SE3910 – REAL TIME SYSTEMS

ROADMAP

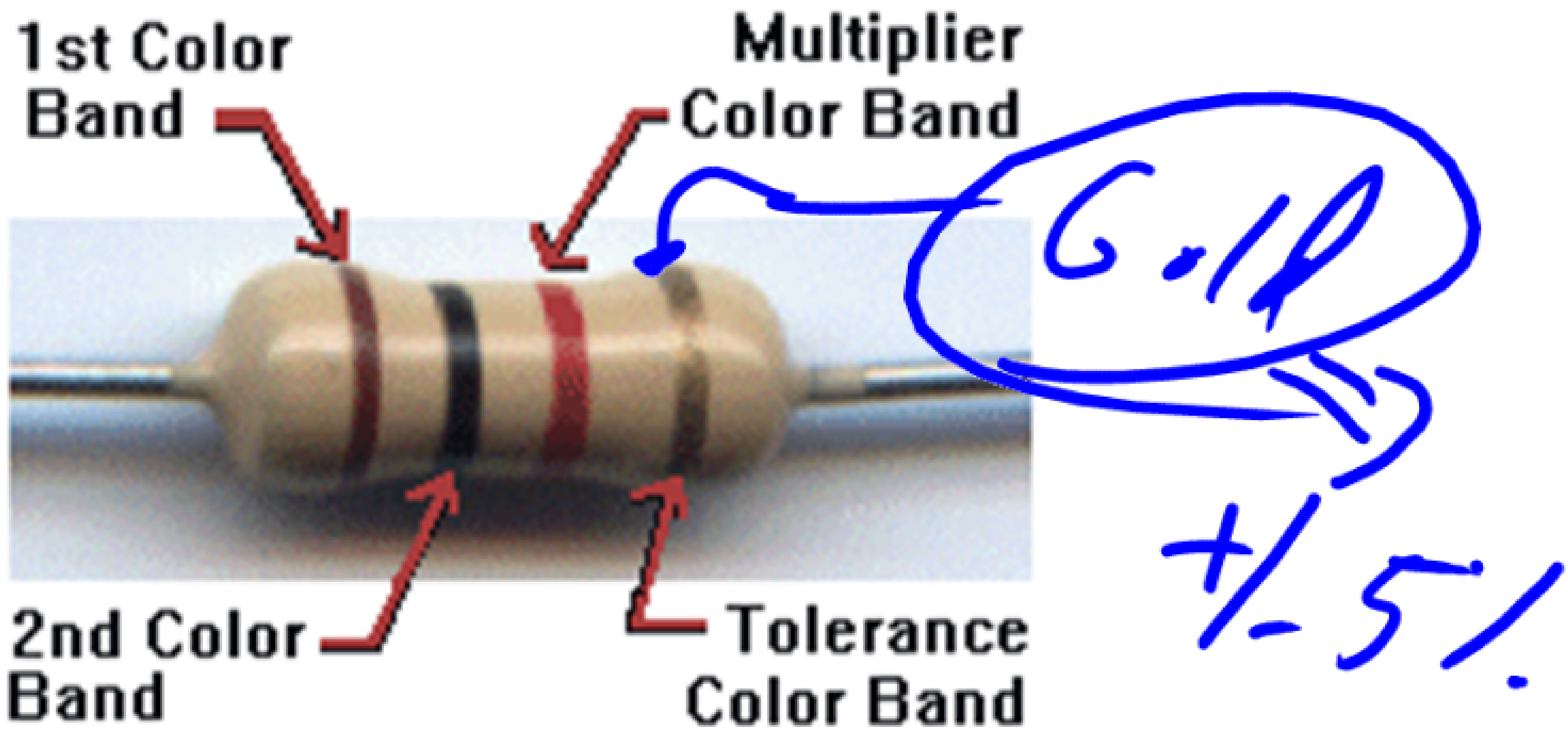
- Today
  - Resistor codes and RTOS definitions
- Monday
  - An in class demo on sockets / C programming tutorials
    - Hint: Will need laptops for this…
- Tuesday / Thursday Labs
  - Sockets on the embedded platform
- Wednesday
  - RTOS Scheduling

OBJECTIVES

- Lab topic
  - Be able to calculate the resistance of a given resistor using the color bands

- Understand the CPU Utilization Factor
- Given a set of processes, calculate the CPU utilization factor
- Define the acronym RTOS
- Explain the role of the kernel in operating systems
- Compare and contrast Polled loops, polled loops with delay, and cyclic code structures
- Explain switch bounce
- Explain how to construct an interrupt only system
- Explain the concept of background and foreground tasks

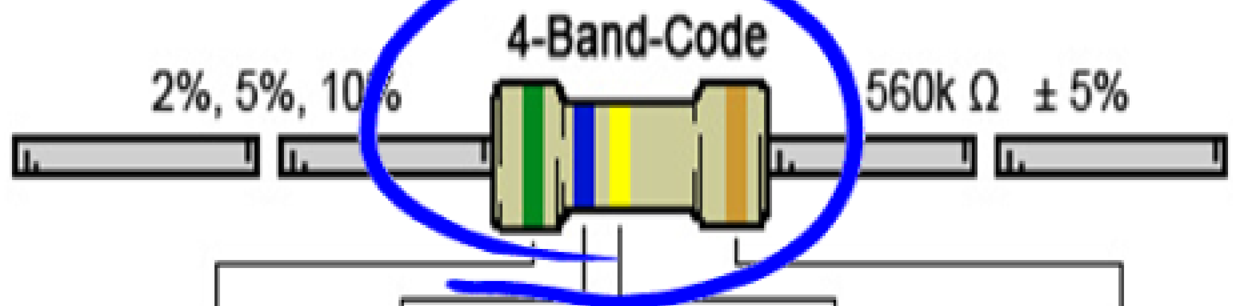
RESISTOR COLOR CODE



- 1st band color gives 1st number
- 2nd band color gives 2nd number
- 3rd band color gives # of zeros
- 4th band color gives tolerance or ± —

RESISTOR COLOR CODE DEFINITION

(HTTP://WWW.DIGIKEY.COM/WEB%20EXPORT/MKT/GENERAL/MKT/RESISTOR-COLOR-CHART.JPG)



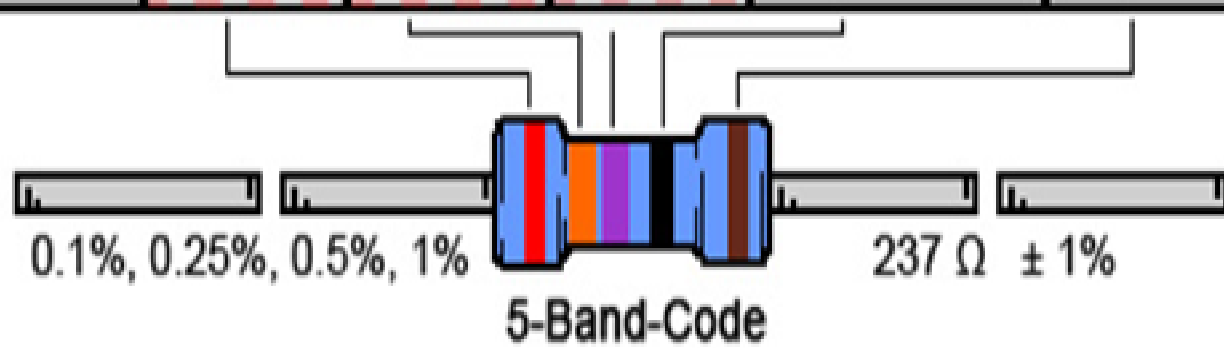
COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

GBYG

56×10k

56×10k

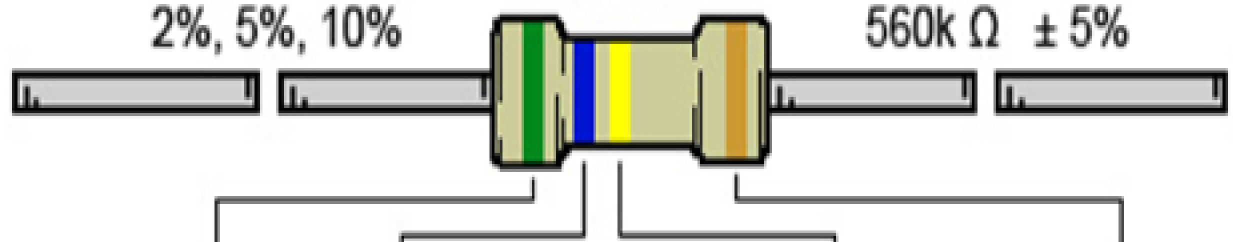
560kΩ



SE3910 REAL TIME SYSTEMS

RESISTOR COLOR CODE DEFINITION

(HTTP://[illegible].DIGIKEY.COM/[illegible]/EB%20EXPORT/MKT/GENERAL/MKT/RESISTOR-COLOR-CHART.JPG)



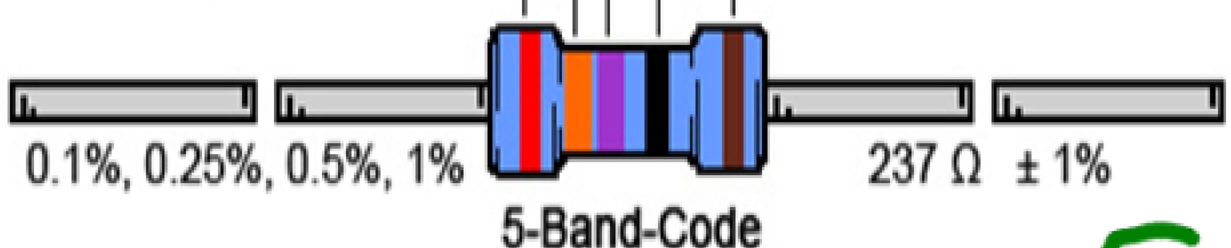
COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Red 2
Black 0
Yellow 10k
Gold
200k

RESISTOR COLOR CODE DEFINITION

(HTTP://WWW.DIGIKEY.COM/WWW/EB%20EXPORT/MKT/GENERAL/MKT/RESISTOR-COLOR-CHART.JPG)

COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

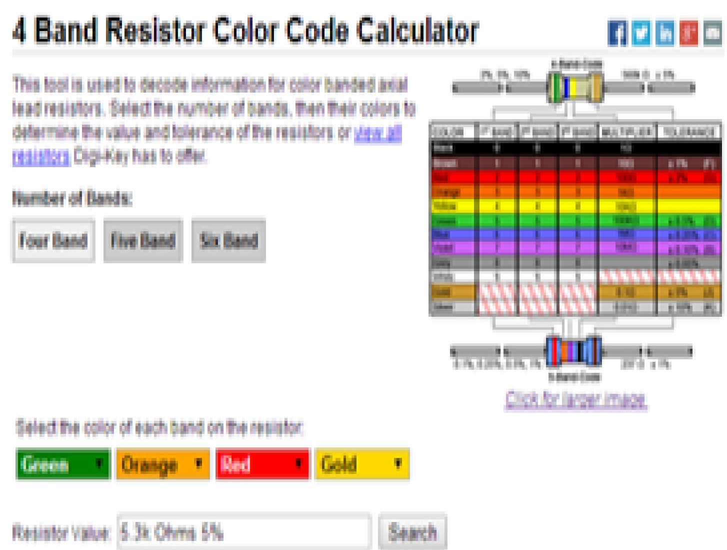


Green
Orange
Red
Gold

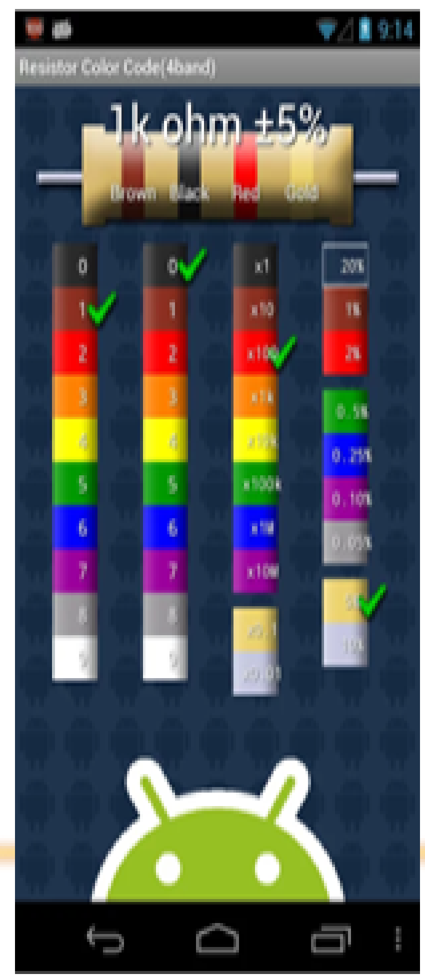
5 3 100Ω
5300 ⇒ 5.3kΩ

WAYS TO FIGURE THIS OUT

- By hand (which I expect you CAN do)
  - You would be provided a chart
- Online
  - http://www.digikey.com/us/en/mkt/calculators/4-band-resistors.html



- On your phone
  - S2 Resistor Color Code



CPU UTILIZATION DEFINITION

- The CPU utilization or time loading factor, U, is a relative measure of the non-idle processing taking place on the processor.

- $u_i = \frac{e_i}{p_i}$
  - P_i=execution period
  - F_i=Execution frequency $(f_i = \frac{1}{p_i}$
  - E_i=worst case execution time

- Overall system utilization
  - $U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{e_i}{p_i}$

Video Decompressing of a frame

$\frac{5ms}{33ms}$

$\frac{5}{33} \Rightarrow \sim 15\%$

Audio processing

$\frac{1ms}{10ms}$

10%

EXAMPLE

- A system has 4 processes in, as is described below. What is the overall utilization?

- Task 1: Measure wheel slip (p1=100ms, e1=12ms)
- Task 2: Measure traction capabilities of wheel (p2=50ms, e2=5ms)
- Task 3: Monitor system diagnostics (p3=200ms, e3=5ms)
- Task 4: Send system messages over network (p4=25ms, e4=1ms)

$$\sum_{i} \frac{12}{100} + \frac{5}{50} + \frac{5}{200} + \frac{1}{25}$$

$$U = \sum 12\% + 10\% + 2.5\% + 4\%$$

$$U = 28.5\%$$

WHY DO WE CARE?

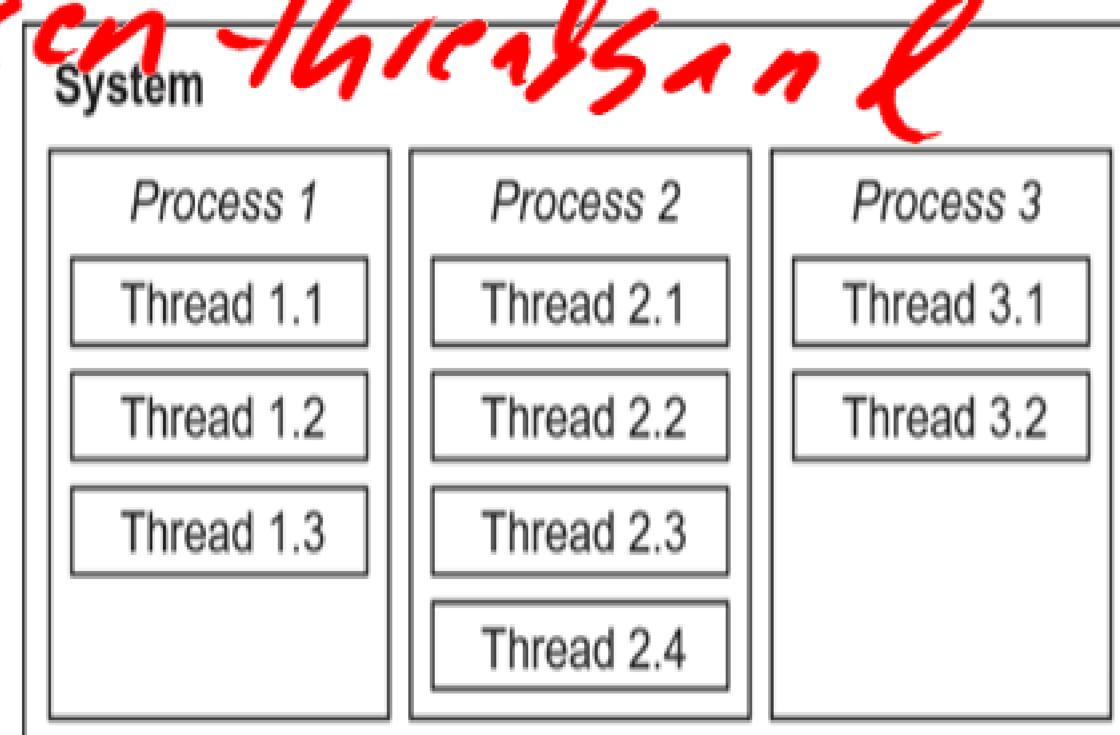
TABLE 1.3. CPU Utilization (%) Zones

Utilization (%)	Zone Type	Typical Application
<26	Unnecessarily safe	Various
26–50	Very safe	Various
51–68	Safe	Various
69	Theoretical limit	Embedded systems
70–82	Questionable	Embedded systems
83–99	Dangerous	Embedded systems
100	Critical	Marginally stressed systems
>100	Overloaded	Stressed systems

THE ROLES OF THE OPERATING SYSTEM

- Task – Piece of the problem we are solving.
  - An abstraction of a running program and a logical unit of work schedulable by an RTOS
- Process
  - A private data structure containing an identity, priority level, state of execution, and resources
- Thread
  - A lightweight process that resides within a process

RTOS ⇒ Do not distinguish between threads and processes.



ESSENTIAL FUNCTIONS OF AN RTOS

- 3 essential functions
  - Scheduling
    - Determines which task will run next
  - Dispatching
    - Starts and stops tasks from running
  - Intertask communication and synchronization
    - Assure that parallel tasks can cooperate effectivly

What is an RTOS?

Real Time Operating System

VxWorks uCos

WinDriver

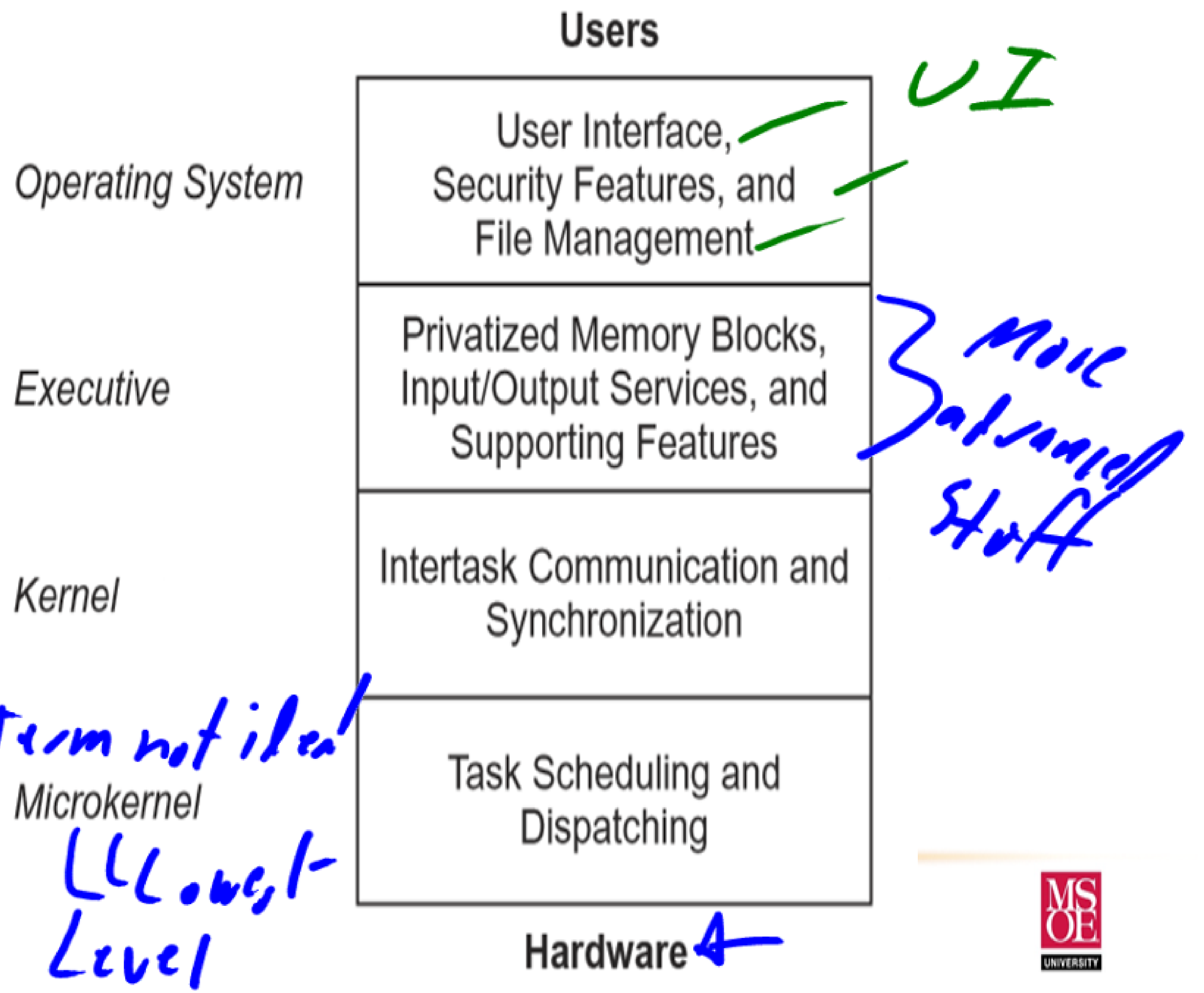
Embedded Linux

ESSENTIAL FUNCTIONS OF AN RTOS

- 3 essential functions
  - Scheduling
    - Determines which task will run next
  - Dispatching
    - Starts and stops tasks from running
  - Intertask communication and synchronization
    - Assure that parallel tasks can cooperate effectivly

Sharing messages, semaphores, etc.

THE TAXONOMY OF AN RTOS



PROBLEM SOLVING

- We have a system that is to determine if a packet of data arrives. The data comes in no more often than once per second. A flag is set when the data arrives. How might we write code that could implement this functionality?

Loop:
check flag
Load something

check for the data. process it if is there?

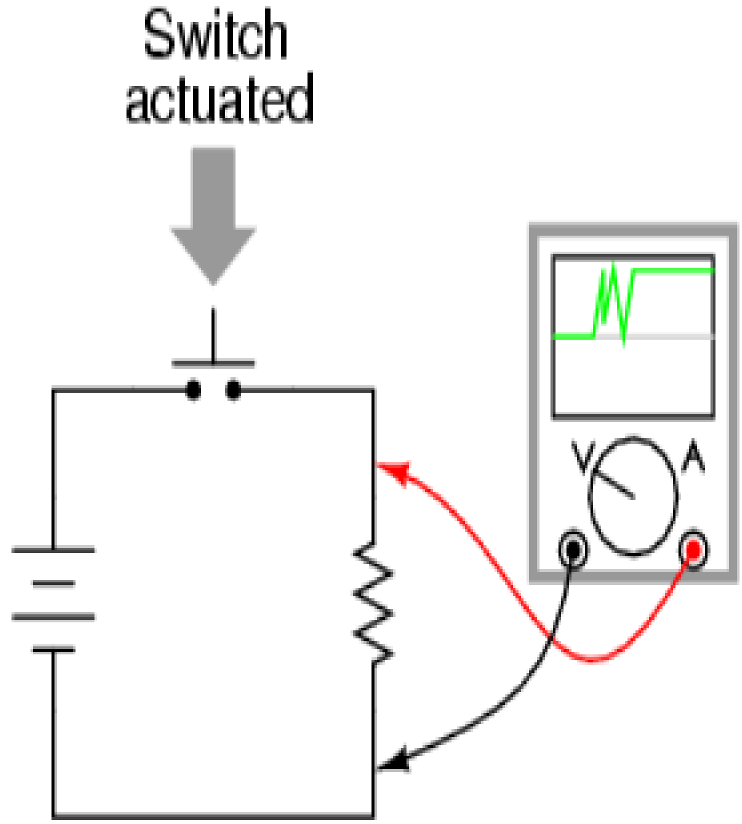
A SIMPLE POLLED LOOP

```
for(;;) {                    /* do forever */
        if (packet_here)     /* check flag */
           {
           process_data();   /* process data */
           packet_here=0;    /* reset flag */
           }
        }
```

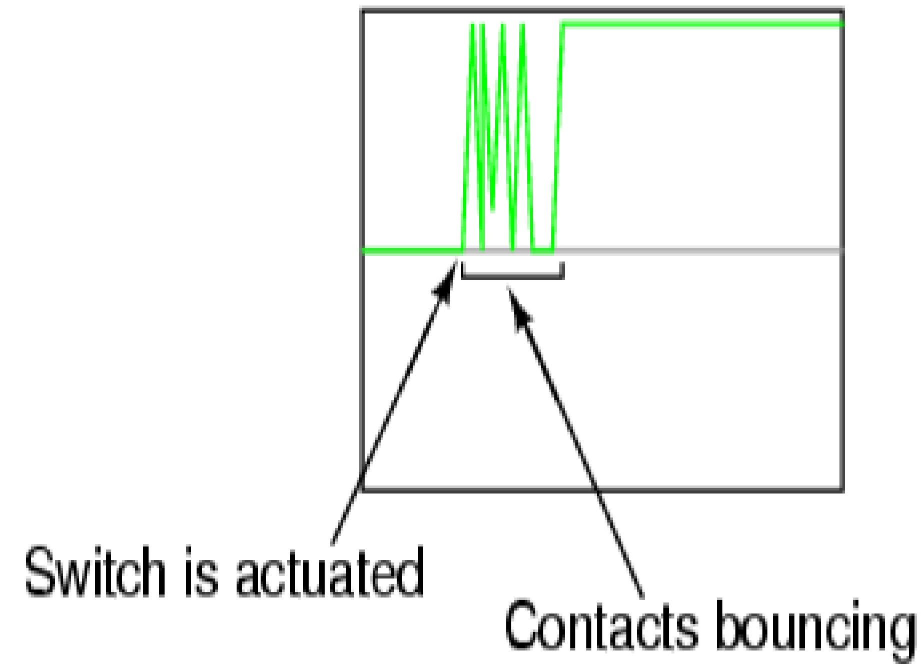
Works well when we have a single CPU Dedicated to doing I/O for some device.

⇒ Can be used for background tasks ...

PROBLEM: SWITCH BOUNCE



Close-up view of oscilloscope display:



SOLUTION: POLLED LOOP WITH A DELAY

```
for(;;) {                      /* do forever */
        if (flag)              /* check flag */
          {
          process_event();     /* process event */
          pause(21);           /* wait 21 ms */
          flag=0;              /* reset flag */
          }
}
```

Wait

20ms for the switch to stabilize

- Question: What if we want to control how often the tasks execute in greater detail?

```
for(;;) {            /* do forever */
        task_1();
        task_2();
        ...
        task_n();
        }
```

```
for(;;) {            /* do forever */
        task_1();
        task_2();
        ...
        task_n();
        task_2();
        }
```

CYCLIC CODE STRUCTURE

```
for(;;) {              /* do forever */
        task_1();
        task_2();
        ...
        task_n();
        }
```

```
for(;;) {              /* do forever */
        task_1();
        task_2();
        ...
        task_n();
        task_2();
        }
```

INTERRUPT ONLY SYSTEM

```
void main(void)
{
  init();             /* system initialization */
  while(TRUE);        /* jump-to-self */
}
void int_1(void)     /* interrupt handler 1 */
{
  save(context);      /* save context to stack */
  task_1();           /* execute task 1 */
  restore(context);  /* restore context */
}
void int_2(void)     /* interrupt handler 2 */
{
  save(context);      /* save context to stack */
  task_2();           /* execute task 2 */
  restore(context);  /* restore context */
}
void int_3(void)     /* interrupt handler 3 */
{
  save(context);      /* save context to stack */
  task_3();           /* execute task 3 */
  restore(context);  /* restore context */
}
```

FOREGROUND / BACKGROUND SYSTEM