

Cálculo en tiempo

¿Cómo funciona un sistema operativo en tiempo real (RTOS)?

Por Leon van Snippenberg, AVIX-RT

Con el uso de un sistema operativo en tiempo real (RTOS), se puede mejorar mucho la estructura del software y las de funciones de temporización de un sistema de microcontrolador. En este artículo describimos el funcionamiento y las funciones específicas de un RTOS.

Cada vez con más frecuencia se usan microcontroladores en los sistemas electrónicos. Hace poco se publicó en Elektor una serie de artículos sobre los microprocesadores de 16 bits de Microchip de la serie 24F, 24H y 33F. Estos modernos y potentes microcontroladores disponen de muchos periféricos y una arquitectura de interrupción avanzada. El uso de un sistema operativo en tiempo real (RTOS) puede ayudar a mantener

controlable la estructura del software y las funciones de temporización de un sistema basado en un potente microcontrolador de este tipo.

En este artículo describimos como funcionan estos RTOS. Los ejemplos aquí mencionados se basan en AVIX, un RTOS que el autor ha desarrollado especialmente para las familias de microcontroladores antes mencionadas. Al final del artículo se reseña una aplicación demostrativa para la placa Explorer-16 de Elektor.

Estructura de software general

El software suele montarse de forma modular. Por eso es importante que los distintos módulos se coloquen oportunamente por orden para que el sistema funcione correctamente como conjunto.

Hemos tomado como ejemplo el sistema de la **figura 1**. Aquí vemos un módulo para leer valores analógicos y un módulo para su procesamiento. Estos módulos funcionan bajo control de un planificador, que aquí no es nada más que una función principal que llama a las demás funciones. Si este sistema debe leer los valores analógicos una vez por milisegundo, el procesamiento debe haber terminado antes de que empiece el siguiente milisegundo. Si dura más, entonces deberá "recortarse" el procesamiento en segmentos, de manera que se cumplan los requisitos de temporización. En tal caso, dentro del módulo de procesamiento se registra qué parte debe activarse en una llamada, de

manera que el conjunto se realice en el orden correcto en la parte exterior del módulo de procesamiento. Dentro del módulo de procesamiento se forma una máquina de estado. Esta estructura puede verse en la **figura 2**.

Ahora, la atribución de capacidades del microcontrolador a los módulos se reparte entre el Planificador y la Máquina de estado. Si se usa un gran número de módulos la estructura de software será más compleja y confusa. Un segundo problema es la temporización del sistema. La temporización correcta depende del tiempo de procesamiento de los distintos (sub)módulos. Este tiempo de procesamiento a menudo sólo se puede determinar de forma experimental. En las adaptaciones de software, la transferencia a un tipo de microcontrolador con otra velocidad o que incluso utiliza otras optimizaciones del compilador, deberá realizarse de nuevo el proceso para pasar a una temporización correcta. Esto es entretenido y sensible a errores.

La planificación de los módulos se produce porque el programador reparte la capacidad del procesador entre los módulos adaptando explícitamente el código. Este es un modelo estático, una vez que ha pasado por el microcontrolador, fija el comportamiento de temporización y ya no se puede modificar dinámicamente. Cuando el planificador llama un módulo, sólo podrá proseguir con su trabajo una vez que el módulo haya terminado. Mientras esté activo el módulo llamado, el planificador no podrá intervenir. Por eso se denomina planificador cooperativo.

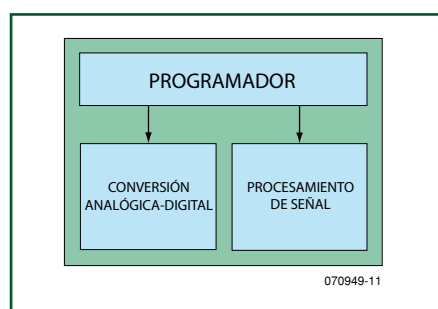


Figura 1. Estructura de software general.

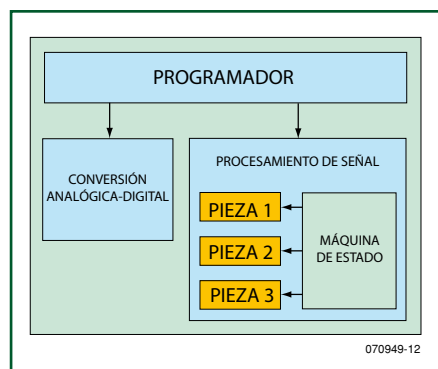
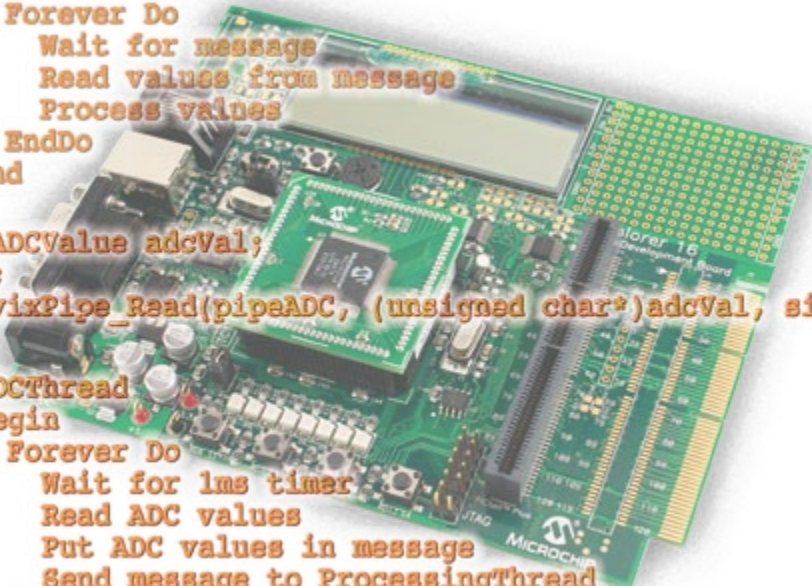


Figura 2. Estructura de software compleja.

real



```
ProcessingThread
Begin
  Forever Do
    Wait for message
    Read values from message
    Process values
  EndDo
End

tADCValue adcVal;
...;
avixPipe_Read(pipeADC, (unsigned char*)adcVal, sizeof(adcVal));

ADCThread
Begin
  Forever Do
    Wait for 1ms timer
    Read ADC values
    Put ADC values in message
    Send message to ProcessingThread
  EndDo
End
```

Un sistema operativo en tiempo real

Estos problemas pueden solucionarse con el uso de un llamado planificador "preemptivo", también denominado RTOS. Con un RTOS la estructura del software queda más clara y la temporización es determinista. Esto es posible porque un RTOS contrariamente al planificador cooperativo sí que puede intervenir en un módulo en curso. La estructura de un módulo no tiene que adaptarse para lograr que el RTOS actúe con la suficientemente frecuencia. Cuando se producen eventos en el sistema ante los que debe reaccionar el RTOS, entonces éste interrumpe el módulo activo para determinar si otro módulo tiene preferencia. Con esto, obtenemos una ventaja importante al utilizar un RTOS. Cada módulo se escribe como un programa lineal sin tener que dividirlo en múltiples fragmentos para la planificación. Los módulos que funcionan bajo control de un RTOS se denominan "hilos". Cada "hilo" se codifica como un programa lineal y en el código de programa no se puede ver donde lo interrumpirá eventualmente el RTOS. Con el uso de un RTOS se da prioridad a los "hilos" y sobre esta base, el RTOS determina si debe activarse el "hilo". Naturalmente, un RTOS ofrece también facilidades para que los "hilos" puedan intercambiarse

```
tADCValue adcVal;
...;
avix"condcto" Read(pipeADC, (unsigned char*)adcVal,
sizeof(adcVal));
```

Figura 3. Sistema básico que utiliza un RTOS.

información mutuamente (como mensajes) y facilidades para trabajar con tiempo (como los temporizadores). Mediante el uso de un RTOS el sistema de la **figura 1** tendría el aspecto en pseudocódigo que se muestra en la **figura 3**.

Aquí tenemos que comentar un aspecto importante del funcionamiento con un RTOS. Ambos "hilos" esperan un evento. El "hilo ADC" espera un temporizador y el "hilo de porceso" espera un mensaje. El RTOS tiene estas funciones de espera, de manera que el RTOS sabe qué espera cada "hilo". El RTOS lo activará en el momento en que se produzca el evento deseado y el "hilo" correspondiente tenga la mayor prioridad. Las funciones de espera no están activas. La espera no se produce porque por debajo corre un bucle hasta que se produce el evento. La espera por medio de un RTOS significa que el "hilo" también está realmente desconectado, y no consume ciclos de CPU. Durante este tiempo el RTOS se encargará de que los demás "hilos" estén activos.

¿Cómo funciona realmente? El estado en que se encuentra un programa, viene determinado por el contenido actual de los registros del procesador. Al ejecutar un código, cambia constantemente el contenido de estos registros. Cuando se produce un evento que lleva a la activación de otro "hilo", el RTOS transfiere el contenido de todos los registros a una parte de la memoria reservada para ello para cada "hilo". Con ello queda asegurado, en principio, el estado del programa en curso.

Téngase en cuenta que uno de los registros es un contador de programa. Es el registro en el que se guarda hasta donde ha avanzado la ejecución del programa. Este registro también se almacena, de manera que determina implícitamente hasta donde ha avanzado la ejecución en el "hilo" en el momento de la interrupción.

Luego, el RTOS cargará los registros del procesador con el contenido del "hilo" que va a activar. El RTOS es quien determina de qué "hilo" se trata, en función de las prioridades de los "hilos".

el programa en curso también puede interrumpirse con un ciclo fijo para la lectura de valores analógicos. Sin embargo, hay varios inconvenientes asociados a ello. Los controladores de 16 bits de Microchip funcionan con una estructura de prioridad de interrupciones. Durante la gestión de una interrupción se bloquearán las interrupciones con una prioridad inferior. Como sin RTOS el manejador de interrupciones debe hacer más y por lo tanto dura más, las interrupciones con una prioridad inferior estarán bloqueadas más tiempo, con lo cual aumentará el riesgo de perder una interrupción.

En segundo lugar, los valores analógicos del manejador de interrupciones se ponen a disposición del manejador de procesamiento. Sin RTOS, se trata de una cuestión compleja. Debe evitarse que la rutina de procesamiento lea justo en ese momento las ubicaciones de memoria que escribe el manejador de interrupciones, de manera que no se usen valores erróneos. Para ello debe desarrollarse un mecanismo complejo o – como ocurre a menudo – la rutina de procesamiento desconectará temporalmente las interrupciones de manera que utilice con seguridad valores correctos. No obstante, esto aumenta el riesgo de perder interrupciones.

Un RTOS utiliza internamente una escala ampliada de interrupciones y ofrece también facilidades para transferir los valores de los manejadores de interrupciones de forma segura hacia el "hilo" responsable de la siguiente gestión.

Las interrupciones forman la base de los eventos antes mencionados. Un ejemplo de ello son los temporizadores. Prácticamente todos los RTOS tienen temporizadores, de manera que los "hilos" puedan esperar un tiempo determinado. Estos temporizadores de RTOS son temporizadores de software. La base de los temporizadores está formada por uno de los temporizadores de hardware del controlador. Este temporizador de hardware generará interrupciones con una determinada periodicidad. En cada interrupción, el RTOS actualizará los temporizadores de software activos y cuando termine activará el mecanismo que determina si debe activarse otro "hilo". De esta forma una interrupción de hardware lleva el recorrido del temporizador de hardware a través de la actualización

de los temporizadores de software hasta el evento del que hemos hablado antes. También se utilizan interrupciones para el control de los periféricos que también pueden llevar a un evento antes de activar un "hilo". Para ello, el RTOS dispone de mecanismos que permiten programar a un nivel bastante elevado y así se encarga de la complejidad de la gestión de las interrupciones. Esto se ilustra en un ejemplo de esta gestión, basada en un mecanismo de "conducto", que también existe en AVIX.

Estos "conducto"s son buffers FIFO con los que los "hilos" pueden intercambiar información entre ellos, o con los manejadores de interrupciones. Un "hilo" que lee un "conducto", se pondrá en espera cuando la cantidad de información deseada (todavía) no esté presente. En la escritura, el "hilo" se pondrá en espera cuando la cantidad necesaria de espacio vacío (todavía) no esté presente. El RTOS desactivará entonces el "hilo" sin que este consuma la capacidad de procesador.

Ahora vamos a ver cómo se pueden leer con un "conducto" los datos de un ADC. La lectura se realiza desde un "hilo" y el código necesario para ello, se muestra en la **figura 5**.

El "hilo" contiene únicamente una operación de lectura y se bloquea cuando los datos deseados no están presentes. ¿Qué ocurre bajo el agua? Se muestra en la **figura 6**.

El transcurso del software es el siguiente:

1. El "hilo" lee del "conducto". Como está vacío, el "hilo" se pone en espera.
2. Desde el mecanismo de "conducto" se llama una función de DRIVER.
3. El DRIVER activa el ADC. El ADC funciona autónomamente y genera una interrupción cuando termina.
4. Esta interrupción activará el manejador de interrupciones perteneciente al ADC. Este lee el resultado de la conversación y:
5. La escribe en el "conducto". Con ello se dispone de los datos que espera el "hilo" y el RTOS se encarga de que:
6. Los datos se transfieran seguidamente al "hilo" que pueda ponerse a trabajar.

Este mecanismo está presente en el programa demo que el autor ha realizado para este artículo.

Un sistema basado en AVIX

Uno de los programas demo de la serie de artículos Explorer 16 (principios de 2007) era un termómetro parlante.

Este programa utiliza una gran cantidad de periféricos que colaboran conjuntamente. La estructura de este programa es poco clara y es difícil adaptarla o ampliarla. Compilar con la mayor optimización provoca que deje de funcionar, porque entonces la temporización es totalmente distinta.

Este programa se ha rescrito para poder trabajar con AVIX, y así poder ilustrar las ventajas de un RTOS. Junto con una versión demo de AVIX, este programa se puede descargar de **www.avix-rt.com**. Tras la instalación se puede montar y programar el programa desde MPLAB en un controlador 24FJ128GA010 en la placa de desarrollo Explorer-16. Para ello se utiliza una tarjeta de sonido Microchip PICtail Plus (AC164125) Se puede ver la estructura en la **figura 7**.

En la demo hay una descripción detallada en inglés de las distintas partes que componen el programa.

(070949)

Enlaces en internet:

www.avix-rt.com

El autor



Leon van Snippenberg ha escrito su propio RTOS, que lleva el nombre de AVIX. Este RTOS se desarrolló especialmente para las familias de microcontroladores PIC24 y dsPIC de Microchip. Su empresa, AVIX-RT, está especializada en el suministro de productos y servicios para el desarrollo de sistemas integrados.