RTOS Timer functions

Reference: Simon Chapter 7

Most embedded systems must keep track of the passage of time

For example:

- To extend its battery life, a cordless bar-code scanner must turn itself off after a certain number of seconds.
- Systems with network connections must wait for acknowledgements to data that they have sent and retransmit the data if an acknowledgement doesn't show up on time.
- Manufacturing systems must wait for robot arms to move or for motors to come up to speed.

Delay is a task blocking operation

- Most RTOSs offer a function that delays a task for a period of time.
- That is... blocks the task until the period of time expires.

Telephone call example

vMakePhoneCallTask

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
    }
}
```

In the US each of the tones that represents a digit must sound for one-tenth of a second, and there must be one-tenth-second silences between the tones.

vMakePhoneCallTask receives a phone number from an RTOS message queue

taskDelay

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
   }
}
```

The function **msgQreceive** copies the phone number from the queue into **a_chPhoneNumber**.

While-loop calls **taskDelay** first to create a silence and then to create a tone of appropriate length for each digit in the phone number.

msgQreceive and taskDelay are VxWorks functions

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
    }
}
```

The functions vDialingToneOn and vDialingToneOff turn the tone generator on and off.

The **msgQreceive** and **taskDelay** functions in this code are from VxWorks.

taskDelay function takes a number of milliseconds as its parameter?

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
   }
}
```

No. The taskDelay function in VxWorks (like most RTOSs), takes the number of system ticks as its parameter.

The length of time represented by each system tick is something you can usually control when you set up the system.

How accurate are the delays produced by the taskDelay function?

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
   }
}
```

They are accurate to the nearest, system tick.

The RTOS works by setting up a single hardware timer to interrupt periodically, say, every millisecond, and bases all timings on that interrupt.

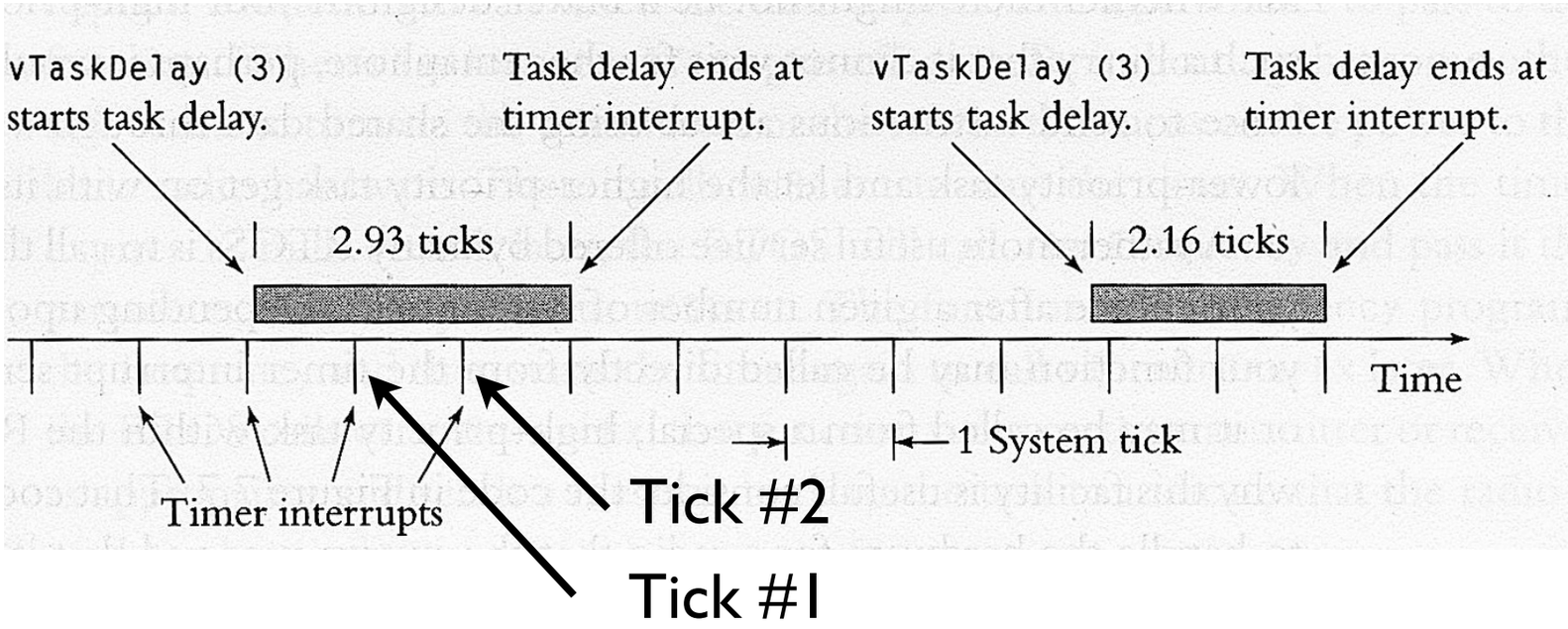
Heartbeat timer

```
//Message queue for phone numbers to dial.
extern MSG_Q_ID queuePhoneCall;
void vMakePhoneCallTask (void)
{
  #define MAX_PHONE_NUMBER    11
  char a_chPhoneNumber[MAX_PHONE_NUMBER];
  //Buffer for null-terminated ASCII number
  char *p_chPhoneNumber;
  //Pointer into a_chPhoneNumber
  while  (TRUE){
  msgQreceive (queuePhoneCall, a_chPhoneNumber,
MAX_PHONE_NUMBER, WAIT_FOREVER);
  //Dial  each of the digits
  p_chPhoneNumber = a_chPhoneNumber;
  while  (*p_chPhoneNumber)
  {
    taskDelay(100); //1/10th of a second silence
    vDialingToneOn(*p_chPhoneNumber  -'0');
    taskDelay(100); //1/10th of a second with tone
    vDialingToneOff ();
    //Go to the next digit in the phone number
    ++p_chPhoneNumber;
    }
   (...)
    }
}
```

This timer is often called the heartbeat timer.

For example, if one of your tasks passes 3 to taskDelay, that task will block until the heartbeat timer interrupts three times.

Timer function accuracy



- The first timer interrupt may come almost immediately after the call to taskDelay or it may come after just under one tick time or after any amount of time between those two extremes.

- The task will therefore be blocked for a period of time that is between just a tiny more than two system ticks and just a tiny less than three system ticks.

Setting up timer hardware

- **Question**: How does the RTOS know how to set up the timer hardware on my particular hardware?
- It is common for microprocessors to have timers.
- If you are using nonstandard timer hardware, then you may have to write your own timer setup software and timer interrupt routine.
- The RTOS will have an entry point for your interrupt routine to call every time the timer expires.
- Many RTOS vendors provide board support packages , which contain driver software for common hardware components.

What is a "normal" length for the system tick?

- There really isn't one.
- The **advantage** of a short system tick is that you get accurate timings.
- The **disadvantage** is that the microprocessor must execute the timer interrupt routine frequently.
- Since the hardware timer that controls the system tick usually runs all the time, whether or not any task has requested timing services, a short system tick can decrease system throughput quite considerably by increasing the amount of microprocessor time spent in the timer interrupt routine.

What if my system needs extremely accurate timing?

- One choice is to make the system tick short enough that RTOS timings fit your definition of "extremely accurate."
- Other choice is to use a separate hardware timer for those timings that must be extremely accurate.
- It is not uncommon to design an embedded system that uses dedicated timers for a few accurate timings and uses the RTOS functions for other timings that need not be so accurate.
- The advantage of the RTOS timing functions is that one hardware timer times many number of operations simultaneously.

Most RTOSs offer other timing services based on the system tick

- **Example #1**: Limit how long a task will wait for a message from a queue or a mailbox.
- **Example #2**: Define how long a task will wait for a semaphore
  - If you set a time limit when your high-priority task attempts to get a semaphore and if that time limit expires, then your task does not have the semaphore and cannot access the shared data.