Digital to Analog Converter

A way to explore Op-amps and GPIO

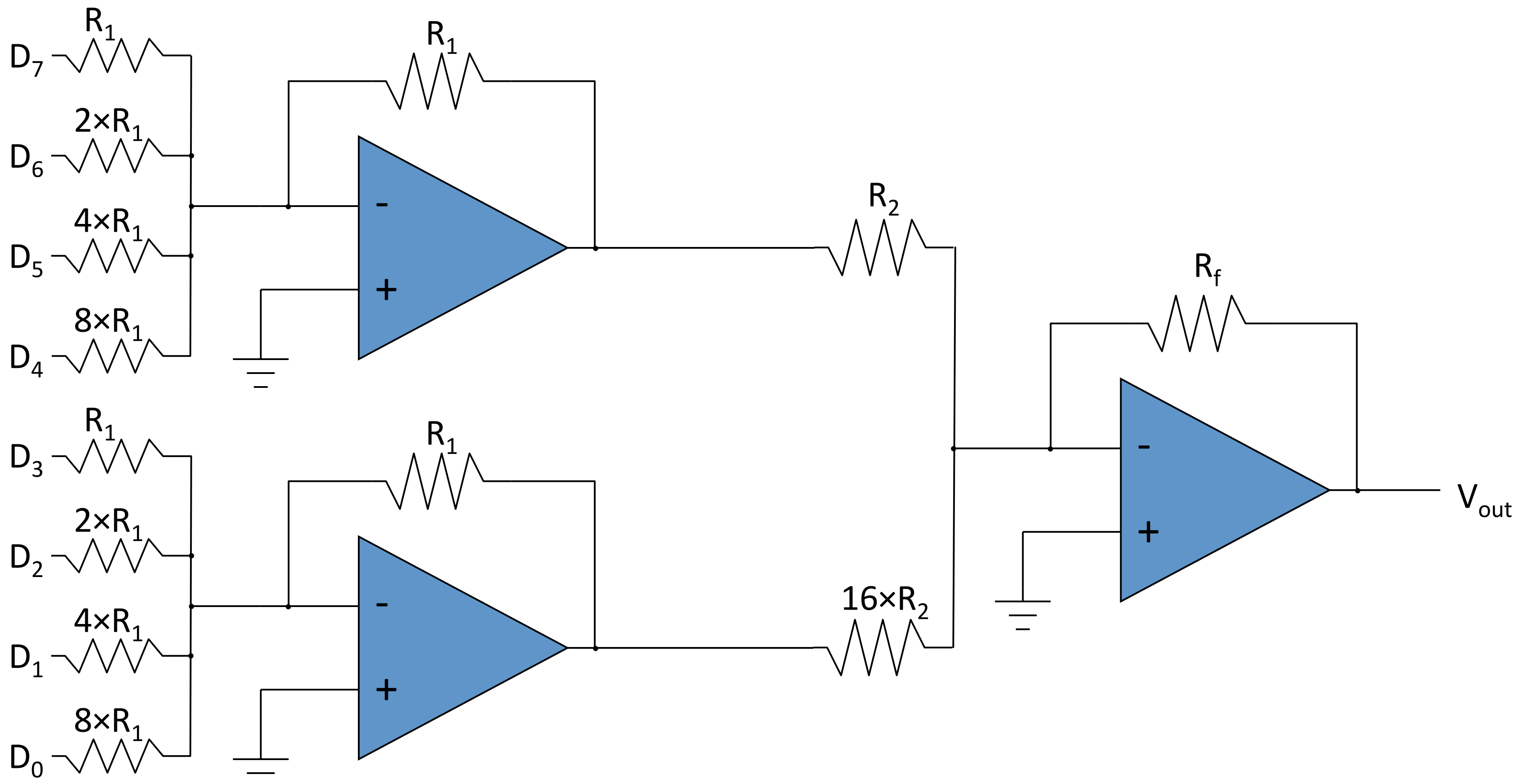
The Point of DAC

- Substantial and growing digital interface to analog world
- Two directions of conversion:
  - ADC: analog to digital converter
    - more common; analog input to computer
    - sensors of all types produce voltage proportional to quantity of interest
  - DAC: digital to analog converter
    - so computer can create analog output (voltage)
    - more fundamental; at core of ADC in guess-and-check scheme

Bit Level

- We'll do 8-bit DAC
  - 256 values; considered pretty crude
  - roughly 20 mV steps if 5-volt range
- 10 bit: 1,024 values; still at low end
- 12-bit: 4,096 values; often a reasonable choice
  - like the ADS1015 unit we used for RTD work
- 14-bit: 16,384 values; seldom need more
  - 0.3 mV resolution at 5 V starts to strain meaningfulness
- 16-bit: 65,536; high-end
  - 75 μV resolution at 5 V; fairly common as 16 bits convenient
  - but often least significant bits lack practical meaning

8-bit DAC Concept: Current Summing

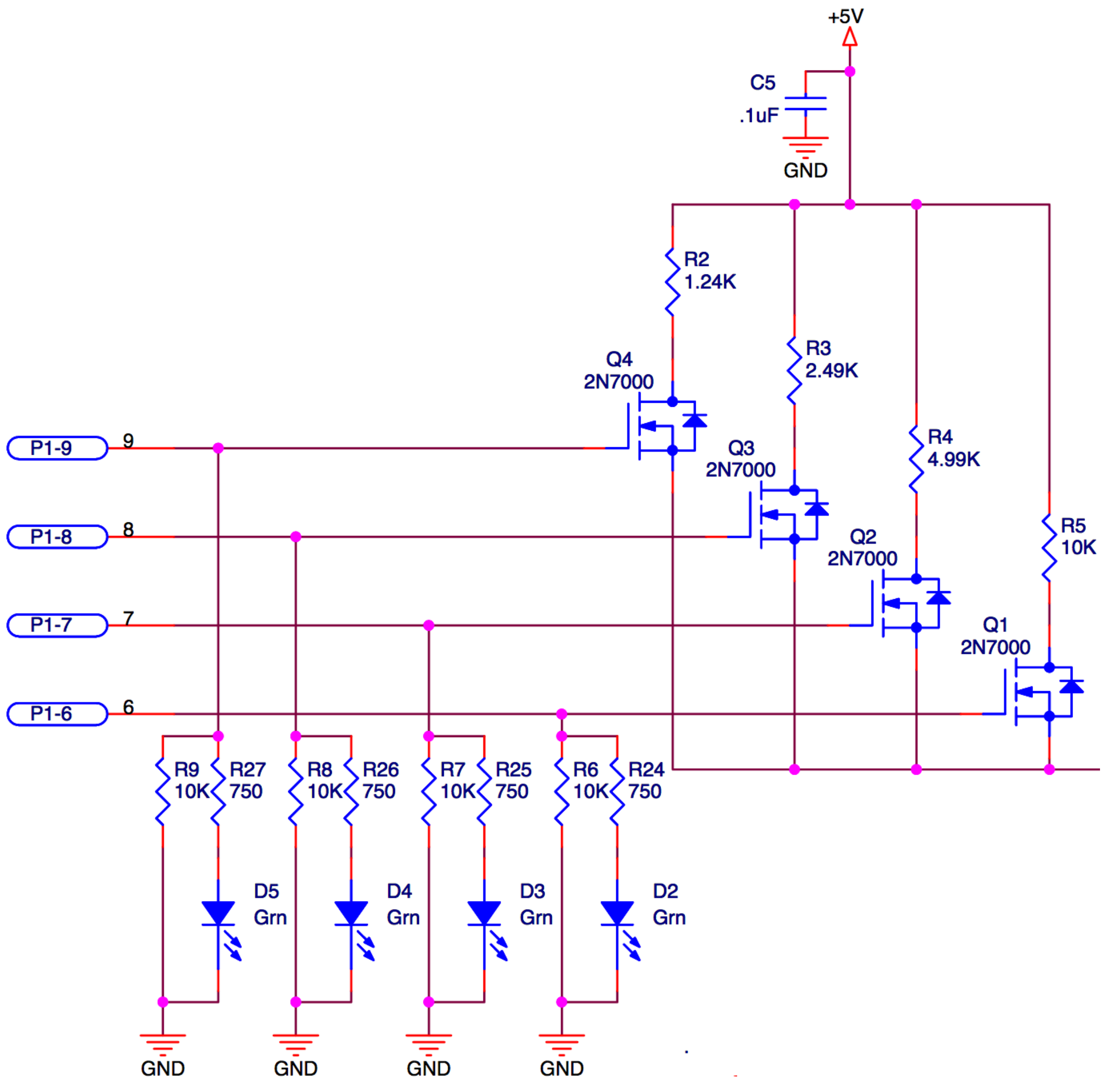


- Factors of 2 in identical 4-bit "nibble" stages
- Factor of 16 in second stage
- Same voltage (or zero) at each D_n input (digital)
  - voltage at intermediate nodes multiple of $-V_{digital}/8$ times integer 0−15
- Can tune final R_f to achieve desired scale

Cleaning Up Noisy Digital Input

- The $D_0 \ldots D_7$ inputs should all be the same voltage, and a known/reliable one
- But digital output from the Pi is not guaranteed to be steady or even the same from one pin to the next
- Want a way to use digital input to “switch-in” a clean reference
- Enter the MOSFET

Input Circuit Preview



- 5V reference at top
- 4-bits of input control MOSFETs
- Note 1:2:4:8 resistors
- 10k resistors pull down to ground when digital zero
- LED & limiter indicate ON

Field-Effect Transistors

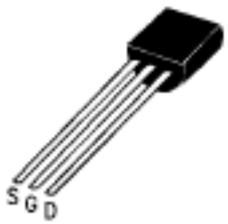
- The "standard" npn and pnp transistors use base-current to control the transistor current
- FETs use a field (voltage) to control current
- Result is no current flows into the control "gate"
- FETs are used almost exclusively as switches
  - pop a few volts on the control gate, and the effective resistance is nearly zero

2N7000 FET

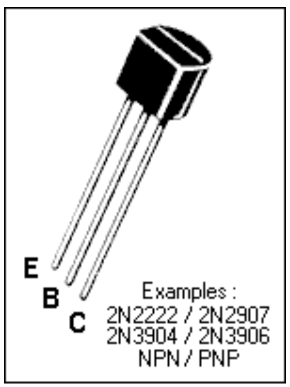
ON CHARACTERISTICS*							
$V_{GS(th)}$	Gate Threshold Voltage	$V_{DS} = V_{GS}, I_D = 1$ mA		0.8	2.1	3	V
$r_{DS(ON)}$	Static Drain-Source On-Resistance	$V_{GS} = 10V, I_D = 0.5A$			1.2	5	Ω
			$T_C = 125°C$		1.9	9	Ω
$V_{DS(ON)}$	Drain-Source On-Voltage	$V_{GS} = 10V, I_D = 0.5A$			0.6	2.5	V
		$V_{GS} = 4.5V, I_D = 75$ mA			0.14	0.4	V
$I_{D(ON)}$	On-State Drain Current	$V_{GS} = 4.5V, V_{DS} = 10V$		75	600		mA
g_{FS}	Forward Transconductance	$V_{DS} = 10V, I_D = 200$ mA		100	320		ms

FET Generalities

FET



BJT

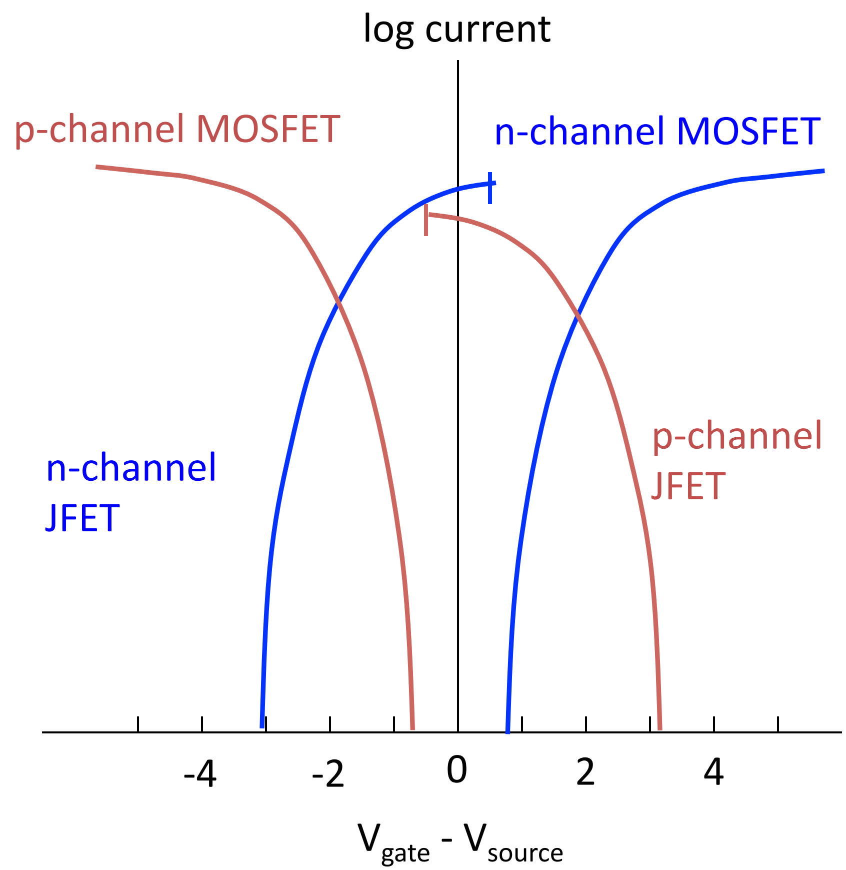


note pinout correspondence

- Every FET has at least three connections:
  - source (S)
    - akin to emitter (E) on BJT
  - drain (D)
    - akin to collector (C) on BJT
  - gate (G)
    - akin to base (B) on BJT
- Some have a body connection too
  - though often tied to source

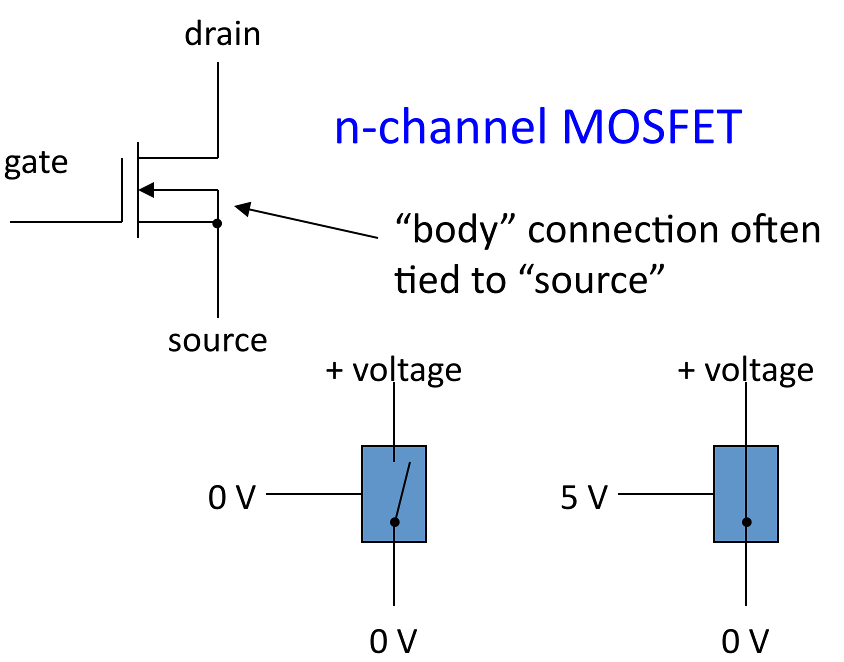
FET Types

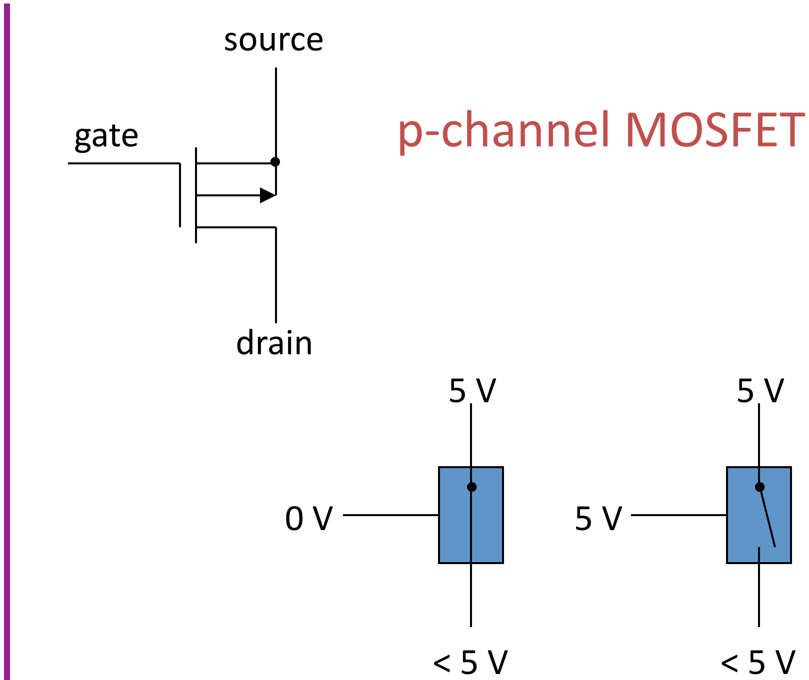
- Two flavors: n and p
- Two types: JFET, MOSFET
- MOSFETs more common
- JFETs conduct “by default”
  - when $V_{gate} = V_{source}$
- MOSFETs are “open” by default
  - must turn on deliberately
- JFETs have a p-n junction at the gate, so must not forward bias more than 0.6 V
- MOSFETs have total isolation: do what you want



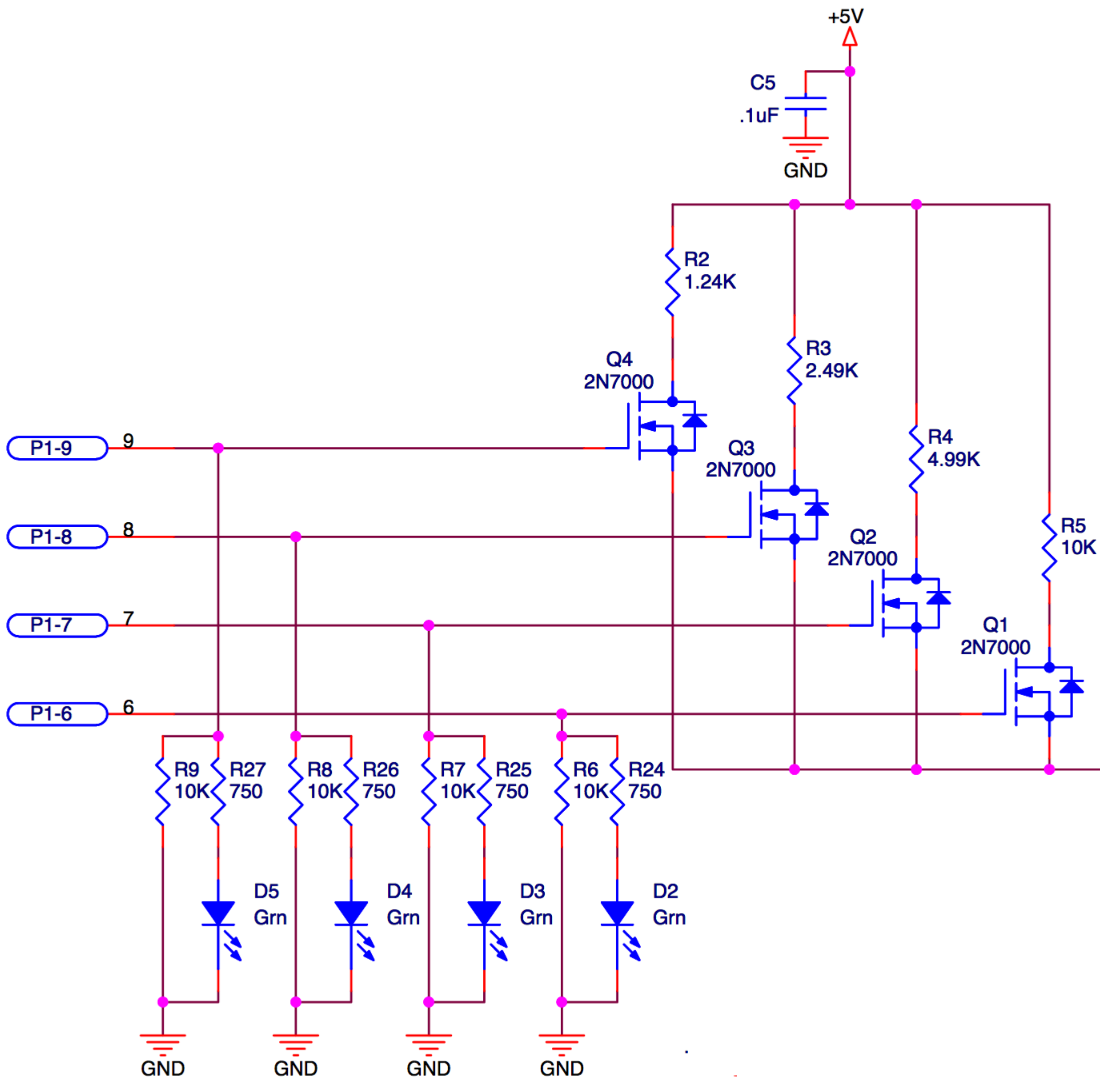
MOSFET Switches

- MOSFETs, as applied to logic designs, act as voltage-controlled switches
  - n-channel MOSFET is closed (conducts) when positive voltage (+5 V) is applied, open when zero voltage
  - p-channel MOSFET is open when positive voltage (+5 V) is applied, closed (conducts) when zero voltage
    - (MOSFET means metal-oxide semiconductor field effect transistor)





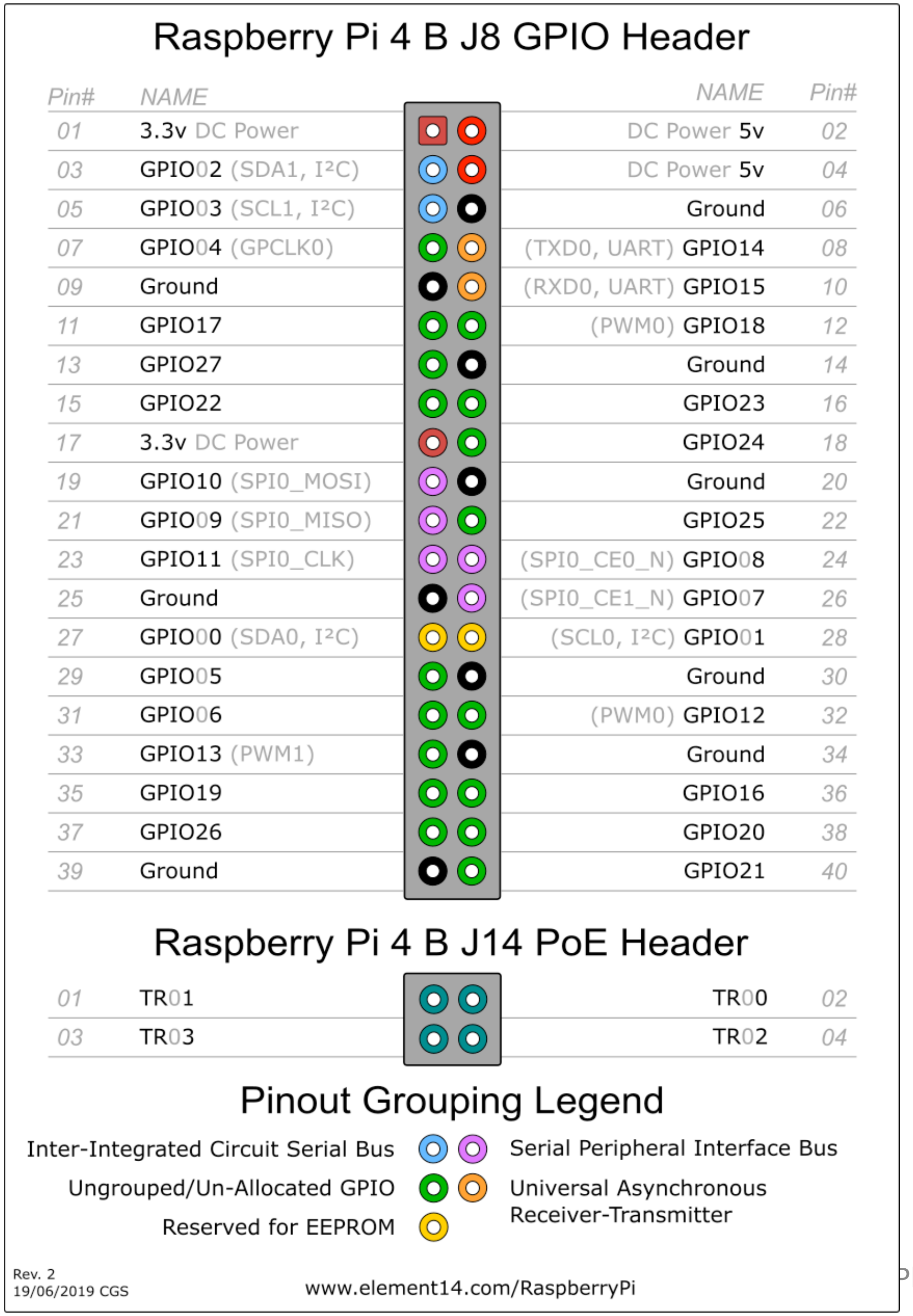
Input Circuit Revisited



- 5V reference at top
- Digital LOW on gate has MOSFET OFF
  - no current
  - drain at 5V
- Digital HIGH makes MOSFET like a short
  - current flows (to op-amp summing junction)
  - drain will be near ground
- 3.3 V from GPIO plenty to switch MOSFETs

Lab 7a

- Build 8-bit input stage and DAC on breadboard and verify operation
  - hits expected target values given input bit pattern
- Ready for Lab 7b; flinging digital data from the Raspberry Pi



RPi Interface

- 40-pin header on side of RPi
- serial is orange (UART)
- I^2C is light blue
- SPI is purple
- GPIO is green
  - and can also use any pin labeled GPIOxx

GPIO on the Raspberry Pi

- 28 pins labeled GPIO00 to GPIO27
  - even those dseignated for SPI, I^2C, UART fair game
  - just not the 12 power and ground pins!
- Numbering scheme is called BCM
  - Broadcom SOC (system on chip)
  - adheres to native numbering of Broadcom CPU used in Pi
  - does not follow 40-pin header numbering; skips around
- Digital values on 3.3 V standard
- Low-level C programs can switch at > 20 MHz
- Python native library switches around 70 kHz
  - see https://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/

Python Interface

- Library called RPi.GPIO installed (by default) on Pi
- Example; toggle pin 40 (GPIO21)

```
import RPi.GPIO as GPIO    # rename for convenience

GPIO.setwarnings(False)    # suppress warning message
GPIO.setmode(GPIO.BCM)     # use BCM labeling
                           # alternative is BOARD, by pin #
GPIO.setup(21,GPIO.OUT)    # declare pin as output

GPIO.output(21,GPIO.HIGH)  # set pin high
GPIO.output(21,GPIO.LOW)   # set pin low
```

`GPIO.LOW` and `GPIO.HIGH` just map to integers 0 and 1, actually

See https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/ for more

Lab 7b

- Purpose: send data to DAC from Pi
  - ultimately, generate creative/custom waveform
    - i.e., something a standard function generator can't do
- Lab 7b nominally Thanksgiving week
  - but may wish to do early; just keep going after 7a
  - single write-up due Dec. 4

Tips for Lab 7b

- Helps a lot to use sequential BCM numbers
  - allows simple `range()` loop to increment
  - makes coding compact, efficient, easy to modify
  - can also send list values to write all at once (later step)
- Determining bit values in integer
  - assume LSB (least significant bit) at bcm_low
  - bitval = (intval >> (bcm_num – bcm_low)) & 0x01
    - bcm_num loops through BCM GPIO numbers
    - right shift appropriate number of bits then mask LSB

Lab 7b, continued

- After initial static outputs (to check behavior), go dynamic
- Initially, just a ramp
  - integer increments 0 to 255; back to 0 and on and on
    - shortcut: `intval += 1` (same as `intval = intval + 1`)
    - then check if > 255 and reset to 0 if so
  - can nest inside: `while True:` for indefinite repeat
    - ctrl-C to terminate
  - at first, will be ratty and spiky due to non-simultaneous bit changes
    - will explore bit order and also list-write
    - then will filter out spikes