# EECS192 Lecture 8
# Mar. 10, 2020

**Notes**:
1. Contingency options- Instructional resilience
2. Quiz 4 steering 3/17
3. Community Spirit: PCB peer review, Piazza, helping fellow students
4. Lab safety/hygiene

Topics

- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  – Linear regulator
  – Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

# Checkpoint 7

C7.1 - emergency stop triggered while the car is in motion. (flag+switch)

C7.1.2 - The vehicle must also be able to be stopped via a remote command

C7.2 - Drive the practice track and collect some data.

C7.2.1 - Live telemetry functionality with logging capability.

C7.2.2 - Both live and logged telemetry data must be able to be viewed, showing at least the detected line position and velocity as line plots.

printf dumps are **not** acceptable

C7.2.3 - Collect telemetry and show us logs of least two runs of the course track with different speed targets. (ie. 0.5m/s and 1m/s).

**You must show us these data plots prior to driving the track.**

C7.2.4 - The vehicle must complete the practice track within 3 minutes' time.

C7.2.5 - As with C6, your vehicle must remain whole during the entirety of the run

C7.3 - Using a PCB editor, motor driver / power supply boards

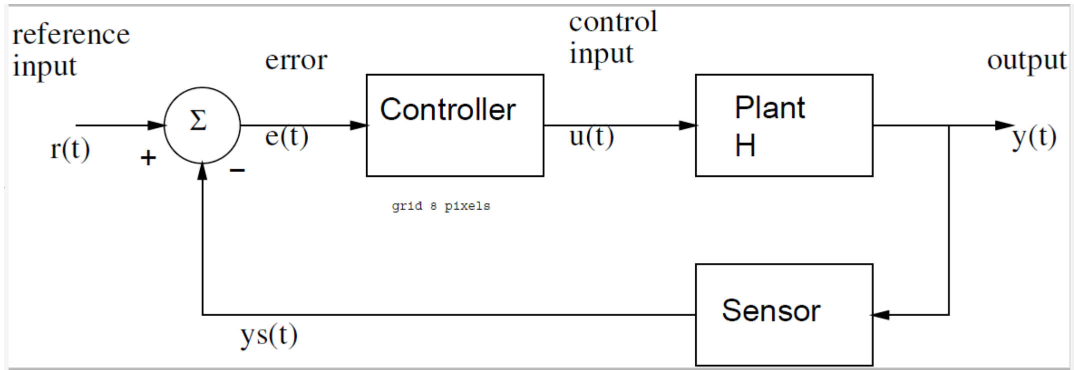
(see project proposal comments)

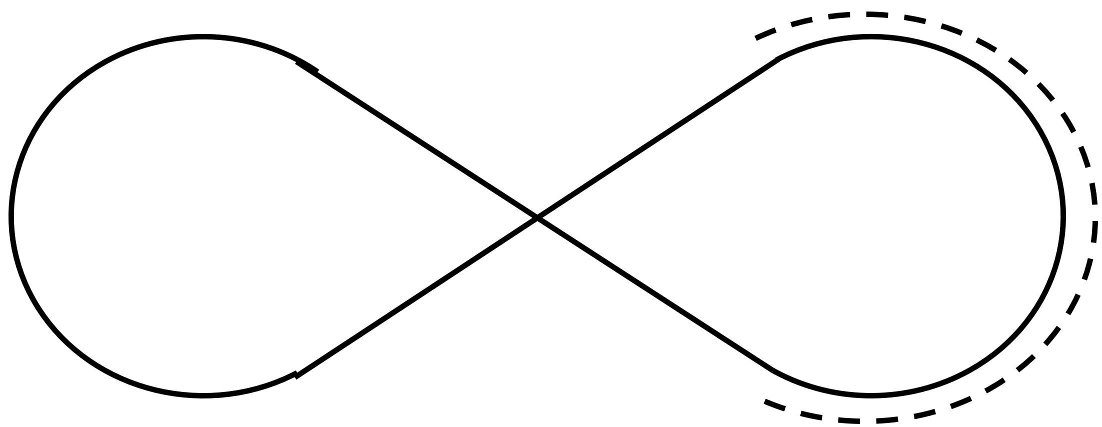
C7.3.1 - We will do a PCB peer review during the March 18 discussion section.

## Topics

- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  - Linear regulator
  - Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

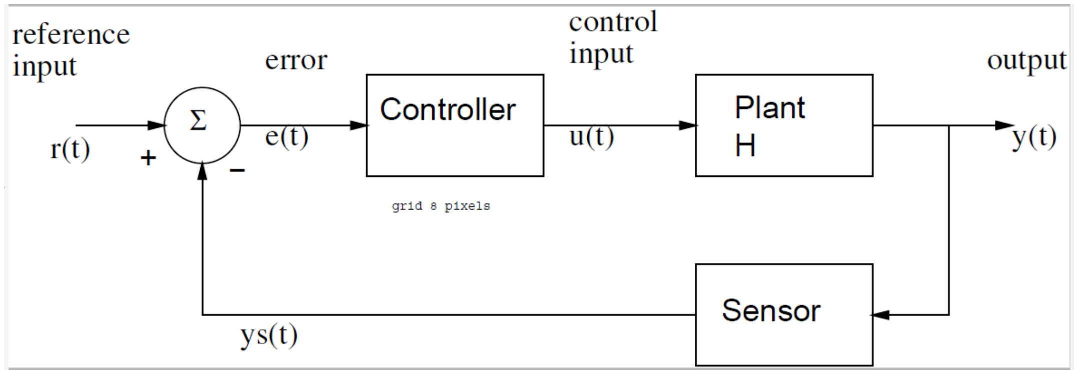
# Proportional + Integral

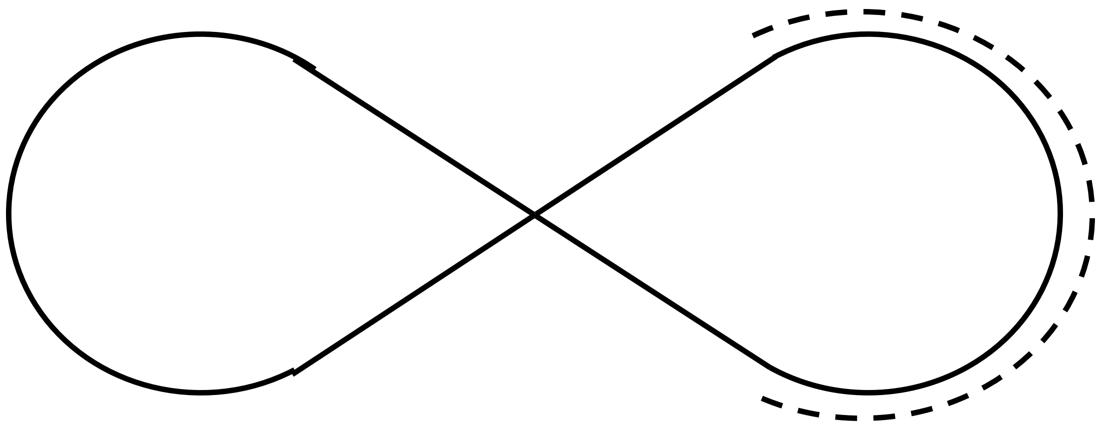


On board Anti-windup

# Feedforward



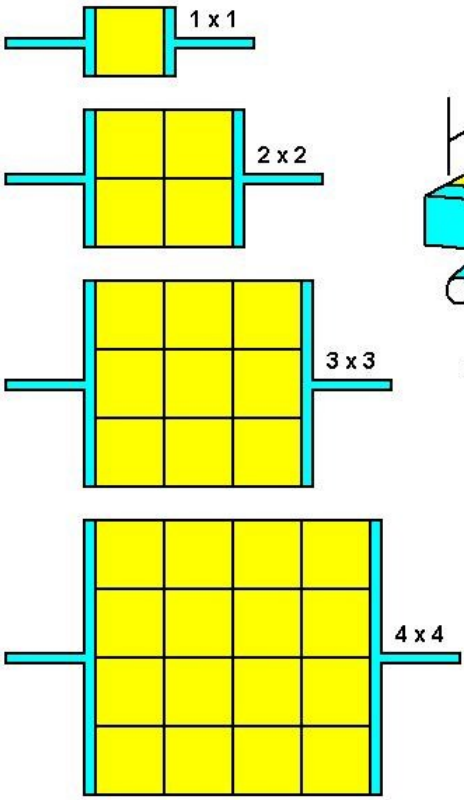



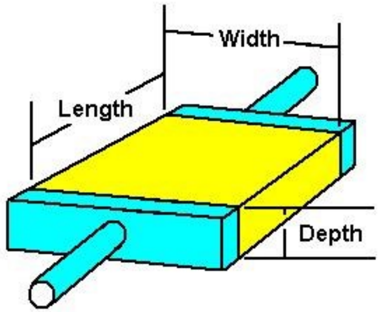
On board

# Topics

- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  - Linear regulator
  - Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

# Ohms/square




$$\text{Ohms} = \frac{\text{Resistivity} \times \text{Length}}{\text{Width} \times \text{Depth}}$$

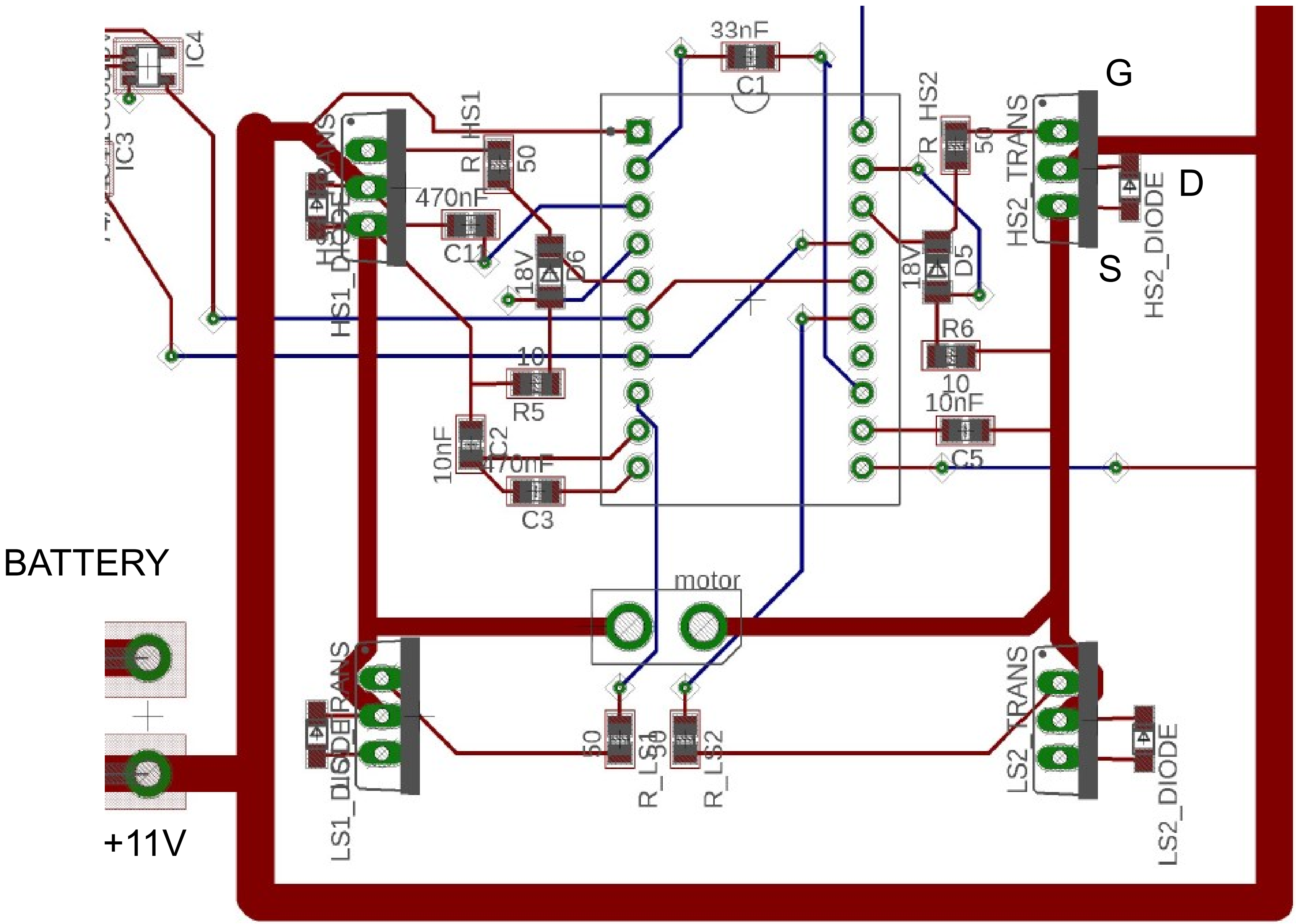
For some given depth, resistance is directly in proportion to length and inversely proportional to width.

Therefore, we can rate the resistive material of constant depth in terms of ohms per square.

| Cu Weight oz. | Thickness mm(mils) | mΩ/Square 25°C | mΩ/Square 100°C |
|---|---|---|---|
| 1/2 | .02 (0.7) | 1.0 | 1.3 |
| 1 | .04 (1.4) | 0.5 | 0.65 |
| 2 | .07 (2.8) | 0.25 | 0.36 |
| 4 | .13 (5.3) | 0.13 | 0.18 |

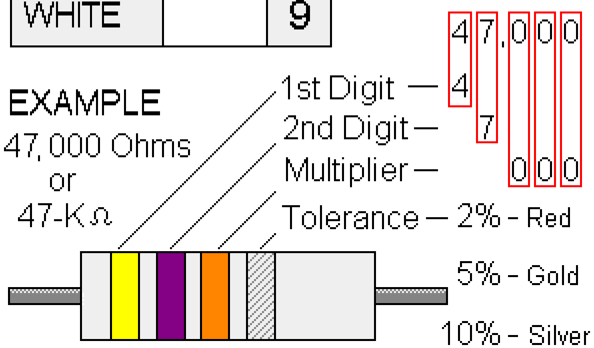
http://www.edn.com/design/components-and-packaging/4411971/Counting-squares--A-method-to-quickly-estimate-PWB-trace-resistance

# PCB Notes- neat but problematic

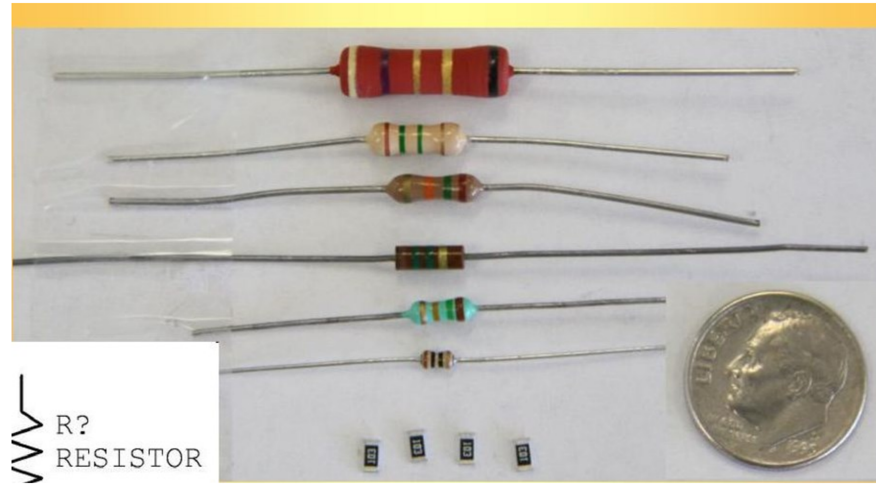


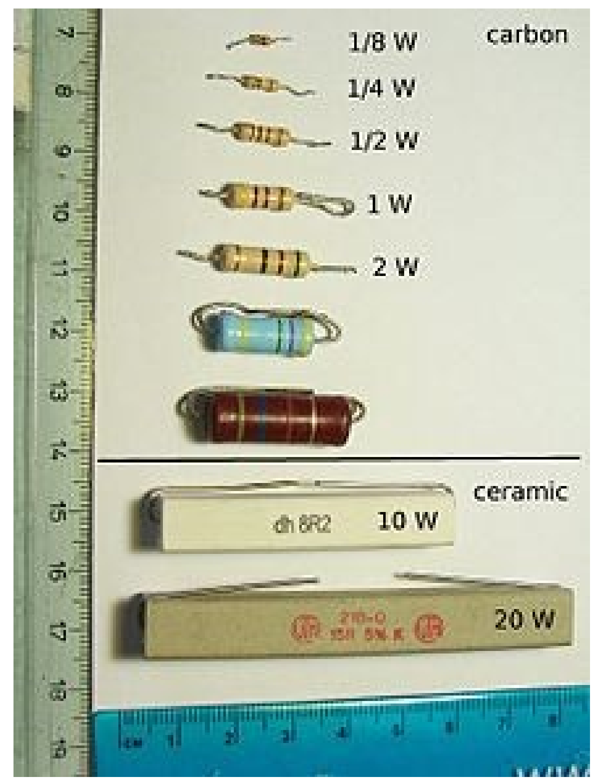
# Electronic Components- Resistors

| Color | Digit | Multiplier |
|---|---|---|
| BLACK | 0 | ______ |
| BROWN | 1 | _____0 |
| RED | 2 | ____00 |
| ORANGE | 3 | ___000 |
| YELLOW | 4 | __0, 000 |
| GREEN | 5 | _00, 000 |
| BLUE | 6 | 000, 000 |
| VIOLET | 7 | |
| GRAY | 8 | |
| WHITE | 9 | |




Yellow | violet | orange| gold



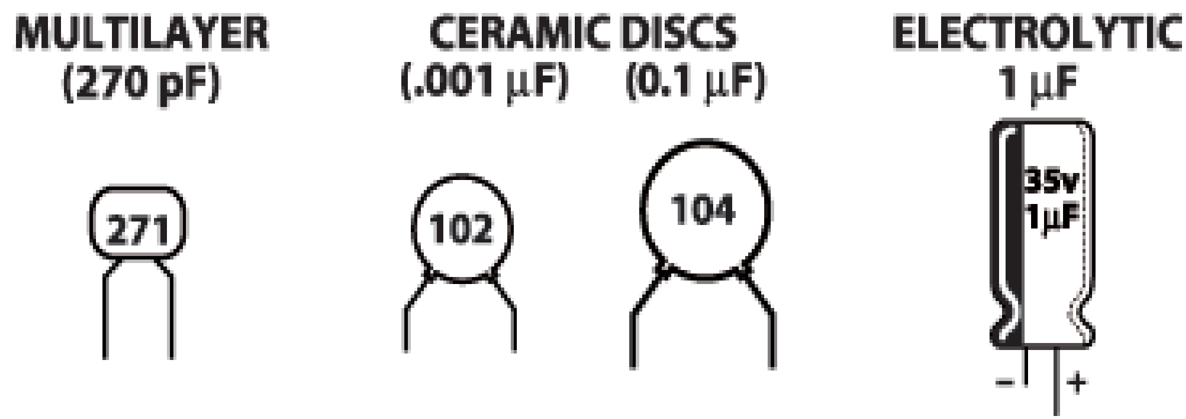




Better be right or your great big venture goes west…

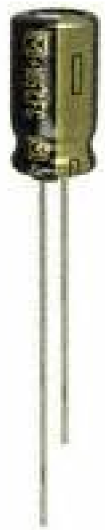
# Capacitor Codes

From:http://www.applefritter.com/sites/default/meta/replicacreation/images/fige-10.png

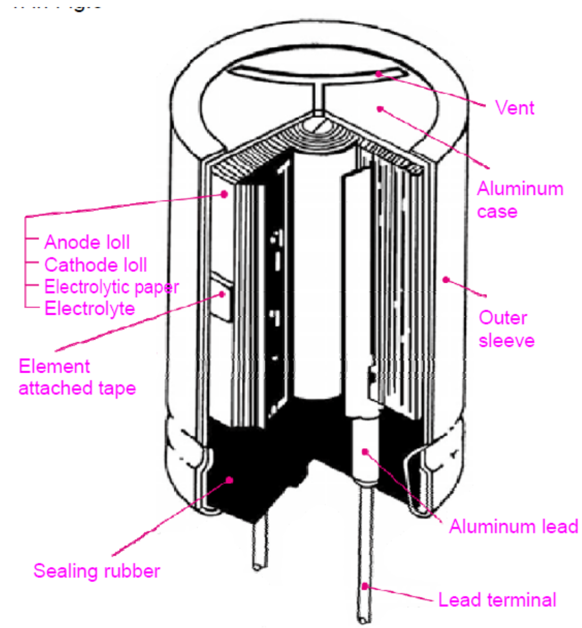
| VALUE | | CODE |
|---|---|---|
| 10 pF | = | 100 |
| 100 pF | = | 101 |
| 1000 pF | = | 102 |
| .001 µF | = | 102 |
| .01 µF | = | 103 |
| .1 µF | = | 104 |



# Capacitor Types- 47 uF 50V

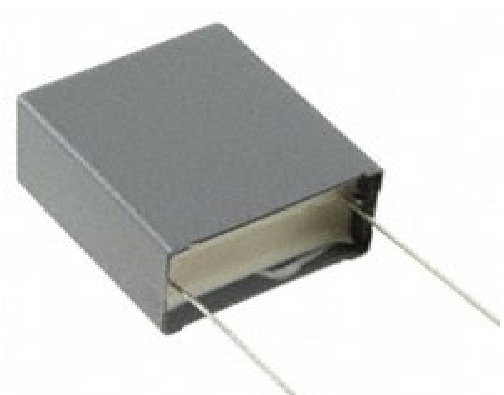


Ripple Current
600mA




Eletrolytic Ripple Current
169mA @ 120Hz

https://industrial.panasonic.com



Metalized film

CAP TANT 22UF 50V
20% 2917

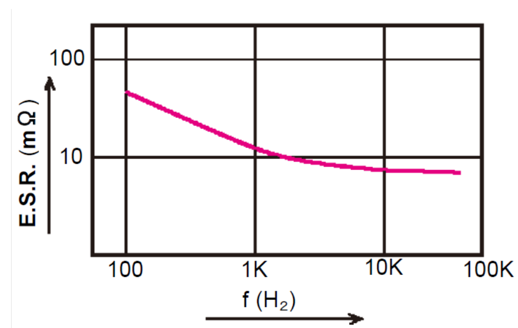


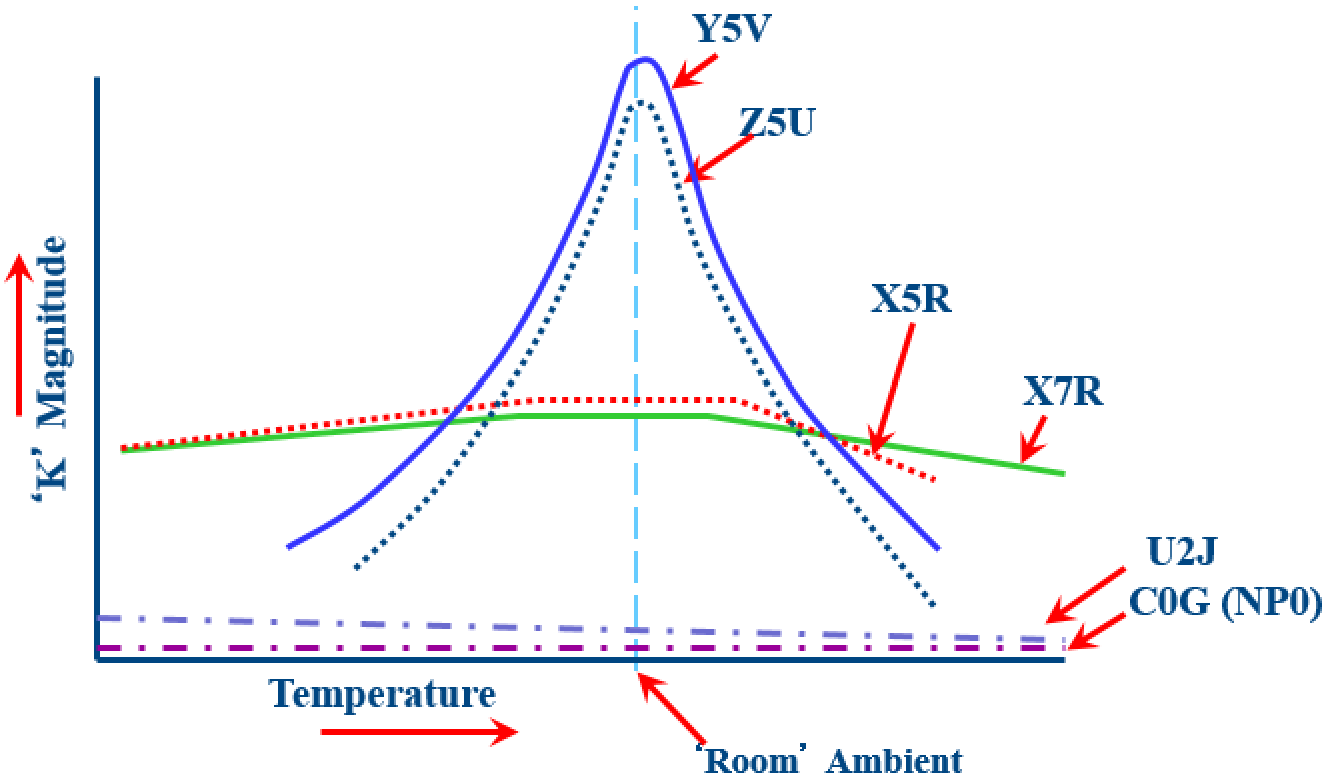

Fig.13 ESR vs frequency

# Capacitor Types-ceramic

CAP CER 0.1UF 50V X7R RADIAL

0.1µF ±20% 50V Ceramic Capacitor Z5U Radial

CAP CER 0.1UF 630V X7R RADIAL



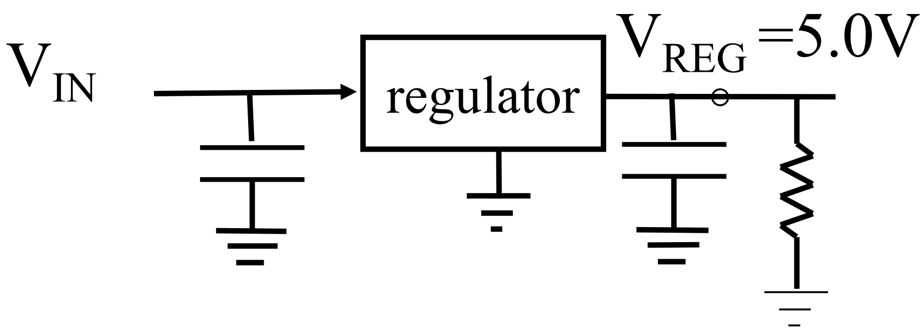

https://ec.kemet.com/wp-content/uploads/2015/12/ceramic-dielectric-comparison-chart.png

CAP CER 0.1UF 50V X7R 0805

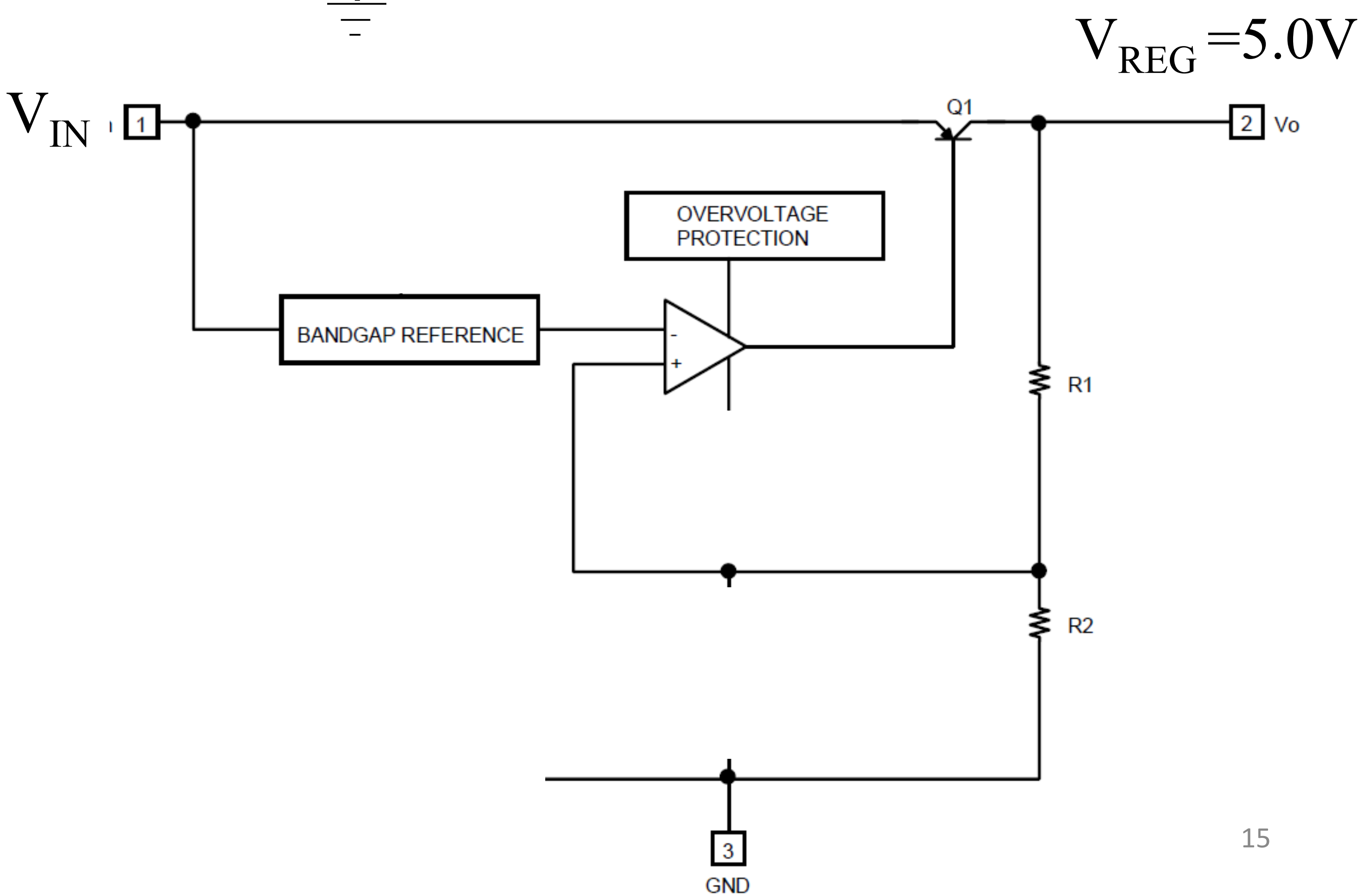
Topics

- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  - Linear regulator
  - Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

# Linear Voltage Regulator



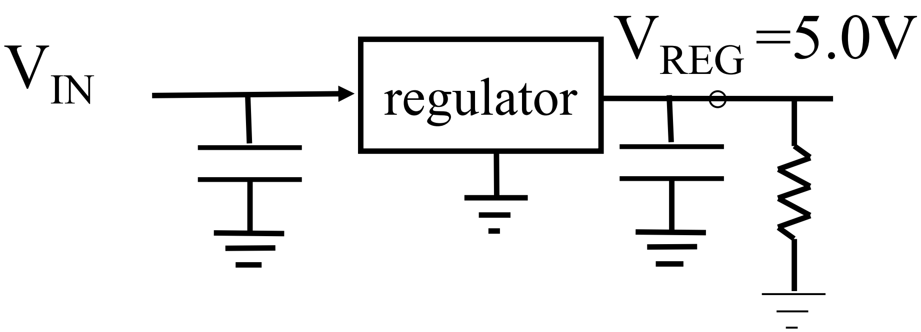

0.5 ohm
(equiv. for 1 amp load)



# Linear Regulator for RC servo power

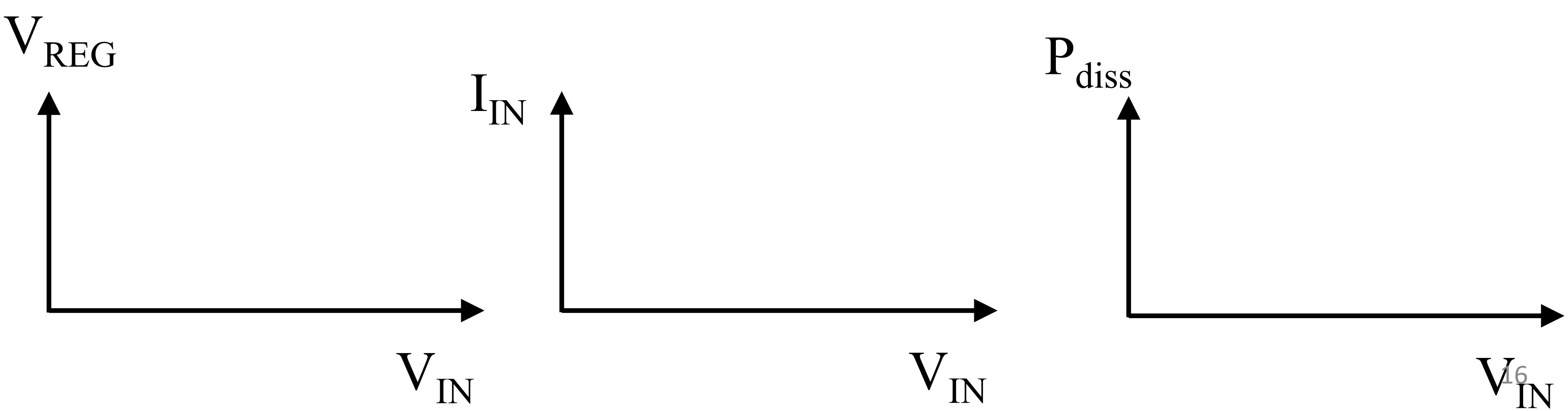
- Power limit? Heat….

Caution: caps required for stability for some voltage regulators

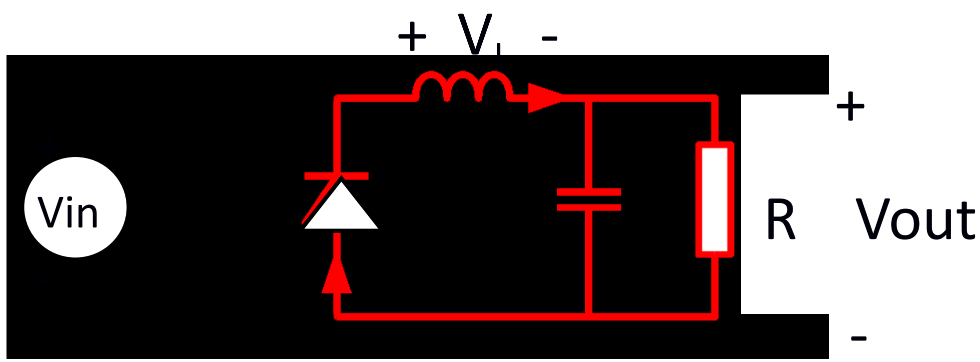



5 ohm
(equiv. for 1 amp load)

$P_{diss}$ =?



# Buck Converter- DC-DC

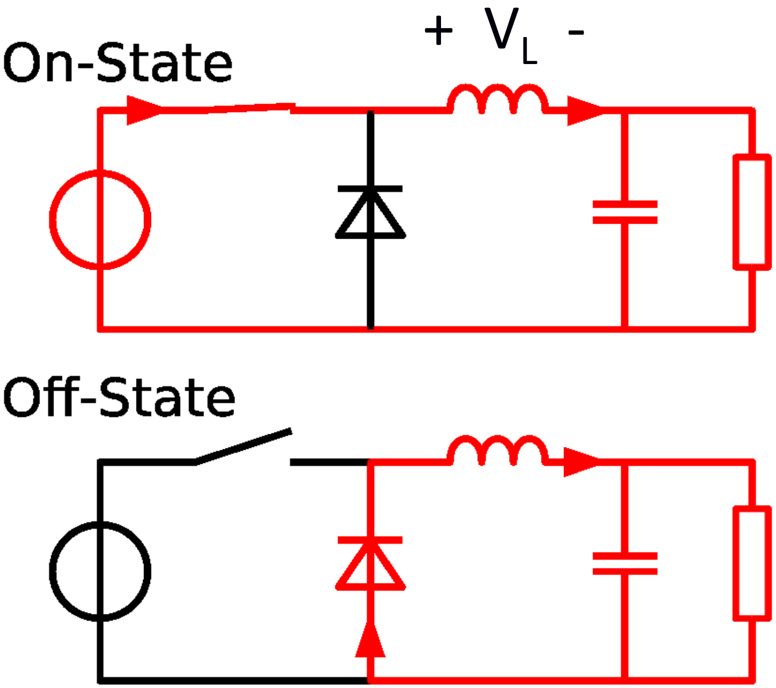


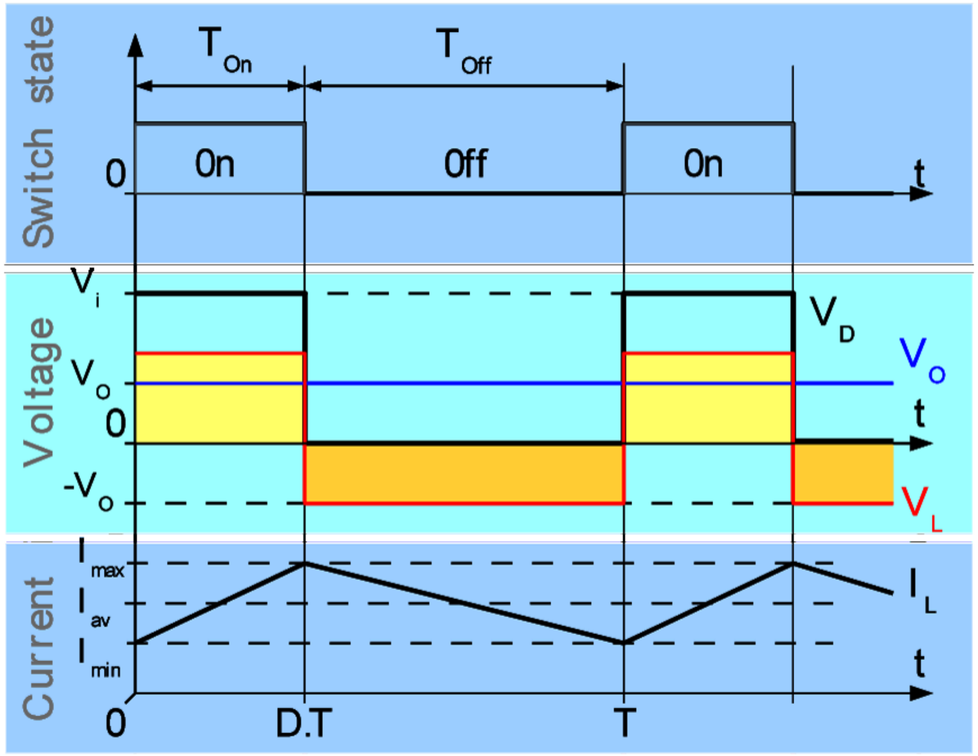

Why? Efficiency ~90%

Waveforms on board (also see buck converter notes.)

Buck: high to low. Boost: low-to-high)

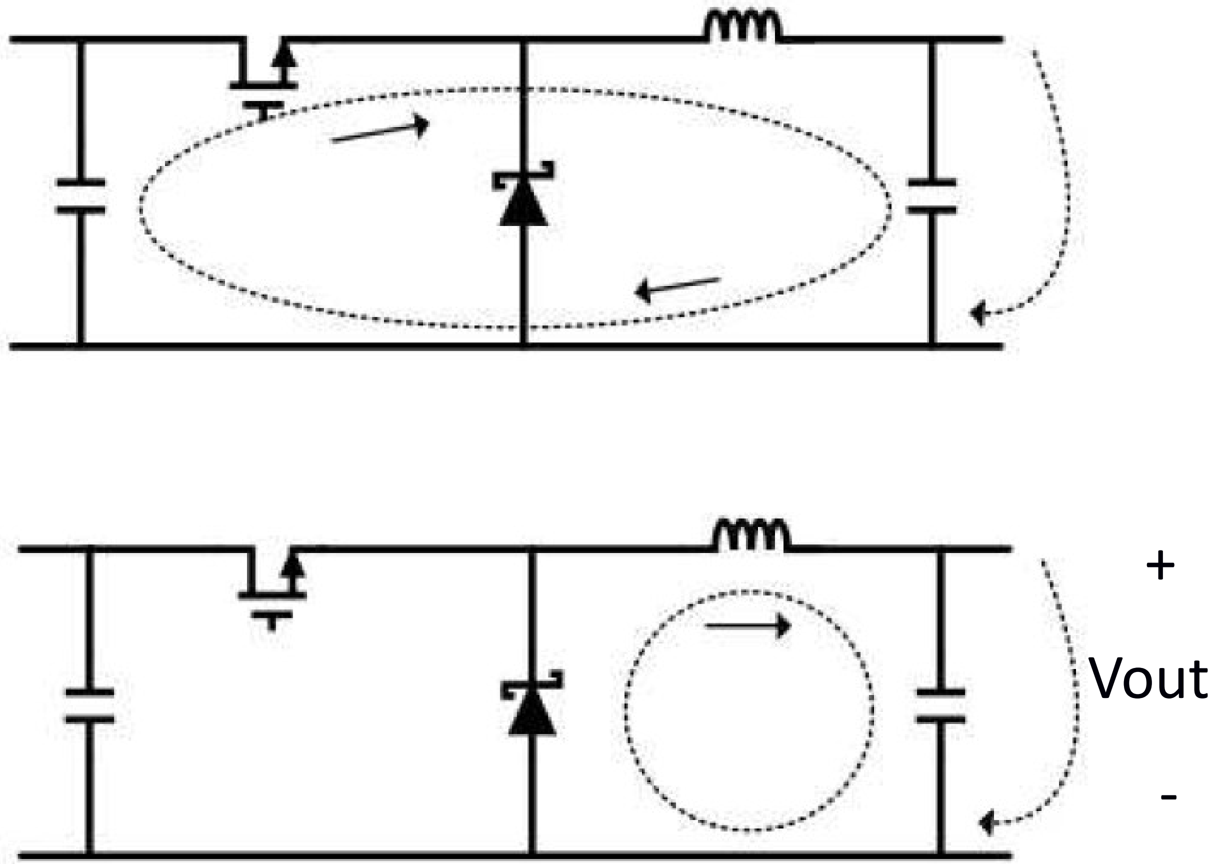
# Buck Converter



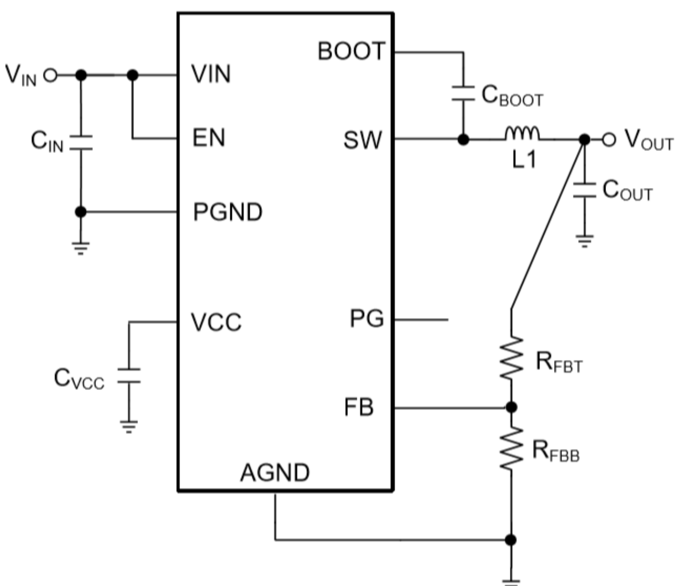



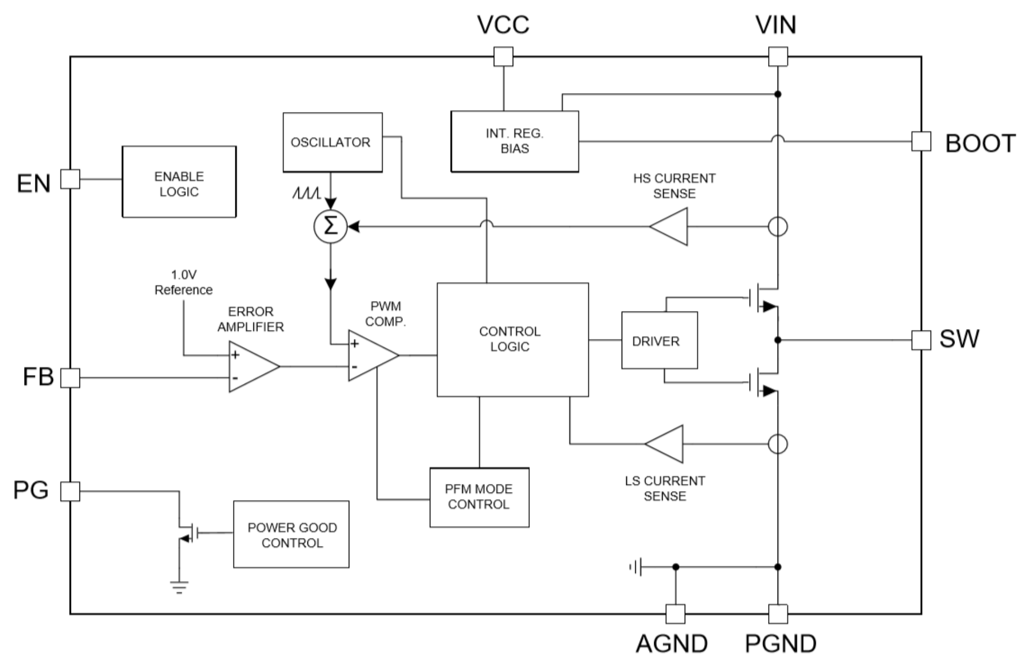

https://en.wikipedia.org/wiki/Buck_converter

# Buck Converter LM2678

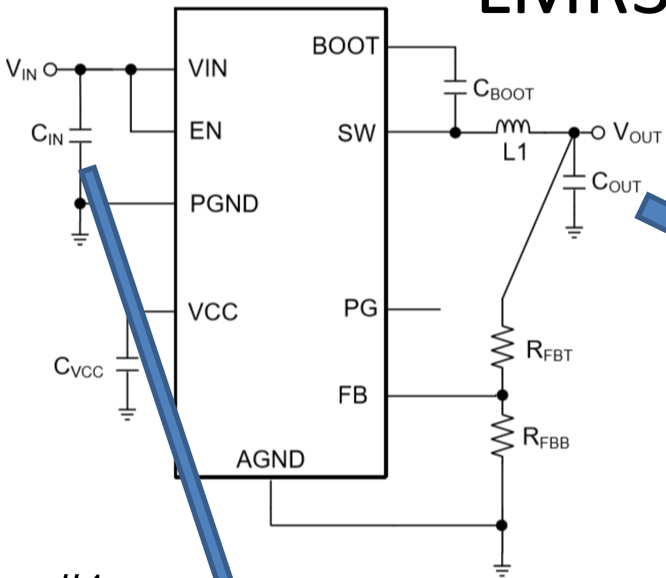


# LMR33630 Buck Converter

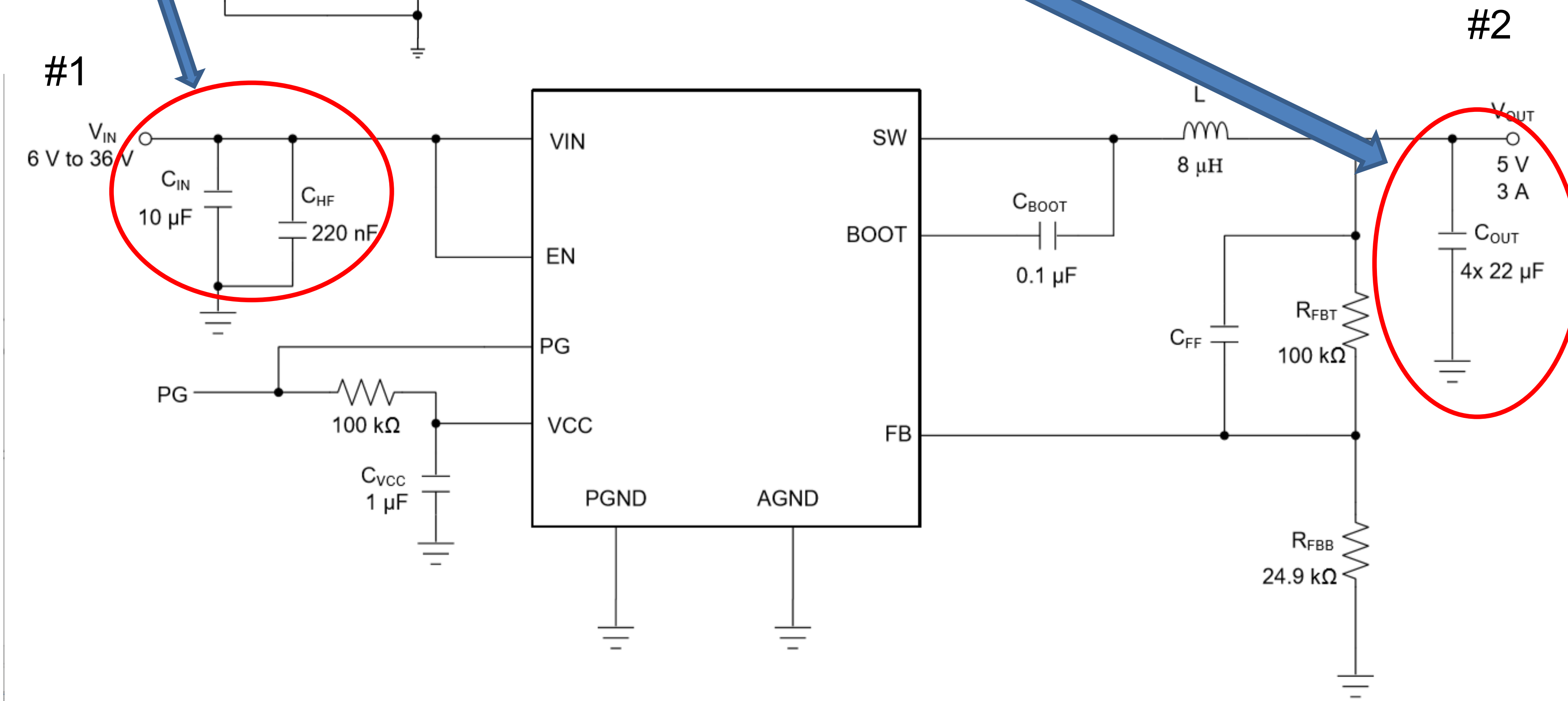


# LMR33630 Buck Converter




… multiple capacitors can be used in parallel to bring the minimum effective capacitance up to the required value. This can also ease the RMS current requirements on a single capacitor.



# Buck Converter Waveforms

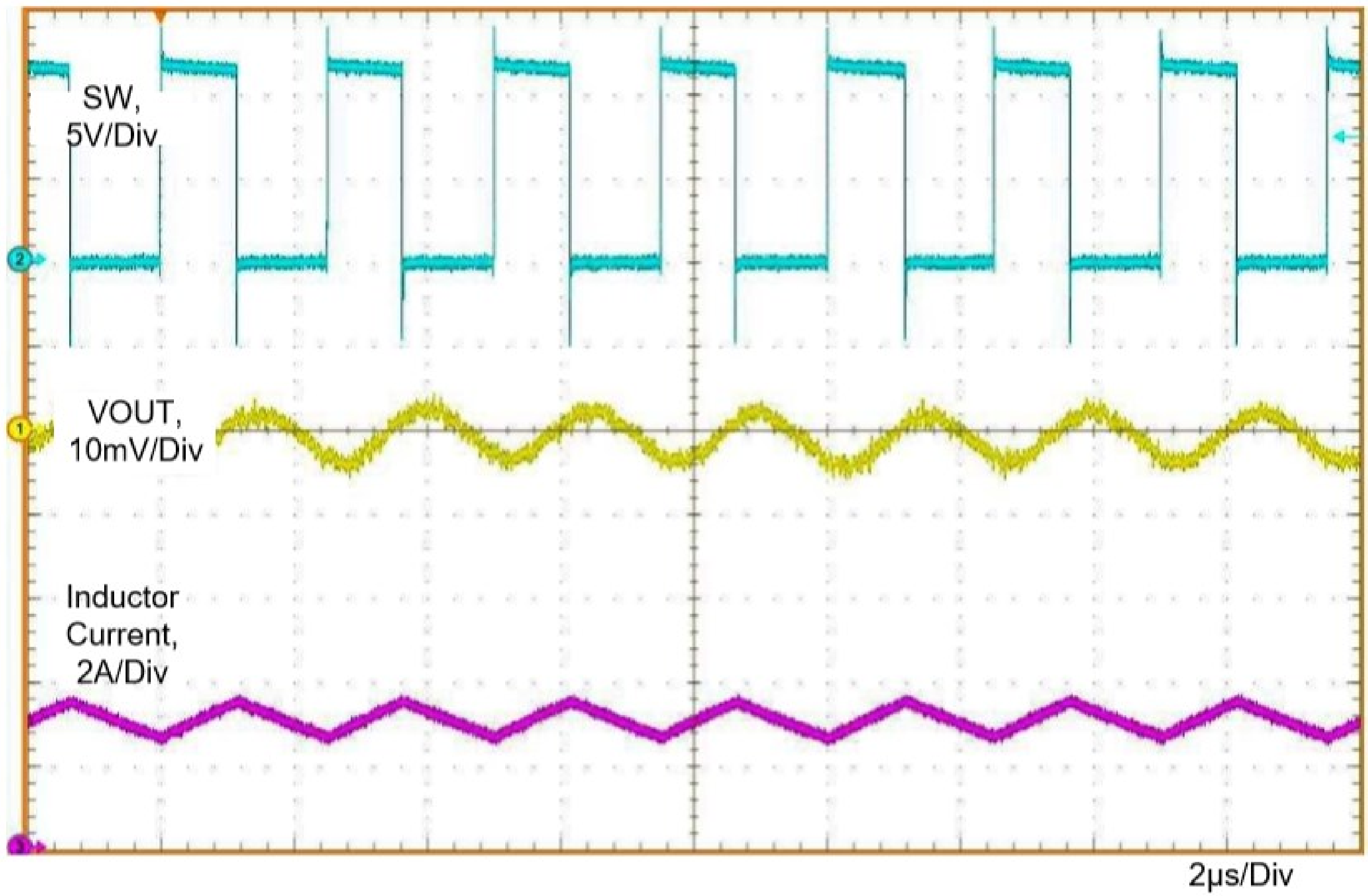



**Figure 14. Typical PWM Switching Waveforms**
**$V_{IN}$ = 12 V, $V_{OUT}$ = 5 V, $I_{OUT}$ = 3 A, $f_S$ = 400 kHz**

Topics

- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  - Linear regulator
  - Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

# Challenge: Embedded real-time programming:

POSIX threads, or Pthreads. Need pthread_mutex_.
(Not provided in librobotcontrol/BeagleBone ☹)
Also see FreeRTOS (used in EE192 2018)

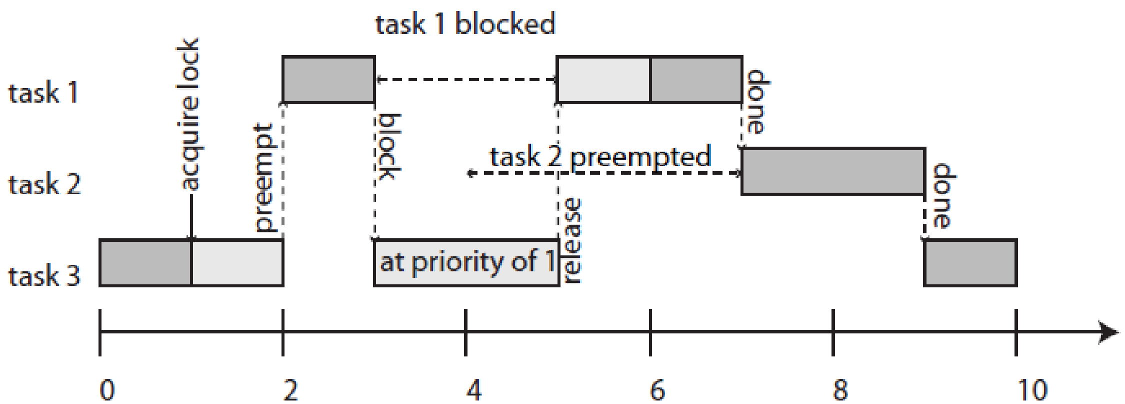



Figure 12.10: Illustration of the priority inheritance protocol. Task 1 has highest priority, task 3 lowest. Task 3 acquires a lock on a shared object, entering a critical section. It gets preempted by task 1, which then tries to acquire the lock and blocks. Task 3 inherits the priority of task 1, preventing preemption by task 2.

See chapter 12 on scheduling.

# Timing range

```
time_min = rc_nanos_since_boot();
angle1 = rc_encoder_read ( int  ch);
time_max = rc_nanos_since_boot();
```

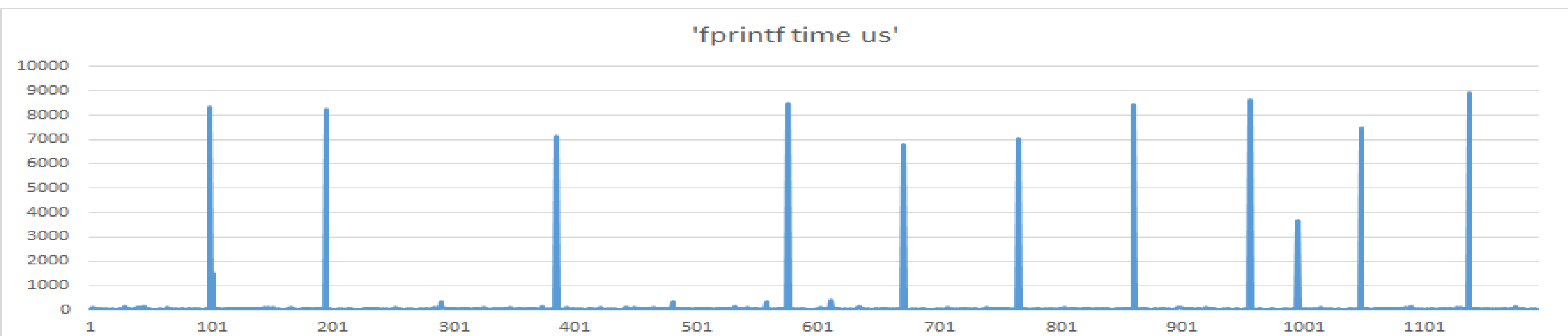
# TimingTest.c: how long does fprintf take?

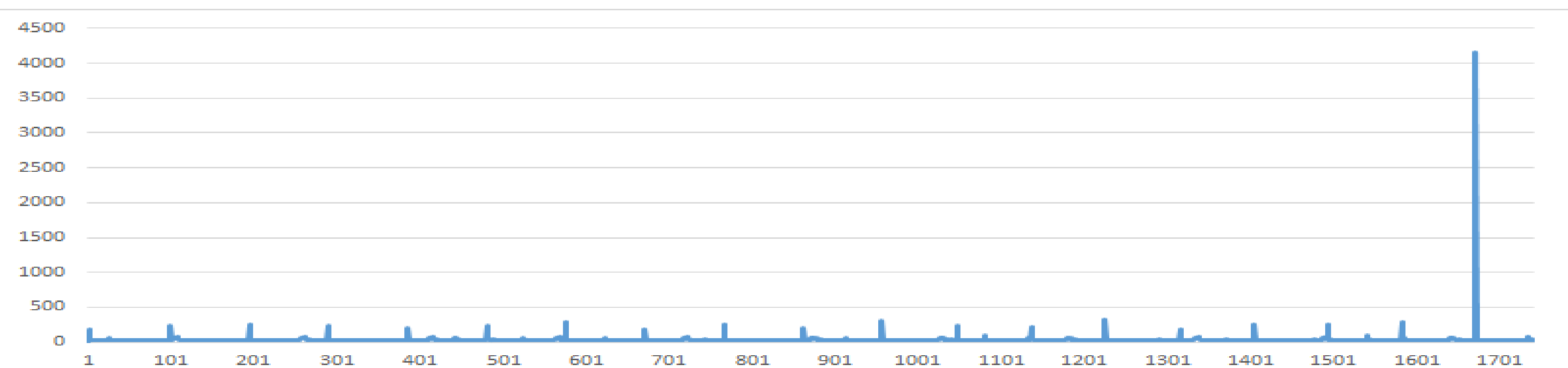
```
while(rc_get_state()!=EXITING)
{  // just data for csv format
        current_time = rc_nanos_since_boot() - start_time;
        old_tick = ticks;
        fprintf(logfile, "%ld, ", old_tick);  // pass value which not changing by other process
        current_time_f = ((double) current_time)/ 1e6; // milliseconds
        run_time_f = ((double) run_time)/1000.0; // us

        fprintf(logfile,"%8.3lf, %8.3lf, ", current_time_f, run_time_f);
        fprintf(logfile, "%" PRIu64 ", ",current_time);
        fprintf(logfile, "%" PRIu64 "\n",run_time);

        end_time = rc_nanos_since_boot() - start_time;
        run_time = end_time - current_time;
        while(old_tick == ticks)
        { rc_usleep(100); // sleep 100 us
        }
  }
```

run_time:  min 20 us, typical 30-50 us, <u>max 6600 us</u>

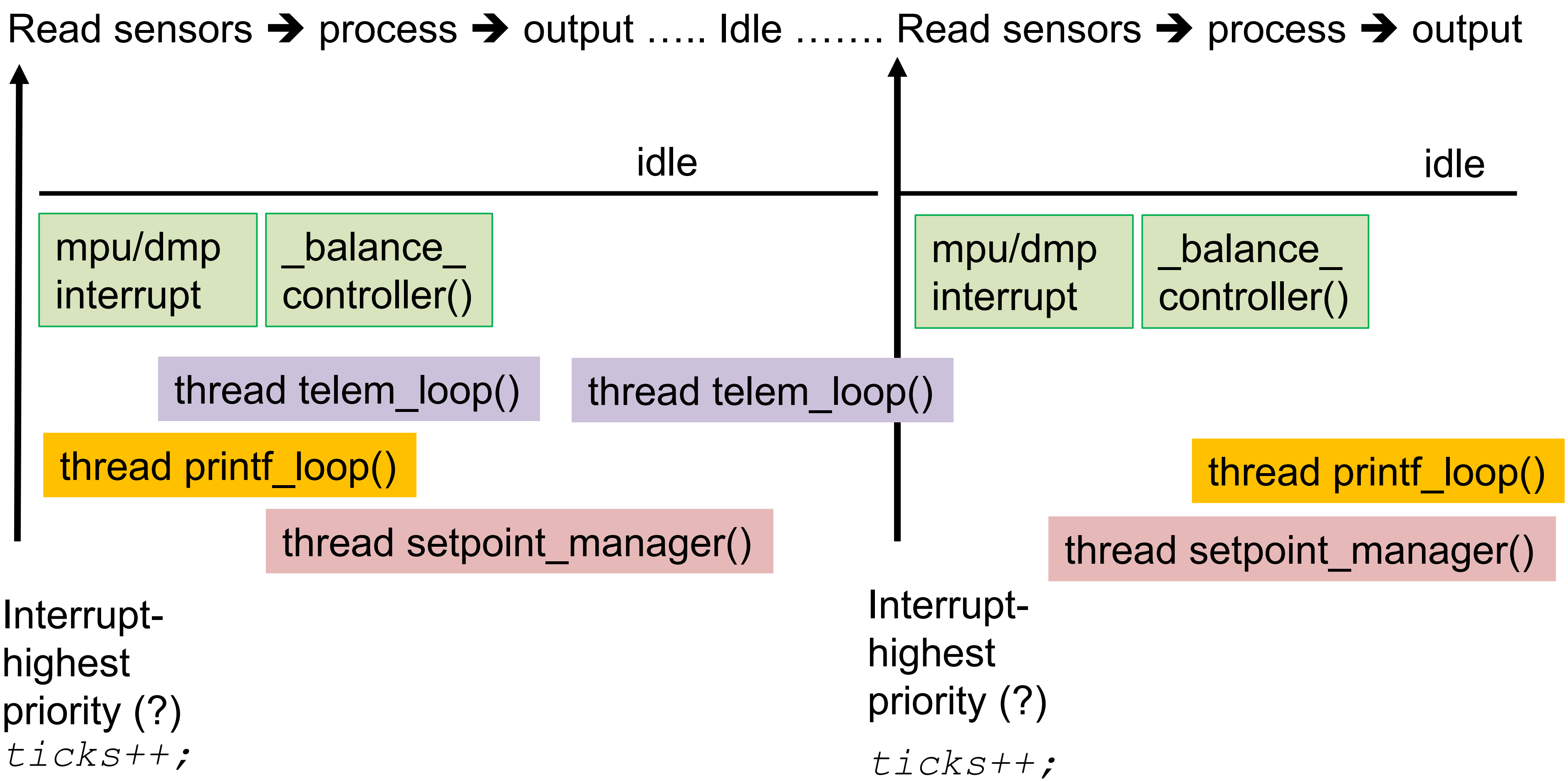
# Debian Processes/Delay

```
htop
# systemctl disable avahi-daemon
# systemctl stop avahi-daemon


sudo kill -9 {avahi-daemon, rc_battery_monitor,
apache2}.
```



# Software Notes- BeagleBone Threads




<u>Threads are asynchronous wrt interrupt!</u>

**`rc_pthread_set_process_niceness() ?`**

# rc_balance2.c using gyro/MPU

When new data is ready in the buffer, the IMU sends an interrupt to the BeagleBone triggering the buffer read followed by the execution of a function of your choosing set with the rc_mpu_set_dmp_callback() function.

```
// set up mpu configuration
rc_mpu_config_t mpu_config = rc_mpu_default_config();
mpu_config.dmp_sample_rate = SAMPLE_RATE_HZ;

// start mpu
if(rc_mpu_initialize_dmp(&mpu_data, mpu_config))

// this should be the last step in initialization
// to make sure other setup functions don't interfere
rc_mpu_set_dmp_callback(&__balance_controller);

// idle while sensing and control done elsewhere
while(rc_get_state()!=EXITING){
      rc_usleep(200000);  }
```

# rc_balance2.c __balance_controller()

```
static void __balance_controller(void)
{ticks++;
 end_time = rc_nanos_since_boot();
 run_time = end_time - start_time;
// time since previous interrupt

/**************************************************************
* STATE_ESTIMATION
* read sensors and compute the state
**************************************************************/
cstate.wheelAngleL =
(rc_encoder_eqep_read(ENCODER_CHANNEL_L) * 2.0 * M_PI) \
/(ENCODER_POLARITY_L * GEARBOX * ENCODER_RES);
/******************************************************
* Send signal to motors
******************************************************/
dutyL = cstate.d1_u - cstate.d3_u;
rc_motor_set(MOTOR_CHANNEL_L, MOTOR_POLARITY_L * dutyL);
}
```

# rc_balance2.c: threads

*// Note that using anything other than SCHED_OTHER with priority 0 is only available to root*

**int** main(**int** argc, **char** *argv[])
{ int c;
  pthread_t setpoint_thread = 0;
  pthread_t printf_thread = 0;
  pthread_t telem_thread = 0;
…
// print thread to print to screen without blocking main

```
rc_pthread_create(&printf_thread, __printf_loop, (void*) NULL,
SCHED_OTHER, 0);
```

…

// start balance stack to control setpoints

```
rc_pthread_create(&setpoint_thread, __setpoint_manager,
(void*) NULL, SCHED_OTHER, 0);
```

…

// telemetry thread to log to file

```
rc_pthread_create(&telem_thread, telem_loop, (void*) NULL,
SCHED_OTHER, 0);
// telem loop could write to file
```

# Topics

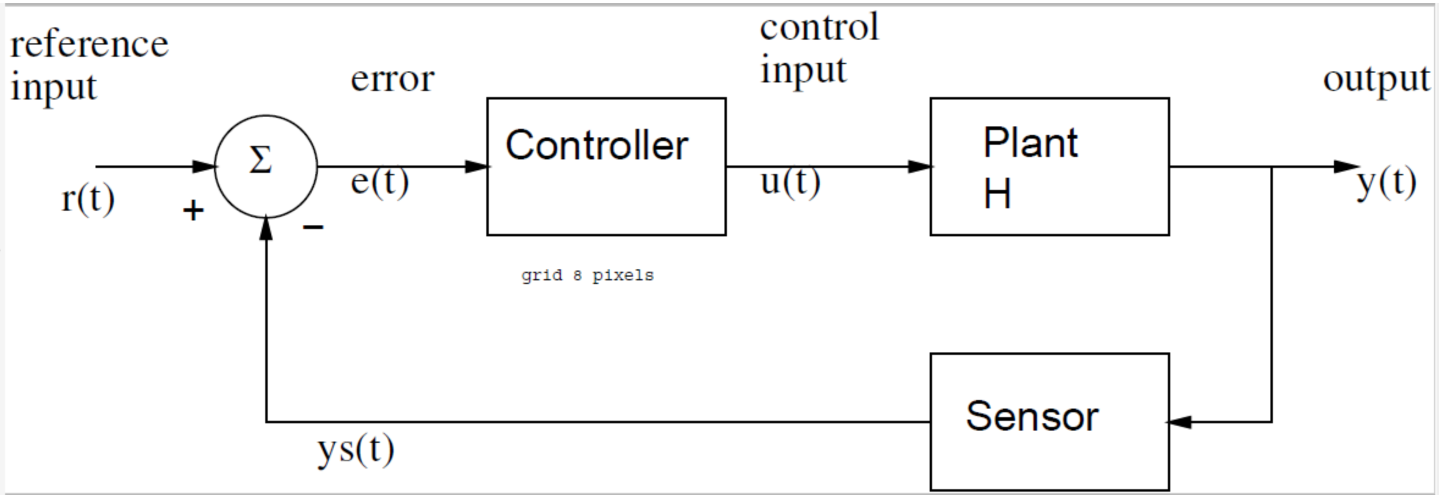
- Upcoming checkpoints
- Quiz 3 soln
- Steering control (advanced)
- PCB highlights
- Power conversion
  - Linear regulator
  - Buck converter
- Software: MPU interrupt + threads
- Telemetry logging
- *Discrete Time control/timing*

# Example logging thread

```
// telemetry thread to log to file
void* telem_loop(__attribute__ ((unused)) void* ptr)
{       long old_tick=0; uint64_t initial_time, startsnap_time, endsnap_time;
        printf("telem thread\n"); fflush(stdout); // empty buffer
        initial_time = rc_nanos_since_boot();
        while(rc_get_state()!=EXITING)
        {       startsnap_time = rc_nanos_since_boot();
                old_tick = ticks;  // ticks set by another process
                // take snapshot- assume assignments are atomic (?)
                log_yaw = cstate.yaw; log_dutyL = cstate.dutyL; log_dutyR = cstate.dutyR;
                log_vBatt = cstate.vBatt;
                endsnap_time = rc_nanos_since_boot(); // bracket how stale data is
                fprintf(logfile, "%ld, %10.3f, %10.3f, %8.3f, %8.3f, %8.3f, %8.3f\n",
                        old_tick,  (double)(startsnap_time-initial_time)/1e6),
                         (double)(endnap_time-initial_time)/1e6),
                        log_yaw, log_dutyL, log_dutyR, log_vBatt);
                while(old_tick == ticks)
                { rc_usleep(100); // sleep 100 us
                }
        }
        rc_usleep(1000000 / PRINTF_HZ);
        return NULL;
}
```

# Extra Slides

# Control Synopsis




State equations: $$\dot{x}(t) = ax(t) + bu(t)$$

Output equations: $$y(t) = cx(t) + du(t)$$

Control Law (P): $$u(t) = k_p e(t) = k_p(r(t) - y(t)).$$

# Control Synopsis

Control Law (P): $u(t) = k_p e(t) = k_p(r(t) \ - \ y(t)).$

New state equations:

$$\dot{x} = ax + bk_p e(t) = ax + bk_p(r - x) = (a - bk_p)x + bk_p r.$$

Zero Input Response (non-zero init condx):

$$x(t) = x(0)e^{(a-bk_p)t} \quad for \quad t \geq 0.$$

$a' = a - b\,k_p$  $b' = b\,k_p$

Total Response (non-zero init condx) by convolution:

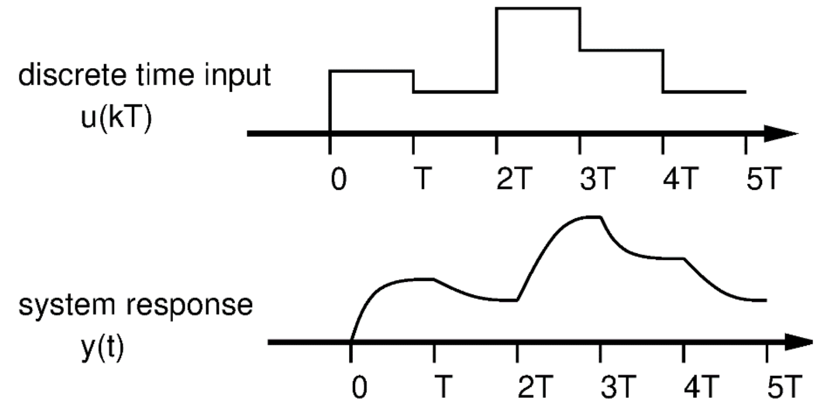
$$x(t_o) = e^{a't_o}x(0) + \int_0^{t_o} e^{a'(t_o-\tau)} b' r(\tau) d\tau \ . \tag{10}$$

Step Response (zero init condx) by convolution:

$$x(t_o) = b' \int_0^{t_o} e^{a't_o} e^{-a'\tau} d\tau \ = \ \frac{-b'e^{a't_o}}{a'} e^{-a'\tau}|_0^{t_o} \ = \frac{b'}{a'}(1 - e^{-a't_o}) \ . \tag{11}$$

# Control Synopsis- Discrete Time

Superposition of Step Responses




$$x((k+1)T) = e^{a(k+1)T}x(0) + e^{a(k+1)T}\int_0^{(k+1)T} e^{-a\tau}bu(\tau)d\tau \ . \qquad (15)$$

$$x(kT) = e^{akT}x(0) + e^{akT}\int_0^{kT} e^{-a\tau}bu(\tau)d\tau \ . \qquad (14)$$

$$x((k+1)T) = e^{aT}x(kT) + e^{a(k+1)T}\int_{kT}^{(k+1)T} e^{-a\tau}bu(\tau)d\tau \ = e^{aT}x(kT) + \int_0^{T} e^{a\lambda}bu(kT)d\lambda \ , \quad (16)$$

# Control Synopsis- Discrete Time

$$G(T) \equiv e^{aT} \quad \text{and} \quad H(T) \equiv b \int_0^T e^{a\lambda} d\lambda \ . \tag{17}$$

State equations:

$$x((k+1)T) = G(T)x(kT) + H(T)u(kT) \tag{18}$$

Output equations:

$$y(kT) = Cx(kT) + Du(kT) \ . \tag{19}$$

Total Response (non-zero init condx) by convolution:

$$x(k) = G^k x(0) + \sum_{j=0}^{k-1} G^{k-j-1} H u(j) \ . \tag{23}$$

# Control Synopsis- Discrete Time

Control Law (P): $U(kT) = k_p\,[r(kT) - x(kT)]$

New state equations:

$$x((k+1)T) = G(T)x(kT) + H(T)k_p(r(kT) - x(kT)) = [G - Hk_p]x(kT) + Hk_p r(kT)\ . \quad (24)$$

$$x((k+1)T) = [e^{aT} + \frac{k_p}{a}(1 - e^{aT})]x(kT) + Hk_p r(kT) \ = \ G' x(kT) + Hk_p r(kT)\ . \quad (25)$$

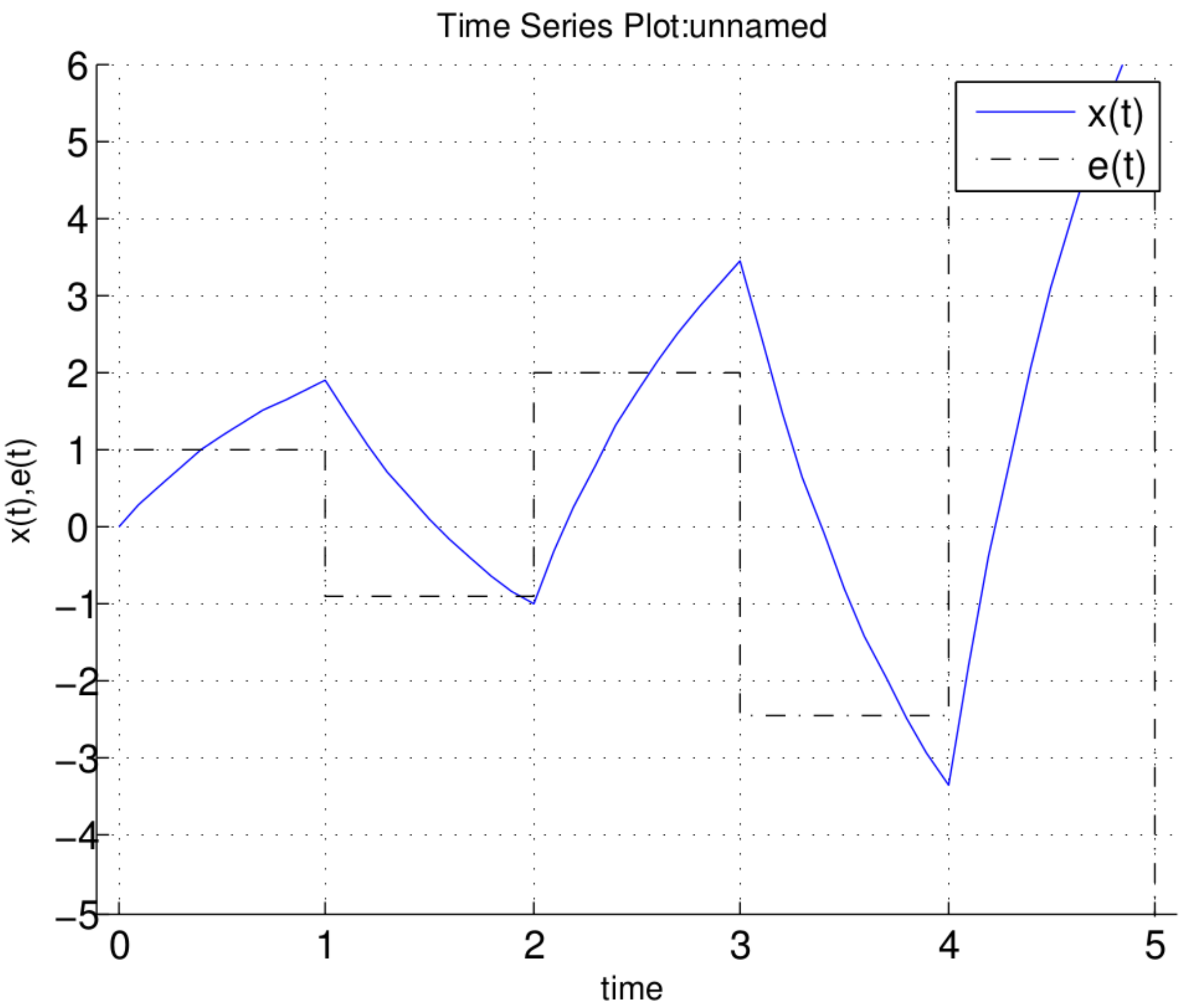
For stability:

$$|e^{aT} - \frac{k_p}{a}(e^{aT} - 1)| < 1. \quad (26)$$

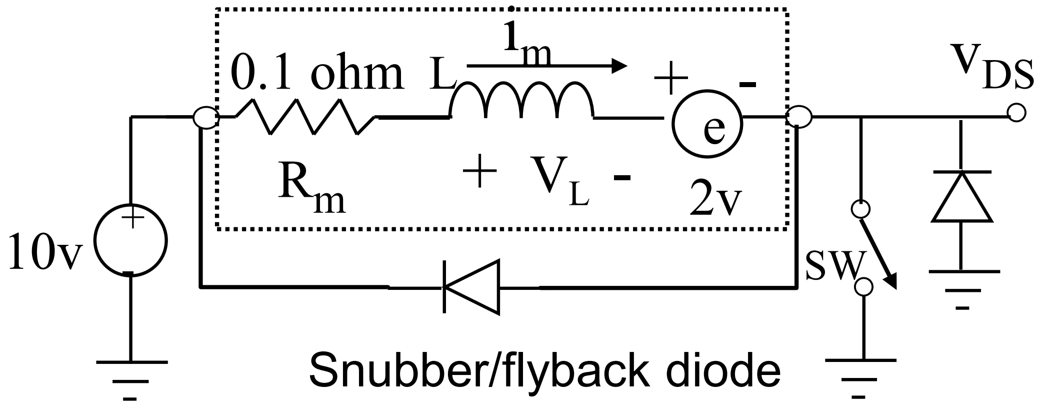
Notes: stability depends on gain **and** T!

# Discrete Time Control

u[k] = kp*(r[k]-x[k])




On board




$$V_{DS} = 10V - V_{DIODE}$$

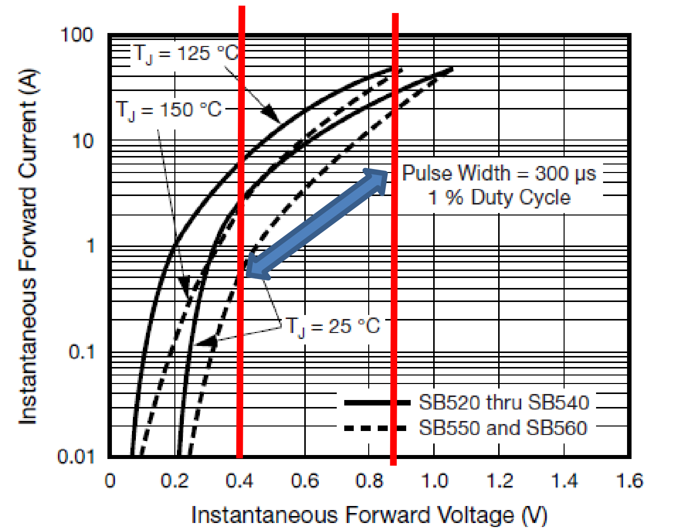
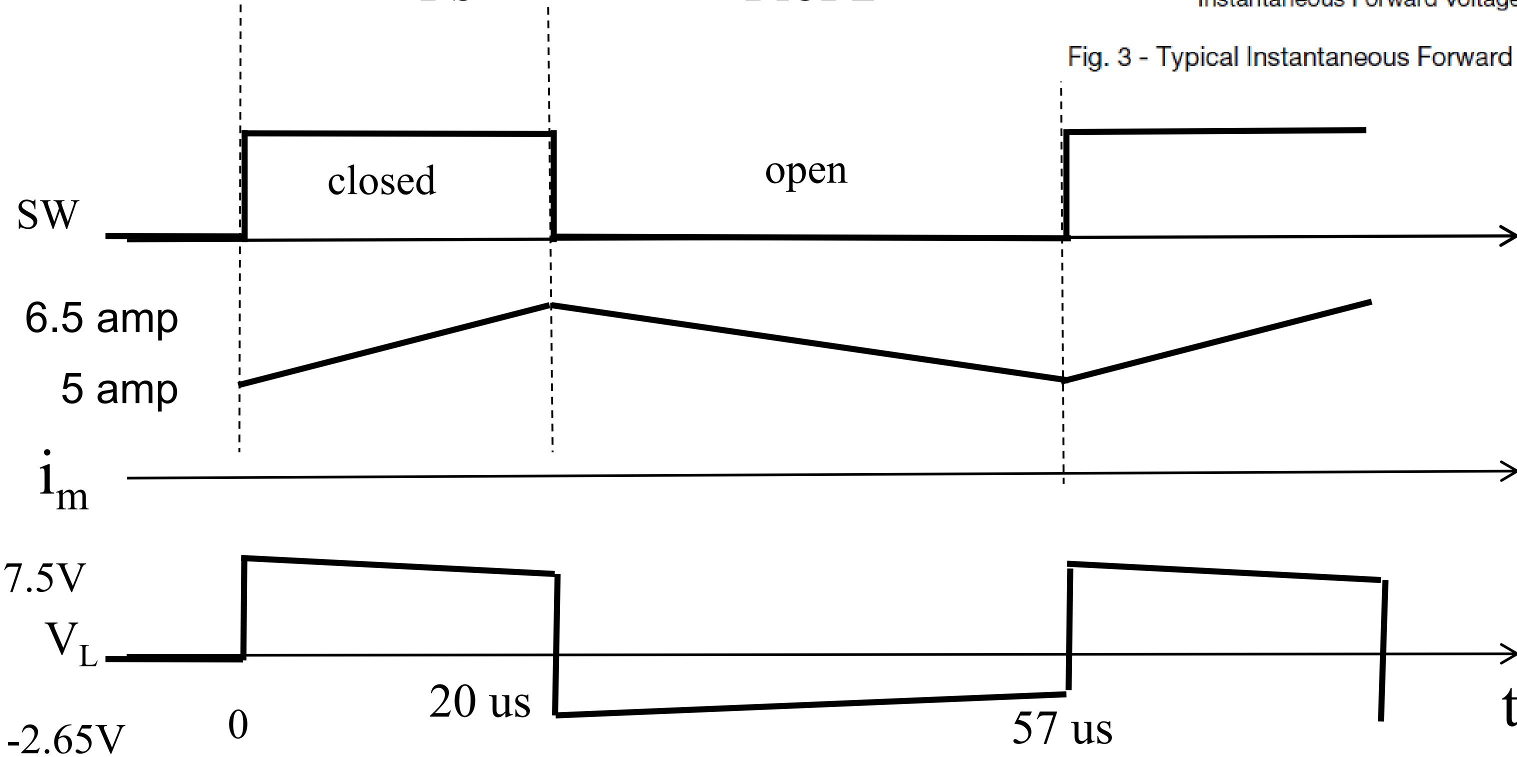



Fig. 3 - Typical Instantaneous Forward Characteristics




# Back EMF velocity sensing