NANO Board GPIO Interface

Harry Li, Ph.D.

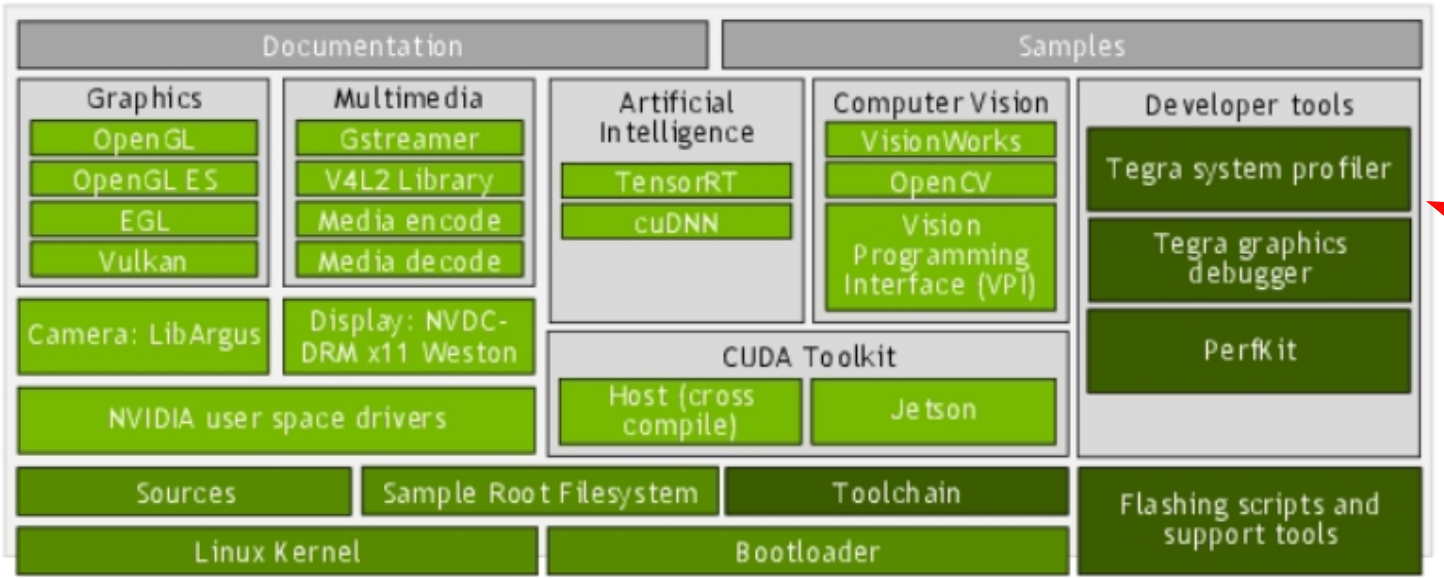
Jetson NANO Adaptation and Bring Up

This is the most comprehensive reference source

https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/adaptation_and_bringup_nano.html#wwpID0E0RR0HA

Nano Boards

Jetson Nano™ devices	Jetson Nano (P3448-0000) *Developer kit version*	Jetson Nano Developer Kit (P3450-0000) †; includes P3448-0000 module
	Jetson Nano (P3448-0002)	
	Jetson Nano 2GB (P3448-0003) *For experimental & educational use only*	Jetson Nano 2GB Developer Kit (P3541-0000, P3541-0001); includes P3448-0003 module



BSP (board support package)

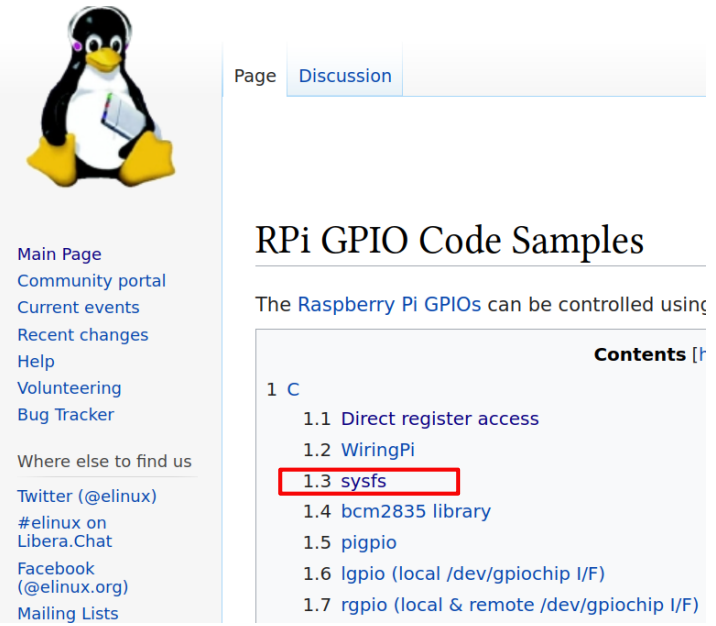
https://www.digikey.com/en/maker/projects/getting-started-with-the-nvidia-jetson-nano-part-1-setup/2f497bb88c6f4688b9774a81b80b8ec2

References on GPIO Interface

Pi and NANO are pin to pin compatible

Jetson Nano GPIO - JetsonHackshttps://www.jetsonhacks.com› ... › GPIO/I2C
Jun 7, 2019 — As you may have heard, the GPIO pin layout on the Jetson Nano is compatible with the 40 pin layout of a Raspberry Pi (RPi).

```
/* blink.c
 *
 * Raspberry Pi GPIO example using sysfs interface.
 * Guillermo A. Amaral B. <g@maral.me>
 *
 * This file blinks GPIO 4 (P1-07) while reading GPIO 24 (P1_18).
 */
//https://elinux.org/RPi_GPIO_Code_Samples#sysfs
```



NVIDIA Jetson Nano J41 Header Pinout

https://www.jetsonhacks.com/nvidia-jetson-nano-j41-header-pinout/

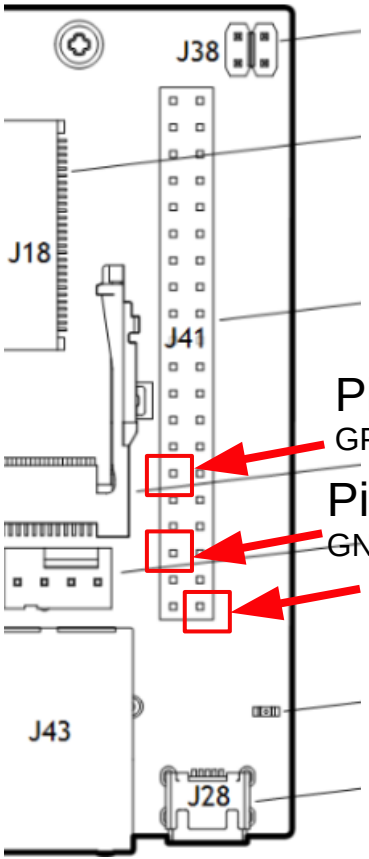
Note: I2C and UART pins are connected to hardware and should not be reassigned. By default, all other pins (except power) are assigned as GPIO. Pins labeled with other functions are recommended functions if using a different device tree.

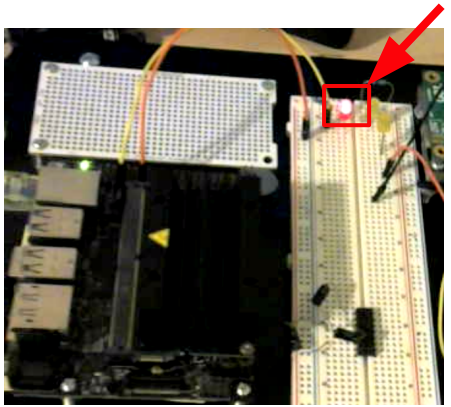
1. take Pin 1 Vcc (3.3V) and Pin 39 GND to test out LED, make sure you can light up a LED with 220 Ohm resistor in series.

Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	**5.0 VDC** *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	**5.0 VDC** *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	**GND**	
gpio216	**AUDIO_MCLK**	7	8	**UART_2_TX** */dev/ttyTHS1*	
	GND	9	10	**UART_2_RX** */dev/ttyTHS1*	
gpio50	**UART_2_RTS**	11	12	**I2S_4_SCLK**	gpio79
gpio14	**SPI_2_SCK**	13	14	**GND**	
gpio194	**LCD_TE**	15	16	**SPI_2_CS1**	gpio232
	3.3 VDC *Power*	17	18	**SPI_2_CS0**	gpio15
gpio16	**SPI_1_MOSI**	19	20	**GND**	
gpio17	**SPI_1_MISO**	21	22	**SPI_2_MISO**	gpio13
gpio18	**SPI_1_SCK**	23	24	**SPI_1_CS0**	gpio19
	GND	25	26	**SPI_1_CS1**	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	**I2C_1_SCL** *I2C Bus 0*	
gpio149	**CAM_AF_EN**	29	30	**GND**	
gpio200	**GPIO_PZ0**	31	32	**LCD_BL_PWM**	gpio168
gpio38	**GPIO_PE6**	33	34	**GND**	
gpio76	**I2S_4_LRCK**	35	36	**UART_2_CTS**	gpio51
gpio12	**SPI_2_MOSI**	37	38	**I2S_4_SDIN**	gpio77
	GND	39	40	**I2S_4_SDOUT**	gpio78

Harry Li, Ph.D.

Testing J41 40 Pin Connector with LED





We can control our LED from the command line. Here are some useful commands:

```
# Map GPIO Pin
# gpio79 is pin 12 on the Jetson Nano
$ echo 79 > /sys/class/gpio/export
# Set Direction
$ echo out > /sys/class/gpio/gpio79/direction
# Bit Bangin'!
$ echo 1 > /sys/class/gpio/gpio79/value
$ echo 0 > /sys/class/gpio/gpio79/value
# Unmap GPIO Pin
$ echo 79 > /sys/class/gpio/unexport
# Query Status
$ cat /sys/kernel/debug/gpio
```

In the above code, the 79 refers to a translation of the Linux sysfs GPIO named gpio79. If we look at the Jetson Nano J41 Header Pinout, we can see that gpio79 is physically pin 12 of the header.

$echo 79 > /sys/class/gpio/export

$ echo out > /sys/class/gpio/gpio79/direction

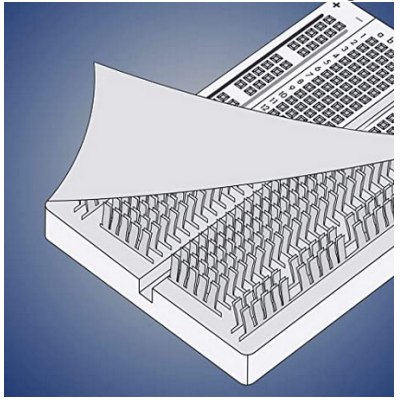
$echo 1 > /sys/class/gpio/gpio79/value

$echo 0 > /sys/class/gpio/gpio79/value

$echo 79 /sys/class/gpio/unexport

$cat /sys /kernel /debug/gpio

Use bread board for testing



Furthr Reference on GPIO

https://developer.nvidia.com/embedded/dlc/tegra-x1-technical-reference-manual

Pads marked "CZ" can be configured to be 3.3V tolerant and driving; and pads marked "DD" can be 3.3V tolerant when in open-drain mode (only.)

Table 22: MPIO Pad Types

Chapter 9, pp. 277

Pad Type	I/O Rail Voltage (V)	Input Buffer	Output Buffer	I/O Voltage Tolerance	Nominal Pull Strength	"Slew Rate" Control
ST	1.8	Schmitt & CMOS	push-pull	VDDIO	100 kΩ	No
CZ	1.8, 3.3	Schmitt & CMOS	push-pull	VDDIO	18 kΩ (pull-up and pull-down)	2 bits, up & d
DD	1.8	Schmitt & CMOS	push-pull & open-drain	3.3V for open-drain, VDDIO otherwise	100 kΩ	No
LV_CZ	1.2, 1.8	Schmitt & CMOS	push-pull	VDDIO	15 kΩ	2 bits, up & d
ST_EMMC	1.2, 1.8	Schmitt & CMOS	push-pull	VDDIO		3 bits, down c
DP_AUX	1.8	Diff/Open Drain	push-pull & open-drain	3.3V	NA	NA

9.13.1 GPIO_CNF_0

Configuration register
Chapter 9, pp. 250

Designates whether each pin operates as a GPIO or as an SFIO programmed to GPIO mode at any stage.

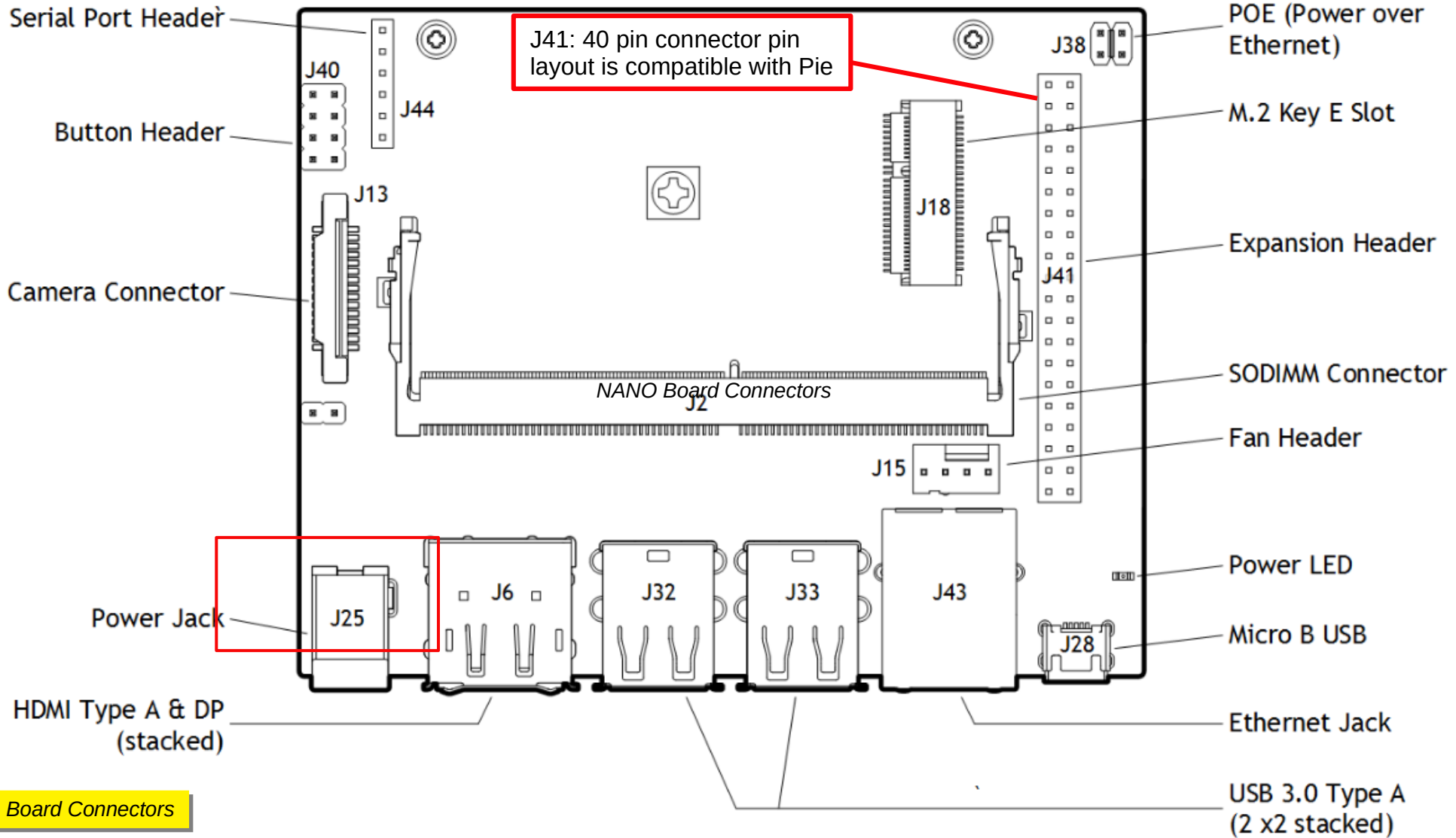
Lock bits are used to control the access to the CNF and OE regist can be programmed ONLY during Boot and get reset by chip res

This is an array of 4 identical register entries; the register fields b

GPIO Port A - D Configuration Registers

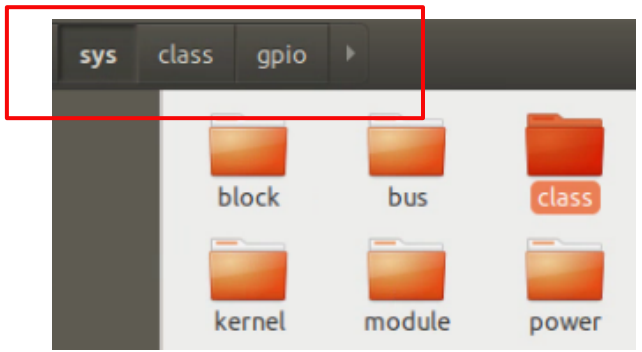
Offset: 0h..fh | Read/Write: R/W | Reset: 0b0000000000000000

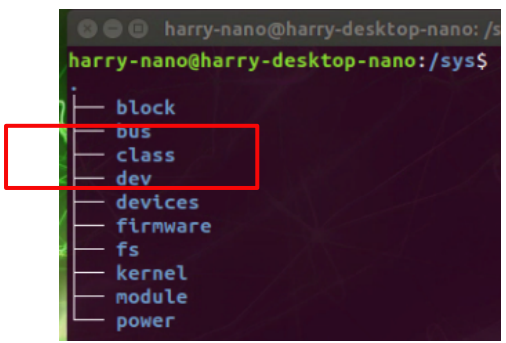
Top View


Serial Port Header
J40
J44
Button Header
J13
Camera Connector
J41: 40 pin connector pin layout is compatible with Pie
J38
POE (Power over Ethernet)
M.2 Key E Slot
J18
Expansion Header
J41
NANO Board Connectors
J2
SODIMM Connector
Fan Header
J15
Power LED
Power Jack
J25
J6
J32
J33
J43
J28
Micro B USB
HDMI Type A & DP (stacked)
Ethernet Jack
USB 3.0 Type A (2 x2 stacked)

GPIO sysfs Files for C/C++ and Python

Use the sysfs file nodes under /sys/class/gpio/ Reading and writing to these files is how languages like C/C++, Python, and other libraries implement device drivers.





Adding p2n 2222 transistor to drive GPIO load

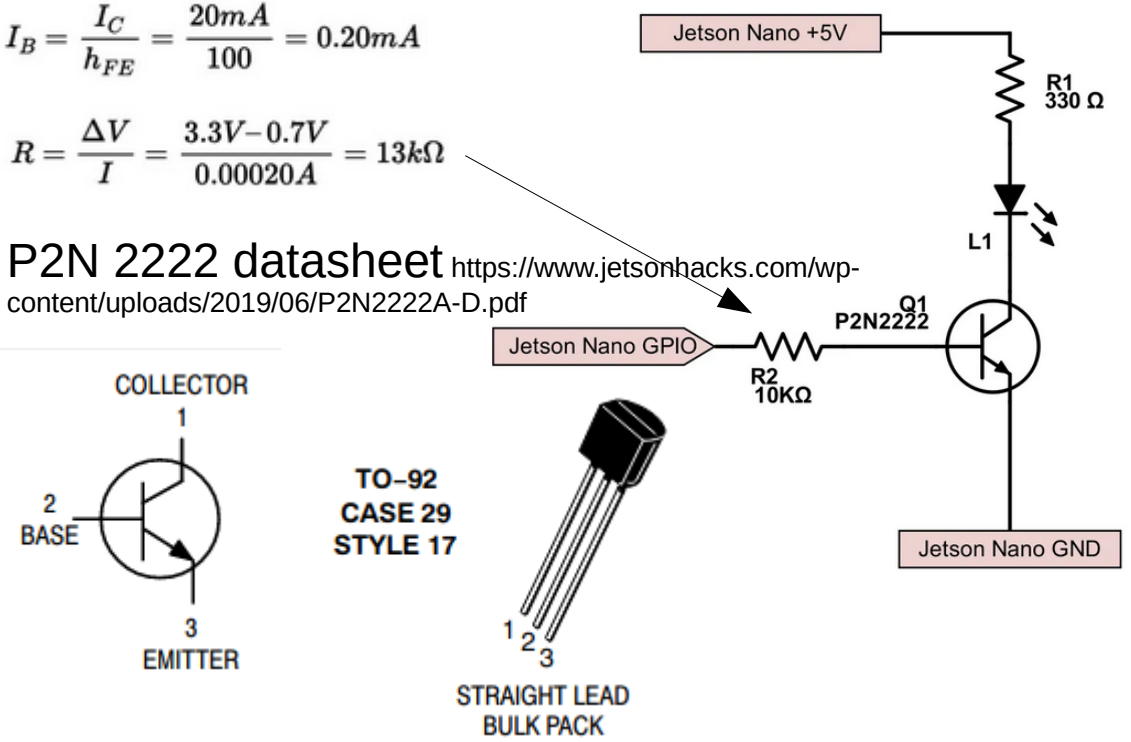
https://www.jetsonhacks.com/2019/06/07/jetson-nano-gpio/

$$I_B = \frac{I_C}{h_{FE}} = \frac{20mA}{100} = 0.20mA$$

$$R = \frac{\Delta V}{I} = \frac{3.3V - 0.7V}{0.00020A} = 13k\Omega$$

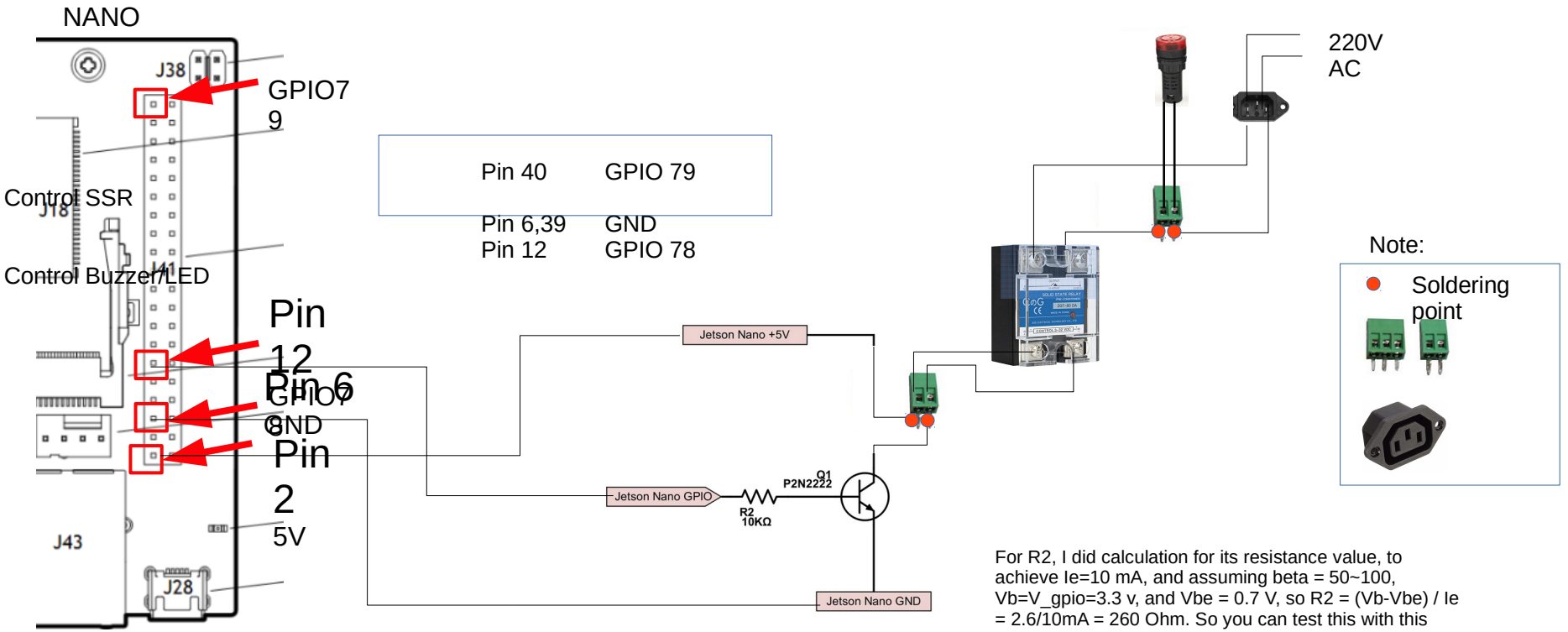
P2N 2222 datasheet

https://www.jetsonhacks.com/wp-content/uploads/2019/06/P2N2222A-D.pdf



AIV100-ACE LED Alarm and 110/220 VAC Control

https://www.jetsonhacks.com/nvidia-jetson-nano-j41-header-pinout/



For R2, I did calculation for its resistance value, to achieve Ie=10 mA, and assuming beta = 50~100, Vb=V_gpio=3.3 v, and Vbe = 0.7 V, so R2 = (Vb-Vbe) / Ie = 2.6/10mA = 260 Ohm. So you can test this with this value, if need adjustment, you can make some change to fine tune R2 value.

Harry Li, Ph.D.

C/C++ GPIO Code

Use the sysfs file nodes under /sys/class/gpio/ Reading and writing to these files is how languages like C/C++, Python, and other libraries implement device drivers.

Python GPIO Code

https://github.com/NVIDIA/jetson-gpio

This document walks through what is contained in The Jetson GPIO library package, how to configure the system and run the provided sample applications, and the library API.

the Jetson GPIO library package contains the following:
1. The lib/python/ subdirectory: The Python modules that implement all library functionality.
(1) The gpio.py module is the main component that will be imported into an application and provides the needed APIs.
(2) The gpio_event.py and gpio_pin_data.py modules are used by the gpio.py module and must not be imported directly in to an application.

2. The samples/ subdirectory contains sample applications to help in getting familiar with the library API and getting started on an application.
(1) The simple_input.py and
(2) simple_output.py applications show how to perform read and write to a GPIO pin respectively;
(3) button_led.py,
(4) button_event.py and
(5) button_interrupt.py show how a button press may be used to blink an LED using (a) busy-waiting, (b) blocking wait and (c) interrupt callbacks respectively.

(a) busy-waiting https://en.wikipedia.org/wiki/Busy_waiting

Busy-waiting, busy-looping or spinning is a technique in which a process repeatedly checks to see if a condition is true, such as whether keyboard input or a lock is available.

(b) blocking wait https://en.wikipedia.org/wiki/Blocking_(computing)

A process is an instance of a computer program that is being executed. A process always exists in exactly one process state. A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. Programming languages and scheduling algorithms are designed to minimize the over-all effect blocking. A process that blocks may prevent local work-tasks from progressing. In this case "blocking" often is seen as not wanted.

(c) interrupt callbacks. In computer system programming, an interrupt handler, also known as an interrupt service routine or ISR, is a callback function in microcontroller firmware, an operating system or a device driver, whose execution is triggered by the reception of an interrupt.

I2C Jetson Nano

Use the sysfs file nodes under /sys/class/gpio/ Reading and writing to these files is how languages like C/C++, Python, and other libraries implement device drivers.

Pan and tilt with servo

https://www.jetsonhacks.com/2019/06/07/jetson-nano-gpio/