
EECE 276

Embedded Systems

Timer functions and events

Timer functions

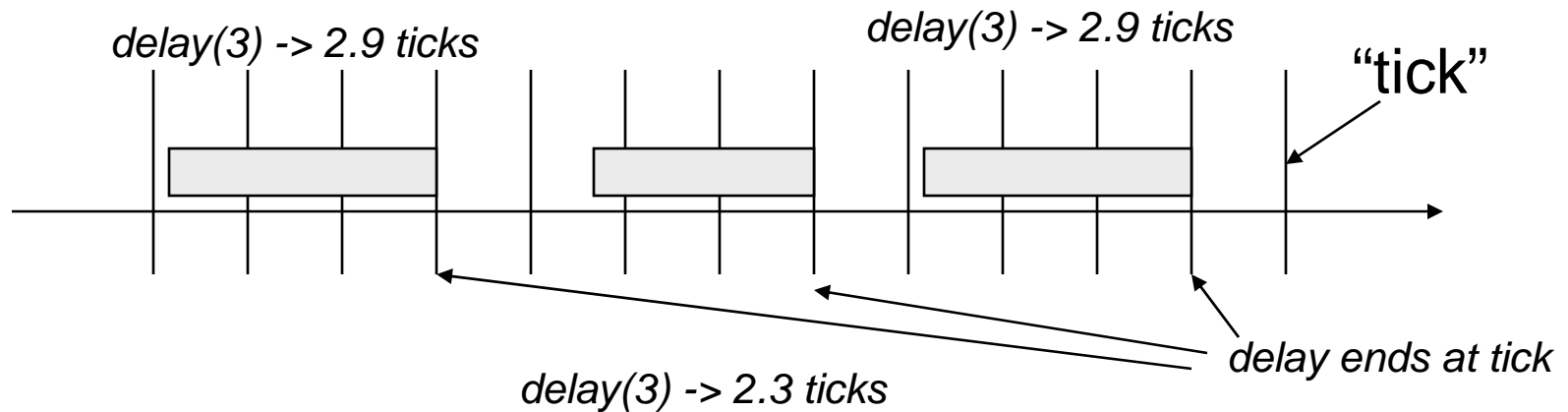
- Real-time systems often must synchronize their activities to the passage of time
 - » “Turn power off if nothing happens for 1 mins”
 - » Dial a digit: 100msec tone+100msec pause
 - » “If temp and press is too high continuously for 3 minutes, then shut down reactor”
- Common technique:
 - » `delay(int units)` – delays task for a “units”
 - » `delay_call(int units,(void)(*func)(void*),void* arg)` – call “func” with “arg” after “units”

Timer example code

```
void phone_dial_task() {
    char phNum[MAX_PH_NUM]; // 0-terminated phone number str.
    char *phNumP;
    while(1) {
        msgq_rcv(queue,&phNum,MAX_PH_NUM); // Get numbers
        phNumP = phNum;
        while(*phNumP) {
            delay(100);                // delay for 100 mSec
            dialToneOn(*phNumP - '0'); // start tone
            delay(100);                // delay for 100 msec
            dialToneOff();             // stop tone
            phNumP++;                 // next digit
        }
    }
}
```

Issues with timers

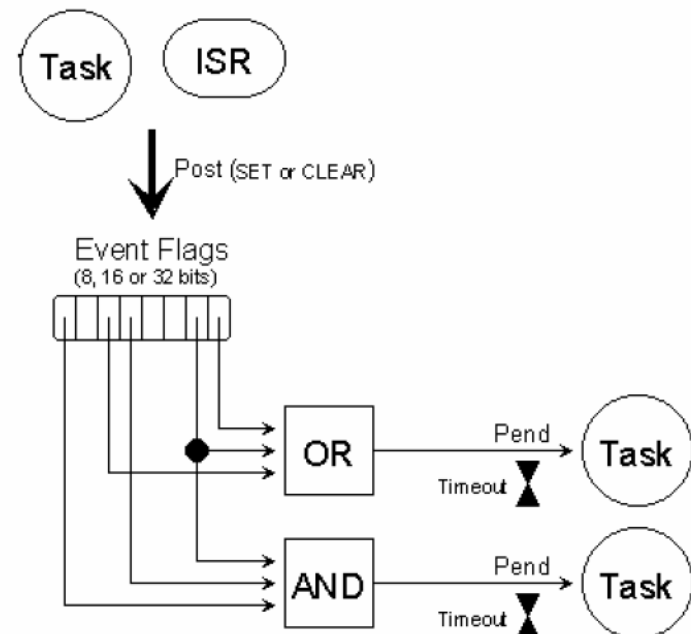
- The `delay(int)` may not precisely delay



- No standard "tick" (read RTOS manual)
- More accurate timing: use hardware timers

Events: Synchronization devices

- “event”: boolean flag
- Multiple tasks can set it/wait for it
- A task can wait for multiple events
 - » AND and OR synchronization
- uCOS: Event flag groups
 - » See App Note



Comparison of task synchronization tools

- Semaphores:
 - » Fast and simple
 - » “Unstructured”
 - » Limited information is carried
- Events:
 - » Slightly more complex than semaphores
 - » Multiple tasks for one event
 - » One task for multiple events
- Queues:
 - » Pass “large” amounts of data
 - » Enqueuing/dequeuing data is expensive