Sol-MakerMKII: Arduino Uno Based Solenoid Controller

Steven A. Bjørnson

March 5, 2013

1 Outline:

The Sol-Maker MKII is a shield designed for the Arduino Uno. This shield allows for individual control of up to twelve solenoids (push, pull, or rotary) utilizing an external power supply; 12 or 24 volts depending on the power requirements of the solenoids. Control of each solenoid is made possible by communication with each of the 12 output pins on the Arduino Uno1 pin per solenoid. The current code compiled on the Ardiuino allows the user to send messages on/off messages from a computer to the Arduino via the USB port.

2 Circuit:

The design of the circuit is quite simple. Each of the 12 digital outputs of the Arduino Uno (outputs 2 to 13) are connected to identical modules. Each module contains a resistor, a TIP120 transistor, and a diode. These modules allow the shared voltage (12 or 24 volt external power supply) to be controlled by the digital output of the Arduino for each solenoid.

In each module the TIP120 acts as a gate which is opened and closed by the 5 volt output from the digital output pins of the Arduino. The resistor is necessary

since the TIP120 provides very low impedance and so without it the Arduino could be damaged.

A diode is run in parallel with each solenoid. This is necessary because solenoids are inductors and after a solenoid is cycled (charged and released) the plunger returns to the starting point (by way of mechanical potential energy; a spring or gravity will return the plunger to home). This returning plunger pushes current the opposite direction in the circuit and thus without the diode this reversed current could potentially damage the transistor.

The second section of each module contains a resistor and an light emitting diode (LED). This section is connected in parallel to each of the digital output pins which control the solenoids. When a solenoid is activated, a corresponding LED is activated. The LEDs are powered by the 5V provided by the digital outputs and do not utilize the external power supply.

A resistor in series with the LED provides the same function as the TIP120's resistor.

The purpose of the LED is to provide visual feedback and, since it will continue to operate when the external power supply is not on and when a solenoid is not connected, it notifies the user which module is being activated. This is advantageous for fault checking during system setup (i.e. finding out which output you are controlling before plugging a solenoid into that modules output).

2.1   Solenoids:

- Sol1 through Sol12 - GUARDIAN ELECTRIC TP6X12-I-24D 24V Solenoids.

Any solenoid with a voltage range of 12 to 24 volts should work with this circuit. There is a possibility of damaging a solenoid by holding current for too long if it is not designed to hold for long periods of time.[1]

---

[1]Remember that the input power supply voltage must match the requirement of your solenoids

2.2   Resistor Values:

- R1 through R12 - 2.2kOhm

- LED-R1 through LED-R12 - ?

2.3   Diodes:

- D1 through D12 - ?

2.4   LEDs:

- LED1 through LED12 - any colour; 5mm;[2]

3   Control:

Control is possible utilizing the USB port on the Arduino. The code (outline below) allows the user to send an on/off byte to each solenoid individually by coupling the byte with an address. Thus, although Max/MSP has been utilized for control, any programming language or software can be used as long as the following serial message format is maintained:

on/off byte (ascii character 48 for 0 and ascii character 49 for 1) + solenoid channel byte (ascii value from 'a' to 'l'; 97 to 108) + 32 (ascii value for blank space) + 13 (ascii value for carriage return)

example: 0 97 32 13 will cause solenoid #1 to turn off.

---

and that your power supply should have a maximum current rating to allow all the solenoids to fire at once. For example, since GUARDIAN ELECTRIC TP6X12-I-24D solenoids require 24V and 198mA each, the power supply must produce 24V and be capable of producing a current of 2376mA.

[2]Resistors LED-R1 through LED-R12 are calculated based on 3.3Vf and 20mA. Thus, if an LED with a different Vf is used a change of it's corresponding resistor is required. The resistance of LED-R1 through LED-R12 was set higher than required in the hopes of preventing any issue problem when using different colour LEDs.

3.1   Arduino Code:

```
char buffer[40]; // stores data from serial.
int index = 0;


// channels represent HIGH/LOW state of each output
int channelA, channelB, channelC, channelD, channelE, channelF,
channelG, channelH, channelI, channelJ, channelK, channelL;


int highLowDeterm(int x) // converts int into HIGH or LOW for digitalWrite
{
  int returnValue;


  if(x > 0)
  {
    returnValue = 0x1;
  }
  else
  {
    returnValue = 0x0;
  }


  return returnValue;
}


void setup()
{
  Serial.begin(115200);
  for(int n = 2; n <  14; n++) //sets pins 2 to 13 to output mode
```

```
  {
    pinMode(n, OUTPUT);

  }

}


void loop()

{

  index = 0;
  do
  {
    buffer[index] = Serial.read();  // get a byte from the serial port

    if (buffer[index]!=-1)

    {
    index = index+1;    // -1 if no byte is present

    }
    else if (buffer[index-1] == 97) // checks for ASCII code 'a'

    {
     // if 'a' convert input to int and convert to HIGH/LOW
      channelA = highLowDeterm(atoi(buffer));

    }
    else if (buffer[index-1] == 98)

    {
      channelB = highLowDeterm(atoi(buffer));

    }
    else if (buffer[index-1] == 99)

    {
      channelC = highLowDeterm(atoi(buffer));

    }
    else if (buffer[index-1] == 100)
```

```
{
  channelD = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 101)
{
  channelE = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 102)
{
  channelF = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 103)
{
  channelG = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 104)
{
  channelH = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 105)
{
  channelI = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 106)
{
  channelJ = highLowDeterm(atoi(buffer));
}
else if (buffer[index-1] == 107)
{
```
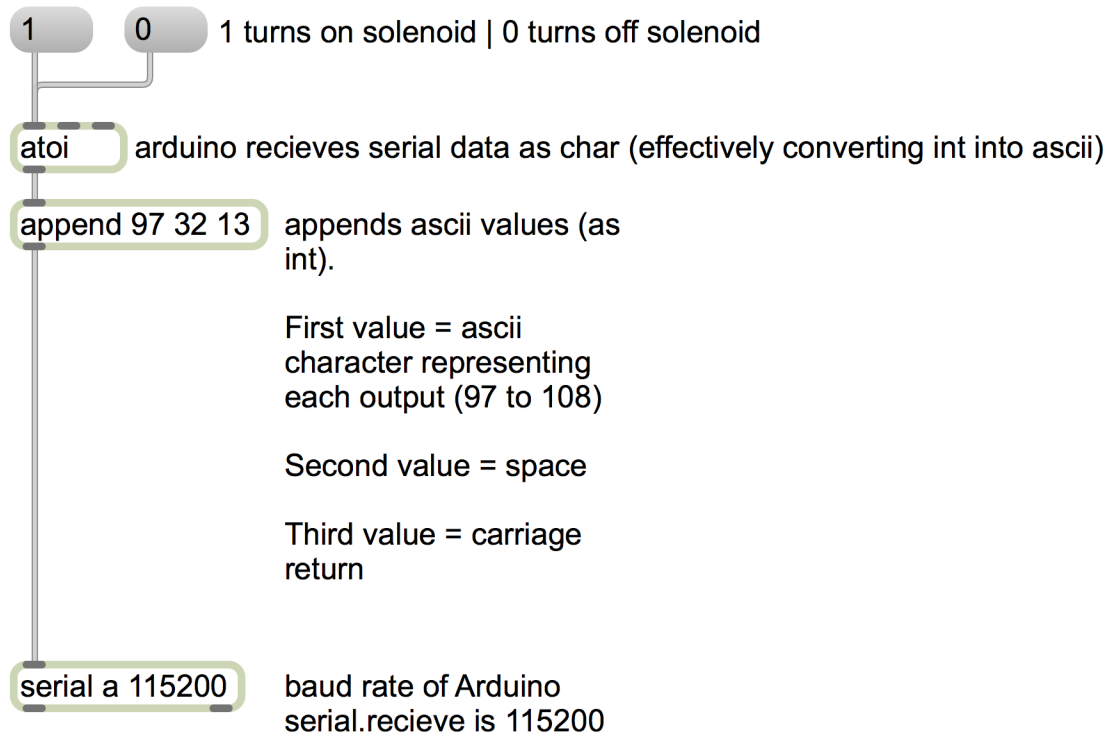
```
        channelK = highLowDeterm(atoi(buffer));

    }

    else if (buffer[index-1] == 108)

    {

        channelL = highLowDeterm(atoi(buffer));

    }


  } while (buffer[index-1] != 32);


    // Solenoids turned on/off
    digitalWrite(2, channelA);

    digitalWrite(3, channelB);

    digitalWrite(4, channelC);

    digitalWrite(5, channelD);

    digitalWrite(6, channelE);

    digitalWrite(7, channelF);

    digitalWrite(8, channelG);

    digitalWrite(9, channelH);

    digitalWrite(10, channelI);

    digitalWrite(11, channelJ);

    digitalWrite(12, channelK);

    digitalWrite(13, channelL);

}
```

3.2 Max/MSP Patch Example:

```
1        0        1 turns on solenoid | 0 turns off solenoid

atoi        arduino recieves serial data as char (effectively converting int into ascii)

append 97 32 13        appends ascii values (as
                       int).

                       First value = ascii
                       character representing
                       each output (97 to 108)

                       Second value = space

                       Third value = carriage
                       return

serial a 115200        baud rate of Arduino
                       serial.recieve is 115200
```

4   Dynamics:

Originally *pulse width modulation* (PWM) was used for all supported pins with the belief that control of the output voltage would be useful for the dynamic capability of the solenoids. This was unfortunate because the Arduino Uno only supports 6 pins for PWM. It became apparent, however, that PWM was unnecessary since the velocity of the solenoids can be effectively controlled with the hold-time–the length of time that current is pushed through a solenoid. Thus, if a solenoid is given current for 10ms it will hit with less force than if current is held for 20ms. Each solenoid is likely to have idiosyncrasies and thus a method of calibration would be useful but a simple scaling of MIDI velocity (0 - 127) with a range of 5 - 200ms should suffice.

5   Future Improvements:

- Multiplexer could be used to increase number of solenoids. An alternative would be to use an Arduino Mega which supports a larger number of digital output pins.

- Since the shield obscures the Arduino completely a serial send/receive light on the board would be useful.

- With an Arduino network adapter shield the system could easily utilize Open Sound Control (OSC) making the whole system more user friendly. Otherwise a simple midi input port would be sufficient for greater usability.

- To reduce the cost of manufacturing, the circuit could be redesigned to fit an Arduino Mini.