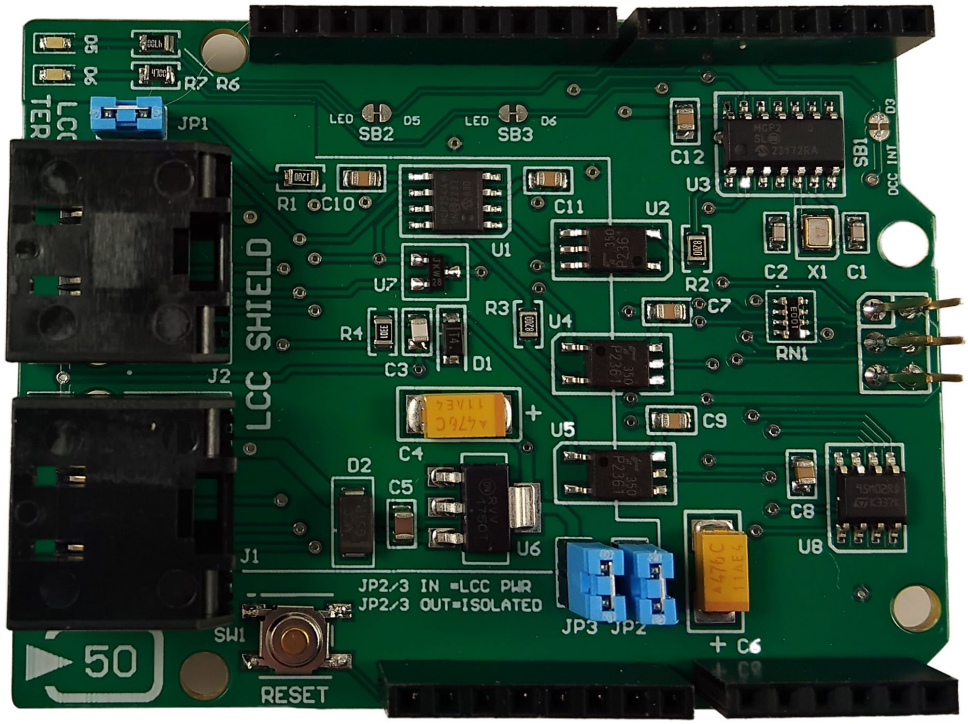
SNOWBALL CREEK
ELECTRONICS

LCC SHIELD

User Guide



Build low-cost, custom LCC Nodes utilizing an Arduino UNO form factor shield and your code. The possibilities are endless.

Contents

Snowball Creek Electronics

2081A Perkins Rd.

Grand Forks, B.C. V0H 1H1

CANADA

www.snowballcreek.com

1. Terms & Definitions

LCC

Layout Command and Control® (or LCC for short) is an NMRA standard for a layout control bus. The standards are created by the OpenLCB group, and then adopted as a standard by the NMRA. This open standard is designed to let all manufacturers connect to the layout control bus and interoperate with each other. For additional information, visit www.openlcb.org.

SPI

Serial Peripheral Interface (**SPI**) is a standard (with many variants) for synchronous serial communication, used primarily in embedded systems for short-distance wired communication between integrated circuits.

2. Features & Capabilities

- **Isolated LCC Interface**

  This allows minimum current draw from your LCC bus. The UNO plus any loads you wish to add are powered from your UNO supply, not the LCC bus.

- **DCC Receiver Circuit**

  This will send the isolated DCC data stream from the LCC ALT port pins to an UNO digital I/O pin. Can be disabled to gain the I/O pin for other uses.

- **256Kx8 EEPROM Included**

  Use the extra SPI accessible 256K memory to store CDI, data, or code.

- **Factory Installed LCC ID**

  Includes its own unique LCC ID programmed into EEPROM by the factory.

- **LCC Bus Terminator**

  Built-in LCC bus terminator. Just set jumper to enable.

- **Flexible Power Source**

  Able to be completely isolated from the LCC bus and use the Arduino power supplies, or the entire Arduino plus any shield(s) can be powered from the LCC bus.

- **Two on-board LEDs**

  Two programmable LED indicators provided. Optionally, you can disable the two LEDs to gain the additional I/O pins for other uses.

- **Designed to be used with LibLCC**

  LibLCC is a C library for LCC that runs on Arduino® and is available through the Arduino Library Manager.

This Arduino® compatible shield has all the hardware required for you to connect an Arduino® UNO or Arduino® Mega to an LCC (Layout Command and Control®) bus.

3. WHAT CAN I DO WITH THIS PRODUCT?

This product is intended for users who are familiar with Arduino® and want to create their own software. For example, custom logic running on the Arduino® can be used to perform more complicated automation than is normally available using standard commercial products.

With the input and output (I/O) of the Arduino board, plus the LCC Shield, you now have all of the hardware necessary to produce custom LCC nodes. Nodes that can produce and respond to LCC events can be programmed to perform virtually any function you may need for your model railroad.

This product is designed to be used with LibLCC, a C library for LCC that can be used on other operating systems(e.g. Linux) in addition to running on the Arduino®. Because LibLCC is available through the Arduino Library Manager, examples are available as normal Arduino® sketches. Currently available examples are:

- Simple IO board
- LCC computer interface
- DCC packet decoder

This shield is also compatible with Alex Shepard's NmraDcc Arduino library, allowing for DCC packet decoding.

The shield itself should also be compatible with the reference OpenLCB single-thread implementation of LCC.

4. Installing the Shield

The LCC Shield installs just like any other Arduino shield. Align all of the pins on the shield to the headers on the Arduino.

Keep in mind, you will need to align the 6-pin ICSP header pins as well.

Press the shield onto the Arduino board. Verify all of the pins are securely seated into their associated receptacles.

Once installed, you may mount other shields as needed. Make sure the pins used for the LCC shield will not also be used on other shields (if installed).

SPI Bus Usage

Note that because of how the SPI bus works on the Uno vs. Mega, the Arduino built-in LED is only available to be used on the Mega. The built-in LED on the Uno will flash when the SPI clock is active (ie. whenever there is a data transfer happening over SPI).

The LCC Shield uses the 6-pin ICSP header for access to the Arduino SPI port. On an Uno, these pins are also connected to D11, D12, and D13. Keep this in mind when you are selecting I/O pins for your device.

Refer to I/O Pin Usage table.

5. UNO & MEGA I/O PIN USAGE

Pin Number	Uno Pin Usage	Mega Pin Usage
D0	Serial RX	Serial RX
D1	Serial TX	Serial TX
D2	MCP2518 IRQ	MCP2518 IRQ
D3	DCC Signal. Cut SB1 to use as GPIO	DCC Signal. Cut SB1 to use as GPIO
D4	GPIO	GPIO
D5	LED. Cut SB2 to use as GPIO	LED. Cut SB2 to use as GPIO
D6	LED. Cut SB3 to use as GPIO	LED. Cut SB3 to use as GPIO
D7	Memory Chip select	Memory Chip select
D8	CAN chip select	CAN chip select
D9	GPIO	GPIO
D10	GPIO	GPIO
D11	MOSI. Unavailable as GPIO.	GPIO
D12	MISO. Unavailable as GPIO.	GPIO
D13	SCK. Unavailable as GPIO.	GPIO / Built-in LED
A0	GPIO / Analog	GPIO / Analog
A1	GPIO / Analog	GPIO / Analog
A2	GPIO / Analog	GPIO / Analog
A3	GPIO / Analog	GPIO / Analog
A4	GPIO / Analog / I2C SDA	GPIO / Analog
A5	GPIO / Analog / I2C SCL	GPIO / Analog

- All remaining I/O pins not listed here are available as GPIO.

6. Power Requirments

Because the shield is designed with an isolated CAN interface, a powered LCC bus is required in order for any packets to be received or transmitted. This isolation circuitry should draw less than 20mA. Note that the board is conservative in its labelling, showing a 50mA draw. If you power the Arduino from the LCC bus as well, the power requirements may be higher. An Arduino Mega powered from the LCC bus is expected to draw around 80mA of power, but may be higher or lower depending on any auxiliary circuitry running from the Mega.

Powering from LCC Bus

To power the shield and the Arduino board (and all connected loads!) from the LCC bus port, install jumpers JP2 and JP3. *By default, these jumpers are NOT installed.* JP2 and JP3 connect ground and +5V from the shield regulator to the Arduino board.

Jumpers shown in 'not installed' position.

Powering from Arduino Supplies

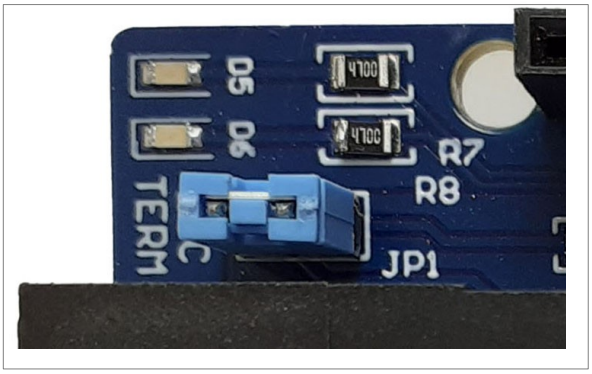
In order to have an isolated LCC port, jumpers JP2 and JP3 must NOT BE INSTALLED. In this mode, the shield LCC port interface circuit is isolated from the other circuitry on the shield. The SPI CAN interface IC (MCP2518) and EEPROM derive power from the Arduino. The CAN driver IC is powered from the LCC port. The Arduino can be powered from either the USB or the barrel connector.

Note: Although no damage will occur, is it recommended to only use one power source at a time. Do not install jumpers JP2 & JP3 AND have a powered Arduino for extended periods of time.

7. Configuring the LCC Interface

LCC Terminator

The LCC Shield has it's own built-in LCC bus terminator. To enable the LCC terminator, simply install jumper JP1. JP1 is located next to the LCC port, and is labeled “LCC TERM”. By default, the shield ships with JP1 installed.



JP1 shown installed.

8. Configuring Necessary Software

Note: All examples assume the pinout shown in section 5.

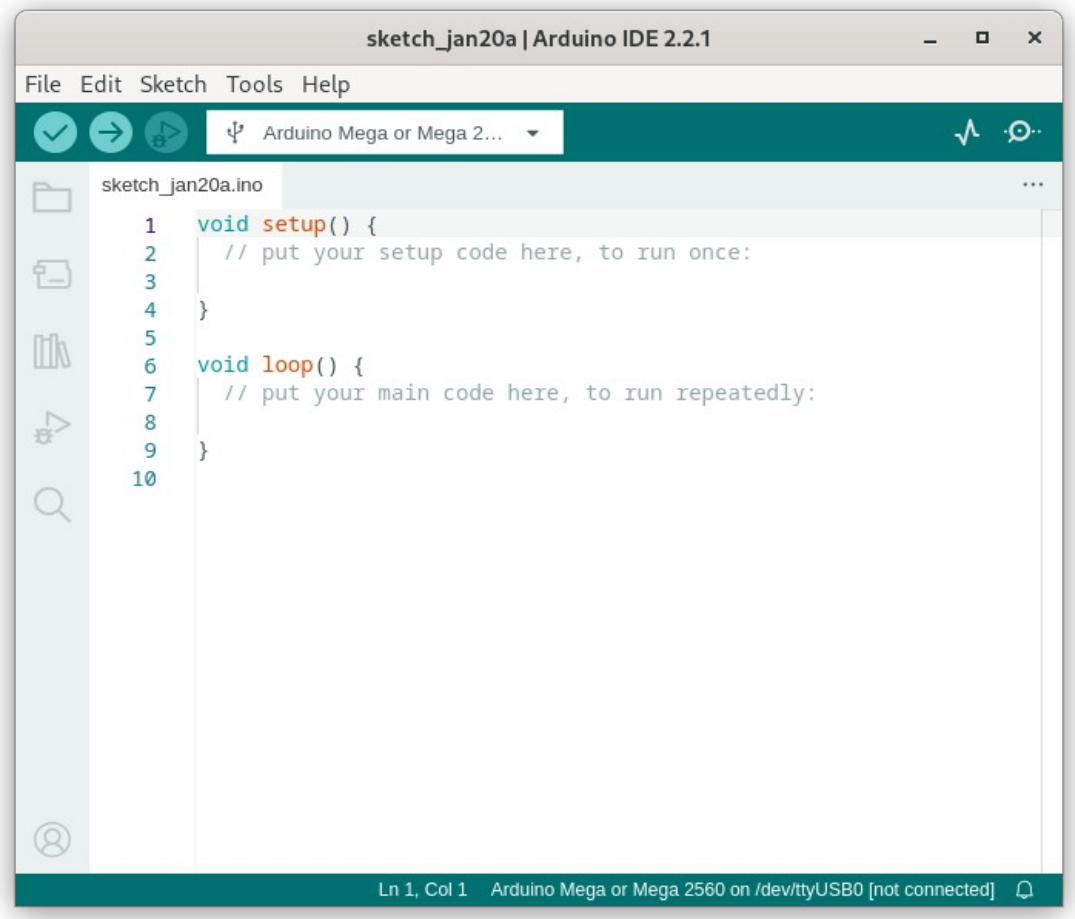
Several Arduino libraries are required to operate the LCC Shield. All of these libraries can be found via the Arduino Library Manager. Required libraries are;

- libLCC , ACAN2517 , M95_EEPROM

Installing LibLCC Through the Arduino Library Manager

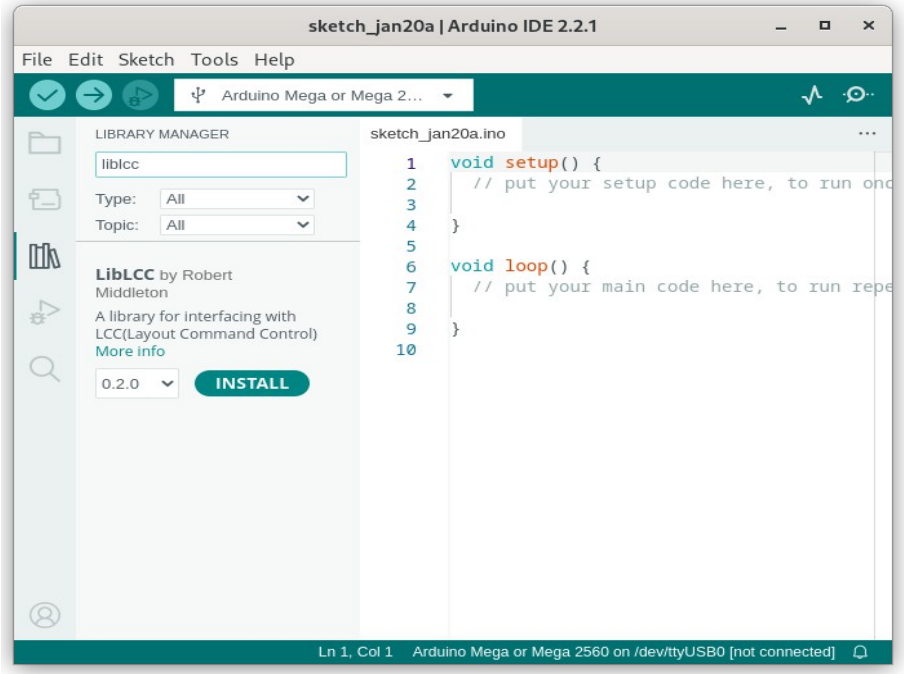
These images were taken with the Arduino IDE(Integrated Development Environment) version 2.2.1 running on Linux, but other platforms(Mac OS, Windows) are similar.

1. Open up Arduino IDE. With a blank project, the IDE should look something like this:



Open the Arduino Library Manager by either going to Tools→Manage Libraries… or by clicking the Libraries button on the left side of the IDE.

Search for libLCC and press the “INSTALL” button.



Now that the library is installed, you may also open up examples and look at them. You may access the examples at any time by going to File→Examples→LibLCC→(selection).

LibLCC – Additional Information

Note that all of the examples for LibLCC utilize the ACAN2517 library from Pierre Molinaro in order to interface with the MCP2518 CAN controller chip. This library is also available through the Arduino Library Manager, and may be installed in the same way as LibLCC.

In addition to the ACAN2517 library, the M95_EEPROM library is used in order to handle the communications with the EEPROM chip on the shield. This library is also available through the Arduino Library Manager.

Now that the LibLCC library and dependencies are installed, programming can begin. We recommend looking at the “Simple Node” example first, as that will initialize the node with a constant LCC ID and send out an event whenever pin D4 changes state(high→low, low→high).

Some important notes on the design of LibLCC:

- The library is designed as a C library. Because of this, it does not follow the same style as other Arduino libraries. If you are familiar with modular C code it should make sense. The reason for a C library is to make it adaptable from high-level systems(such as Linux) down to small micro controllers.
- Reading and writing are handled by callbacks
    - In order to push data to the library, you must call lcc_context_incoming_frame with an lcc_can_frame when a frame is available
    - The library calls a callback when it has a frame to write to the bus.
    - Because of the above points, it is adaptable to any concrete implementation of sending and receiving CAN frames.
- You may create different contexts in order to handle different parts of the LCC spec. When using an Arduino, these contexts are statically allocated. Due to the way that GCC works, if you do not create a context it will not be used, and thus will not take any memory. Contexts are:
    - Datagram context – handles reading/writing datagrams
    - Memory context – handles reading/writing datagrams. Required in order to send CDI information to a querying node.
    - Event context – handles sending and receiving LCC events

9. Accessing the 256K SPI EEPROM

The SPI EEPROM may be accessed by creating a new M95_EEPROM class as so:

```
static const byte EEPROM_CS = 7;
M95_EEPROM eeprom(SPI, EEPROM_CS, 256, 3, true);
```

The layout of memory in the EEPROM is application-defined. The specific EEPROM chosen has a 256-byte read-only page that is factory-programmed with the following information:

Offset	Length(bytes)	Usage	Default Value
0	8	Unique LCC ID	N/A
8	2	ID page version	1
10	32	Manufacturer	Snowball Creek Electronics
42	21	Part Number	SCE-230900
63	12	Hardware Version	Rev.4

The following code may be used to read and display the LCC ID and other manufacturing data from the EEPROM:

```
#include <M95_EEPROM.h>
#include <SPI.h>

static const byte EEPROM_CS = 7;
M95_EEPROM eeprom(SPI, EEPROM_CS, 256, 3, true);

struct id_page{
  uint64_t node_id;
  uint16_t id_version;
  char manufacturer[32];
  char part_number[21];
  char hw_version[12];
};

void print_node_id(uint64_t node_id){
    char buffer[3];
    sprintf(buffer, "%02X", (int)((node_id & 0xFF00000000000l) >> 40));
```

```
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
    sprintf(buffer, "%02X", (int)((node_id & 0x00FF000000000l) >> 32));
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
    sprintf(buffer, "%02X", (int)((node_id & 0x0000FF0000000l) >> 24));
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
    sprintf(buffer, "%02X", (int)((node_id & 0x000000FF00000l) >> 16));
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
    sprintf(buffer, "%02X", (int)((node_id & 0x00000000FF00l) >> 8));
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
    sprintf(buffer, "%02X", (int)((node_id & 0x0000000000FFl) >> 0));
    Serial.print(buffer[0]);
    Serial.print(buffer[1]);
}

void setup() {
  struct id_page id;

  Serial.begin (9600) ;
  while (!Serial) {
    delay (50) ;
    digitalWrite (LED_BUILTIN, !digitalRead (LED_BUILTIN)) ;
  }

  pinMode(EEPROM_CS, OUTPUT);

  SPI.begin();
  eeprom.begin();

  if(!eeprom.exists()){
    Serial.println(F("Unable to find EEPROM: PANIC!"));
    while(1){}
  }

  uint64_t node_id;
  eeprom.read_id_page(sizeof(id), &id);
  node_id = id.node_id;

  Serial.print("LCC ID: ");
  print_node_id(node_id);
  Serial.println();
  Serial.print("ID page version: ");
  Serial.println(id.id_version);
  Serial.print("Manufacturer: ");
  Serial.println(id.manufacturer);
  Serial.print("Part number: ");
  Serial.println(id.part_number);
  Serial.print("HW Version: ");
  Serial.println(id.hw_version);
}
```

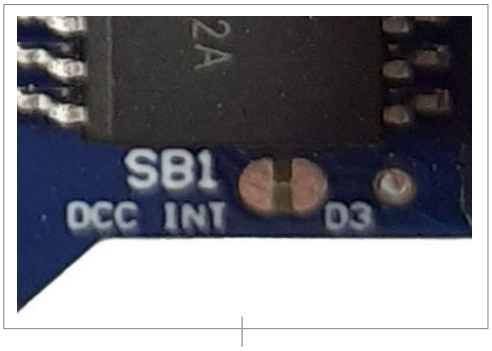
```
void loop() {
  // put your main code here, to run repeatedly:

}
```

10. Receiving DCC data from the LCC ALT port

LCC uses pins 4 and 5 of the RJ45 connector in order to pass the DCC signal to connected devices. The LCC shield has these pins connected to the Arduino in order to provide DCC decoding capabilities. The DCC Decode example of LibLCC uses the NmraDCC Arduino library in order to decode switch commands sent by the command station. The NmraDCC library is available through the Arduino library manager.

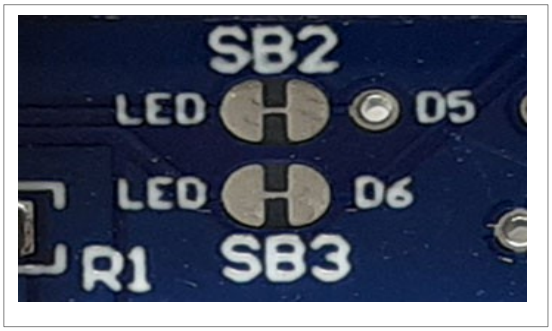
If you are not going to utilize the DCC decoding feature, it can be disconnected, thus gaining the extra I/O pin D3 for other uses. To disconnect, use a sharp hobby knife (X-acto) and cut the small trace between the two larger pads of SB1. If you need to re-connect the DCC to pin D3, cover both pads of SB1 with solder, creating a solder bridge.



If you do not have an LCC command station, you may simply modify a CAT5(or equivalent) cable to connect pins 4 and 5 directly to your track.

11. Configuring & Operating the On-Board LEDs

By default, Arduino header pins D5 and D6 are connected to two on-board LEDs located near the LCC port connector. If pins D5 and D6 are needed for other purposes, you can disconnect the LEDs by cutting the small trace between the larger pads of SB2 and/or SB3. To reconnect, add solder to the larger pads to create a solder bridge.



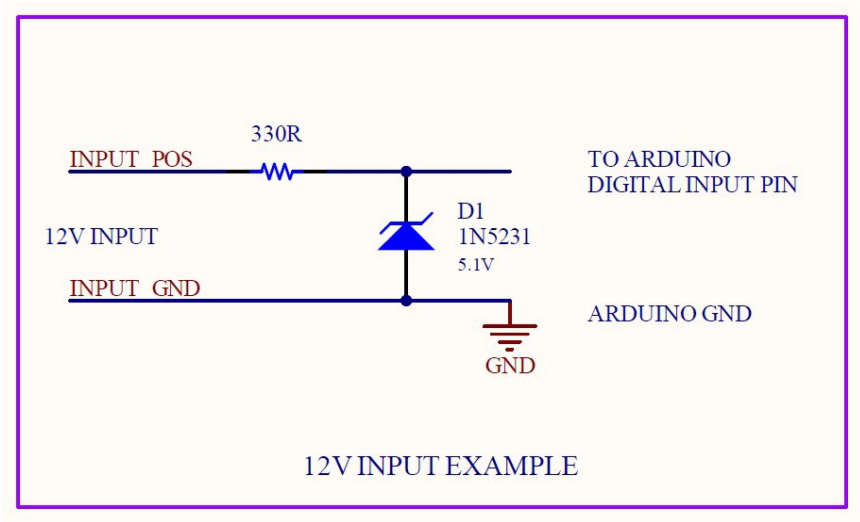
The following code will turn ON both LEDs.

```
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
digitalWrite(5, 1);
digitalWrite(6, 1);
```

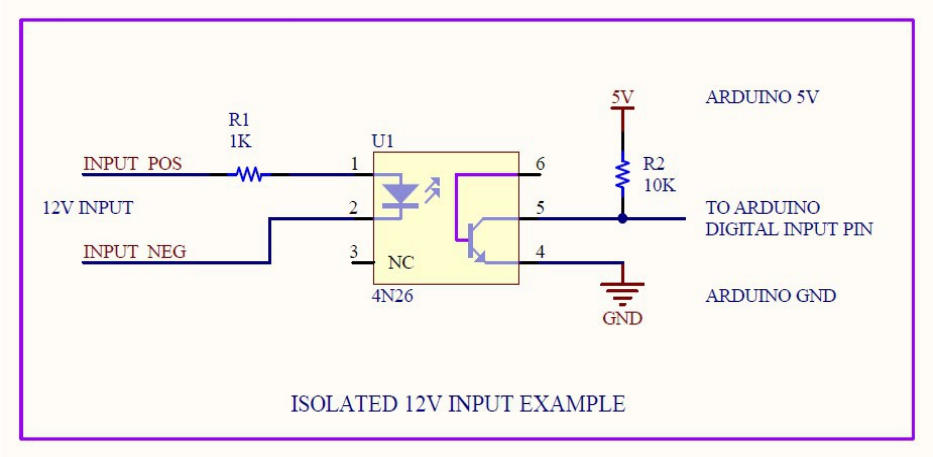
12. EXAMPLE CIRCUITRY FOR INTERFACING

There are many online sources with examples of how to interface an Arduino with various inputs and outputs. Here are a few basic circuits.

INTERFACING WITH 12V INPUT

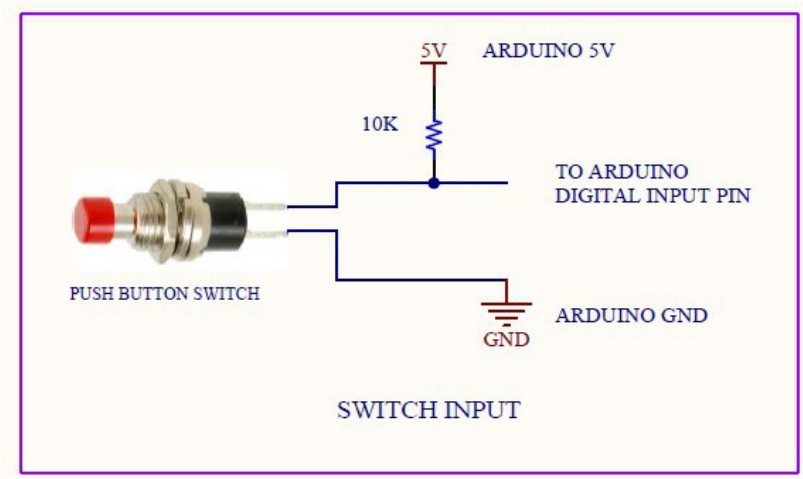


12V input connected directly to Arduino



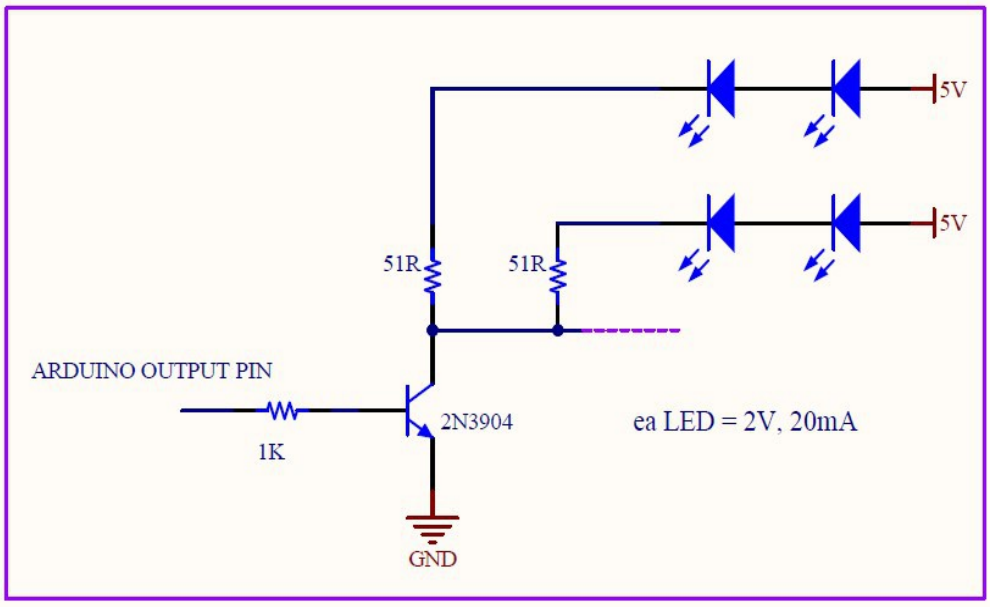
12V input connected via isolation (recommended)

SWITCH INPUT EXAMPLE



Push button switch input

MULTIPLE OUTPUT LEDS



Multiple LED's driven by Arduino & NPN transistor (40mA)

Note: Resistor value depends on power supply voltage, LED current rating, and LED Forward Voltage drop. The 2N2904 can drive up to 500mA total current.

13. SHIELD SCHEMATIC DIAGRAM