ACM Soda Machine
Hardware Documentation

Authored by: Nathan Szanto
Contact info: ns6cf@mst.edu

Introduction

In this document, the reader will find information on the theory of operation and design of the ACM Soda Machine. Aspects of the soda machine that will be discussed in detail include the controller and the overall wiring of the machine. After reading this document, the reader will have an understanding of how the controller interfaces with machine components such as buttons and motors.

Overview

A block diagram representation of the overall system can be seen in Figure 1.

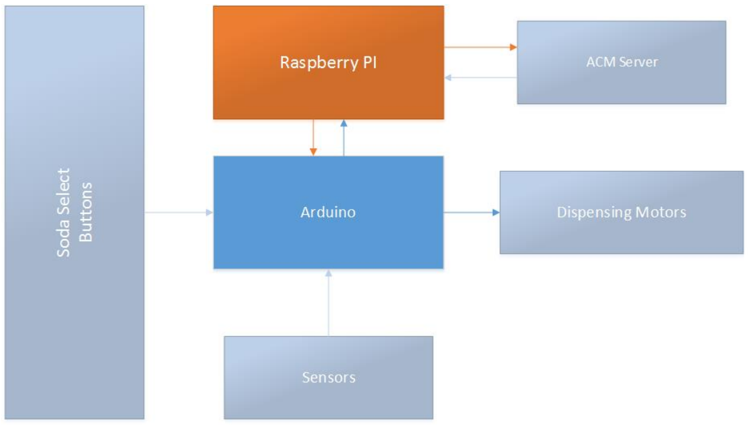

Figure 1: System Block Diagram

The controller is represented by the block labeled "Arduino." The controller implements an Arduino Mega 2560 to read inputs as well as to control the motors. Furthermore, it communicates with the Raspberry Pi in order to receive authorization as well as to relay information concerning soda stock. This document will begin by explaining the theory behind motor operation and control, continue to discuss the inventory sensors and soda select buttons, and conclude with a discussion of the controller itself. The discussion on motor operation and soda select buttons must be read with the Arduino environment in mind; in particular, it is recommend that the reader be familiar with the basics of the Arduino (in particular, the important concept is the idea of using digital pins to read inputs as well as to control outputs).

The overall process of dispensing a soda in terms of confirmations, error checking, and general protocol (confirming payment, receiving authorization from the Raspberry Pi, waiting for a button press. . .) will not be discussed in this document; rather, the operation of each sub-system will be discussed from a hardware perspective.

Throughout this document, signals may be referred to by their signal names; these names correspond to the names that are used in the Eagle Cad design files for the controller.

Motor Operation

The soda machine has eight motors (Figure 2), each with identical modes of operation. All eight motors require $120\,VAC$ in order to turn on; when a give motor is running, it will draw $1.15\,A$. Any design modification must take these parameters into consideration. Because of the way that the software is written, it should not be possible for more than one motor to be active at a given time.

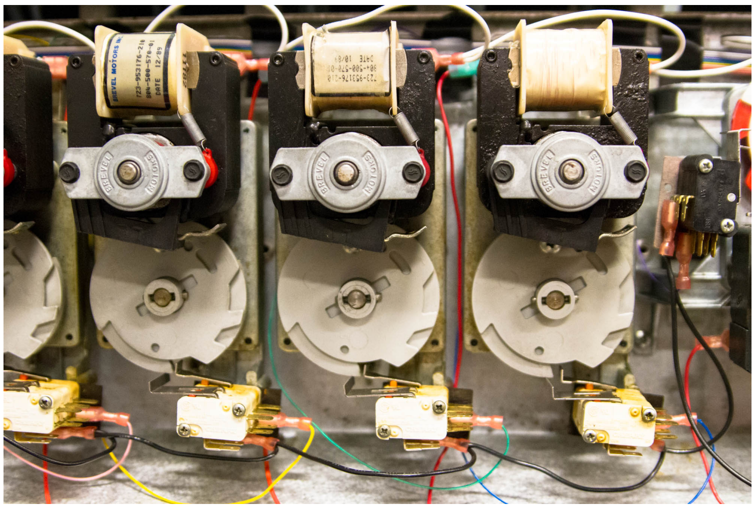

Figure 2: Machine Motors

The red wires that connect to the left end of the coil windings are connected to the controller; when a motor is to be activated, it is through the red wire that the controller will supply the $120\,VAC$. All of the motors are tethered to each other via the white wire. The white wire connects directly to the controller, where it is then connected to the neutral line of the $120\,VAC$ power from the socket.

The motor control works by implementing the rotational disks and the switches that are on the motors (Figure 3).

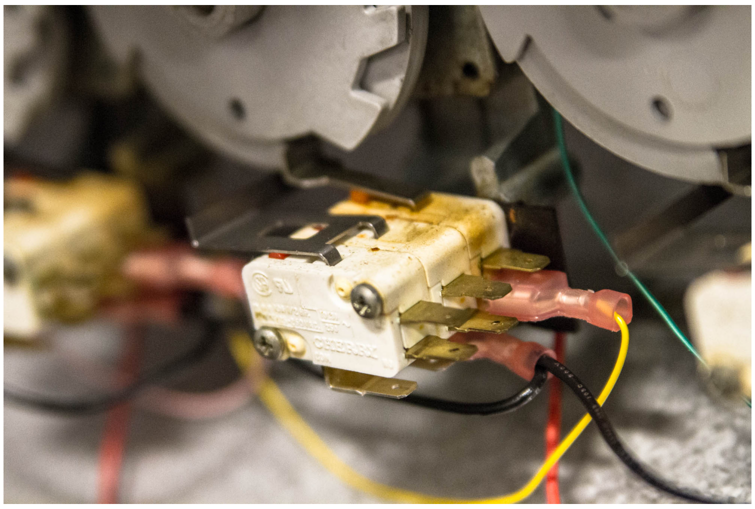

Figure 3: Motor Control Switch

When a soda is to be dispensed, $120\,VAC$ must be supplied to the appropriate motor for a period of time that is long enough to allow the disk to make a full revolution. In order to achieve the appropriate amount of time, the controller monitors the state of the switch as the motor rotates. When a motor is initially turned on, the controller waits for a short period of time (just enough time for the disk to rotate to a degree where the disk closes the switch) and then it begins monitoring the state of the switch; once the disk makes a full revolution, and when the switch snaps back open, the controller ceases to supply power to the motor.

The black wires that connect each of the switches together are signal ground of the controller. The colored ribbon cable wire of each switch connects to a specific pin on the Arduino through the controller; the pins are held HIGH via internal pull-up resistors in the Arduino. When a motor is mid-rotation and the switch is closed, the ribbon cable wire will short to ground and the controller will supply power for as long as it reads a signal ground from the switch.

The motor numbering convention is 1-8, going from left right, facing the machine. A summary of the wire colors and signal names for the motor system is given in Table 1; a summary of the control signal (ribbon cable wires) mapping to the Arduino's digital pins is given in Table 2. Details concerning the method by which power is supplied to the motors will be provided in the section on the controller itself.

Motor Sub-system	Wire Color	Signal Name
Power	Red	MOTOR1...MOTOR8
Power	White	NEUTRAL
Control	Assorted (Ribbon Cable)	CYCLE_SENSE1...CYCLE_SENSE8
Control	Black	GND

Table 1: Motor System Wiring

Signal Name	Arduino Pin Number
CYCLE_SENSE1	41
CYCLE_SENSE2	40
CYCLE_SENSE3	39
CYCLE_SENSE4	38
CYCLE_SENSE5	37
CYCLE_SENSE6	36
CYCLE_SENSE7	35
CYCLE_SENSE8	34

Table 2: Control Signal Pin-mapping

Inventory Sensors

As the soda cans sit in their slots, they press down on one side of a lever, causing the opposite side to elevate; the opposite side of the lever is the side that presses (or depresses when there is no soda) a mechanical switch. These mechanical switches are the inventory sensors (Figure 4).

Each sensor has two sets of switches that are adjacent to each other: Orienting the switch such that the metal tabs are facing you, the set of tabs on the right-hand side correspond to the inventory stock; the ribbon cable connected to the right-hand side switch connects to pins on the Arduino through the controller.

The inventory switches essentially operate the same way that the motor control switches operate: One end of each inventory switch is connected to the controller's ground and the other end of the switch is connected to a specific pin on the Arduino; the Arduino pin is held HIGH with an internal pull-up resistor and, when a particular soda runs out, the switch is closed and the signal shorts to ground.

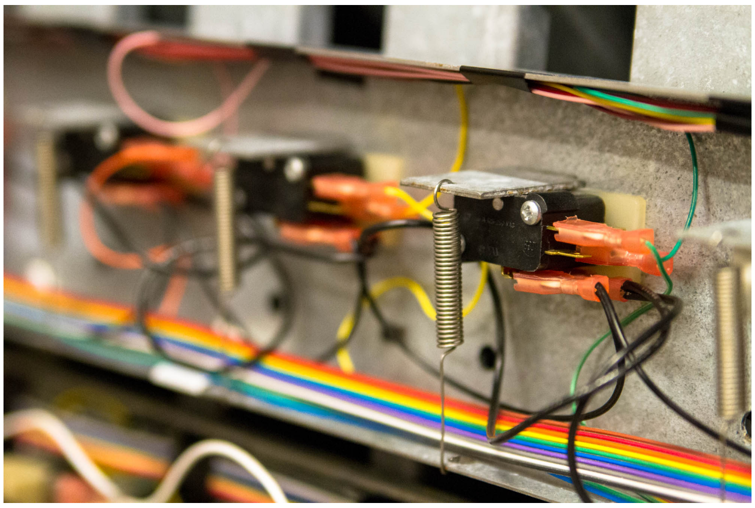

Figure 4: Inventory Sensors

The Arduino pins used for the stock sensors are analogue pins, as opposed to digital pins (for the purposes of this project, there is little bearing on whether a pin is “digital” or “analogue”). The numbering convention of the inventory sensors is identical the motor numbering convention: From left to right, they are 1-8. Table 3 summarizes the inventory signals and their corresponding Arduino pin numbers.

Signal Name	Arduino Pin Number
SENSOR1	A0
SENSOR2	A1
SENSOR3	A2
SENSOR4	A3
SENSOR5	A4
SENSOR6	A5
SENSOR7	A6
SENSOR8	A7

Table 3: Inventory Sensor Pin-mapping

The set of tabs on the left-hand side of the inventory switch corresponds to the LEDs of the buttons; these signals do not correspond to any pins on the Arduino. Each LED switch has one end connected to the controller’s $+5\,V$ pin and the other end of the switch is connected to a particular LED’s anode; the cathode of each LED is connected to the controller’s ground. When a soda is in stock, the LED will be supplied with power; however, when a particular soda runs out and the lever presses down on the switch, the LED will be disconnected from the power supply and the LED will turn off.

NOTE: At it its current state, there is a set of wires, one green (which connects to ground) and one blue (which connects to $+5\,V$), running from the LED switches to the controller. At the controller end, the wires have pins on their ends, but they are not plugged into the connector that would mate with the $+5\,V/GND$ headers on the controller. DO NOT neglect obtaining the proper connector before connecting the wires to the board; if you do, you risk shorting the $+5\,V$ pin on the Arduino with the GND pin, which could cause the Arduino to fry if it goes unnoticed. The appropriate connector that is needed for this is a Molex connector (MFG Part#: 22-01-2025). It might be possible to obtain free samples of this connector directly from Molex; if not, they are available on Mouser.

Soda Select Buttons

The soda select buttons (Figure 5) essentially operate the same way that the motor control and inventory switches operate: One end of each button is connected to the controller's ground and the other end of the button is connected to a specific pin on the Arduino; the Arduino pin is held HIGH with an internal pull-up resistor and, when a button is pressed, the Arduino recognizes the short to ground.

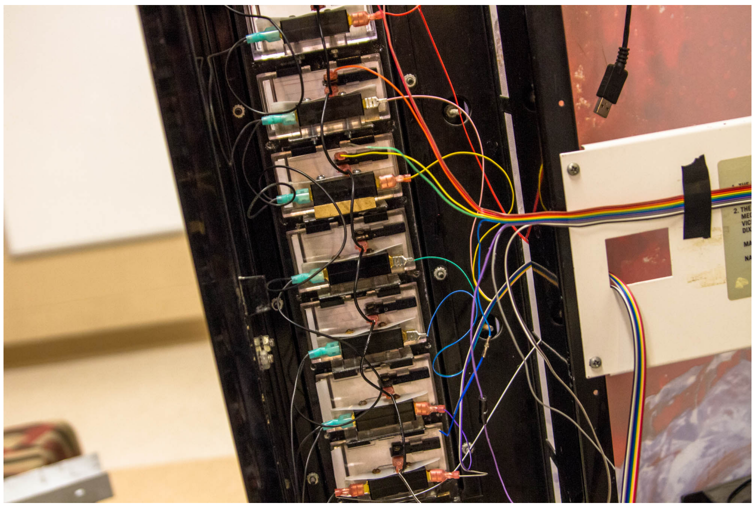

Figure 5: Soda Select Buttons

The ribbon cable labeled as "Buttons" is the one that corresponds to the Arduino pins involved with monitoring the button select system; the Arduino pins used for the buttons are analogue pins, as opposed to digital pins. The button numbering convention is 1-8, going from top to bottom. Table 4 summarizes the button signals and their corresponding Arduino pin numbers.

Signal Name	Arduino Pin Number
BUTTON1	A8
BUTTON2	A9
BUTTON3	A10
BUTTON4	A11
BUTTON5	A12
BUTTON6	A13
BUTTON7	A14
BUTTON8	A15

Table 4: Soda Select Button Pin-mapping

The Controller

The controller can be seen in Figure 6.

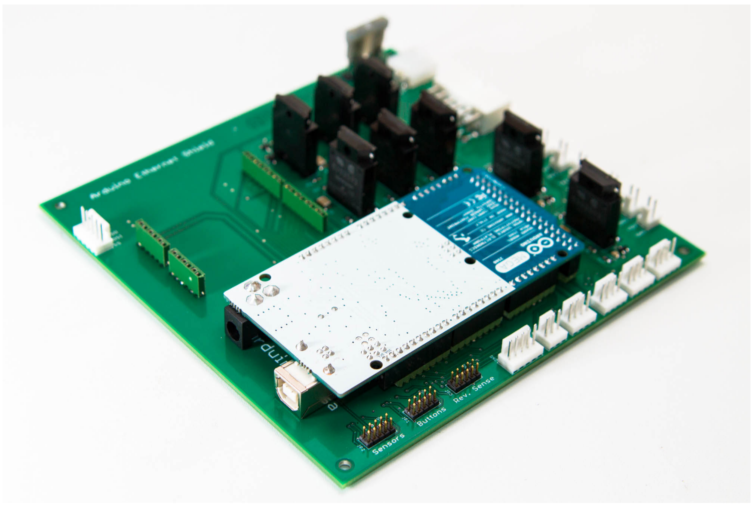

Figure 6: The Controller

Power Connector

The power connector can be seen in Figure 7.

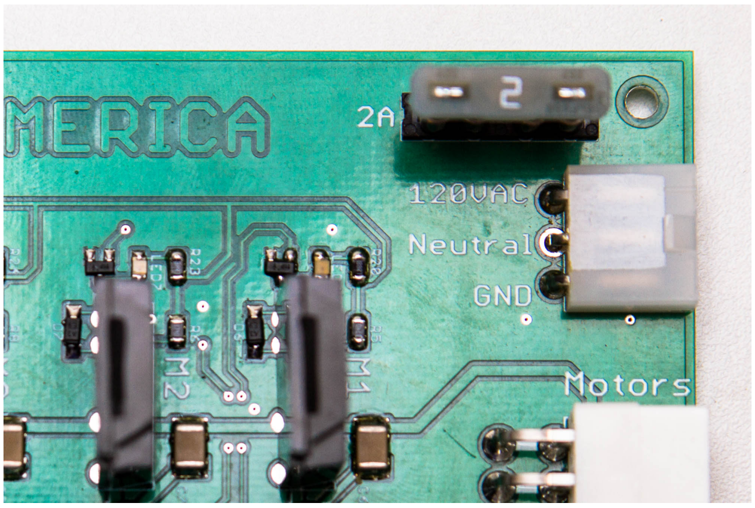

Figure 7: Power Connector

The Neutral line goes immediately to a pin on the adjacent Motors connector, from where it goes to the motors. At its current state, the GND line goes directly to a pin on the Motors connector, but to nowhere else; in the future, a wire could be run from this GND pin and connect to the machine chassis via a round terminal. The $120\,VAC$ line is distributed to one of the output pins of each motor relay (Figure has the $120\,VAC$ signal highlighted).

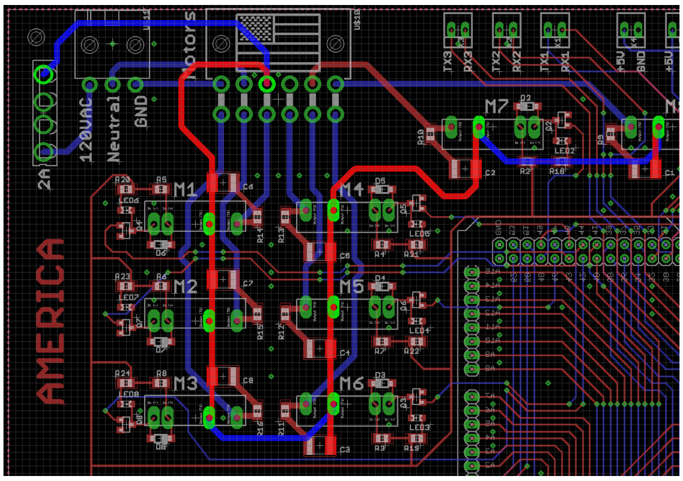

Figure 8: 120 VAC Highlighted

However, before the 120 VAC line is distributed, it goes through a fuse; the fuse protects the board from current surges. The 120 VAC traces have widths of 40 mils; using a copper thickness of 1 oz/ft^2, the current carrying capability of the traces is about 2.4 A. The fuse must be able to carry more than 1.15 A, but MUST NOT BE ABLE TO CARRY MORE THAN 2.4 A. A 2.0 A fuse is recommended. The fuse holder takes ATC blade fuses.

The second output pin of each relay is connected to a pin on the Motors connector; these pins are what connect to the motors through the red wires.

Motors Connector

The numbering convention for the pins on the Motors connector follows what is indicated in Figure 9. Table 5 gives the signal names corresponding to the pins on the Motors connector.

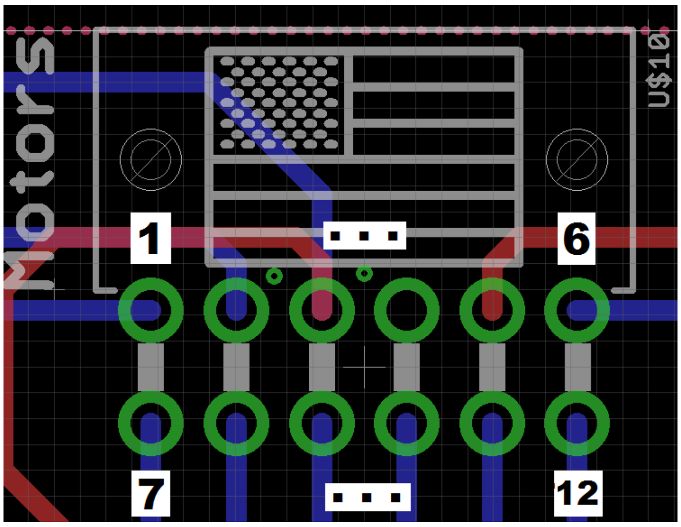

Figure 9: Motor Connector Numbering

Connector Pin Number	Signal Name
1	120VAC-GND
2	NEUTRAL
3	120VAC
4	*NC*
5	MOTOR7
6	MOTOR8
7	MOTOR3
8	MOTOR2
9	MOTOR1
10	MOTOR4
11	MOTOR5
12	MOTOR6

Table 5: Motors Connector Pin-out

Relay Circuitry

The relays that are used to interface between the Arduino and the motors are solid state relays made by Sharp Microelectronics (MFG Part #: S202S01F). Between the input and output sides, the relays provide $4.0\,kV$ of isolation and they are rated for $8.0\,A$. The relay circuitry for a single motor is shown in Figure 10.

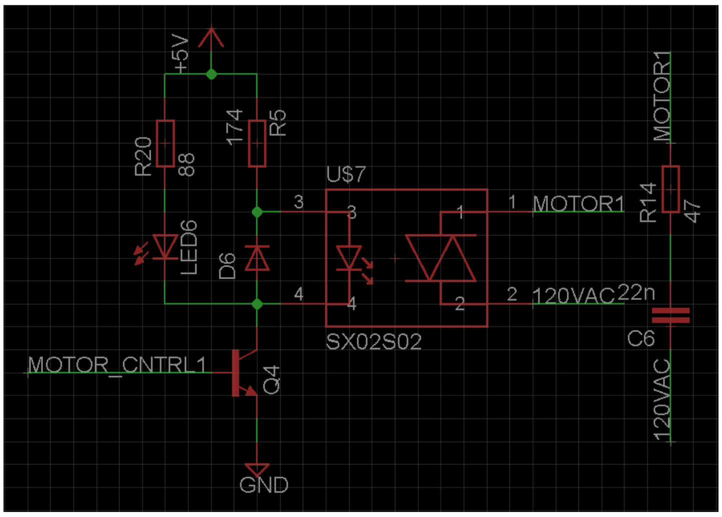

Figure 10: Relay Circuitry

Essentially, in order to turn a motor on, the internal LED of the relay needs to be turned on. This is accomplished by first supplying $+5\,V$ to the anode pin of the relay while also using a current-limiting resistor to protect the internal LED. The signal, MOTOR_CNTRL1, connects to a digital pin on the Arduino. When this signal goes HIGH, the transistor turns on, completing the input circuit of the relay and allowing current to flow through the relay LED. When this happens, MOTOR1 (which connects to the first motor through the Motors connector) is connected to the $120\,VAC$ line, turning the motor on. The motor control signal (in this case, MOTOR_CNTRL1) remains HIGH until the motor makes a full revolution; that is, it remains HIGH until the Arduino ceases to read a LOW state from the motor control signal (in this case, the first motor would remain on until CYCLE_SENSE1 stopped registering as LOW). When a motor is to be turned off, the motor control signal is set to LOW, essentially creating an open-circuit condition in the relay's input circuit.

Table 6 summarizes the motor control signals and their corresponding Arduino pin numbers.

Signal Name	Arduino Pin Number
MOTOR_CNTRL1	44
MOTOR_CNTRL2	43
MOTOR_CNTRL3	42
MOTOR_CNTRL4	45
MOTOR_CNTRL5	46
MOTOR_CNTRL6	47
MOTOR_CNTRL7	48
MOTOR_CNTRL8	49

Table 6: Motor Control Pin-mapping

Communication and Auxiliary Pins

The controller has easy access to the Arduino's communication pins; the communication pins are summarized in Table 7.

Signal Name	Arduino Pin Number
MISO	50
MOSI	51
SCK	52
SS	53
TX3	14
RX3	15
TX2	16
RX2	17
TX1	18
RX1	19

Table 7: Communication Pin-mapping

Furthermore, the controller has six auxiliary ports that provide easy access to all other unused Arduino pins. Table 8 gives the Arduino pin numbers corresponding to each auxiliary pin.

Auxiliary Port 1

Signal Name	Pin Number
AUX1.1	31
AUX1.2	32
AUX1.3	33

Auxiliary Port 2

Signal Name	Pin Number
AUX2.1	28
AUX2.2	29
AUX2.3	30

Auxiliary Port 3

Signal Name	Pin Number
AUX3.1	25
AUX3.2	26
AUX3.3	27

Auxiliary Port 4

Signal Name	Pin Number
AUX4.1	22
AUX4.2	23
AUX4.3	24

Auxiliary Port 5

Signal Name	Pin Number
AUX5.1	21
AUX5.2	20

Auxiliary Port 6

Signal Name	Pin Number
AUX6.1	10
AUX6.2	11
AUX6.3	12
AUX6.4	13

Table 8: Pin-mapping for Auxiliary Ports

Establishing Communication With the Raspberry Pi

Perhaps the simplest way to establish communication between the Raspberry Pi and the Arduino would be to take advantage of the Arduino's I^2C pins, which can easily be accessed with auxiliary pins AUX5.1 (SCL) and AUX5.2 (SDA).

NOTE: DO NOT use this method unless the Raspberry Pi is being run as a "master" and the Arduino is being run as a "slave." If the situation is otherwise, you will need to use a different method that incorporates a logic level converter to interface between the Raspberry Pi and the Arduino.

The header which has AUX5.1 and AUX5.2 is a three-pin header which also includes GND. In order to connect with the Raspberry Pi, simply connect the Arduino's GND to the Raspberry Pi's GND, AUX5.1 with the Pi's SCL pin, and AUX5.2 with the Pi's SDA pin.

For cable assembly: The wires on the Arduino end of the cable will need to be fitted with crimp terminals (Molex Part #: 39-00-0039) which will be fed into a mating connector (Molex Part #: 22-01-2035). Be sure to orient the wires into the connector correctly so that the pins on the Arduino connect to the correct pins on the Raspberry Pi.

Further details can be found here: http://blog.oscarliang.net/raspberry-pi-arduino-connected-i2c/.

Final Notes

During the final testing of the controller, it was noticed that if the machine is turned off and then turned back on too quickly, there is a possibility of blowing the fuse. For this reason, it is recommended that, when the machine is turned off, it be given a chance to “cool down” for about 10 seconds before turning it back on.