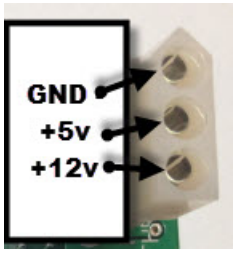
Physical Connections

There are 5 power receptacles.

- Pass Through - This connector is a fused pass through connector. The PIG 2 does not control this output. It is there as a convenience for you to connect another 'non PIG 2' mod without needing a splitter.
- OUT1 through OUT4

Each output has 3 pins. Some people find this confusing. You can use each output to control a mod that requires +5v and/or +12v. If your mod only requires 12v, you would wire your mod's Molex connector to use pins 1 and 3.

If your mod requires +5v, you wire pins 1 and 2. If you require +12v and +5v (not common), then you wire all three pins. The PIG 2 will control all of the mods on that connector the same. There is no difference between OUT1 5v or OUT1 12v as far as the PIG 2 is concerned.



- Pin 1 is ground
- Pin 2 is +5v
- Pin 3 is +12V

Many mods for Stern pinball machines are being created using this connector. You may find that those mods plug in without any problems.

Input Configuration

The default behavior of PIG 2 inputs is for each input to control its corresponding output. IN1 controls OUT1, IN2 controls OUT2, etc. If you would like to change how the Inputs work, you need to edit the PIG.TXT file that exists on the microSD card on the PIG 2. You will need a PC, a microSD card, and a text file editor like "notepad.exe" on Windows PCs.

There are other variables that can be changed for each output. These are the variables followed by the explanations.

```
# INPUT 1
invertIN1   = 0
mapIN1-OUT1 = 1
mapIN1-OUT2 = 0
mapIN1-OUT3 = 0
mapIN1-OUT4 = 0
bufferIN1   = 5
```

invertIN# [0 = OFF, 1 = ON]

This variable inverts the logic. If it is set to "1" then the logic is inverted. OFF is ON and ON is OFF.

mapIN#-OUT# [0 = OFF, 1 = ON]

By default, IN1 controls OUT1, IN2 controls OUT2, etc. You may want one input to control multiple outputs. For example, if you want IN1 to turn on two mods, one connected to OUT1 and one connected to OUT3, then these settings should be made:

```
mapIN1-OUT1 = 1
mapIN1-OUT2 = 0
mapIN1-OUT3 = 1
mapIN1-OUT4 = 0
```

One output can be controlled by multiple inputs as well. If

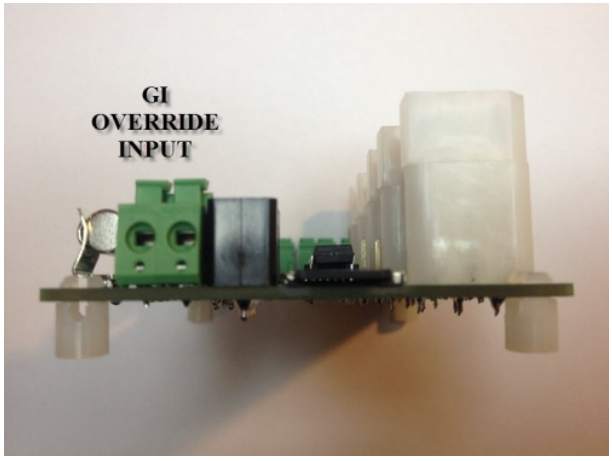
```
mapIN1-OUT1 = 1
mapIN4-OUT1 = 1
```

OUT1 will turn on if either IN1 **OR** IN4 are on.

bufferIN# [0 to 32767]

Electronic signals in pinball machines can be noisy. Small blips caused by interference or any other number of reasons can trigger an input to turn on. You can raise this value, in milliseconds, to "squelch out" noise. There is a balance. If this number is too high, then the input might ignore good signals. The best strategy is to start at a very low number (like 5) and raise it until there are no more "false triggers".

PIG 2 GI / Override Input



The GI / Override input is a unique input. It has the capability to override the actions of any or all of the outputs.

If you connect the GI / Override input to a General Illumination bulb on your game and set the mapGI# for an output to "1", then that output will only work when the GI is on. Many games use the general illumination to add to the effect of playing. Imagine a game that goes dark when the ball drains, but 2 or 3 of your lit mods are lit, blazing away when the rest of the game is dark. With the GI / Override input connected it will stop that from happening.

Another use for that input is to connect it to the ball trough switch furthest away from the shooting lane. During attract mode on a game lights go on and off in different patterns. You may have effects, such as figures on Servos that you do not want to move during attract mode. With the GI / Override connected to the last ball in the ball trough, you essentially turn off the defined mods when the game is not being played because all of the balls are in the trough.

PIG.TXT Variable: mapGI#

mapGI# = [0=Do Not Map, 1=Only when GI is ON, 2=Only when GI is OFF]

Example

```
mapGI1 = 0
mapGI2 = 1
mapGI3 = 1
mapGI4 = 2
```

- OUT1 is connected to a Topper that we want on at all times.
- OUT2 is connected to under cabinet lighting which we want OFF when GI is OFF.
- OUT3 is connected to a lit sign inside the game which we want OFF when GI is OFF.
- OUT4 is connected to a special light that highlights a toy when GI is OFF. We only want this to go on when GI is OFF.

• Inverting the Input for switches

- From DK Pinball
- Jump to: navigation, search
- In the PIG.TXT file, there is an **"invert"** option on each input

```
• # INPUT 1
• invertIN1    = 0
• mapIN1-OUT1 = 1
```

- `mapIN1-OUT2 = 0`
- `mapIN1-OUT3 = 0`
- `mapIN1-OUT4 = 0`
- `bufferIN1   = 10`

- Why do switches need you to set "invertIN# = 1"? The short answer is: It just does. Don't worry about it. If you connect a mod and it's doing the opposite of whatever you want it to do, set "invertIN# = 1" and that will fix it. If that's good enough for you, there's no need to read anymore.

-

- But you want to know why. I've been trying to come up with a good explanation and I won't be surprised if I change this page a few times to get the explanation right. The PIG 2 inputs use a little bit of electricity. About 1.2v and between 2.2ma and 11ma depending on the source voltage.
- When you connect a PIG 2 input to a light bulb, normal operation is that when the bulb is ON, electricity flows through the bulb to ground. The PIG 2 input will sense that electricity flowing (using a little bit of that electricity for itself) and will say that input is "ON". When the bulb is off, there is 0v on the positive side of the bulb and 0v on the negative side of the bulb. No electricity flows through the bulb or the PIG 2, so the PIG 2 says that input is "OFF".
- This is easy enough to understand. The bulb is ON, the PIG 2 is ON. The bulb is OFF, the PIG 2 is OFF.
-

  Imagine you have a switch connected between a 9v battery and a light bulb. The switch is there to hold back the electricity from flowing. It's a valve. If you connect the PIG 2 input to this valve, it behaves differently. Remember how I said that the PIG 2 uses just a little bit of electricity? When the switch is open, there is still 9v of electricity on the positive side and ground on the negative side.
- The PIG 2 sips a little bit of that electricity says that the input is "ON".
- If you close the switch, it's like a dam broke. All of that electricity flows through the switch, which has almost 0 resistance. The electrical current rushes through that switch and the PIG 2 input "dries up". All of the current flows through the switch using all of the voltage, not leaving enough voltage for the PIG 2, and the PIG 2 says that the input is "OFF".
- The end result of all of this is that on "Normally Open" Switches

- `Switch = ON  | PIG 2 = OFF`
- `Switch = OFF | PIG 2 = ON`

- That's probably not what you want. If this is what you want, Great! If not, set **invertIN#=1** and that will make

- `Switch = ON  | PIG 2 = ON`
- `Switch = OFF | PIG 2 = OFF`

Output Configuration

The default behavior of PIG 2 outputs is for each output to be controlled by its corresponding Input. IN1 controls OUT1, IN2 controls OUT2, etc. If you would like to change how the Outputs work, you need to edit the PIG.TXT file that exists on the microSD card on the PIG 2. You will need a PC, a microSD card, and a text file editor like "notepad.exe" on Windows PCs.

The default behavior of each output is to instantly turn completely on, and then instantly turn completely off. The meat of making interesting effects using the PIG 2 is done in configuring the Outputs.

Output Variables

```
# OUTPUT 1
powerMaxOUT1    = 254
powerMinOUT1    = 0
fadeOn1         = 0
fadeOff1        = 0
delayOn1        = 0
delayOff1       = 0
andOUT1         = 0
mapGI1          = 0
noInterruptOUT1 = 0
```

```
repeatOUT1      = 0
repeatDelayOUT1 = 0
repeatDecayOUT1 = 0
repeatResetOUT1 = 0
```

Output Variables Explained

```
NOTE: Variables based on time are set in Milliseconds.
1 Second = 1000 milliseconds.


This may seem overly precise but you'll find that the
precision is necessary.  Most pinball actions happen in
less than a second.
```

powerMaxOUT# and powerMaxIN#

powerMaxOUT and powerMinOUT *[Minimum 0, Maximum 254]*

You may not want your mod to get full power when it's on. Or you may not want your mod to be completely off when the input is off. Output power is set with a number 0 to 254. 0 is no power, 254 is full power.

If you want a mod on IN1 to be dimly lit, and then go to full power you set the variables:

```
powerMaxOUT1 = 254
powerMinOUT1 = 100
```

You may want a small motor connected to OUT4 to spin slowly, but when IN4 is active, you want it to spin faster, but not at full speed. You would then use the variables:

```
powerMaxOUT1 = 100
powerMinOUT1 = 50
```

You will have to experiment with the settings for your particular mod to get the look you want.

fadeOn and fadeOff

fadeOn and fadeOff [*Minimum 0, Maximum 65535*]

fadeON and fadeOff control the speed at which the output turns on and off in milliseconds. By default, they are both set to 0, which means the output goes on INSTANTLY and off INSTANTLY. If you would like your mod to go from minimum power to maximum power more slowly, you raise that number.

A 1 second fade on with a 3 second fade off for OUT1 would look like this

```
fadeOn0 = 1000
fadeOff0 = 3000
```

delayOn and delayOff

delayOn and delayOff *[Minimum 0, Maximum 65535]*

delayOn and delayOff control how long after the input is triggered (turned on or turned off) that the output will react. If you want OUT3 to go ON when a switch is closed and then STAY ON for 5 seconds after the switch is opened, you would use the setting

```
delayOff2 = 5000
```

You may have an insert light that blinks in 1/2 second intervals and then goes solid, but you only want the mod on OUT2 to turn on when the light is solidly lit. In that case you would use the setting

```
delayOn1 = 550
```

This would only turn on the output if the insert light is on for longer than half a second, or 500 ms. For this example I added a 50 ms buffer to make certain the timing works.

andOUT

andOUT *[0 = OFF, 2 - 4]*

If you have IN2 and IN4 controlling OUT4 then the default action is that if IN2 *OR* IN4 are on then OUT4 will turn on. You may want OUT4 to only turn on when IN2 *AND* IN4 are on. In that case you would turn on the "AND" logic for that output.

We do this by choosing how many inputs have to be ON before the output will go ON. If you have two inputs controlling one output but you want to both be on before the output turns on, you would use

```
andOUT4 = 2
```

If you want three IN's to be on before the OUT goes on, then it would be

```
andOUT4 = 3
```

There is an added configuration to make this interesting. If you have three INs controlling one OUT, you can set the andOUT to be only 2. This gives the added ability to turn on OUT when only 2 out of 3 inputs is ON.

mapGI

mapGI *[0 = OFF, 1 = ON only when GI is ON, 2 = ON when GI is OFF]*

If you want your mod to turn on and off with the General Illumination on your machine, you turn on mapGI.

```
mapGI0 = 1
mapGI1 = 0
mapGI2 = 2
```

- OUT1 will now turn on and off with the assigned input and GI.
- OUT2 will ignore the GI lights and turn on and off with its assigned inputs only.
- OUT3 will now only turn on when the GI is off.

noInterruptOUT

noInterruptOUT *[Minimum = 0, Maximum = 65535]*

Changes in the state of an input will interrupt the current action. noInterruptOUT sets how long the output will lock out interruptions.

Assume you set OUT3 to fade on for 3 seconds and then turn off. If a switch is pressed but 1 second after being pressed, the switch is released, the fade will stop after 1 second and the mod will turn off.

You may want a momentary press to go through the whole 3 second fade on before the mod turns off. You would set

```
noInterruptOUT3 = 3000
```

Repeat Variables

Repeating is how we create effects like blinking or pulsing. Once you understand what is actually repeating, everything else should make sense.

If you set repeatOUT# to a number higher than 0, then the triggering changes. Lets use IN1 and OUT1 with all of the default values as the example. Assume IN1 is connected to a bulb.

When the bulb turns on, IN1 is triggered as "ON". The PIG 2 then uses this signal to turn OUT1 "ON". As long as that bulb is on, OUT1 will stay "ON". Once the bulb turns off, IN1 is triggered as "OFF" and it turns OUT1, "OFF".

To make OUT1 turn ON then OFF, you require two triggers. An "ON" trigger and an "OFF" trigger.

If you set repeatOUT1 to more than 0, the PIG 2 will now **assume both triggers happened**.

```
delayOFF1 = 500
repeateOUT1 = 20
repeateDelayOUT1 = 1000
```

What this will now do is use the "ON" trigger from IN1 and make OUT1 turn ON. It assumes the the "OFF" Trigger happens immediately after the "ON" Trigger. We turn the light off 1/2 a second later (delayOFF1 = 500). Because we defined a repeateDelayOUT1 = 1000 ms, we'll wait another 500ms and then repeat (500ms Delay off + 500ms waiting to repeat the ON/OFF triggers = 1000ms total). The PIG 2 will repeat the 20 times (repeatOUT1 = 20).

The effect you will see when the lamp on the game turns on (the trigger) is the mod connected to OUT1 blink on and off 20 times. On for 1/2 a second and off for 1/2 a second.

repeatOUT#

repeatOUT# *[0 - 32767]*

If this is set to a number higher than 0, it will take the entire ON/OFF cycle and repeat it the number of times you define.

repeatDelayOUT#

repeatDelayOUT# ***[0 - 32767]***

Defines how much time to wait between ON / OFF cycles.

repeatDecayOUT#

repeatDecayOUT# ***[-32767 to 32767]***

Defines how much time to subtract from the repeatDelayOUT value after each ON/OFF cycle.

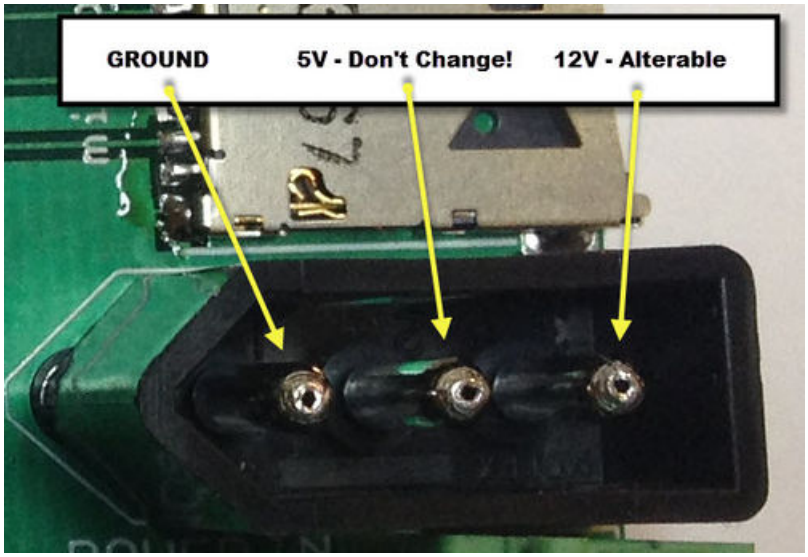
Using this will cause the blink to speed up. A negative number will result in a [blink]..... [blink].....[blink]....[blink]..[blink].[blink][blink] type of pattern.

A positive number will result in a blink that slows down. A [blink][blink]..[blink]....[blink]..... [blink].....[blink].....[blink] type of pattern.

repeatResetOUT#

repeatResetOUT# ***[0 = OFF, 1 = ON]*** Set to 1 (ON) this will reset the repeat count each time the IN is activated. If set to 1, and a switch is held for 30 seconds, the ON/OFF cycle will repeat for 30 seconds + the repeatOUT count.

Advanced Power Concepts



This is for Pro Users who understand electronics. If you don't have a firm grip on voltages, current, and how to calculate power, do not alter your power input!

There are three pins on the power input. The connector has a flat end and a pointy end. The ground pin is **Pin #1** and is up at the "pointy" end. I remember this because one day I was working with my friend Chad and he said, "Remember that you always 'point down to the ground'". It makes absolutely no sense really, but it stuck in my head and I hope it will now stick in yours.

Pin #2 is for 5v DC. **This cannot be changed!** ***EVER!***. All of the logic circuits and the micro-controller all require 5v.

Pin #3 is for 12v DC. It works out great because Stern has graciously provided we modders with a very nice 3 pin power plug that provides 5v and 12v, which are two great voltage levels that allow us to use so many great devices. LED strips used 12v. Many powered mods use 12v because we re-use a lot of lighting mods that come from cars which also run on 12v. However,

the 12v on Pin #3 isn't used for anything that the PIG 2 requires.

While it is doing it in interesting and creative ways, all the PIG 2 is doing is switching the 12v on and off at the outputs. If you want to control a few 9v mods, or even a 24v mod, you can feed 24v to PIN 3. PIN 3 on every output will then be a 24v pin. You can't mix and match. PIN 3 on the Power Input directly affects PIN 3 on every OUT.

Things to consider

Whether or not you stick with the 5v/12v or decide to go 5v/9v or 5v/24v, you need to consider the current that will be running through the PIG 2. There are two, 2amp fast blow fuses on the PIG 2. The PIG 2 has been designed to handle 2.5 amps of continuous current on the 5v lead and the 12v lead. Most mods will not use anything near that. You can run 100 LEDS without worrying about exceeding the specifications of the PIG 2. However, if you plan on running a motor, or a coil, or anything else that draws a lot of current, make sure you are keeping those specifications in mind.

At a later date we will be doing stress testing to find the real world maximums, but at this time as long as you keep the fuses at 2.5a or lower, you will not hurt the PIG 2.

Pulsing Flipper Buttons and Undercabinet Lights

OUT1 is connected to the Flipper Buttons
OUT2 is connected to the Under cabinet Lighting
GI / Override input is connected to a GI bulb.

To get this effect, we make the following changes to the PIG.TXT. We want the pulsing to be ON always, except when GI is off. We want it to slowly fade on and then slowly fade to "DIM", not completely off ever. We want the effect to repeat over and over as long as the GI is on. Knowing these things we set the following variables for the flipper buttons:
invertIN1 = 1 With nothing connected to IN1, it is always off. If we invert the input, ON = OFF and OFF = ON. Now with nothing connected, it will always be ON.
powerMinOUT1 = 5 This is DIM but not completely off. Different lights and bulbs will look different with the same levels. This requires some experimentation.
fadeON1 = 1500 Go from 5 to 255 (min to max) over 1500 ms (1.5 seconds).
fadeOFF1 = 1500 Go from 255 to 5 (mac to min) over 1500 ms (1.5 seconds).
mapGI1 = 1 When GI goes OFF, set the lights to the powerMinOUT1 value of 5.
noInterruptOUT1 = 1500 This one is a little tricky. We're going to set repeat to 1. This means go from ON to OFF automatically, and then repeat. This variable allows the FadeOn to complete before fading off again. Without this, the light would not go on at all.
repeatOUT1 = 1 This means we want the cycle to repeat 1 time. Since the Input is inverted and nothing is connected to it, it will always be "ON". It will now repeat the cycle over and over unless the GI is off.
repeatDelayOUT1 = 3000 This delay gives the fade cycle the time it needs to complete before starting the cycle again. It should be equal to (fadeON + fadeOFF).

Configuration for the undercabinet lights is similar. The only difference is that we change the pulse timing from 1500 to 2000.

```
Lit Flipper Buttons
# INPUT 1
invertIN1   = 1
mapIN1-OUT1 = 1
mapIN1-OUT2 = 0
mapIN1-OUT3 = 0
mapIN1-OUT4 = 0
bufferIN1   = 10
# OUTPUT 1
powerMaxOUT1    = 254
powerMinOUT1    = 5
fadeOn1         = 1500
fadeOff1        = 1500
delayOn1        = 0
delayOff1       = 0
andOUT1         = 0
mapGI1          = 1
noInterruptOUT1 = 1500
```

```
repeatOUT1      = 1
repeatDelayOUT1 = 3000
repeatDecayOUT1 = 0
repeatResetOUT1 = 0
Under Cabinet Lighting
# INPUT 2
invertIN2   = 1
mapIN2-OUT1 = 0
mapIN2-OUT2 = 1
mapIN2-OUT3 = 0
mapIN2-OUT4 = 0
bufferIN2   = 10
# OUTPUT 2
powerMaxOUT2    = 254
powerMinOUT2    = 20
fadeOn2         = 2000
fadeOff2        = 2000
delayOn2        = 0
delayOff2       = 0
andOUT2         = 0
mapGI2          = 1
noInterruptOUT2 = 2000
repeatOUT2      = 1
repeatDelayOUT2 = 4000
repeatDecayOUT2 = 0
repeatResetOUT2 = 0
```

Configuration Example: Flashing Input / Solid Output

Imagine a lit mod that you want to key on an insert light. You want the output to be a constant "ON" when the insert light is on. However, sometimes the insert light flashes on and off.

You want your output to act like this

```
INPUT - OFF       OUTPUT - OFF
INPUT - FLASHING  OUTPUT - SOLID ON
INPUT - SOLID ON  OUTPUT - SOLID ON
```

To achieve this, you set the "delayOFF#" variable. The delay should be longer than the "off" time of the flashing insert light. More than likely, this will require some experimenting to get perfect. In our case the flashing is ON for 300ms (.3 seconds), OFF for 300ms (.3 seconds), ON for 300ms (.3 seconds), repeat...

To keep the mod lit through the 300ms "OFF" time of the insert, we set the "delayOFF#" variable to 350ms. I added 50ms just to be sure the timing is right.

Here is the sample config for a mod connected to OUT2.

```
# OUTPUT 2
powerMaxOUT2    = 254
powerMinOUT2    = 0
fadeOn2         = 0
fadeOff2        = 0
delayOn2        = 0
delayOff2       = 350
andOUT2         = 0
mapGI2          = 0
noInterruptOUT2 = 0
repeatOUT2      = 0
repeatDelayOUT2 = 0
repeatDecayOUT2 = 0
repeatResetOUT2 = 0
```

This is an example. The Flynn's sign is keyed off of the Flynn's insert light. Even though the insert light flashes, the sign stays solidly lit.

Configuration Example: IN1 AND IN2 controlling OUT1

Imagine a mod that lights a 12v sign saying "MULTIBALL READY" when two balls are locked. This would require using the "AND" logic of the PIG 2.

```
Connect IN1 to the Ball 1 Lock Switch.
Connect IN2 to the Ball 2 Lock Switch.
Connect the "MULTIBALL READY" to OUT1 12v and GND pins.
```

You then change the PIG.TXT configuration file.

```
Change both invertIN1 and invertIN2 to "1".  Switches need to be inverted or
"OFF" will actually be "ON".
Change the map of IN2 to be OUT1.
Change the andOUT1 variable to 2 - Meaning 2 inputs must be on in order for
OUT1 to turn on.
# INPUT 1
invertIN1   = 1
mapIN1-OUT1 = 1
mapIN1-OUT2 = 0
mapIN1-OUT3 = 0
mapIN1-OUT4 = 0
bufferIN1   = 5
# INPUT 2
invertIN2   = 1
mapIN2-OUT1 = 1
mapIN2-OUT2 = 0
mapIN2-OUT3 = 0
mapIN2-OUT4 = 0
bufferIN2   = 5
# OUTPUT 1
powerMaxOUT1    = 254
powerMinOUT1    = 0
fadeOn1         = 0
fadeOff1        = 0
delayOn1        = 0
delayOff1       = 0
andOUT1         = 2
mapGI1          = 0
noInterruptOUT1 = 0
repeatOUT1      = 0
repeatDelayOUT1 = 0
repeatDecayOUT1 = 0
repeatResetOUT1 = 0
```