PinLightShield / SwitchShield

User Guide

Version 0.2

06. Feb. 2018

1 Introduction

In 2011 a member of the German Flippermarkt forum (www.flippermarkt.de) came up with a little project, where he built a shaker into his STTNG and added an RGB LED strip to the back of the backbox as kind of a surround lightshow. The shaker as well as the LEDs were controlled by a popular Arduino™ board which used various signals from the pinball machine as inputs (flasher, lamp matrix, coils). A little piece of software running on the Arduino then used those inputs to control the outputs, which via a few power transistors then drove the output devices (shaker and LEDs). Of course there was additional hardware needed between the pin and the Arduino and also between the Arduino and the output devices.

When I came across this project I found it so interesting that I immediately started to implement my own lightshow for my AC/DC Premium pinball machine. The result was so amazing, that I was sure that many pinheads with a little programming skill might be interested in this. But not everybody likes to wire and solder breadboards so I started thinking about a solution which should be as "solder-free" as possible to make it attractive for as many pinheads as possible. Not everybody is into programming either, but software can be shared while sharing the soldering effort is a bit more difficult. ☺

The PinLightShield is the result of these thoughts and this User Guide shall explain what it does and how it can be used in a pinball machine.

2 Overview

The PinLightShield is a so-called Arduino Shield. A "shield" is an extension to an Arduino Uno board which sits directly on the Arduino via the respective connectors. No cables etc. needed.

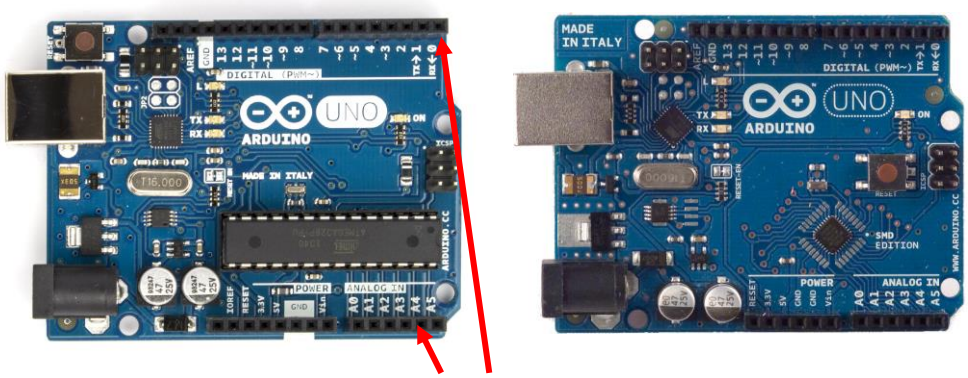


Figure 1: Arduino Uno R3 boards (see connectors on top and bottom to connect shields to the board)

The Arduino Uno has 16 digital ports (D0-D13) and 6 analog ports (A0-A5) which can be used as inputs or outputs (defined by software). The PinLightShield can use all ports as digital inputs with the following exceptions:

D0: not used, because it´s used internally by the Arduino for serial communication
D1: normal input or AC input to use the GI signal from pinball machines (determined by jumper W1)
D3, D5, D6: outputs to drive the power transistors (PWM)
D2, D4, D7, D8: can be used either as input or to drive addressable LED strips via J9
D9, D10, D11: input or output (PWM), determined by jumpers W2, W3 and W4

So why do we need the PinLightShield when the Arduino can already do all those nice things? Well, on the one hand the signals that come from a pinball machine are usually not directly compatible with the Arduino, so they need to be converted to lower levels. On the other hand we want to drive loads higher than those 50 or 100 mA the Arduino can drive. That´s what the PinLightShield does.

This is a block diagram of how all parts are connected:

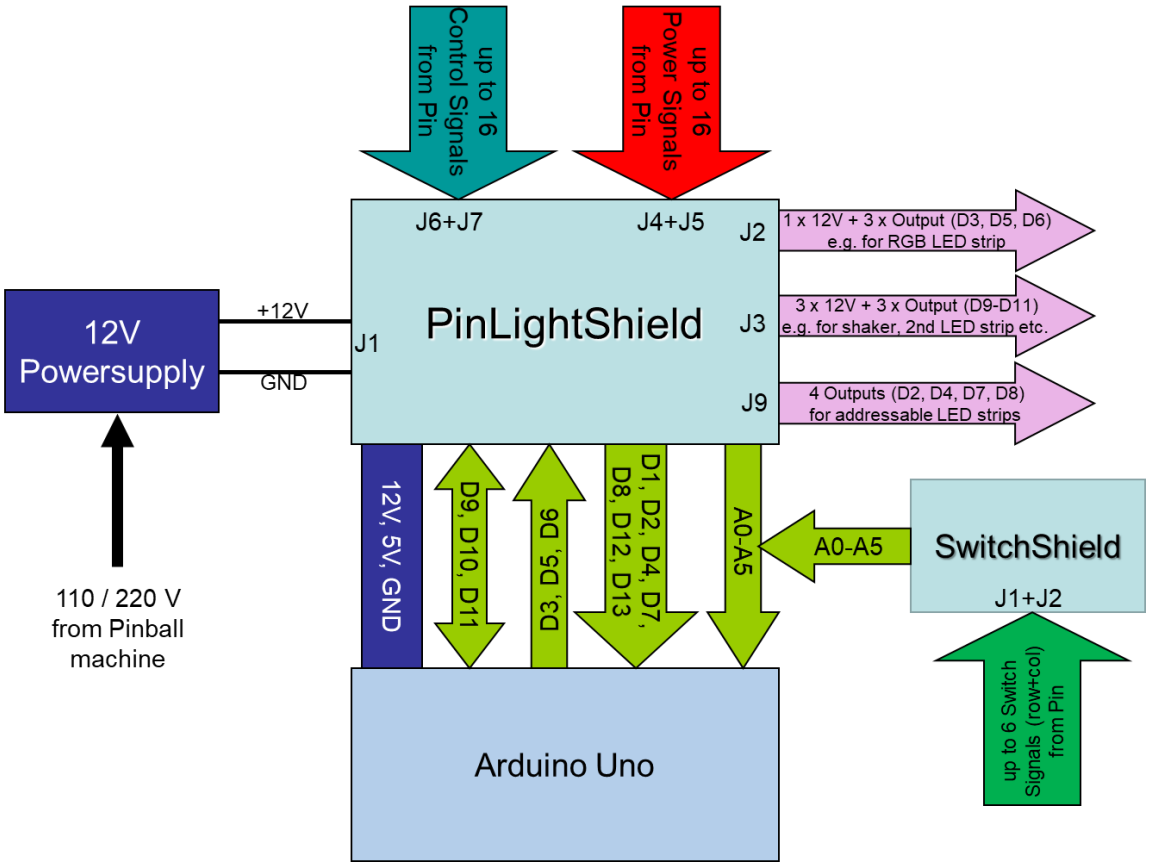


Figure 2: block diagram

From this picture it should become pretty clear what else you need to use the PinLightShield in your pinball machine – a 12V power supply, a few cables to connect to the input signals and a few cables to connect the devices you want to drive from the shield. And the software program that ties it all together. I´ll explain all that in the next chapters of this guide.

But first of all a picture of a fully assembled PinLightShield with descriptions of the various connectors and sections on the board:

3 Assembly and installation in your pinball machine

3.1 Assembling the PinLightShield

If you´ve ordered a kit you should have received one board and bag full of parts accompanied by a parts list that looks like this (for a rev2 board).

	PinLightShield Parts List		
Count	**Part**	**Value**	
15	R1-R15	2,2k	
1	R16	1,0k	
1	R17	1,2k	
1	R18	220 Ohm	
1	R19	680 Ohm	
2	RN1, RN2	8 x 10k	
3	C1, C2, C3	100µF 16V	
16	D1-D16	1N4148	
3	D17-D19	1N4004	
1	BR1	B40D	
4	OK1-OK4	ILQ1 or CNY 74-4	
6	Q1-Q6	STP16 or BUZ 71	
2	J1, J8	MKDSN1,5/2-5,08	
1	J2	Molex KK-156-4	
1	J3	Molex KK-156-6	
4	J4-J7	Molex 22-23-2081	
1	J9	Molex 22-27-2041	
1	Pololu D24V50F5 or D24V25F5	**optional**	
1	Arduino Stackable Headers	2 x 6, 2 x 8	
5	Jumper 2,54mm		
3	Header 3 pins		
1	Header 2x3 pins		
4	IC socket 16 pins		
1	LED red 3mm		
1	LED green 3mm		

Figure 3: Parts list for PinLightShield rev2

Assembling the board should be fairly self-explaining based on the silk screen print on the board. There´s just a few things to watch:

1. Correct orientation of all diodes
2. Correct orientation of the resistor arrays RN1 and RN2
3. Correct orientation of the capacitors C1-C3
4. The LEDs need to be installed with the longer leg (anode) facing upwards

3.2 Mounting the PinLightShield on the Arduino

Just insert the long pins that come out of the backside of the Shield into the respective headers on the Arduino so that the whole assembly looks like this:

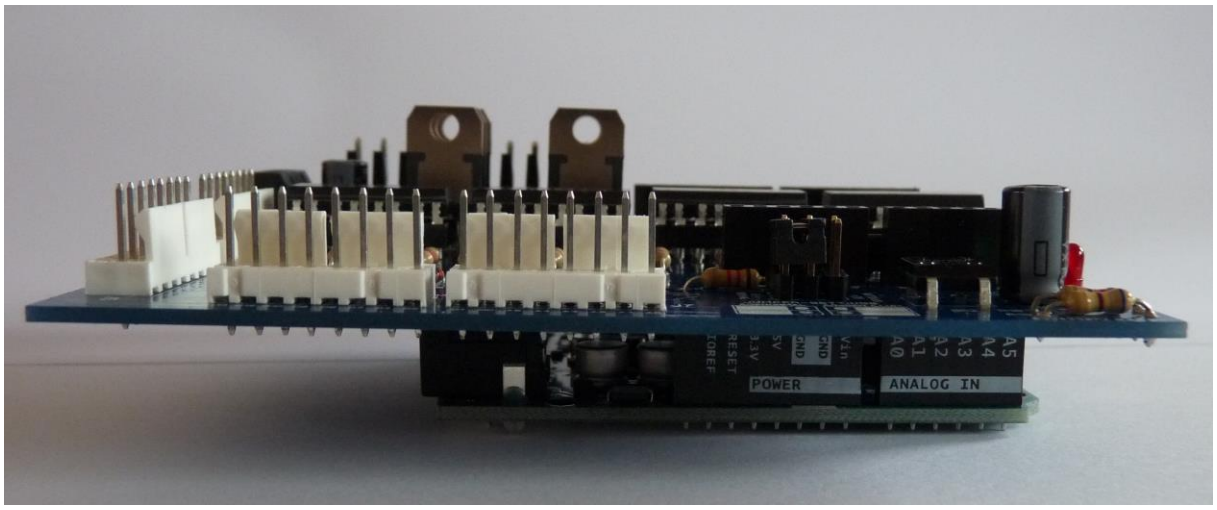


Figure 4: PinLightShield stacked onto Arduino board

Watch out that all pins really found their way into their respective socket.

Note: on the first version (Rev 1.0) of the PinLightShield it is possible that the USB connector of the Arduino touches the solder pads of the power transistors causing shorts and possible damage to the shield. In that case protect the shield by applying some insulating tape to the top of the USB port. This has been corrected in rev2 of the PinLightShield.

3.3 Power Supply

What Power Supply you need mainly depends on the consumption of the devices you want to drive. You can draw the power from the pinball machine or use an external power supply, but it must provide 12 VDC which will drive the Arduino, the PinLightShield and all devices connected to the Shield. The Power Supply connects to J1 on the PinLightShield.
If you use an external power supply I´d recommend the primary side of the Power Supply to connect to the power switch of the pinball machine, ideally to the "switched side" of it so that it only consumes power when the pin is on.

Be careful when you make that connection because you´re dealing with potentially lethal voltages of 110 or 220 V. Only do this with the pin completely disconnected from power!!!

It is also recommended to put a fuse between the Power Supply and PinLightShield just for additional safety.

3.4 Input Signals

The PinLightShield can potentially consume a number of different input signals, but so far only System11, WPC, DE and newer Stern machines have been tested. This section covers how you would connect the various input signals to the shield.

3.4.1 Physical Connection

To keep the pinball machine as untampered as possible it´s a good idea not to solder the wires directly to the flasher, coils, inserts etc. Instead I started with self-built splitters using IDC connectors that fit exactly on the headers in the backbox. Here´s an example I built for my AC/DC machine, where I had to connect the PinLightShield to four connectors in the backbox to get the signals I wanted to use for my lightshow:

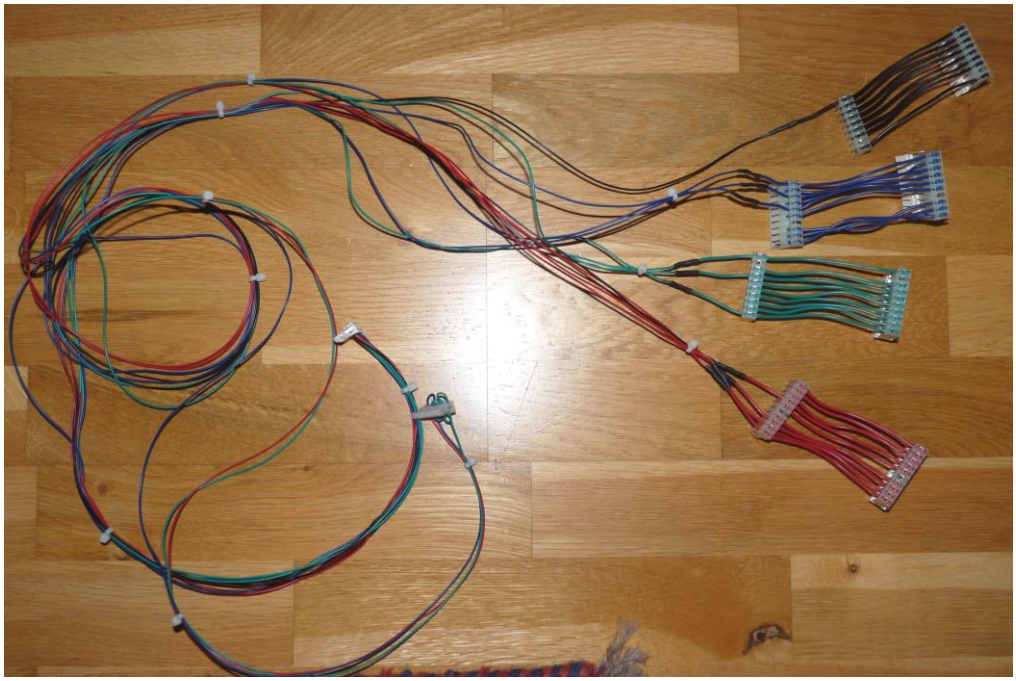


Figure 5: Typical wiring with IDC connectors

Recently I´ve built little adapters for the typical connectors that are used in pinball machines.

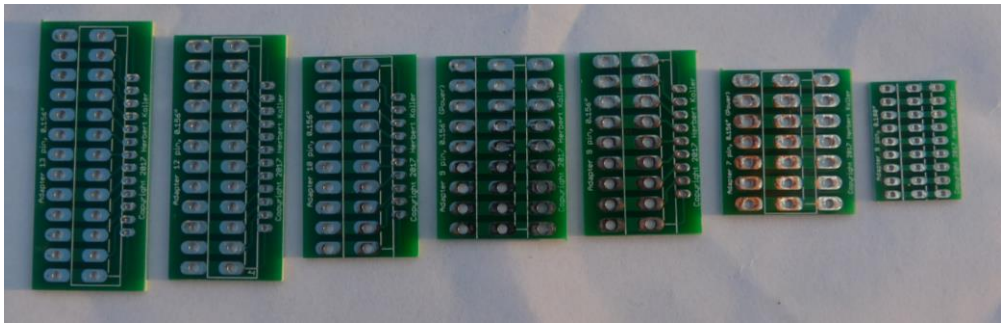


Figure 6: Adapter boards

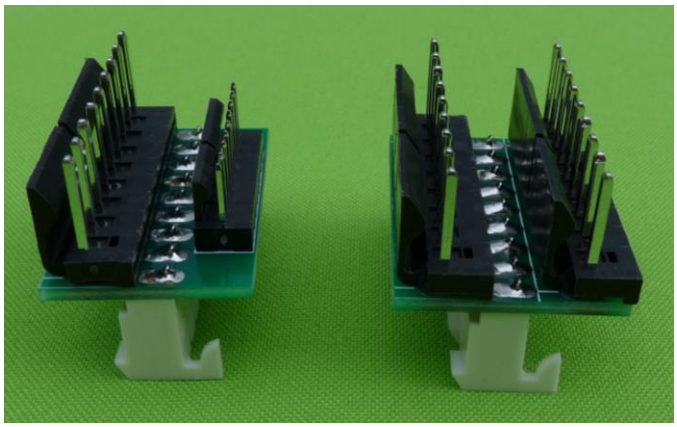


Figure 7: assembled 9-pin adapter board (left for normal signals, right for POWER signals)

These adapters can be installed on the original headers of the pin providing now two headers instead of one. One header is for the original cable, the other for the connection to the PinLightShield.

This is the section of the PinLightShield the signals from the pin get connected to:

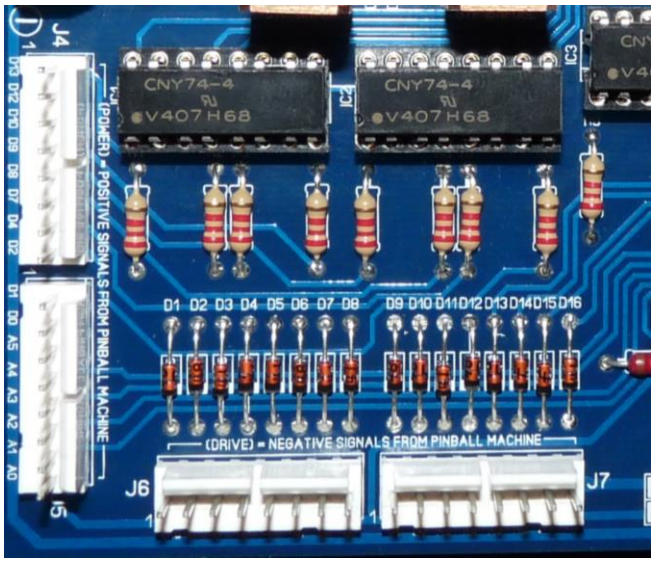


Figure 8: Connectors on the shield

The two connectors on the left are the ones for the Power Signals, the two on the bottom are for the Drive (Control) Signals. Every pin carries the name of the Arduino input signal it´s connected to (e.g. D13 is connected to J4-1 and J6-1).

3.4.2 Coils and Flashers

For coils and flashers there´s usually a “Power” and a “Drive” or “Control” signal coming from the pin. The Power Signal is always high using different voltages depending on the particular pin (20VDC, 50VDC etc.). The Drive (or Control) Signal is also high as long as the coil/flasher is not activated. To activate it the pin pulls the Drive Signal low allowing the current to flow through the coil or flasher.

There´s two ways to connect the PinLightShield to these signals. In the following we assume that you want to use D13 as your input channel:

1.) Connect the Power Signal of the device you want to monitor from the pin to the Power connector J4 pin 1 on the shield (that would be the topmost pin in Figure 8). Then connect the Drive Signal of the same device from the pin to the Drive connector J6 pin 1 on the shield (the leftmost pin of J6).
2.) The other – and much easier - option which also works for these kinds of signals is to simply connect the Power connector J4 pin 1 to the 12V of the PinLightShield. The Drive signal of the device you want to monitor still needs to be connected to the pin as above. CAUTION: if you use this option you have to make sure, that GND of the PinLightShield is connected to GND of the Pinball Machine!!

This will cause D13 to become HIGH every time this particular flasher or coil gets activated by the pinball machine. And this in turn can be detected by the software (see below “Software”) and then acted upon.

3.4.3 Shakers

Shakers are usually driven by a roughly rectified AC signal of various voltages. Apart from that the mechanics are pretty similar to flashers and coils. There is a Drive Signal which needs to be connected to the PinLightShield. The Power Signal is an AC signal which we can´t use on the PinLightShield. But we can simply use one of the Power Signals from the flashers or coils instead. That will work perfectly well and the PinLightShield will reliably detect the shaker.
Alternatively we can use the PinLightShield 12V to supply the Power Signal just as explained above in chapter 3.4.2. (Option 2.)

3.4.4 Inserts (Lamp Matrix)

All pins with a Lamp Matrix have arranged the insert lamps in rows and columns. When the pin wants to activate an insert lamp it applies a positive signal to the column side and at the same time pulls the row side to ground. This can be detected by the PinLightShield and then consumed by the software.
To use an insert signal we have to connect the positive side (the column) to the Power connector on the PinLightShield and the negative signal of the insert (the row) to the Drive connector on the PinLightShield.

So for example if you want to use the Extraball insert on a Scared Stiff which is lamp 48 in the matrix (meaning it is connected to column 4 and row 8 of the matrix) and you want again to use D13 as the input channel you would have to make the following connections:

PDB J121-4 to PinLightShield J4-1 (the column signal = positive signal)
PDB J125-9 to PinLightShield J6-1 (the row signal = ground signal)

This connection would cause D13 to become HIGH whenever the Extraball insert lamp on the pin is activated.

I have not yet experimented with the inserts on more recent pinball machines like my AC/DC or others. Many inserts there are RGB LEDs so in fact 3 lamps in one. Therefore it will probably be more complicated to use these lamps as input to the PinLightShield than on older machines.

3.4.5 GI (General Illumination)

The GI is usually driven by an AC voltage in the range of ~6 volts. On the PinLightShield we have a special input channel for a GI signal and that is D1. If you set both jumpers on W1 to "D1 = Gen.Illu.", which is the left position, then the GI signal will first be rectified and then fed into D1 to determine whether GI is currently ON or OFF.
You can see the red jumpers in the picture below in the bottom right part of the board.

Figure 9: Jumper W1 for the GI signal

On some pinball machines I found that the GI signal is also switched on and off with a relay which in turn is controlled via a coil signal. In that case you can also use the coil signal to determine the status of the GI in your pin (see section 3.4.2).

Be careful when you use the D1 signal with the software in debug mode. In that mode D1 is used for the communication with the Arduino IDE on the PC and therefore using D1 can render unpredictable results in the software.

3.4.6 Switches (Switch Matrix) and the SwitchShield

The switch matrix is a bit more tricky to deal with, because you need to analyze two signals to determine whether a switch is closed or not – the row and the column signal of that switch.

It could probably be done by the PinLightShield but with a few serious downsides:

- due to timing issues it might not be totally reliable

- we could use interrupts to add reliability but there´s only two of them on the Arduino
- we would waste two input channels for each switch that needs to be monitored

And not all pinball machines really require analyzing switches to create a great lightshow. On my AC/DC for instance I´m perfectly fine with just using flasher, coil and the shaker signal.

Therefore I decided to create the SwitchShield Extension for the PinLightShield which is able to read up to six switches and connect them to the inputs A0 to A5 on the PinLightShield from where it gets directly into the Arduino input. The extension is a fully compatible Arduino shield that can sit on top of the PinLightShield, but can also work with just an Arduino, if the purpose is something else than driving big output loads (e.g. if you only want to drive addressable LED strips and only need switches as inputs then you can do that with a SwitchShield sitting on top of an Arduino).

Every input that you use for a switch can of course no longer be used for other signals: so if you use A5 on the extension board for a switch detection the A5 connector on the PinLightShield must stay empty!!

3.4.6.1 Assembly of the SwitchShield

If you´ve ordered a kit you should have received one board and bag full of parts accompanied by a parts list that looks like this (for a rev2 board).

	Switch Shield Parts List		
Count	**Part**	**Value**	
7	R1-R6, R20	10k 0204	
12	R7-R18	6,8k	
1	R19	220 Ohm	
2	RN1, RN2	6 x 4,7k	
1	LED1	LED green 3mm	
3	IC2-IC4	74HC74	
1	IC1	74HC14	
6	C1-C6	1nF	
4	C7-C10	100nF	
1	Arduino Stackable Headers	2 x 6, 2 x 8	
1	IC5	MC33164	
2	J1, J2	Molex 22-23-2061	
4	IC sockets 14 pin		

Based on the silkscreen text on the board assembly should be fairly self-explanatory. The only part that´s not quite clear on the silkscreen is the LED which needs to be mounted with the long leg (=anode) facing left.

Here you can see the fully assembled SwitchShield:

Figure 10: Switch Shield fully assembled

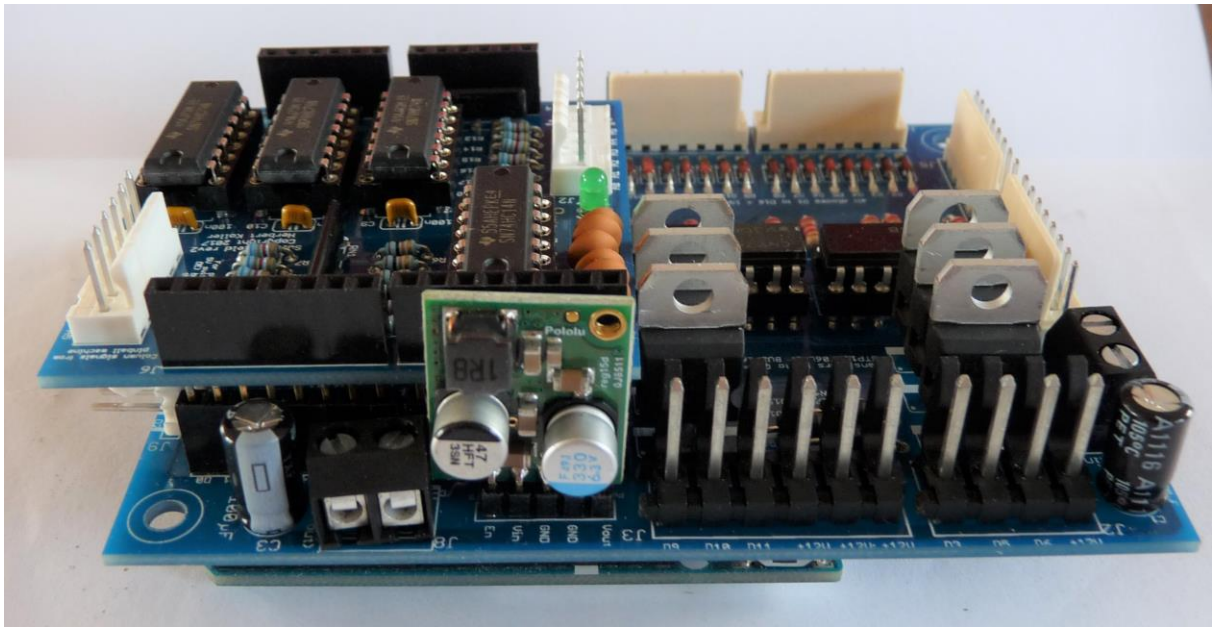


Figure 11: Side view (SwitchShield on top of PinLightShield on top of Arduino Uno

You just need to connect the row signal of the switch you want to monitor to J2 and the column signal of the same switch to the same input on J1 (e.g. A0). You can take the signal from the respective headers in the backbox (on WPC you even don´t need a splitter/adapter because the switch signals are available on two headers J206-J209) or you can take them directly from the switch (make sure to take the signal in front of the switch diode). When the switch is closed the Arduino input will read a HIGH and vice versa.

Note: There is an important difference between the Switch Matrix on WPC pins and for example Stern pins, which must be accommodated for by the Switch Shield. The WPC switch matrix operates at 12V while Stern and some other pins (Data East I believe) work with 5V.
Therefore the Switch Shield must be configured according to the machine it shall run in. If it runs in a machine with a 12V Switch Matrix then 6.8kΩresistors must be put on the shield for R7-R18. In case of a 5V Switch Matrix those resistors must be replaced

with 0Ωresistors or wire bridges. On rev2 of the SwitchShield there are solder pads on the backside of the board that can be used instead of bridges or 0Ωresistors.
Caution: never use a SwitchShield that is configured for a 5V switch matrix in a pin with a 12V switch matrix. This could destroy the SwitchShield.

3.5 PWM Output Signals

The PinLightShield can drive as many as 6 PWM output signals each one operating at 12 VDC. Three of these 6 are configurable so if you don´t need them you can use them as inputs. This is configured via the jumpers W2, W3 and W4 (see picture below).

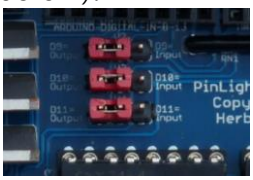


Figure 12: Jumpers W2, W3 + W4 to configure D9, D10, D11

All six output lines support PWM signals controlled by the software on the Arduino. The output signals are made available via the headers J2 and J3 which are 0,156" Molex headers.
We used these pretty big connectors because the shield is able to drive as much as 7A per output header. This is roughly equivalent to about 10m of RGB LED strips with 300 LEDs/m shining bright white.

Figure 13: PWM Output section of the shield (J2 + J3)

As you can see in Figure 13 J2 is a 4-pin header providing the 12 VDC plus 3 PWM signals (D3, D5 and D6) and therefore is ideally suited to drive RGB LEDs and LED strips. Which PWM signal drives which color of the RGB LED strip is entirely up to you and can easily be adapted in the software.
J3 is a bit more flexible as it provides three 12 VDC outputs as well as three PWM signals (D9, D10 and D11). Whether these can actually be used depends on the setting of the headers W2, W3 and W4 as described earlier in this paragraph.
So J3 can be used for a second RGB LED strip or it can be used for three totally separate devices (lamps, motors etc.) that run on 12 VDC.

3.6 Digital Output Section

Since rev2 of the PinLightShield there also is a Digital Output Section which can be used to control addressable LED strips. These strips need a stable 5 VDC power supply and a digital input signal that comes directly from the Arduino. To avoid the

necessity of a second power supply (in addition to the 12V the PinLightShield runs on) we´ve added the option of a Pololu 5V step-down voltage regulator on the PinLightShield. This regulator is a little expensive (5A version ~15$) but very efficient. For that reason it´s usually not included in the kit because not everybody needs it. This section can stay empty without affecting the functionality of the rest of the board.

Here is the new section on the board:

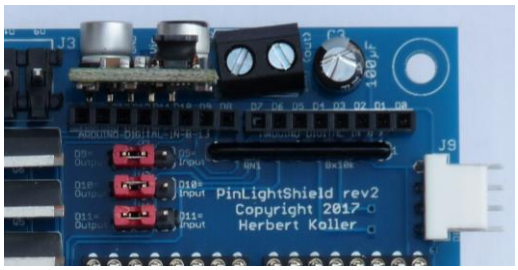


Figure 14: new 5V section with header J9 to drive addressable LED strips

You can see the Pololu on the top left of the picture with the 5V voltage connector to its right. The white header J9 provides 4 digital signals that can be used for up to 4 separate addressable LED strips. The output signals available for this are D2, D4, D7 and D8. Signals that are used for addressable LED strips can of course no longer be used as input signals.

3.7 Software

The software is the heart and soul of this whole thing. You´ll have to write it with the Arduino IDE that can be downloaded from the Arduino website (http://arduino.cc/en/Main/Software) and installed on Windows, Mac and Linux.
It´s a C-like language and you´ll find the reference, documentation, tutorials etc. also on http://arduino.cc. Therefore this User Guide will not be a replacement of all the documentation that is available out there about programming for the Arduino platform. We will only explain a few basic concepts and give a few hints and tips about what to do and what not.

3.8 Defining Inputs and Outputs

Usually the first thing you´ll do in your program is to tell the Arduino which inputs and outputs to use and what for.

Note: even though the Arduino has digital and analog input channels we use all of them as digital inputs, meaning that for us they are either HIGH or LOW, nothing in between.

Note: the Arduino language is **case-sensitive!!** That means that for the Arduino compiler #define is something completely different than #Define!!

To make your program more readable (and maintainable) I suggest to use a number of #define statements for this.

Here´s an example from my AC/DC sketch:

```
// The sketch processes the following input signals
// Digital Pin 0 shouldn´t be used because it´s used by the Arduino serial interface
// Digital Pin 1 is also used by the serial interface but I found it still can be used as input
#define BANDCOILPIN A0
#define CANNONCOILPIN A1
#define GIPIN A0
#define DETONATORPIN 2
#define SHAKERPIN 4
#define RIGHTSIDEFLASHERPIN 7
#define TRAINFLASHERPIN 8
#define BUMPERPIN 12
#define LEFTSIDEFLASHERPIN 13

// The sketch controls two sets of LED strips thru the following output signals
#define BLUEPIN 3
#define REDPIN 5
#define GREENPIN 6
#define BANDBLUEPIN 9
#define BANDREDPIN 10
#define BANDGREENPIN 11
```

The lines starting with "//" are comment lines. For the rest of your sketch you only use the real names of the signals (e.g. BANDCOILPIN) and don´t have to remember that this is connected to A0.

Then you actually tell the Arduino which signals are outputs and which are inputs. This happen in the setup() routine, which is always the first routine in any Arduino sketch. It is called once at startup of the Arduino or after you´ve downloaded a new sketch to it.

Here´s an example using the definitions above:

```
void setup() {

pinMode(REDPIN, OUTPUT);
pinMode(GREENPIN, OUTPUT);
pinMode(BLUEPIN, OUTPUT);
pinMode(BANDREDPIN, OUTPUT);        // jumper W2 must be in the left position for this
pinMode(BANDGREENPIN, OUTPUT);    // jumper W1 must be in the left position for this
pinMode(BANDBLUEPIN, OUTPUT);

pinMode(SHAKERPIN, INPUT);
pinMode(GIPIN, INPUT);
pinMode(TRAINFLASHERPIN, INPUT);
pinMode(BANDCOILPIN, INPUT);
pinMode(CANNONCOILPIN, INPUT);
```



```
pinMode(DETONATORPIN, INPUT);
pinMode(BUMPERPIN, INPUT);
pinMode(RIGHTSIDEFLASHERPIN, INPUT);
pinMode(LEFTSIDEFLASHERPIN, INPUT);
}
```

So this for example tells the Arduino, that we want to use its channel REDPIN (=digital channel 5) as an output and its channel BANDCOILPIN (=analog channel A0) as an input.
You can do more stuff in your setup() routine (like seeding random generators etc.) but remember that this routine is only called once at startup.

Immediately after the setup routine follows the code that does the actual work. In the Arduino world this is the so called loop() routine. As the name already implies the processor runs through this routine again and again in an infinite loop until it´s powered off or somebody presses the reset button.
This is the place where you read the inputs from the pinball machine, do whatever logic is needed and then write to your outputs to activate whatever devices you´ve connected to the PinLightShield.

3.9 Reading Inputs

As noted above the PinLightShield uses all Arduino inputs as digital inputs which only have two states: LOW or HIGH. Therefore we use the Arduino function digitalRead() to read an input signal and determine whether it´s LOW or HIGH.

This for example can happen in an IF statement:

```
if (digitalRead(SHAKERPIN)==True)
  {
  ..........;          // do something
  ..........;          // do some more
  ..........;          // and even more
  }
else
  ..........; // do something else
```

This code block will read the SHAKERPIN signal and if it´s HIGH (=True), meaning that the shaker is currently active, it will execute the three statements in the first block. If it´s LOW (=False) it will execute the one statement in the second block.

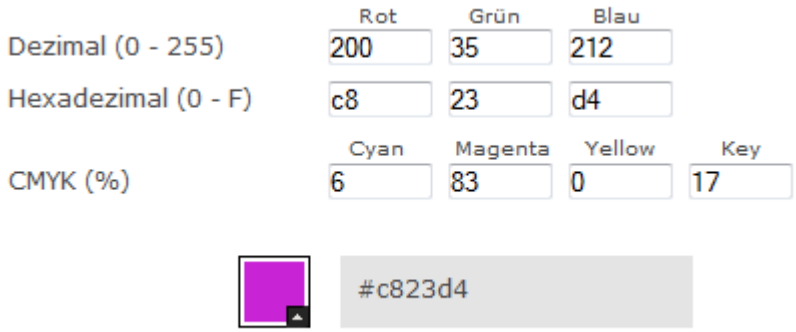
3.10 Writing Outputs

The Arduino outputs that are used by the PinLightShield are so-called PWM outputs (PWM = Pulse Width Modulation). Without going into too much detail this basically means that we can drive signals with varying intensity through them. This is

accomplished by the Arduino function analogWrite(pin, value). The two arguments in parentheses have the following meaning:

pin: the Arduino pin to write to, e.g. 3 (=D3)
value: intensity of the output signal between 0 (=completely off) and 255 (=completely on)

So an LED driven by the command analogWrite(3, 200) will shine approximately four times brighter than an LED driven by analogWrite(3, 50).
The commonly available RGB LEDs do actually consist of three independent LEDs in the colors red, green and blue. By driving these LEDs with different combinations of intensities we can create almost any color we like. There is a German website that allows you to get a first idea which combination of red, green and blue you need to create your desired color. The website is http://www.farbtabelle.at/farben-umrechnen/ and it allows you to pick your desired color and then shows you the RGB values you can start with. Here´s a screen shot of the webpage where I´ve selected some kind of purple:



So the following code would create this color, provided that REDPIN, GREENPIN and BLUEPIN are defined correctly and your LED is connected to the corresponding outputs:

```
analogWrite(REDPIN, 200);
analogWrite(GREENPIN, 35);
analogWrite(BLUEPIN, 212);
```

So RGB LEDs and LED strips always occupy three output signals on the PinLightShield. For those you would typically use J2 as output.

From J3 you can of course drive another RGB LED strip, but you can also use it to drive devices that need only one output like a shaker for example. For that reason J3 has three individual 12V pins to make connections easier, especially if you want to drive several devices from it.

3.11 The PinLightShield library

To make things a little easier for novice programmers I´ve built a little library, that hopefully takes away some of the dirty groundwork. This library contains functions to read all kinds of inputs and control all kind of output devices, most notably adding multiple pre-configured effects for RGB LEDs and LED strips.
There´s a separate document describing that library and how to install and use it in your own sketches.
You can find the library including its documentation on github: https://github.com/BigLebowski59/PinLightShield

3.12 Driving addressable LED strips

To drive addressable strips I refer you to Adafruits Neopixel webpages (https://learn.adafruit.com/adafruit-neopixel-uberguide/software) and also Eric Lyons' work for the Pinduino on github (https://github.com/elyons/pinduino).

3.13 Some hints and tipps

This chapter contains hopefully helpful hints and tipps for writing your first Arduino-PinLightShield sketch. This chapter will grow over time.

3.13.1 The millis() function

This function returns the time in milliseconds for which the program has been running since the last start. It is very helpful use this function if you want certain events to run for a certain time. So for example you want to trigger a flash of your LEDs from a flasher signal in the pinball machine, but you want the flash to last longer than the flasher in the pin flashes.

In that case you´d store the current time with a statement like this:

```
StartFlash = millis();
```

And then in the routine that lights the LED strip you can check wether the desired time is over:

```
if (millis() – StartFlash > 300)     // if 300 ms are over
        .....                        // do this
else
        .....                        // do something else
```

I usually store the current time at the beginning of the loop() function in a global variable so I can always refer to it:

```
CurrentTime = millis();
```


3.13.2 Avoid the delay() function

You could accomplish something similar as above with the delay() function. But this function has a significant downside: your program basically stops running while it´s in the delay function. So if you have a statement like this in your sketch:

```
delay(1000);              // do nothing for 1 second
```

your program will not register any events for that whole time and you might miss something important (a switch being closed, a flasher flashing etc.).

3.13.3 The sequence of events

It is very important to understand that the main program of the Arduino (the loop() function) is running in an infinite and very fast loop (hence the name) and just sequentially executes all statements over and over again. So you need to decide which events are the most important to you and then put those first in your loop() function and protect them from being overwritten by some subsequent events.
For example if you consider the shaker event the most important one which you want to enhance with an orange colored flash from your LED strips then you need to put that event as the first one in your loop() function and then put everything else into the "else"-part of your statement:

This could look something like this:

```
if (digitalRead(SHAKERPIN))
    {
    MakeOrangeFlash(); // a little function lighting the LEDs orange
    }
   else
    if (digitalRead(TRAINFLASHERPIN))
     {
     MakeBlueFlash();    // light LEDs  blue
     }
    else
     if (digitalRead(RIGHTSIDEFLASHERPIN))
      {
      MakeRedFlash();    // light LEDs red
      }
     else
          .................
          .................
```

There´s many other ways to make sure that the right thing (and **only** the right thing) happens at the right time. This is just one example which I wanted to discuss to explain the importance of the sequence of events.

