Santec OEM Switch User Guide

Last Updated: September 18, 2023
Revision: 1.3

Glossary

Term	Definition
I2C	Inter-integrated circuit protocol
driverboard	PCB with motor control circuitry

Table 1: Glossary

Getting Started

Calibration

Each motor will be provided with a pre-calibrated driverboard that is matched to the motor. Calibration data is saved in EEPROM. The channels can be accessed using the commands detailed below.

Connections

- Connect the I2c harness to the I2C master.
- Connect the GND from the I2C harness to the same GND as the I2C master
    - (not required if the I2C master and the Santec OEM driverboard share the same power supply)
- Connect the 12V power to an appropriate +12V power supply (see Appendix B)
- Connect the optical fiber port labeled *COM* to a light source (ideally a visible red-light source for debugging and testing purposes)

Addressing

Address is set by the dip switch *SW1* on the driverboard. By default every unit is shipped at dip switch address 0 (0000 or off,off,off,off) If more than 1 switch is to be controlled on the same I2C bus, each switch should be given its own unique address.
The address is a 4 bit integer. LSB is labelled 1, MSB is 4. For example, a pattern of “1000” or “on,off,off,off” (see Fig. 2) would be address 1.

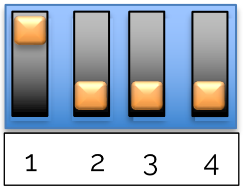


Figure 1: Dip switch at Address 1

Communications

The I2C bus should be run at <100kHz.

Command Byte Format

7 1	0	7 0	7 0
ADDRESS	R/W	CMD	PARAMETER

ADDRESS=0x50 + dip | R/W
For example:
At dip switch address 0, to write:
ADDRESS = 0x50<<1 | 0x00 = 0xA0
At dip switch address 0, to read:
ADDRESS = 0x50<<1 | 0x01 = 0xA1

The following table shows the correct address bytes to send depending on dip switch setting and whether the command is a read or a write.

					ADDRESS	
Decimal	**Dip switch**				**Write**	**Read**
0	0	0	0	0	0xA0	0xA1
1	0	0	0	1	0xA2	0xA3
2	0	0	1	0	0xA4	0xA5
3	0	0	1	1	0xA6	0xA7
4	0	1	0	0	0xA8	0xA9
5	0	1	0	1	0xAA	0xAB
6	0	1	1	0	0xAC	0xAD
7	0	1	1	1	0xAE	0xAF
8	1	0	0	0	0xB0	0xB1
9	1	0	0	1	0xB2	0xB3
10	1	0	1	0	0xB4	0xB5
11	1	0	1	1	0xB6	0xB7
12	1	1	0	0	0xB8	0xB9
13	1	1	0	1	0xBA	0xBB
14	1	1	1	0	0xBC	0xBD
15	1	1	1	1	0xBE	0xBF

Command Set

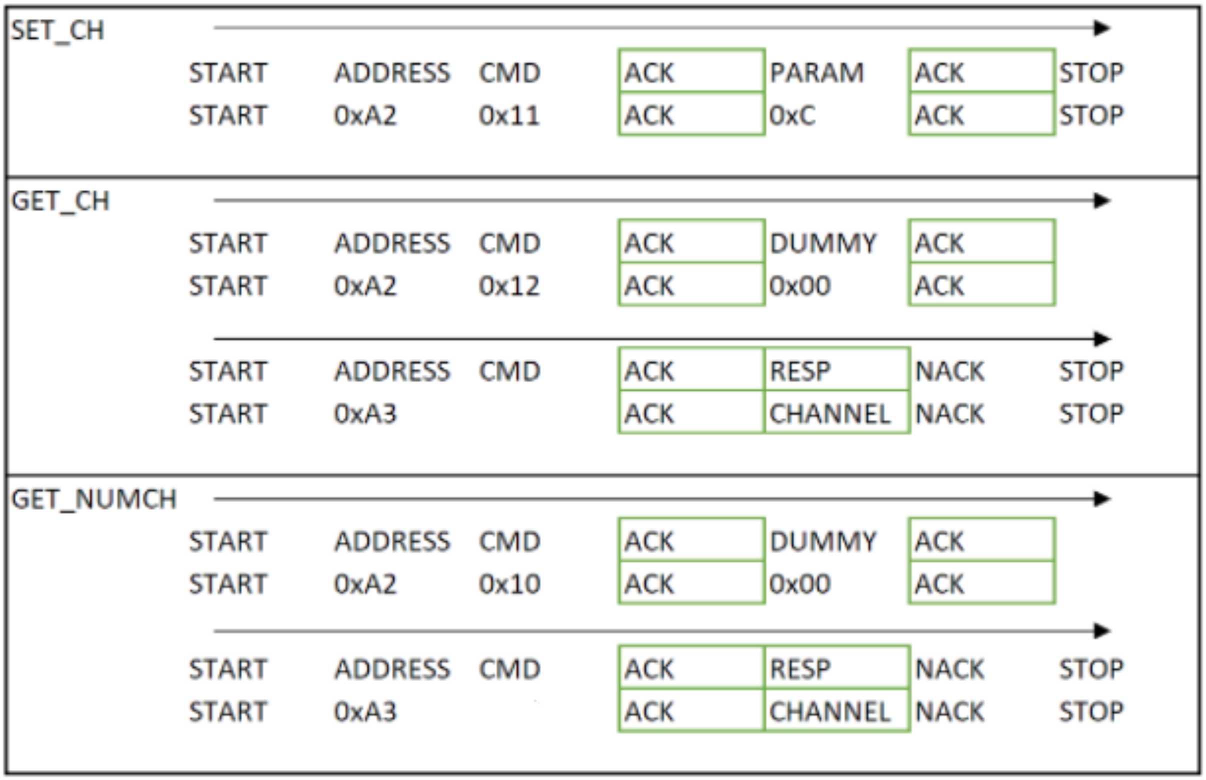
Command Byte	**Command Description**
0x11	Set channel

0x12	Get current channel
0x10	Get total number of channels

Example

Assuming dip switch is set to address 1. The command byte sequence for I2C is the following for each of the 3 commands. The green (bolded) boxes indicate places where the master should wait for a response from the driverboard. The first line of each set is the generic structure. The second line shows the actual bytes to be sent.

Note: This is the procedure for “bit-banging” I2C. If you use a library (e.g. Arduino Wire.h) the start stop and ack are already taken care of.



SET_CH							
	START	ADDRESS	CMD	ACK	PARAM	ACK	STOP
	START	0xA2	0x11	ACK	0xC	ACK	STOP
GET_CH							
	START	ADDRESS	CMD	ACK	DUMMY	ACK	
	START	0xA2	0x12	ACK	0x00	ACK	
	START	ADDRESS	CMD	ACK	RESP	NACK	STOP
	START	0xA3		ACK	CHANNEL	NACK	STOP
GET_NUMCH							
	START	ADDRESS	CMD	ACK	DUMMY	ACK	
	START	0xA2	0x10	ACK	0x00	ACK	
	START	ADDRESS	CMD	ACK	RESP	NACK	STOP
	START	0xA3		ACK	CHANNEL	NACK	STOP

Getting Started with Arduino Uno

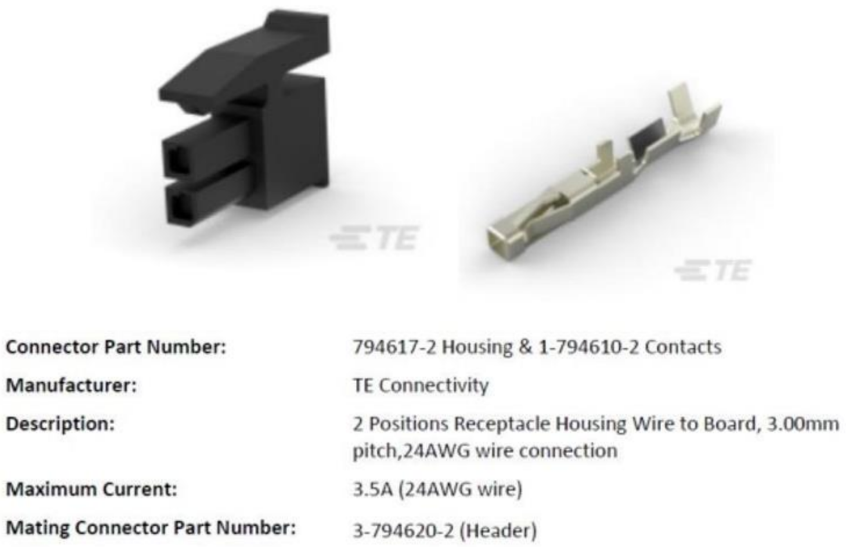
To test communications, an Arduino Uno can be used.
Load the sketch in Appendix D.
Connect the I2C header on the motor driver board to the I2C SDA and SCL lines on the Arduino. Connect the GND cable to the Arduino GND.

Appendix A: Optical specifications

Parameter	Specification	
	Single-mode	Multimode
Wavelength Range (nm)	1250 to 1670	840 to 1350
Insertion Loss (dB)	< 0.7	
Backreflection (dB)	< -60	< -40
PDL (dB)	< 0.05	N/A
Repeatability (dB)	± 0.005	
Crosstalk (dB)	< -80	

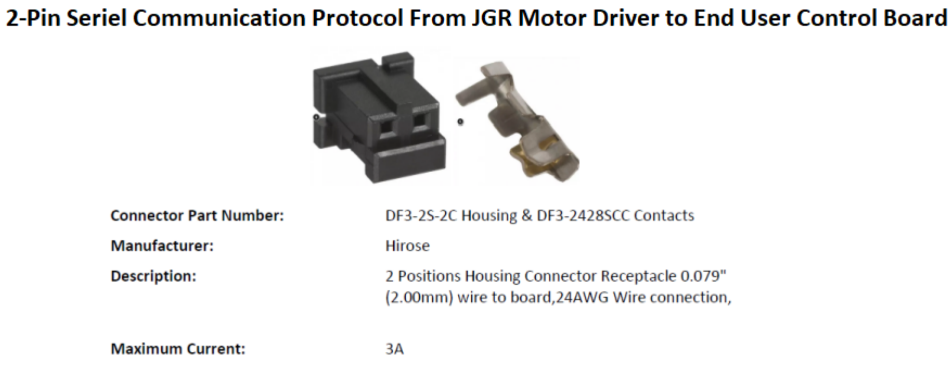
Appendix B: Electrical specifications

Parameter	Specification	
	Single-mode	Multimode
Switching Time (ms)	<300	
Control	I2C	
Input Voltage	12VDC with <120mV$_{PP}$ ripple	
Power Consumption	<1.3A @12V	
Switch Life (cycles)	10^8	



Connector Part Number:	794617-2 Housing & 1-794610-2 Contacts
Manufacturer:	TE Connectivity
Description:	2 Positions Receptacle Housing Wire to Board, 3.00mm pitch,24AWG wire connection
Maximum Current:	3.5A (24AWG wire)
Mating Connector Part Number:	3-794620-2 (Header)

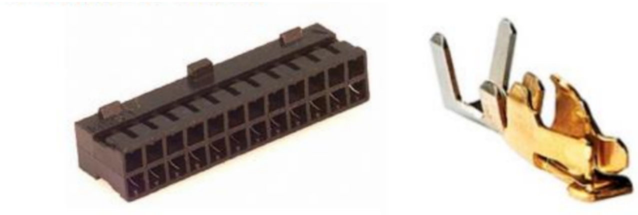
Figure 2: 12V power supply header

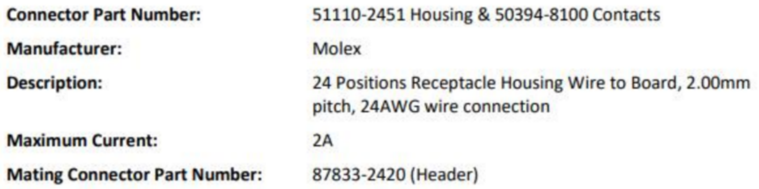


2-Pin Seriel Communication Protocol From JGR Motor Driver to End User Control Board

Connector Part Number:	DF3-2S-2C Housing & DF3-2428SCC Contacts
Manufacturer:	Hirose
Description:	2 Positions Housing Connector Receptacle 0.079" (2.00mm) wire to board,24AWG Wire connection,
Maximum Current:	3A

Figure 3: 2 wire I2C header





Connector Part Number:	51110-2451 Housing & 50394-8100 Contacts
Manufacturer:	Molex
Description:	24 Positions Receptacle Housing Wire to Board, 2.00mm pitch, 24AWG wire connection
Maximum Current:	2A
Mating Connector Part Number:	87833-2420 (Header)

Figure 4: I2C header connection. Note this connection is provided. The 2 wire I2C will be broken out into a 2-pin Hirose connector from pins 9 and 10.

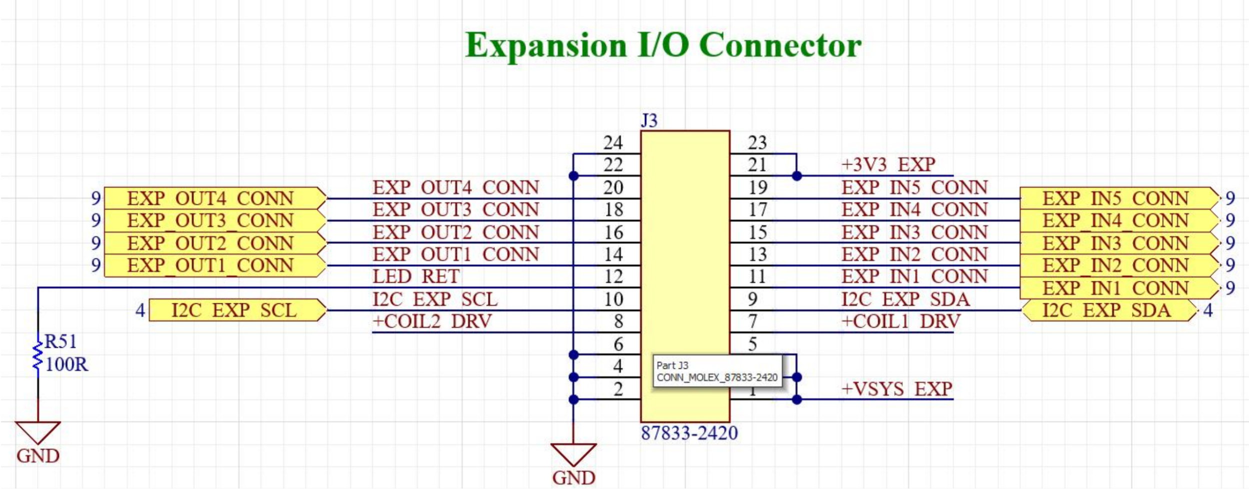


Figure 5: Sensor header pinout showing I2C connection pins 9 and 10

Appendix C: Mechanical Specifications

Parameter	Specifications			
	Extra Small Switch	Small Switch	Medium Switch	Large Switch
Channel Count	26	36	54	80
Dimensions Length A (mm)	100	120	154	175
Dimensions Width C (mm)	52	60	84	110
Dimensions Height E (mm)	41	54.5	78	102.5
Dimensions Mounting Holes B x D (mm)	90 x 42	110 x 50	144 x 74	165 x 100
Mounting Hole Diameter F (mm)	3.2	3.2	3.4	3.4
PCB Standoffs Included	No	Yes	Yes	Yes
Recommended Fiber Area (mm)	200 x 125	200 x 125	200 x 160	200 x 200
Shipping Box Dimensions W x H x D (cm)	36 x 33 x 18	36 x 33 x 18	36 x 33 x 18	36 x 33 x 37
Unit Weight (kg)	0.3	0.5	1.2	1.8
Total Shipment Weight (kg)	1.1	1.6	2.0	2.6
Operating Temperature (ºC)	0 to 55			
Humidity (Non-condensing)	Maximum 95% RH from 0 to 40ºC			

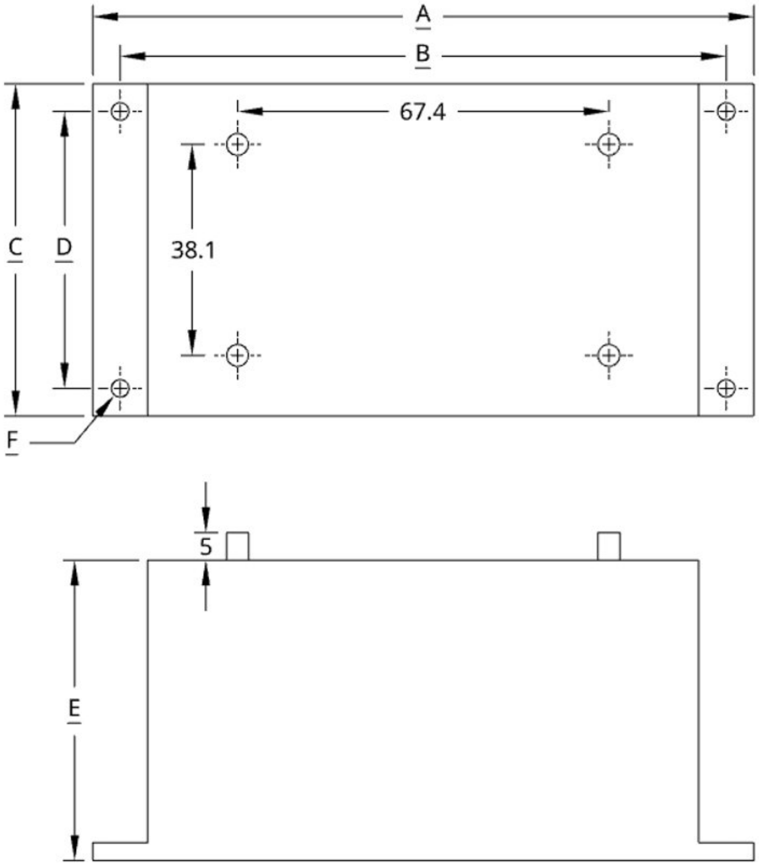


Figure 6: Switch motor mount hole locations

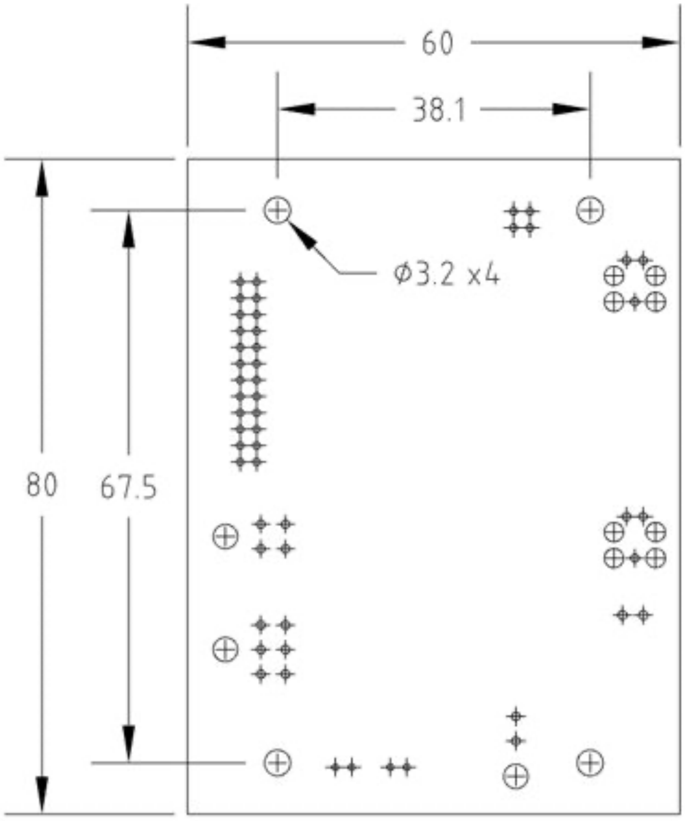


Figure 7: Switch motor control board PCB dimensions

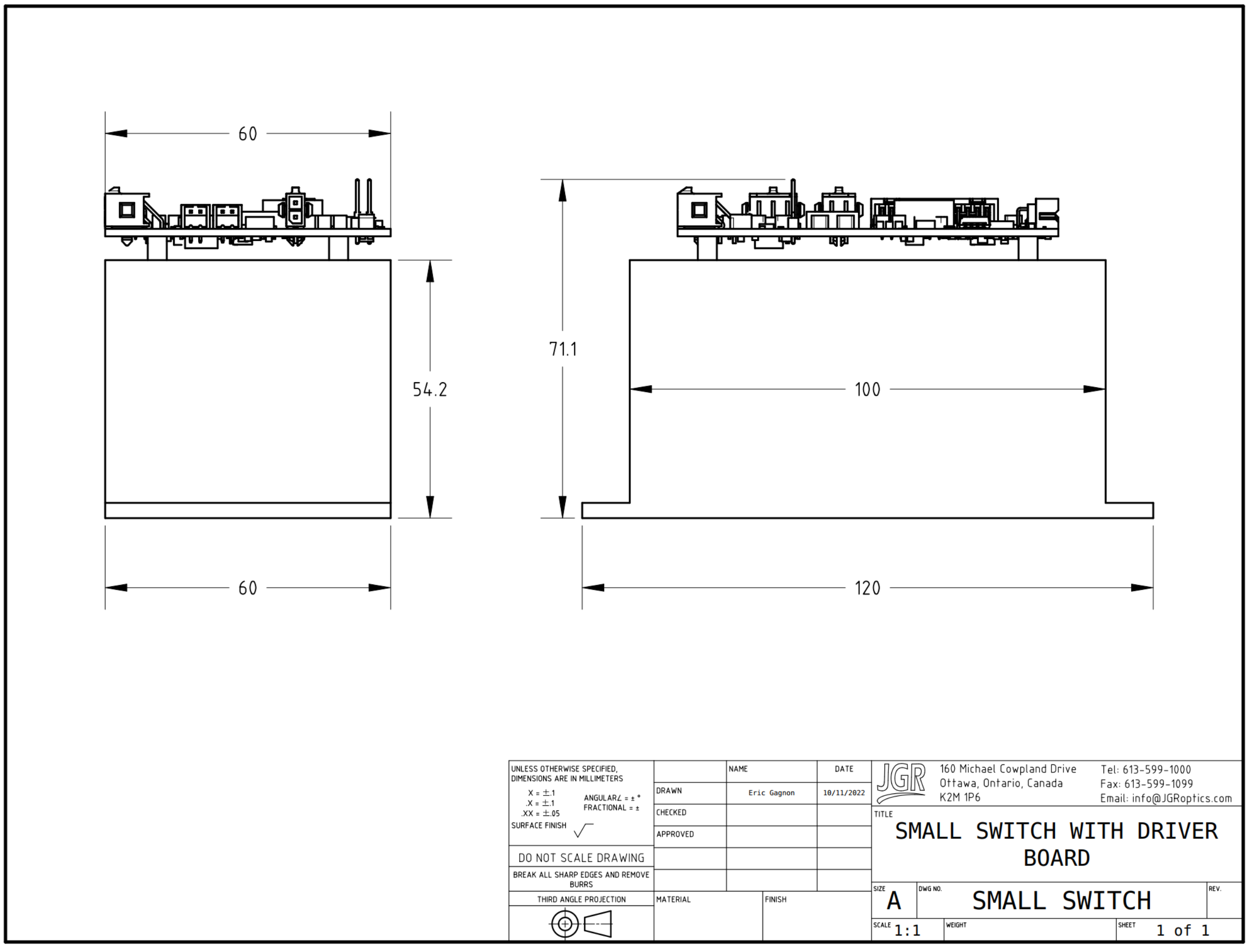


Figure 8: Small motor <36 channels

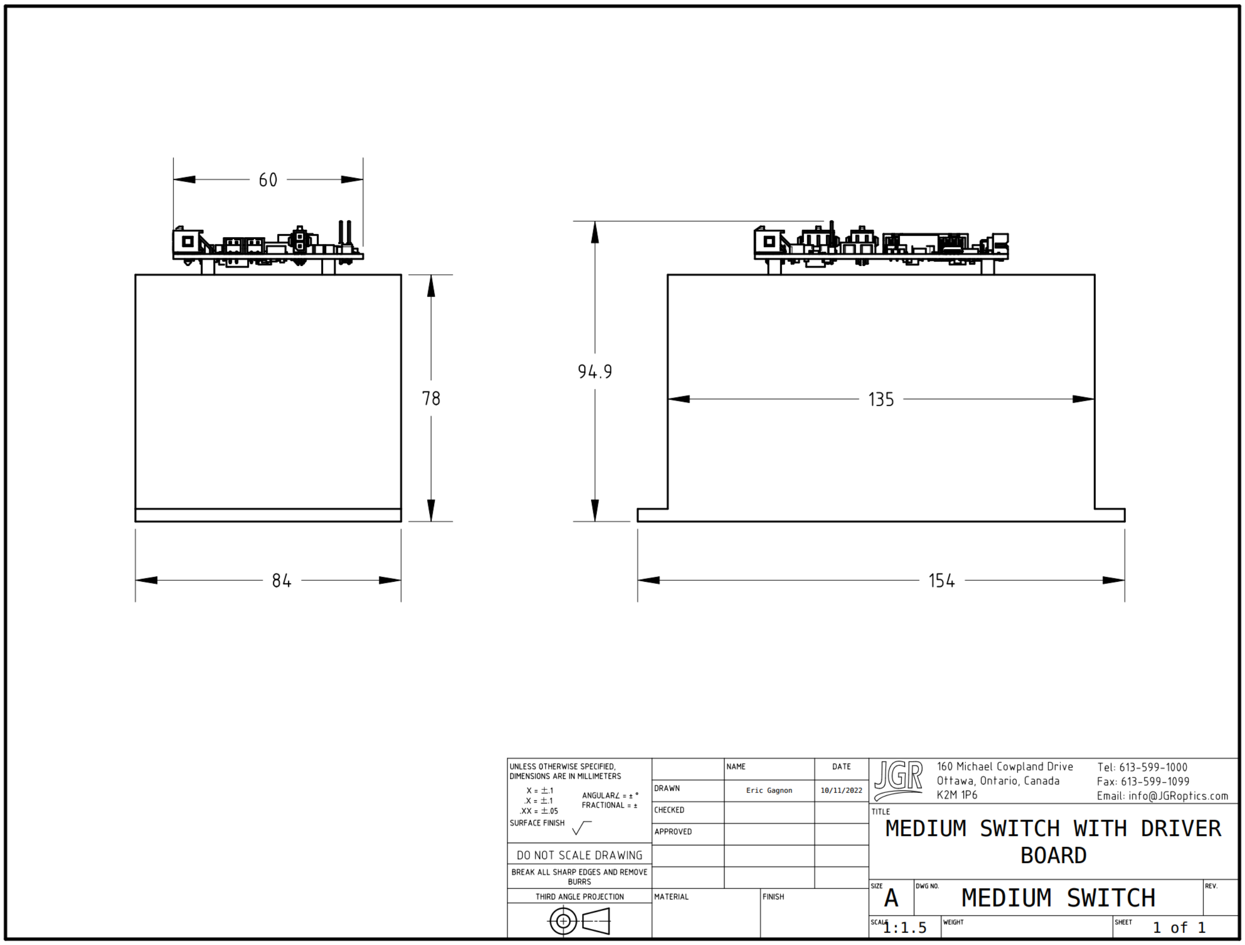


Figure 9: Medium motor <54 channels

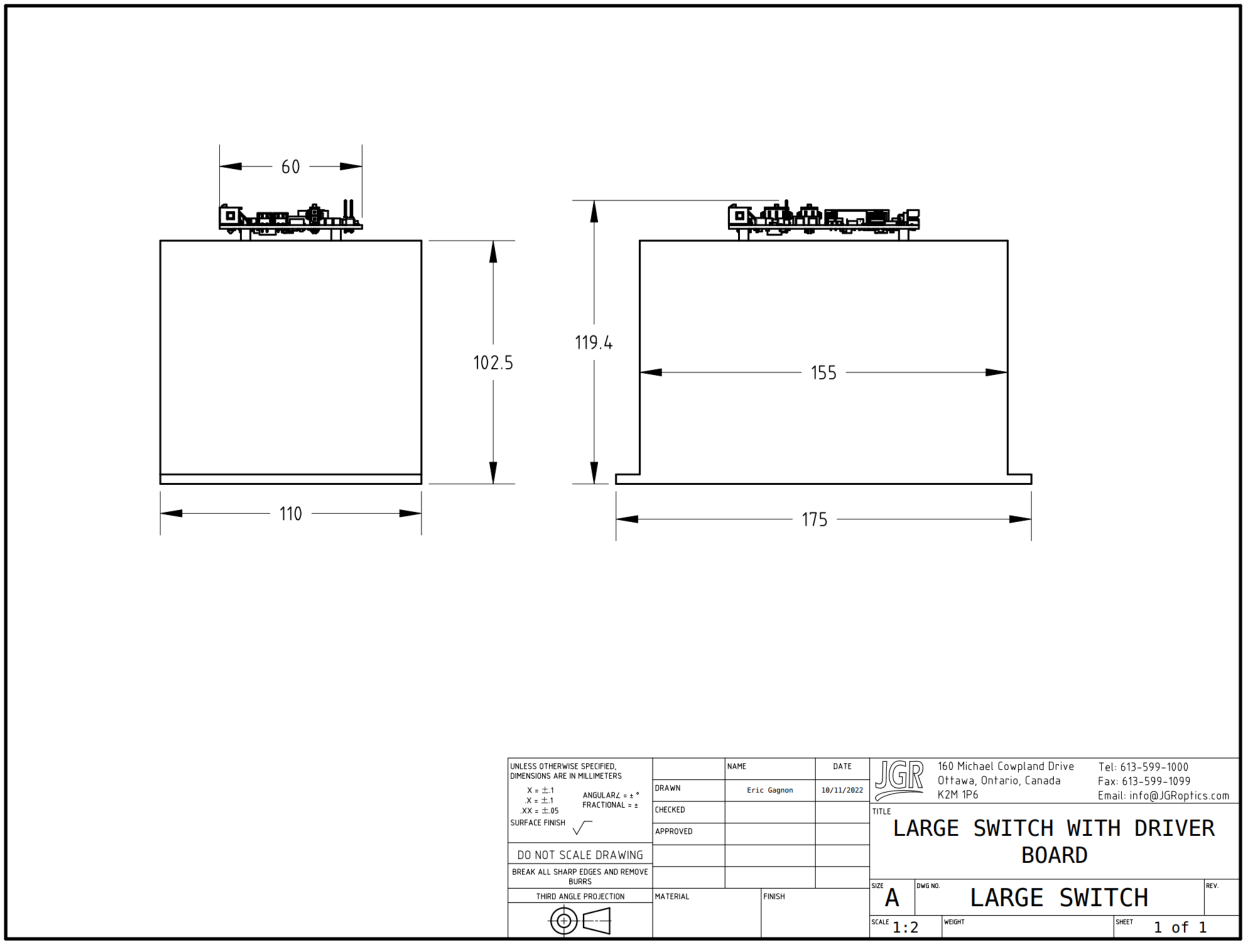


Figure 10: Large motor <80 channels

Appendix D: Arduino Uno Sketch

```
/*****************************************************************************
IMPORTANT NOTES

(1) The Wire library uses 7 bit addresses throughout. Must drop the low bit
    (i.e. shift the value one bit to the right), yielding an address between 0 and
    127. However the addresses from 0 to 7 are not used because are reserved so the
    first address that can be used is 8.

(2) The Wire library uses a 32 byte buffer, therefore any exceeding bytes in a single
    transmission will just be dropped.

(3) endTransmission(stop) ends a transmission to a peripheral device
    that was begun by beginTransmission() and transmits the bytes that were
    queued by write().

(4) Wire.requestFrom(address, quantity, stop)

    stop: true or false. True will send a stop message, releasing the bus after
          transmission. False will send a restart, keeping the connection active.
*****************************************************************************/

#include <Wire.h>

#define DIP_SWITCH 0X00
#define MOTOR_BOARD_ADDR (0x50 + DIP_SWITCH) << 1
#define MOTOR_WRITE (MOTOR_BOARD_ADDR | 0x00) >> 1  // see Note (1)
#define MOTOR_READ (MOTOR_BOARD_ADDR | 0x01) >> 1   // see Note (1)

// Define command bytes
#define CMD_SET_CH 0x11
#define CMD_GET_CURR_CH 0x12
#define CMD_GET_TOT_CH 0x10

// Define dummy param byte
#define PARAM_DUMMY 0x00

static int totChannelCount = 0;
String select = "none";

void setup() {
  Wire.begin();
  Serial.begin(9600);
  totChannelCount = getChannelCount();  // automatically get total channel count
  Serial.println("Select a function to execute ...");
  Serial.println("Enter 'get' to query current channel");
  Serial.println("Enter 'set' to change the channel");
  Serial.println("Enter 'total' to get total channel count");
}

void loop() {
  // wait for user input
  while (Serial.available() == 0) {}
  select = Serial.readStringUntil('\n');

  // call function based on user input
  if (select.equals("total")) {
    totChannelCount = getChannelCount();
    if (totChannelCount > 0) {
      Serial.println((String) "Total Number of Channels = " + totChannelCount);
    } else {
      Serial.println("Failed to get total number of channels ...");
    }
  } else if (select.equals("get")) {
    int currChannel = getCurrCh();
    if (currChannel >= 0) {
      Serial.println((String) "Current Channel = " + currChannel);
    } else {
      Serial.println("Failed to get current channel ...");
    }
  } else if (select.equals("set")) {
    Serial.println("Enter channel number");
    while (Serial.available() == 0) {}
    int channelNum = Serial.parseInt();
    if (channelNum < 0 || channelNum > totChannelCount) {
      Serial.println((String) "Invalid channel number ...");
    } else {
      setCh(channelNum);
    }
  }
}

/****************************************************************************

FUNCTION DEFINITIONS

****************************************************************************/

void setCh(int channelNum) {
  Serial.println("Setting switch ...");

  Wire.beginTransmission(MOTOR_WRITE);  // begin with slave device address
  Wire.write(CMD_SET_CH);               // command byte
  Wire.write(channelNum);               // param byte
  Wire.endTransmission(true);           // see Note (3)

  Serial.println((String) "Set switch to channel " + channelNum);
}

int getCurrCh() {
  int currCh = -1;
  Serial.println("Getting current channel ...");

  Wire.beginTransmission(MOTOR_WRITE);  // begin with slave device address
  Wire.write(CMD_GET_CURR_CH);          // command byte
  Wire.write(PARAM_DUMMY);              // param byte
  Wire.endTransmission(false);          // see Note (3)

  Wire.requestFrom(MOTOR_READ, 1, true);  // see Note (4)
  while (Wire.available()) {
    currCh = Wire.read();
  }

  return currCh;
}

int getChannelCount() {
  int chCount = -1;
  Serial.println("Getting total channel count ...");

  Wire.beginTransmission(MOTOR_WRITE);  // begin with slave device address
  Wire.write(CMD_GET_TOT_CH);           // command byte
  Wire.write(PARAM_DUMMY);              // param byte
  Wire.endTransmission(false);          // see Note (3)

  Wire.requestFrom(MOTOR_READ, 1, true);  // see Note (4)
  while (Wire.available()) {
    chCount = Wire.read();
  }

  return chCount;
}
```