the code. A disclaimer also states that the program itself is software for free redistribution and modification.

Line 28 of the code begins the definition of the various libraries that will be used to run the thermocycler. Of particular interest is the PID library [1]. PID stands for proportional-integral-derivative. What this library does is calculates the output power required to heat the heater to the desired temperature setpoint as quickly as possible with minimal overshoot. The other libraries are the libraries to control the various components of the device such as the keypad and the display.

The code from lines 34 to 59 defines the keypad to have four rows and three columns. This part of the code also defines an object “keypad” with the class “Keypad” from the previous library definition [2]. When keypad is called later within the program it uses the input of the row and column to define the variable “keypad” to be one of the values ‘0’-‘9’, ‘\*’, or “#” based on which key is pressed on the actual keypad. Also the definition of the pins the keypad will use on the microcontroller is here. The keypad uses pins 7-13.

After this, the variables for the PCR process are defined. These are the variables that change based on the specific recipe required for the primers that are being used. The variables ‘INITIALIZATION’, ‘DENATURATION’, etc. are the variables that will be used for the temperatures for the heating block for that specific step (the same step that the name denotes). Likewise, the variables ‘INIT\_TIME’, ‘DENAT\_TIME’, etc. are the variables to denote the amount of time the heating block will be held at the temperature for that step before moving on to the next step.

The next set of lines is the group of function prototypes that will be used at multiple points later in the code. These will be covered more extensively at a future point. Lines 98 to 114 initialize many variables that will be used throughout various functions. The number of cycles begins at zero and is incremented each cycle.

Lines 101 and 102 define the output pins for the heater and buzzer. In the code they are defined as 3 and 5, but there is an important distinction to be made here. These pin numbers do not refer to the pins on the microcontroller itself, but rather Digital Input/Output pin number that can be found on the microcontroller’s pinout diagram. The pin number for Digital I/O 3 is pin 1, and the pin number for Digital I/O 5 is pin 9. The pin to which the buzzer is connected is pin 26 on the microcontroller, but instead of using the pin number, ‘A3’ is written for the current setup as specified by the pinout diagram [3].

The final part before the setup() function defines the ‘myPID’ variable in the PID class. This is the code’s variable for changing the temperature based on the variables input, output and setpoint variables defined in line 117. Line 122 defines the myPID variable itself with the address operator (&) to the input, output, and setpoint variables. The numerical values in line 122 are the values for Kp, Ki, and Kd, respectively, which control the rise time, overshoot, settling time, and steady-state error of the PID controller [4].

*Operating Procedure:*

Lines 124 through 219 comprise the operating procedure for the code. This includes the setup() and loop() functions for the code. From within these functions, all the other functions of the code are called, and those will be described later. The setup() function gets the desired recipe from the user. The loop() function within the code repeats the procedure for the desired PCR recipe by using the PID function to change the temperature of the heating block as necessary.

The setup() function begins with line 124 of the code. This function begins serial communication at the specified frequency, here 9600 baud. Also, the pins for the heater and fan are set as output pins using the pinMode() function. The setup() function also calls the recipeChoice() function to determine which recipe preset will be used when running the PCR process. Using the selected recipe, the heater then brings the heating block to the selected “INITIALIZATION” temperature and delays the program until that temperature has been reached while outputting the “WAIT FOR HEATERS” message from line 139. Lines 143 to 150 are implemented to give the user time to put the vials into the PCR device; the LCD screen is cleared and outputs the given message and waits for the user. The output message says to press 1 to start the process, but the code allows for any button to be pressed to continue with the waitforkey() command. The final part of the setup() function prints the headers to the LCD screen as well as to the Serial monitor and turns on the PID.

The next function, which begins on line 166, is the loop() function which runs the PCR thermocycler itself following setup(). The arguments of the settemperature() function are first the temperature to which the heating block will be set followed by the amount of time it will be held at that temperature. First, the heating block is brought to the initialization temperature for the given amount of time as the beginning denaturation step. Then, the “for” loop beginning in line 173 runs the thermocycling itself based on the recipe. Line 179, the denaturation step, splits the DNA into single strands; line 180 bonds the primers for the DNA onto the single strands produced in the denaturation step; line 181 binds the complementary DNA onto the single strands in the extension step. For every iteration of the process, line 176 adds one to the current value of “i” until the program has gone through the correct number of cycles as specified by the recipe.

After all the cycles have been finished, the heating block is brought to the final elongation temperature for the given time in line 188. The final hold step (refrigeration) occurs in line 190. As soon as all the steps have finished, the code displays a message on the LCD screen to remove the vials and press a key to end. During this time, the buzzer is being sounded from line 200 until any key on the keypad is pressed. Realistically, the code from lines 193 to 204, the “remove vials” message as well as the buzzer sound, should be moved just after the final elongation step in line 188 for the device as the low final hold temperature cannot usually be reached with the current setup. This way, the user can remove the vials manually and refrigerate them in with some other means. Finally, in the current code, after the key has been pressed to end the buzzer, a final message loops indefinitely saying that the run has been completed and to refrigerate the vials.

*Functions:*

The final section of the code for ND PCR includes the rest of the functions used within the setup() and loop() functions. These functions are employed to control the temperature of the heating block and output data to the LCD display and serial output.

Following the thermocycling loop() function is the recipeChoice() function which begins on line 221. A simple message is output to the LCD display asking the user to choose a number one through three based on what recipe they desire. Currently recipes one and two could simulate realistic PCR processes although the recipes might need to be changed for slight temperature differences or slight differences in time for a certain step based on the specifications for the primers that are being used. Recipe number three exists as a test cycle that can be run through fully within a couple minutes rather than waiting for an entire PCR process to run which takes multiple hours. This allows for testing the heating block temperature for each step, the time on each step, and the overall number of cycles that were run for one process to ensure robustness of the code. After one of the buttons has been pressed to select a recipe, a confirmation message as to which recipe has been chosen is output to the LCD display.

The settemperature() function which begins on line 312 controls the heating block to the desired temperature when called. Lines 315 and 316 use the millis() function, which determines the time in milliseconds since the current program began running. The while loop which begins running in line 320 and ends in line 337 moves the heating block from the last step’s temperature to the current step’s temperature. It also outputs to the screen every second. Once the temperature for that step has been reached within one degree Fahrenheit, the program maintains that temperature for the desired time, “settime.” This time is added to the current time of the program which gives the value of “time2” in line 341, and the temperature is held until the new time is reached; then the function ends.

When the readTemps() function is called, the board takes a 5-bit value for the analog voltage read from each of the temperature sensors on the aluminum block, from pins 22 and 23 (A7 and A0 respectively for the Analog/Digital pin values). These values represent the voltage across the sensor to ground. Lines 371 takes the sensor value and calculates the resistance of the thermistor at that time given that the thermistor is part of a voltage divider. The value of the other resistor is known to be 10k Ohms, so the value of the thermistor can be calculated based on that and the value of the voltage across the thermistor. This value is then taken and put into line 372 which is the Beta equation for calculating the temperature of a thermistor:

T={B\over { {\ln{(R / r_\infty)}}}}

The value 3560 is the Beta value for the specific thermistor being used [5], and the value for r**∞** is a constant based on the Beta value from the following equation:

r_\infty=R_0 e^{-{B/T_0}}.

where R0 is the resistive value of the thermistor at room temperature. The final output is in Celsius, so 273.15 is subtracted from the temperature in Kelvin. The same process is used in lines 373 and 374 to determine the temperature of the other thermistor. To simplify the results that are achieved, the two temperatures are averaged to give the subsequent temperature of the heating block.

The third to last function of the program is the printData() function. The outputs to the LCD display are the time remaining as well as the current temperature of the heating block, separated by a space. The outputs to the serial display, as can be easily seen in the code, are the current step’s desired temperature (setTemp) followed by the two temperatures of the thermistors, the “Output” from the myPID variable, and finally the time remaining on the step.

The second to last function in the code is the controltemp() function. This is called from within the settemperature() function described above. The temperature of the heating block is read in line 418 from the previous readTemps() function, and the fan is either turned on from lines 422 to 426 if the temperature is too high, or is turned off if the temperature of the heating block is too low. Line 430 changes the new input to the myPID variable to the current temperature from the readTemps() function and the next increment in temperature is computed in line 433. The corresponding output in temperature is then sent to the heater pin in line 436 using analogWrite().

The final function of the code is the calcTimeRemaining() function which returns an integer guess as to how much time in minutes is remaining for the entire PCR process. The total time for all the steps (“totTime”) is computed in line 433 by adding the initialization step time to the final elongation time, the hold time, and the number of cycles multiplied by the quantity of the denaturation step time, the annealing step time, and the extension step time. Line 446 calculates the approximated time for the heating up and cooling down between steps to be about 0.07 times the total time for all the steps. This number is added, in line 449, to the “totTime” variable to give us the approximated time for the entire process including ramp up and down times in temperature, “totalTime.” The final line returns the value of the amount of time that has passed subtracted from the total time to give us the estimated amount of time remaining until the whole process has been completed.

***Section 2: Schematics***

The two boards that are currently being used to run ND PCR can be broken down into eight different sections by their various purposes. These sections are as follows: power supply, microcontroller, heater and serial output, LCD interface, temperature monitoring equipment, keypad, serial to USB converter, and buzzer alert. The first six were implemented on the parent board while the last two were added to the functionality of the overall device as a result of the daughter board. Each of these sections are described and discussed in greater detail in the following segments of this document.

*Power Supply:*

The power supply and voltage regulator for the ND PCR is depicted in Section A of Figure 1 in Appendix B. Part J2 in this section is the connector for the input power from the battery. The 12V from the power jack connects directly to the rest of the 12V wires on the board as well as connecting to the voltage regulator; any wire connecting to a voltage source with V+ has 12V on it. The voltage regulator can be seen as part IC2 on the schematic. This is a MC7805CT voltage regulator which takes the 12V from the power jack and steps it down to 5V. A schematic for the inner workings of the MC7805CT can be found on its data sheet and involves a complex network of over 20 transistors in order to make this step down in voltage. The output of the voltage regulator connects to all wires with a VCC denotation and thus has 5V on it relative to ground.

*Microcontroller:*

Section B of the parent board comprises the microcontroller and support components. Pin 29 connects to the reset button on the ICSP (in-circuit serial programmer) for when the chip needs to be reset when being programmed. Pins 4, 6, and 18 connect to the 5V from the voltage regulator to power the chip; pins 3, 5, and 21 connect to the ground pins on the chip. Across pins 7 and 8 is a 16 MHz crystal oscillator which is required for the clock of the microcontroller.

The microcontroller that is used is the ATMEGA328P with TQFP package.

*Heater and Serial Output:*

The connector pins denoted by JP5 in Section C of the parent board are for serial output communications. Also in this section are the two thermistors that are being used to heat the current heating block. They are NTC2K radial leaded thermistors. As stated in Section 1, they are connected into a voltage divider setup, and the voltage across each of them is read as a 5-bit value into the microcontroller to determine their temperatures to be heated or cooled appropriately. Finally, a battery check is in place to ensure power is coming from the battery. This connects the voltage across a 50k Ohm resistor to pin 19 on the microcontroller.

*LCD Interface:*

Section D of the parent board consists of a set of 12 pin headers, 5 of which connect to pins on the microcontroller. Pins 1 and 2 on the pin header are connections to 5V and ground, respectively. Pin 3 on the pin header is connected to a potentiometer. The main device that is connected to this pin header is the LCD display which uses three of the pins. A temporary buzzer is also connected to one of the pins for testing but will be implemented otherwise in the future.

*Temperature Monitoring:*

Section E consists of two sets of connectors to other components used in the PCR device. The JP4 connector pins are used for the heater to control the heating block. One of the pins is connected to 12V and the other is connected to the collector of the NPN transistor. This one, the TIP120, is a Darlington transistor which is a single device which has two cascaded NPN transistors sharing a common collector and with the emitter of one connected to the base of the other. The advantage of this device is that it allows for a much higher DC current gain than a single transistor could. For the TIP120, the current gain, also denoted by hFE, has a minimum value of 1000. The 500 ohm resistor connected to the base of the Darlington transistor ensures that the device operates in saturation. Similarly, JP1 is the set of connector pins to the fan. The fan is also connected on one side to the 12V power source but is instead connected to a PN2N2222A silicon transistor. This is only a single NPN transistor with minimum hFE values ranging from only 30-100. Likewise with this transistor, a resistor (this time 1K Ohm) is connected to the base to ensure the transistor operates in saturation when on. A capacitor is connected across both of these circuits, for the fan and heater controls, to provide positive feedback to them.

*Keypad:*

The final part of the parent board, Section F, contains the pin connectors for the keypad as well as pin connectors for the in-circuit serial programmer.

*Serial to USB converter:*

The daughter board consists of two sections itself. The first, Section G, is a USB to TTL serial converter. The chip that is used is the FT232RL chip. The setup and implementation for this circuit can be found on www.sparkfun.com/tutorials/108.

*Buzzer alert:*

Section H on the daughter board is the circuit for the buzzer as well as connector pins. The circuit for the buzzer is similar to the one for the fan on the parent board, having the pins across the buzzer connect to VCC and to the collector of the NPN transistor. This time though, a BC547 transistor is used as well as a 10k Ohm resistor going from the input pin to the base to ensure the device operates in saturation. The final set of connector pins has a pin for serial transmission, serial receiving, the buzzer input, VCC, and ground.

***Section 3: Future Work***

There are a few additional operations which ND PCR will hopefully be running by next summer. The first of these would be getting the serial communications to work so that data from the device can be sent to a computer. With the addition of the serial to USB converter this past summer, getting these communications running should require little, but still some, testing.

Another goal for ND PCR in the near future would be to get multiple wells working for the PCR process whereas only one well of the current heating block works successfully. There are likely two steps that need to be done so that this will work well. First the wells will likely need to be run separately instead of attempting to run three (or however man) on the same heating block. Dissipation of heat is not even across the current heating block. For multiple wells to work in the future, a more powerful microcontroller will need to be used to control separated single wells, but this will also require a redesign of the cooling/fan system in the current device.

The final goal for ND PCR in the near future is the joining of the device with the testing device used in the lab run by Dr. Hsueh-Chia Chang. With the uniting of the two devices, not only would DNA be amplified but it would also be tested for being of a specific type of animal all at once.

Appendix A

1. /\* PersonalPCR, code for Arduino based PCR
2. Copyright (C) 2011 Chris Templeman <templemanautomation.com>
3. 2012 Scott Howard <showard@nd.edu>
4. Authors: Chris Templeman
5. Matt Brittan
6. Alex Toombs
7. Elizabeth Hunschke
8. Frank Kuhny
9. Date Last Modified: 5/19/13
10. This program is free software: you can redistribute it and/or modify
11. it under the terms of the GNU General Public License as published by
12. the Free Software Foundation, either version 3 of the License, or
13. (at your option) any later version.
14. This program is distributed in the hope that it will be useful,
15. but WITHOUT ANY WARRANTY; without even the implied warranty of
16. MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17. GNU General Public License for more details.
18. You should have received a copy of the GNU General Public License
19. along with this program.
20. If not, see <http://www.gnu.org/licenses/>.
21. \*/
22. #include <PID\_v1.h>
23. #include <math.h>
24. #include <SoftwareSerial.h>
25. #include <serLCD.h>
26. #include <Keypad.h>
27. //Four rows
28. const byte ROWS = 4;
29. //Three columns
30. const byte COLS = 3;
31. //Define the Keymap
32. char keys[ROWS][COLS] = {
33. {
34. '1','2','3' }
35. ,
36. {
37. '4','5','6' }
38. ,
39. {
40. '7','8','9' }
41. ,
42. {
43. '\*','0','#' }
44. };
45. //connect to the row pinouts of the keypad
46. byte rowPins[ROWS] = {
47. 12, 7, 8, 10 };
48. //connect to the column pinouts of the keypad
49. byte colPins[COLS] = {
50. 11, 13, 9 };
51. Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
52. // PCR Temperatures in C, to be set by recipe or user with keypad
53. // Presets set in recipeChoice function
54. double INITIALIZATION = 95;
55. double DENATURATION = 95;
56. double ANNEAL = 55;
57. double EXTENSION = 72;
58. double FINAL\_ELONGATION = 72;
59. double HOLDT = 10;
60. // Number of cycles, input in recipeChoice
61. int NUMBER\_OF\_CYCLES = 35;
62. // PCR cycle times, corresponding to above double temps.
63. // Set in recipeChoice function.
64. double INIT\_TIME = 30;
65. double DENAT\_TIME = 30;
66. double ANNL\_TIME = 60;
67. double EXT\_TIME = 60;
68. double FIN\_TIME = 300;
69. double HLD\_TIME = 20;
70. // Definie the serial LCD object and communication pin
71. const int pin = 19;
72. serLCD lcd(pin);
73. // core function protoypes
74. // Directs user to select a recipe
75. void recipeChoice();
76. // Calls controltemp() to bring temp to a new level, hold it for a set time, calls printData() to display what is happening
77. void settemperature(double settemp, long settime);
78. // Prints data to LCD screen and serial monitor (time remaining in minutes, current temp)
79. void printData(double setTemp, int timer);
80. // Reads voltages from thermistor/resistor voltage dividers and calculates temperatures
81. void readTemps();
82. // Uses PID control with measurements from readTemps to set power to heater and fan
83. void controltemp(boolean fan\_on, unsigned long timerinit, unsigned long timer, double settemp);
84. // track number of cycles
85. int cycle = 0; //"global" because used by serial print
86. // heater, fan, and buzzer pins
87. const int heater = 3;
88. const int fan = 5;
89. const int buzzer = A3;
90. // temperature for heater 1
91. double temp1 = 0;
92. // temperature for heater 2
93. double temp2 = 0;
94. // averaged temps
95. double temp = 0;
96. // will output every one sec
97. unsigned long pulse=0;
98. // Define PID variables we'll be connecting to
99. double Setpoint, Input, Output;
100. //Specify the links and initial tuning parameters, also do so in
101. //loop() with SetTunings
102. // tunings are setup and material specific
103. PID myPID(&Input, &Output, &Setpoint,33.01,0.35,120.55, DIRECT);
104. // Set up variables, recipe choice, and headers
105. void setup()
106. {
107. // Begin serial communication at 9600 baud
108. Serial.begin(9600);
109. // set output pins to output
110. pinMode(heater, OUTPUT);
111. pinMode(fan, OUTPUT);
112. // Get recipe to control cycle
113. recipeChoice();
114. // print delay message until heaters are at INITIALIZATION point
115. while(temp > INITIALIZATION) {
116. lcd.selectLine(1);
117. lcd.print("WAIT FOR HEATERS");
118. }
119. // after heats are heated, wait for input before starting recipe
120. lcd.clear();
121. lcd.selectLine(1);
122. lcd.print("Load vials");
123. lcd.selectLine(2);
124. lcd.print("Hit 1 to start");
125. // wait until any key is pressed.
126. keypad.waitForKey();
127. // Print headers on LCD
128. lcd.clear();
129. lcd.print("min\_left T");
130. lcd.selectLine(2);
131. // Print to Serial Monitor on Laptop
132. Serial.print(" s cyc sp T1 T2 output time\_rem");
133. //turn the PID on
134. myPID.SetMode(AUTOMATIC);
135. }
136. // Doesn't actually loop like it's supposed to. This will only loop once, in theory
137. void loop()
138. {
139. // Set the temperature to remain at INITIALIZATION for INIT\_TIME seconds
140. settemperature(INITIALIZATION, INIT\_TIME);
141. // Loop for PCR recipe while current cycles is less than total number of cycles, as specified in recipe
142. // Real loop is here!
143. for (int i = 0; i < NUMBER\_OF\_CYCLES; i++)
144. {
145. // increment cycle
146. cycle++;
147. // step through each step of process
148. settemperature(DENATURATION, DENAT\_TIME);
149. settemperature(ANNEAL, ANNL\_TIME);
150. settemperature(EXTENSION, EXT\_TIME);
151. }
152. cycle++;
153. // PCR process loop ended; elongating, holding, then buzzing
154. // Final
155. settemperature(FINAL\_ELONGATION, FIN\_TIME);
156. // Hold at HOLD\_TEMPERATURE for a long time
157. settemperature(HOLDT, HLD\_TIME);
158. // Play buzzer until button hit, as PCR is done, tubes should be removed
159. lcd.clear();
160. lcd.selectLine(1);
161. lcd.print("REMOVE VIALS!");
162. lcd.selectLine(2);
163. lcd.print("Hit Key For End");
164. // play annoying buzzer until they hit a key to alert for vial removal
165. digitalWrite(buzzer, HIGH);
166. // buzzer should play until a key is pressed
167. keypad.waitForKey();
168. digitalWrite(buzzer, LOW);
169. lcd.clear();
170. // display end message, loop infinitely
171. while(1) {
172. lcd.selectLine(1);
173. lcd.print("Fridge vials!");
174. lcd.selectLine(2);
175. lcd.print("Run complete.");
176. // delay to prevent flicker
177. delay(500);
178. }
179. }
180. // Either choose a preset recipe for temperatures
181. void recipeChoice() {
182. // Clear LCD, select line 1
183. lcd.clear();
184. lcd.selectLine(1);
185. // Welcome message displayed
186. delay(1000);
187. lcd.print("Welcome to NDPCR");
188. delay(2000);
189. lcd.clear();
190. lcd.clear();
191. lcd.selectLine(1);
192. lcd.print("Enter choice:");
193. lcd.selectLine(2);
194. lcd.print("(1 thru 3)");
195. // Recipe choice input (char digit)
196. char c2 = keypad.waitForKey();
197. // Sets recipes, can be 0-9, # and \* as chars (up to 12 recipes stored)
198. if(c2 == '1') {
199. // Edit these fields to change recipe
200. INITIALIZATION = 95;
201. DENATURATION = 95;
202. ANNEAL = 55;
203. EXTENSION = 72;
204. FINAL\_ELONGATION = 72;
205. HOLDT = 10;
206. NUMBER\_OF\_CYCLES = 35;
207. // Change timing, set in seconds
208. INIT\_TIME = 30;
209. DENAT\_TIME = 30;
210. ANNL\_TIME = 60;
211. EXT\_TIME = 60;
212. FIN\_TIME = 300;
213. HLD\_TIME = 20;
214. lcd.clear();
215. lcd.print("Rcp 1 Chosen");
216. delay(500);
217. }
218. else if(c2 == '2') {
219. // Edit these fields to change recipe
220. INITIALIZATION = 90;
221. DENATURATION = 96;
222. ANNEAL = 50;
223. EXTENSION = 74;
224. FINAL\_ELONGATION = 73;
225. HOLDT = 12;
226. NUMBER\_OF\_CYCLES = 20;
227. // Change timing, set in seconds
228. INIT\_TIME = 30;
229. DENAT\_TIME = 30;
230. ANNL\_TIME = 60;
231. EXT\_TIME = 60;
232. FIN\_TIME = 300;
233. HLD\_TIME = 20;
234. lcd.clear();
235. lcd.print("Rcp 2 Chosen");
236. delay(500);
237. }
238. else if(c2 == '3') {
239. // Edit these fields to change recipe
240. // This recipe currently used to test lcd display
241. INITIALIZATION = 35;
242. DENATURATION = 30;
243. ANNEAL = 25;
244. EXTENSION = 27;
245. FINAL\_ELONGATION = 27;
246. HOLDT = 23;
247. NUMBER\_OF\_CYCLES = 3;
248. // Change timing, set in seconds
249. INIT\_TIME = 10;
250. DENAT\_TIME = 10;
251. ANNL\_TIME = 10;
252. EXT\_TIME = 5;
253. FIN\_TIME = 5;
254. HLD\_TIME = 5;
255. lcd.clear();
256. lcd.print("Test RCP!");
257. delay(500);
258. }
259. }
260. // Set temperature of heaters
261. void settemperature(double settemp, long settime)
262. {
263. unsigned long timer, time2, timerinit;
264. timer = millis();
265. timerinit=timer;
266. Setpoint = settemp;
267. // while temperature is off from settemp, keep looping
268. while (abs(temp-settemp) > 1.0)
269. {
270. // control PID and fan with other function
271. controltemp(1, timerinit, timer, settemp);
272. // get current time in ms
273. timer=millis();
274. // print to screen every second
275. if(timer>pulse)
276. {
277. // increment by 1 second
278. pulse+=1000;
279. // output to screen
280. printData(settemp, 0);
281. }
282. }
283. // Grabs time to control step duration
284. timerinit=timer;
285. time2=settime\*1000+timerinit;
286. // Hold at temperature while at current step
287. while (timer < time2)
288. {
289. // control PID output and fan on/off by using controltemp function
290. controltemp(1, timerinit, timer, settemp);
291. // Every second, output to screen
292. timer=millis();
293. if(timer>pulse)
294. {
295. pulse+=1000;
296. // print data to LCD/serial monitor
297. printData(settemp, (timer-timerinit)/1000.0);
298. }
299. }
300. }
301. // read temperatures from digital temperature sensors on aluminum block
302. void readTemps()
303. {
304. double r1, r2;
305. int sensorValue1, sensorValue2;
306. //read all three sensors
307. sensorValue1 = analogRead(A0);
308. delay(1); //delay to prevent excess ringing, from datasheet
309. sensorValue2 = analogRead(A7);
310. //convert all thermistors from "counts" voltage to C
311. r1 = 1000.0/((1023.0/sensorValue1) - 1);
312. temp1 = 3560.0/log(r1/0.0130444106) - 273.15;
313. r2 = 1000.0/((1023.0/sensorValue2) - 1);
314. temp2 = 3560.0/log(r2/0.0130444106) - 273.15;
315. // Average temperatures of two sensors to control fan and PID
316. temp = (temp1 + temp2) / 2;
317. }
318. // displays data to LCD monitor and serial monitor
319. void printData(double setTemp, int timer)
320. {
321. char buff[17];
322. // buffered charString that contains data printed to LCD
323. sprintf(buff, "%3.d ", calcTimeRemaining());
324. // Prints out time remaining in minutes and current averaged temp (always on line 2)
325. lcd.selectLine(2);
326. lcd.print(buff);
327. lcd.print(" ");
328. lcd.print(temp);
329. // Print to serial monitor as well
330. // Appearance: ## ##.## ##.## ##.##
331. Serial.println();
332. Serial.println();
333. Serial.print(buff);
334. Serial.print(" ");
335. Serial.print(int(setTemp));
336. Serial.print(" ");
337. Serial.print(temp1);
338. Serial.print(" ");
339. Serial.print(temp2);
340. Serial.print(" ");
341. Serial.print(Output);
342. Serial.print(" ");
343. Serial.print(calcTimeRemaining());
344. }
345. // Control temperature by calculating PID and turning fan on or off.
346. void controltemp(boolean fan\_on, unsigned long timerinit, unsigned long timer, double settemp)
347. {
348. // Read temperatures into global variables
349. readTemps();
350. // if temp too high, turn fan on to cool. else, turn fan off.
351. // NOTE: fan\_on is a stupid variable that is always 1 when this function is called
352. if (fan\_on && temp>settemp+1)
353. {
354. // turn fan on
355. digitalWrite(fan, HIGH);
356. }
357. else digitalWrite(fan, LOW);
358. // Send temperature to PID controller for predictive contrl
359. Input = temp;
360. // Compute next PID step
361. myPID.Compute();
362. // Control heater based upon PID compute
363. analogWrite(heater, Output);
364. }
365. // return time remaining on this PCR recipe
366. // return: time remaining in minutes, int
367. int calcTimeRemaining() {
368. // total process time in seconds, excluding ramp times
369. int totTime = INIT\_TIME + (NUMBER\_OF\_CYCLES \* (DENAT\_TIME + ANNL\_TIME + EXT\_TIME)) + (FIN\_TIME + HLD\_TIME);
370. // "fudge factor" for ramp times, etc.; needs to be more elegant later
371. int fudgeFactor = totTime\*.07;
372. // Calculate final time
373. int totalTime = totTime + fudgeFactor;
374. // returns time remaining in minutes as an integer
375. return (totalTime - (millis()/1000))/60;
376. }

Appendix B

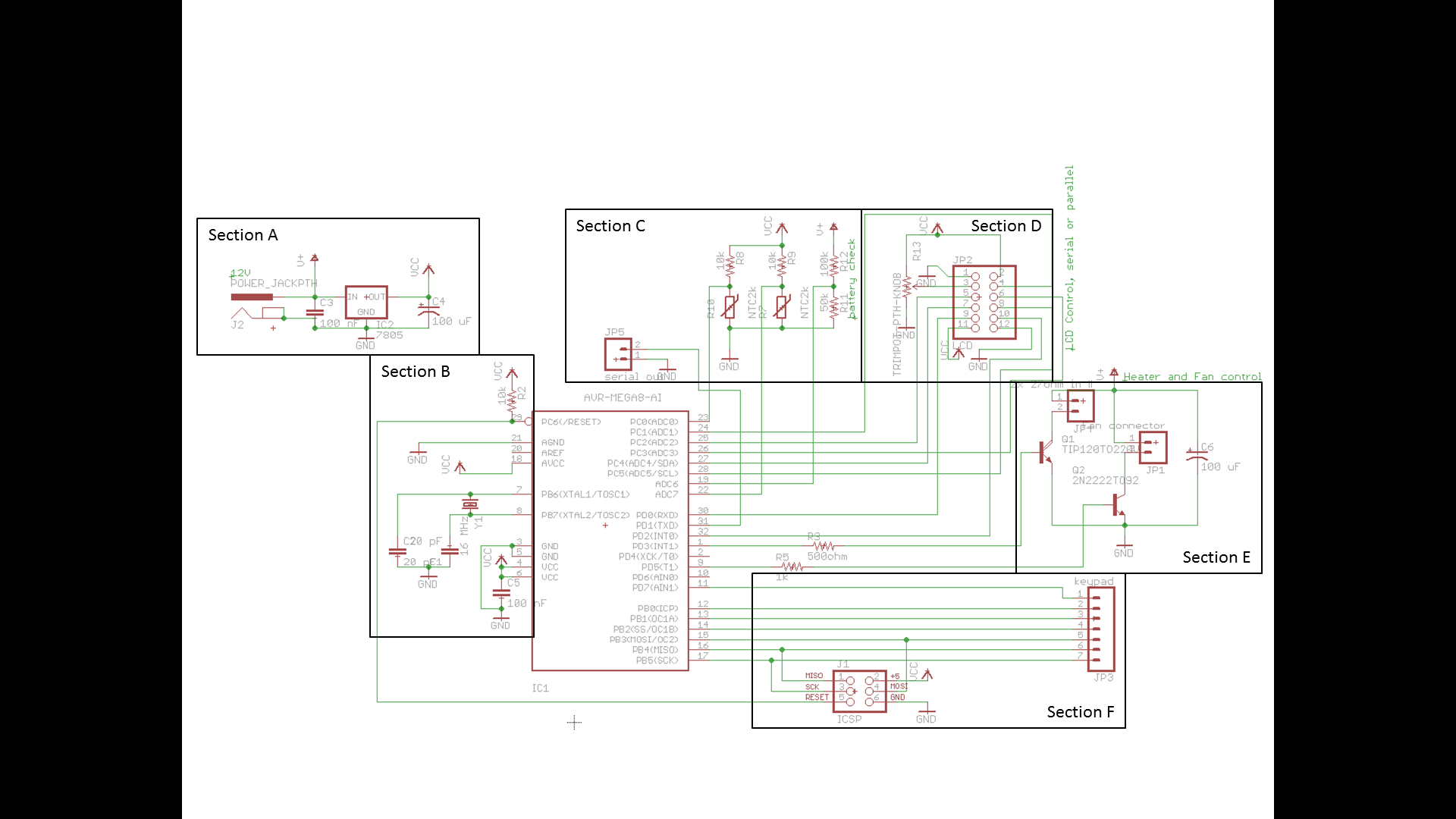


Figure 1. Schematic for Parent Board

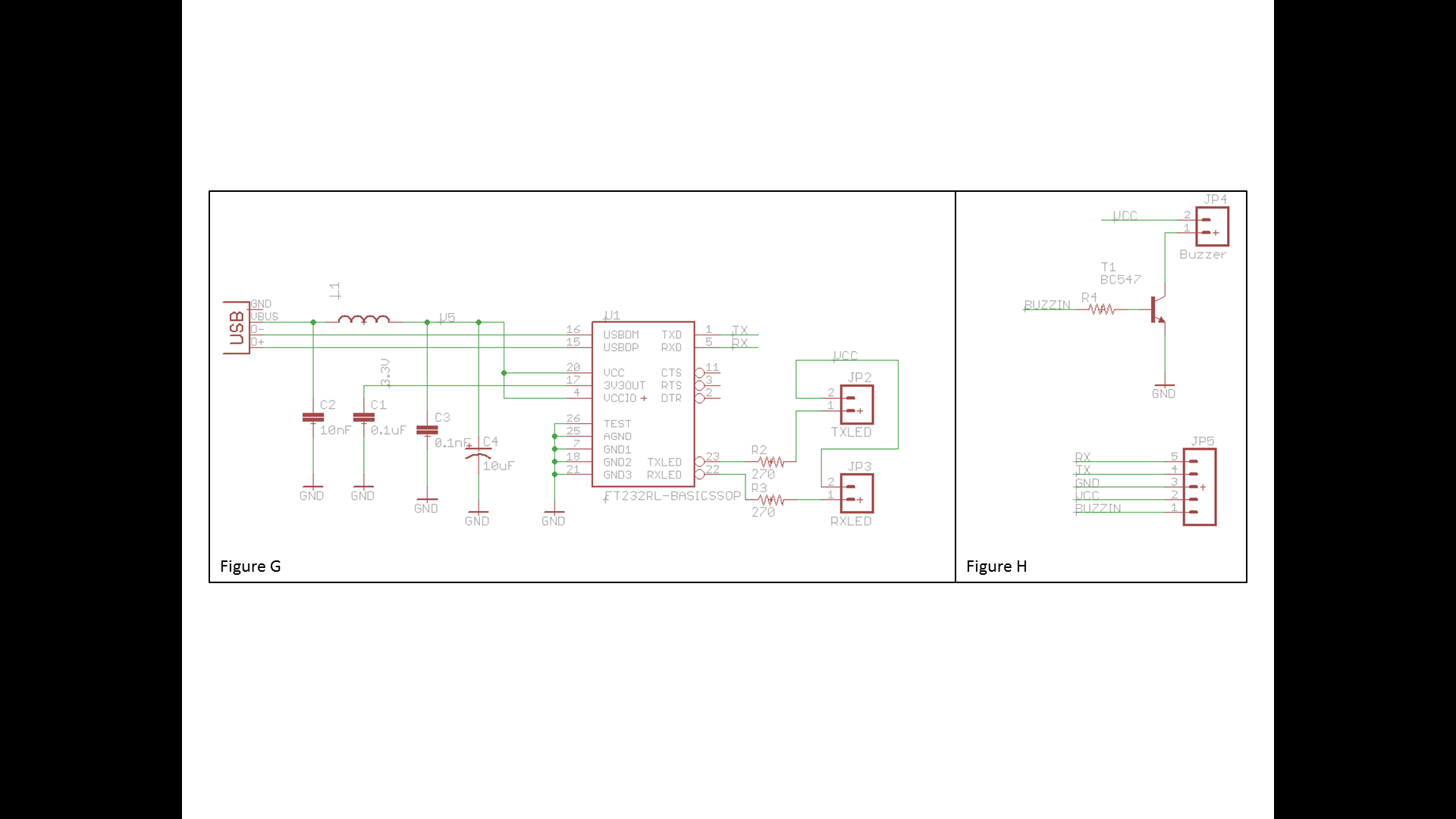


Figure 2. Schematic for Daughter Board

Works Cited

[1] <https://github.com/br3ttb/Arduino-PID-Library/blob/master/PID_v1/PID_v1.h>

[2] <http://playground.arduino.cc/Code/Keypad#CurrentVersion>

[3] <http://www.hobbytronics.co.uk/arduino-atmega328-pinout>

[4] <http://wwwdsa.uqac.ca/~rbeguena/Systemes_Asservis/PID.pdf>

[5] <http://www.epcos.com/inf/50/db/ntc_09/MiniSensors__B57861__S861.pdf>