Método para el Diseño e implementación de software para sistemas embebidos orientado a la eficiencia en el uso de la CPU utilizando la técnica Polling Loop. Caso de estudio: Receptor Xmodem

Freddy Alexander Díaz González
Escuela de ciencias exactas e ingenierias
Universidad Sergio Arboleda, Bogotá, Colombia, freddy.diaz@correo.usa.edu.co
María Paula Gómez Prieto
Universidad Sergio Arboleda, Bogotá, Colombia, mariapau.gomez@correo.usa.edu.co
Jairo Andrés Quiñones Buitrago
Universidad Sergio Arboleda, Bogotá, Colombia, jairo.quinones@correo.usa.edu.co

Abstract

The embedded systems software has increase its complexity, therefor is needed to find a method with wich to carry out the desing and implementation of the software for embedded systems taking its eficiency as a important fact when implementing critical systems.To do this without using an operating system, it is used Polling Loop technic consisting on constantly consulting the task if needs the CPU or not, using tools set by SysML y UML RT for the design of sequence diagrams, state machines and composite structure diagram, this last one is fundamental to identify actors in the RT sequence diagram, and implementing it within C language and using Switch control structure.

Keywords: Embedded systems, critical systems, state machines, modeling formalisms, embedded software.

Resumen

Con el avance de los sistemas embebidos tambien ha aumentado la complejidad del software para estos dispositivos, por lo tanto se ve la necesidad de formular un método con el cual se pueda llevar a cabo el diseño y la implementación de software teniendo como base el uso eficiente de la CPU y considerando éste como un factor muy importante al momento de implementar un sistema crítico. Para lograr esto sin el uso de un sistema operativo, se usó la tecnica *Polling Loop,* usando herramientas de modelamiento planteadas por SysML y UML RT para el diseño del software con características de tiempo real. En este artículo se presenta el modelo de diseño e implementación de software para sistemas embebidos y un ejemplo de uso del modelo, orientado a la implementación en lenguaje C.

Palabras claves: Sistemas embebidos, sistemas críticos, máquinas de estado, formalismos de modelamiento, software embebido.

Introducción

Con la mejora de los microcontroladores, ()DSPs y demás sistemas embebidos, se ha aumentado la cantidad de funcionalidades que estos dispositivos deben soportar, como lo son la administración de pantallas táctiles y gráficas, control de sensores MEMS() y módulos de comunicación como USB y Ethernet. Por lo cual, el software para los sistemas embebidos también ha incrementado su complejidad. Adicionalmente, este tipo de dispositivos generalmente son usados para la implementación de sistemas críticos en áreas de automatización, domótica y

telecomunicaciones, las cuales requieren de aplicaciones con atributos de calidad como eficiencia en el uso de los recursos, seguridad y bajos tiempos de respuesta.
Con el aumento de la complejidad del software para los sistemas embebidos, también ha aumentado la complejidad en las actividades de diseño, modelamiento e implementación de este. No obstante, la mayoría de las metodologías para desarrollo de software están orientadas a aplicaciones que son implementadas sobre máquinas con sistemas operativos como Windows o Linux, las cuales no están enfocadas a sistemas críticos, ni contemplan actividades de modelamiento orientadas a la administración de los recursos del sistema a bajo nivel, como los requeridos por los sistemas críticos, que pueden exigir tiempos de latencia por debajo de los milisegundos. Varios autores proponen el modelamiento del software para sistemas embebidos basados en máquinas de estado, pero no son explícitos en cómo realizar la implementación para obtener una buena administración de la CPU y demás recursos.

En este documento se propone un método para el desarrollo de software para sistemas embebidos de sistemas críticos, que atienda las restricciones de seguridad, bajos tiempos de respuesta y eficiencia en el uso de la CPU; utilizando formalismos de modelamiento de máquinas de estado y autómatas temporizados, que orienta la implementación del software, sin la necesidad de emplear sistemas operativos para administra los recursos.
Este documento está estructurado en siete partes. En la primera se presenta la introducción del documento. En la segunda se realiza una revisión literaria sobre las técnicas de administración de la CPU de modelamiento para sistemas embebidos. En la tercera se plantea la estructura en bloques funcionales para el software de sistemas embebidos, diferenciando entre los módulos de recursos y de procesos. En la cuarta parte se presenta el método para el modelamiento, diseño e implementación de software para sistemas embebidos sin uso de sistemas operativos. En la quinta parte se presenta un caso de estudio en el que se aplica el método planteado. En sexta parte se presentan los resultados de la implementación del caso de estudio. Finalmente, en la séptima parte se presentan las conclusiones y trabajo a futuro.

Administración y uso eficiente de la CPU

Generalmente las aplicaciones que son implementadas sobre sistemas embebidos son de uso específico, no escalables por los usuarios y se le pueden definir detalladamente sus entradas y salidas, por lo cual las tareas que se implementen sobre un sistema embebido no deben competir por la CPU con otras aplicaciones, como es el caso de las aplicaciones implementadas sobre maquinas tipo desktop o similares que soportan aplicaciones de software comercial.

Para la administración de la CPU en sistemas embebidos también existen herramientas como los sistemas operativos y en el caso de aplicaciones para sistemas críticos que deben ser determinísticos y predecibles, se han desarrollado los sistemas operativos en tiempo real, sin embargo, si es baja la cantidad de tareas que se requieren para implementar la aplicación, no se hace necesario el uso de un sistema operativo. El determinar cuando la cantidad de tareas, su complejidad o recursos de desarrollo requieren el uso de un sistema operativo, queda a criterio del desarrollador. Este documento se centra en el desarrollo del software para sistemas embebidos sin el uso de sistemas operativos.

La técnica sobre la cual se desarrolla el software embebido sin sistemas operativos se basa en consultarle constantemente a la tarea, si requiere o no de la CPU, a esto se le denomina Polling (encuesta). El Polling se implementa con un loop infinito (while(TRUE) o for(;;)) dentro de cual se hace el llamado los métodos o subrutinas que constituyen las tareas, con un orden específico que no puede ser alterado en tiempo de ejecución, en este método todas las tareas tiene la misma prioridad y se garantiza que a todas se les asignara tiempo de CPU para su ejecución, siempre y cuando ninguna tarea se apodere de la CPU indefinidamente.

Para implementar una tarea que no se apodere de la CPU indefinidamente, esta debe tener un flujo continuo y determinístico en su ejecución, sin realizar ningún tipo de espera por eventos. En los sistemas embebidos las tareas más sencillas son activadas por un solo evento, el cual dispara o no su ejecución.

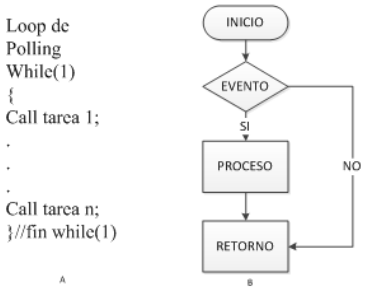

Figura 1. Llamada a la tarea1 en el Polling Loop

En la figura 1-A, muestra como en el Polling Loop es llamada la tarea 1, la cual es activada por un evento, si el evento es verdadero se ejecuta la tarea y luego retorna, si no la tarea retorna inmediatamente, como se muestra en la figura 1-B. El evento puede ser la bandera que indica el fin de una conversión análogo-digital, la bandera de desborde de un timer o la llegada de un mensaje a una cola. Para la mayoría de los casos, la encuesta dará como negativo el evento, lo cual puede hacer suponer una pérdida de tiempo en la consulta a las tareas en el Polling Loop, sin embargo este tiempo es despreciable si se compara con el tiempo que toma la ejecución del scheduler (planificador) en un sistema operativo.

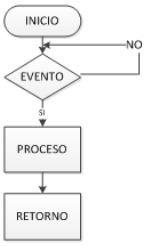

Figura 2. Caso de mal uso de la CPU

Para evitar que una tarea haga mal uso de la CPU, se debe evitar el caso mostrado en la Figura 2, en el cual la tarea se queda esperando por el evento que dispara su ejecución, ese tiempo de espera puede ser aprovechado para ejecutar otra tarea.

En la técnica de Polling tiempo de ejecución de una tarea corresponde al peor caso de ejecución que deba realizar la tarea una vez que se ha cumplido la condición para su ejecución. En la figura 3 el recorrido identificado por la línea “A” es el peor caso de la tarea.

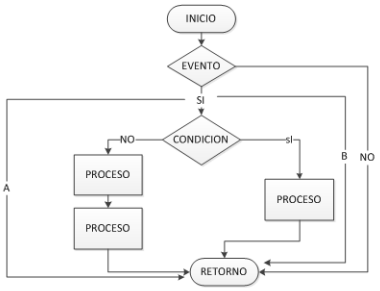

Figura 3. Peor caso de ejecución de una tarea

Para verificar que el sistema cumpla con los tiempos de latencia exigidos por la aplicación, se suman los tiempos de ejecución de todas las tareas que se encuentren en el Polling Loop, y este resultado debe ser menor al tiempo más pequeño de latencia exigido.

Son muy pocos los sistemas que exigen tiempos de latencia menores a milisegundos, por ejemplo los cambios de temperatura en una incubadora no tiene tasa de variación inferiores a 1 segundo o no tiene sentido que la frecuencia de muestreo de un teclado sea superior a 100 Hz ya que una persona promedio, no digita con frecuencias superiores a 50Hz, por lo cual la latencia del teclado seria de 10 milisegundos.

En la tabla 1 se muestran las tareas que componen un Polling Loop con sus respectivos tiempos de ejecución y sus ejecuciones en un segundo. La cantidad de veces que se ejecuta la tarea en 1 segundo está condicionada al evento que la dispara, la frecuencia del evento da la latencia de la tarea, para el caso de eventos periódicos. En general los sistemas son periódicos, más adelante se discute el caso de los no periódicos.

Tabla 1. Tiempos de ejecución de cada tarea

Tarea	Tiempo de ejecución de la tarea en micro segundos	Cantidad de ejecuciones por segundo	Latencia en milisegundos
Tarea 1	800	10	100
Tarea 2	730	12	83.33
Tarea 3	100	50	20
Tarea 4	600	20	50
Tarea 5	400	4	250

No es de extrañar que la ejecución de una tarea este dada en microsegundos, debido a las velocidades de los sistemas microprocesados actuales, que como el caso de los ARM Cortex M3, el cual es la gama baja de los Cortex, pueden alcanzar hasta 60 DMIPS (Joseph Yiu, 2007) y siguiendo las buenas prácticas de Ingeniería de software, una tarea solo debe cumplir con una sola funcionalidad para más funcionalidades se implementan más tareas.

De la tabla 1 el tiempo total de ejecución de las tareas suma 2.63 milisegundos, el cual es inferior al menor de los tiempos de latencia que es de 20 milisegundos. Aunque no siempre todas las tareas se ejecutan en el mismo ciclo del Polling Loop, el cálculo para la respuesta a la latencia, también se realiza contemplando el peor caso del Polling Loop, el cual es el caso en que todas las tareas se ejecutan en el mismo ciclo.

Una de las principales ventajas de utilizar la técnica de Polling Loop es que la implementación queda totalmente predecible, esto asegura que se pueden cumplir con los tiempos de respuesta del sistema y que la implementación corresponda con los modelos comportamentales realizados.

El porcentaje de uso de la CPU para las tareas se calcula por el tiempo total que la tarea usa el recurso de la CPU en un segundo. (S., 2009)

Ecuacion1: porcentaje de uso de la CPU

$$timeCPUt_n = timefec * CVt_n$$

Tabla 2. Resultados de uso de la CPU

Tarea	Tiempo de ejecución de la tarea (micro segundos)	Cantidad de ejecuciones de la tarea en un segundo	Suma de los tiempos de ejecución de la tarea en un segundo (Segundos)	%de uso de la CPU de la tarea en un segundo
Tarea 1	800	10	0,008	0,8
Tarea 2	730	12	0,00876	0,876
Tarea 3	100	50	0,005	0,5
Tarea 4	600	20	0,012	1,2
Tarea 5	400	4	0,0016	0,16

La tabla 2 muestra los resultados de uso de la CPU para las tareas de la tabla 1.

La suma de los tiempos de ejecución de las 5 tareas es de 35.36 milisegundos, lo que representa el 3,536% del tiempo de CPU en un segundo, esto indica que el restante 96,464% del tiempo la CPU está en espera por un evento que dispare alguna de las tareas para ejecutarla.

Para el caso en que el tiempo de latencia de alguna de las tareas sea inferior a la suma de los tiempos de ejecución de todas las tareas, se puede asignar más tiempo de CPU a la tarea, llamándola más de una vez en el Polling Loop. Por ejemplo de la tabla 1, si el tiempo de latencia de la tarea 3 no fuera de 20 milisegundos sino de 2 milisegundos, la latencia sería inferior a la suma de los tiempos de ejecución de las 5 tareas (2.63 milisegundos), para satisfacer el nuevo tiempo de respuesta de la tarea 3 no se requiere rediseñar las tareas para acortar su tiempo de ejecución, lo que se puede hacer es llamar la tarea 3 dos veces en el Polling Loop como se muestra en la figura.

```
Loop de
Polling
While(1)
{
Call tarea 1;
Call tarea 2;
Call tarea 3;
Call tarea 4;
Call tarea 5;
Call tarea 3;
}//fin while(1)
```

Figura 4. Caso en el que la tarea 3 tiene un tiempo de latencia mayor

Con la nueva distribución del Polling Loop los tiempos de ejecución entre la tarea 3 son de 1.63 y 1.1 milisegundos, los cuales son inferiores a la latencia de la tarea que es de 2 milisegundos, garantizando que la aplicación responda a los tiempos exigidos por el sistema.

Responder a los tiempos de latencia, cuando los eventos que disparan las tareas son periódicos, puede ser sencillo de calcular y modelar, sin embargo, cuando los eventos no son periódicos o cuando la latencia es muy pequeña, las tareas pueden ser disparadas por interrupciones, estas no se ubican dentro del Polling Loop y a la suma del peor caso del tiempo del Polling Loop se le debe adicionar el tiempo de ejecución de la interrupción.

Cuando los eventos que disparan las tareas no son periódicos, es recomendable llamar la tarea dentro del Polling Loop, para los casos en que el porcentaje de uso de la CPU lo permita. Para los eventos no periódicos con latencias muy bajas, lo recomendable es activar la tarea por interrupción y a la suma del peor caso del tiempo del Polling Loop se le debe adicionar el tiempo de ejecución de la interrupción.

Estructura en bloques funcionales para el software de sistemas embebidos

En el proceso de desarrollo de software para sistemas embebidos, se debe conocer detalladamente el comportamiento del sistema que va a interactuar con la aplicación, para ellos se pueden seguir las técnicas y herramientas planteadas por SysML y UML RT (Selic & Rumbaugh, 1998), en las cuales se contemplan esquemas de diagramas de comunicación con tiempos de respuesta, diagramas de secuencia y modelos dinámicos de las plantas que componen el sistema entre otros. Una vez identificadas las características del sistema, se procede a definir la estructura de los bloques funcionales del software, para esto se propone clasificar los bloques en tres tipos, el primero denominado bloque de tarea, el segundo bloque de recurso software y el tercero bloque recurso hardware. Los bloques de tareas corresponden a los módulos software que son llamados desde el Polling Loop. Los bloques de recursos software son ejecutados solo cuando un bloque de tarea lo decide. Los bloques recursos hardware representan los módulos hardware que se encuentren dentro del sistema embebido sobre el cual se está implementado la aplicación, como lo son los Timer, USART y ADC, sin incluir los componentes del procesador del sistema .

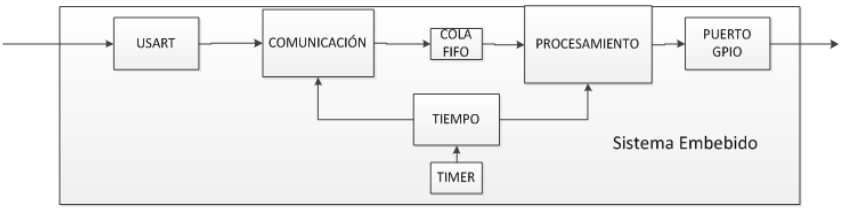

Figura 5. Diagrama de estructura compuesta

En la figura 5 se muestra el diagrama de estructura compuesta de una aplicación sencilla en un sistema embebido, en la cual hay 3 recursos hardware, 1 recursos software y 3 tareas. La tarea 1 (comunicaciones) consulta periódicamente el recurso HW1 (USART), para leer los datos y almacenarlos en el recurso SW1, el cual es una cola de datos tipo FIFO, que puede almacenar los datos en una parte de la memoria RAM, a la que solo este recurso tiene acceso; La tarea 2 periódicamente saca los datos de la cola FIFO, los procesa y los muestra en el recurso HW2, que es un puerto GPIO. La tarea 3 es el módulo de tiempo que envía eventos de disparo a las tareas 1 y 2, según la frecuencia que ellas requieran, estos eventos son banderas software que activa la tarea 3 y limpian las tareas 1 y 2 respectivamente; Finalmente, la señal de sincronía de la aplicación, la da el recurso HW3, que es un Timer consultado por la tarea 3.

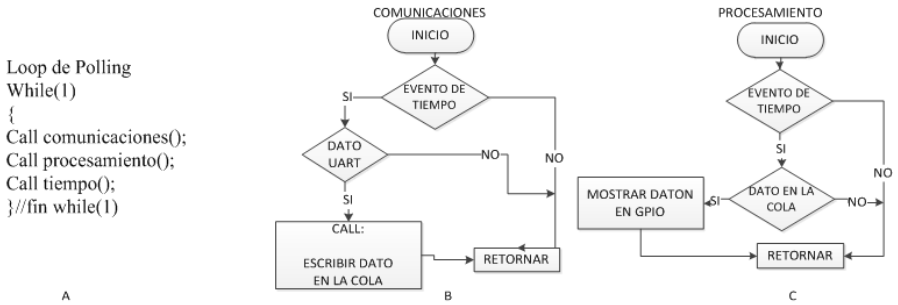

Figura 6. Polling Loop para el diagrama de estructura compuesta

En la figura 6-a se muestra el Polling Loop para el diagrama de estructura compuesta de la figura 5, es importante resaltar que solo los bloques que son tareas, son llamados desde el Polling Loop, los bloques de recursos de software no son llamados. Las tareas 1 y 2 se ejecutan periódicamente, sin embargo, la ejecución real se da por un segundo evento, que es generado por el recuso HW1 en el caso de la tarea 1 y por el recurso SW1 para la tarea 2. Figura 6-b y 6-c respectivamente.

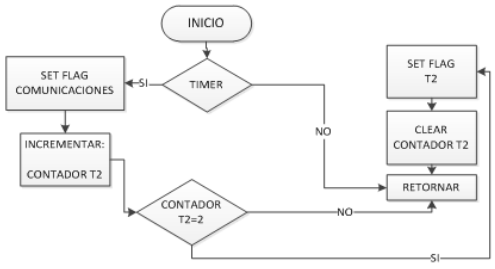

Figura 7. Flujo de ejecución de la tarea 3

En la figura 7 se muestra el flujo de ejecución de la tarea 3, para este caso esta se implementa solo con el recurso HW3 debido a que los eventos de disparo de las tareas 1 y 2 se pueden generar con la misma base de tiempo, para los casos en que los eventos no se puedan generar con una sola base de tiempo, se debe utilizar otro módulo Timer.

Interfaces de comunicación entre los bloques software del sistema.

Las interfaces de comunicación entre los bloques pueden variar dependiendo de las necesidades propias de la aplicación, por ejemplo para el diagrama de la figura 5, la comunicación entre la tarea 1 y el recurso SW1, está definida según el tipo de datos que se almacenen en la cola FIFO, adicionando los campos necesarios para indicar si se pudo o no almacenar el dato y demás información requerida; lo mejor para este caso es encapsular todas las variables en una estructura de datos y pasarla al método (función) con el que se implementa el SW1, bien sea paso de datos por referencia o por copia (O'REILLY, 2009), según sean los recursos de RAM del hardware.

Al definir las interfaces de comunicación, es importante mantener la independencia de las variables que pueden afectar el funcionamiento de un bloque, por ejemplo, no es necesario que los demás bloques puedan acceder a las variables con las que se administra la secuencia de entrada y salida de los datos en el recurso SW1 (cola de datos FIFO), porque podrían modificar la secuencia de administración de los datos.

Tareas que deben realizar esperas

Generalmente el intercambio de información entre computadores de escritorio y sistemas embebidos, se orienta a periféricos seriales como RS232, RS485 y USB, para la capa física y protocolos industriales, como lo son ModBus, Xmoden y CAN entre otros, para la capa de enlace (Gajski, 2010);debido a su naturaleza serial, el paquete que espera la capa de enlace (protocolo) requiere de la llegada de un numero finito de datos en la capa de física (Gajski, 2010), para lo cual la tarea que debe cumplir con la función comunicación en la aplicación del sistema embebido, debe gestionar el bloque hardware serial y las características propias del protocolo de la capa de enlace, esta gestión implica que la tarea pueda ejecutarse parcialmente y quedar en espera por diferentes tipos de eventos. Para el diseño, modelamiento e implementación de este tipo de tareas se recomienda utilizar las máquinas de estado, que pueden ser implementadas en leguaje C mediante la estructura de control Switch Case (Pont, 2002),en la cual cada *case* representa un estado de la máquina y la expresión que se evalúa, es gobernada por la variable de estado.

En el campo de electrónica digital se manejan dos tipos de máquinas de estado Mealy y Moore. Básicamente los dos tipos son autómatas finitos, sin embargo, la diferencia entre ellas radica en que el tipo Mealy las acciones se realizan en las transiciones entre estados y en la Moore las acciones de realizan dentro de los estados (Gajski, 2010),las dos comparten en que pueden existir múltiples transiciones a estados con varias condiciones.

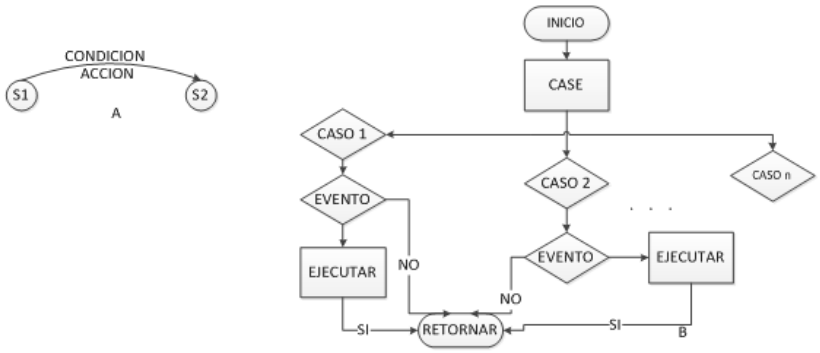

Figura 8. Máquina de estado Mealy

En la figura 8-a se presentan un caso modelado en la máquina de estado Mealy y en la figura 8-b el diagrama de flujo que guía la implementación de la máquina para software embebido. De igual manera se presenta el mismo caso en la figura 9, pero utilizando la máquina de estados Moore.

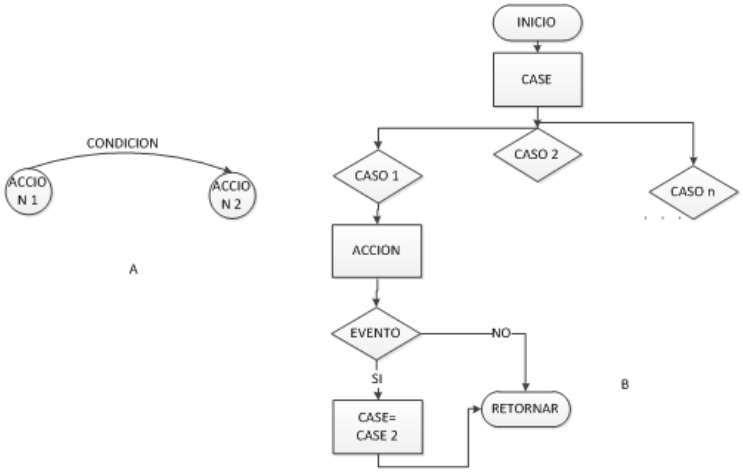

Figura 9. La máquina de estados Moore

Debido a que los sistemas embebidos generalmente son empleados en sistemas críticos, el tiempo consumido por cada ciclo de la maquina infiere directamente en los tiempos de respuesta del sistema, por lo cual es indispensable que la implementación de las tareas se realice estrictamente con la cantidad de instrucciones necesarias y suficientes (Kamal Hyder, 2005),por lo cual en el modelo presentado en la figura 8, se hace explicito que la acción del estado solo puede ser realizada si hay un evento que la dispare, por lo contrario, el modelo presentado en la figura 9, la maquina siempre ejecuta una acción, así está ya haya sido realizada en una ejecución anterior y el evento solo es evaluado para realizar las transiciones entre estados; por estas razones se propone que Para diseño, modelamiento e implementación de las tareas en sistemas embebidos se utilice las máquinas de estado tipo Mealy (Sharan, 2012) de la figura 8.

Casos de máquinas de estado

Dependiendo la funcionalidad de la tarea que desea desarrollar con máquinas de estado, se pueden presentar varios casos de diseño, a continuación se presentan los caso básicos que guían el diseño de software para sistemas embebidos basado en máquinas de estado, que será implementado sin sistema operativo utilizando la técnica de Polling Loop.

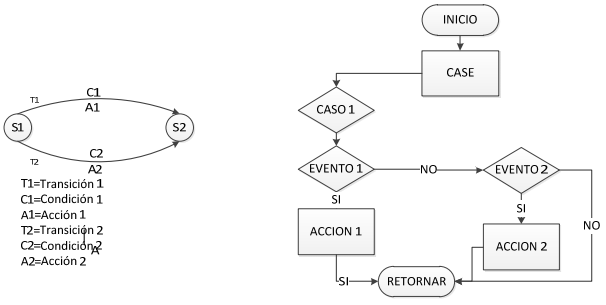

Figura 10. Modelo con dos transiciones

En la figura 10-a se presenta un modelo con dos estados, en los que las transiciones han sido etiquetadas con los índices T1 y T2, en el grafo en la parte superior de las transiciones se han etiquetado las condiciones para que determinan el cambio de estado (C1 y C2), finalmente las acciones se han etiquetado debajo de las transiciones (A1 y A2).

Las condiciones para la transición entre estados son el conjunto de eventos que disparan la ejecución de la tarea dependiendo en el estado que esta se encuentre, las condiciones pueden estar formadas por uno o más eventos de

diferentes tipos, por ejemplo la bandera hardware que indica la llegada de un dato a al periférico USART, puede ser una condición; del mismo modo una bandera software como la que comunica la tarea 3 con la tarea 1 de la figura 5. En el caso de un estado con opción de múltiples transiciones a otros estados (figura 10-a), la implementación debe anidar las condiciones para evaluarlas según la prioridad que el desarrollador le asigne a cada una de ellas, como se muestra en la figura 10-b.

Las acciones que se realizan cuando se dispara el evento en el estado, deben cumplir con el principio de buen uso de la CPU presentado en la figura 1, adicionando que, al terminar las instrucciones propias de la acción, se debe actualizar la variable de estado que permitirá que en la próxima entrada a la tarea, la máquina de estados evalué las condiciones del nuevo estado al que se dio la transición.

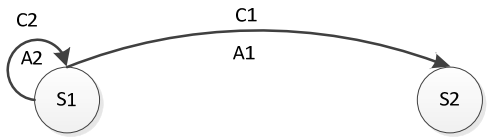

Figura 11. Ciclo de acciones, donde termina en el mismo estado

Existen casos en los que la tarea debe realizar un ciclo de acciones, como por ejemplo esperar la llegada de una cantidad finita de datos a la USART para completar una trama de información, para ello se plantea el esquema en el que la transición de la maquina termina en el mismo estado, como se muestra en la figura 11. En este ejemplo la acción es realizada pero no se actualiza la variable de estado. Adicionalmente con disparo de otro evento se puede ejecutar el cambio de estado.

Un ejemplo de diseño e implementación de una máquina de estados compleja, es el caso de la tarea de comunicaciones de la figura 5, este bloque representa el receptor para la descarga de información desde un computador, que utiliza un protocolo Xmodem. Para el diseño de una máquina de estados es necesario conocer el comportamiento del bloque con los demás elementos del sistema. Para lo cual se utiliza las técnicas de modelamiento UML y SysML (CoFluent Design, 2009) (J. P. Calvez,1993), para este caso, se propone la construcción de diagramas de secuencia que modelan el comportamiento de la tarea de comunicaciones con los demás bloques del sistema, para todos los casos que se puedan presentar en una descarga de datos que utilice el protocolo Xmodem.

El protocolo Xmodem está orientado byte, en el cual el transmisor fragmenta la información en paquetes de 128 bytes de 8 bits, que envía al receptor utilizando la trama base que se muestra en la figura 12, la cual contiene x campos

Inicio de trama	Numero paquete	Numero paquete complemento 1	datos	Checksum

Figura 12. Trama base

En el primer campo, está el inicio de trama, el cual contiene un carácter SOH que sirve como bandera señalizador de comienzo de trama (Start of Header) y el último campo *Checksum*, contiene un carácter EOT, que sirve como señalizador de final de trama, a diferencia del SOH.

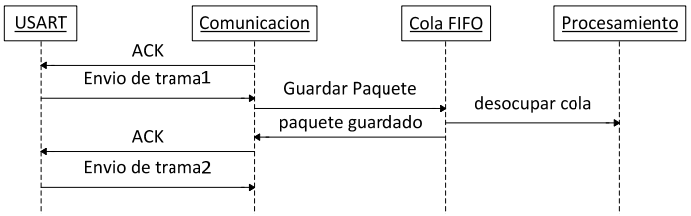

Figura 13. Caso de envió de trama sin error

En un caso de envió de trama en la cual no se presenta ningún error, como se observa en la figura 13, la tarea de comunicación envía un mensaje de reconocimiento (ACK) a la USART, ese mensaje indica que está listo para recibir la siguiente trama, la USART en el momento que recibe el mensaje responde enviando la trama, la tarea de comunicación recibe los datos y los almacena en la cola FIFO, paralelamente y como un proceso independiente, la tarea de procesamiento periódicamente saca los datos de la cola FIFO, para que termine el envió de la trama con éxito, la tarea de comunicación envía el ACK a la USART nuevamente, indicando que el paquete ha sido guardado satisfactoriamente y pueden continuar con el envió de la siguiente trama.

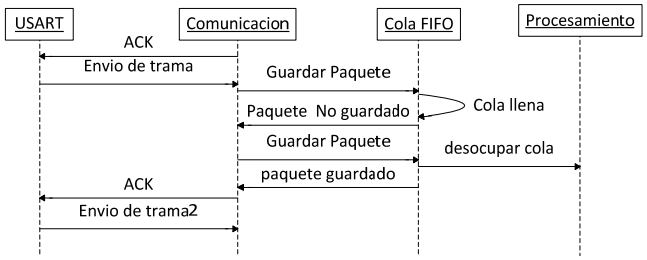

Figura 14.caso de envió, en el que la cola está llena.

En dado caso que la cola FIFO no tenga espacio suficiente, a la tarea de comunicación le es imposible guardar el paquete en la cola, sin embargo vuelve a internarlo, como se muestra en la figura 14, si en ese tiempo el bloque de procesamiento ya ha desocupado la cola, la tarea de comunicación guarda el paquete, y envía el ACK.

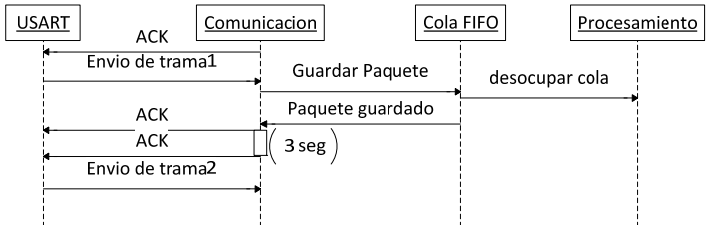

Figura 15. Caso en el que no se envía ninguna trama

Como muestra la figura 15, cuando la tarea de comunicación envía el ACK a la USART, se espera un determinado tiempo para una respuesta, si en ese tiempo no envía respuesta alguna, la tarea de comunicación envía de nuevo el ACK.

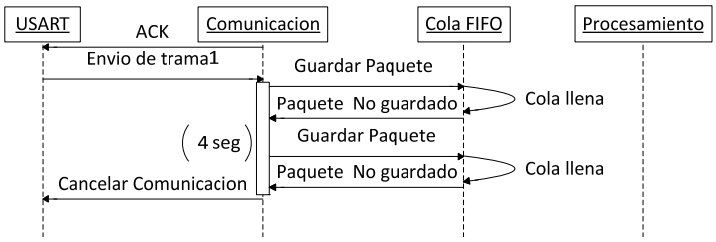

Figura 16. Caso error CAN (se cancela la comunicación)

En el momento en que la cola FIFO se encuentra llena, y la tarea de comunicación no pueda guardar el paquete en ella, se espera un determinado tiempo para que el bloque de procesamiento desocupe la cola, si en ese tiempo la cola continua llena, se cancela la comunicación (CAN)

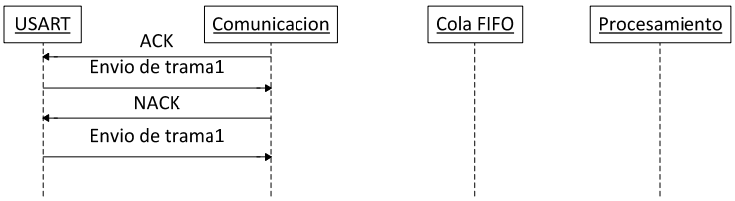

Figura 17. Caso de error en la trama

En la figura 17 se muestra una relación únicamente entre USART y comunicación, esto se debe a que al momento de llegar la trama a la tarea de comunicación, se ha encontrado algún error en esta, de ser así, la tarea de comunicación no pasa a guardar el paquete en la cola FIFO, sino que en su lugar, envía un NACK.

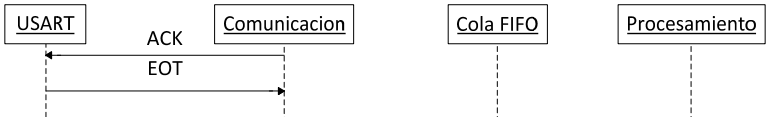

Figura 18. Carácter de fin de transmisión (EOT)

Si en la transmisión se da el caso en el que todas las tramas están completas, la USART envía a la tarea de comunicación el carácter de fin de transmisión (EOT), para indicar que todas las tramas han sido enviadas.
En la tabla 3 se menciona y explica los diferentes eventos que son nombrados en los diferentes diagramas de secuencia.

Tabla 3. Descripción de los eventos

Evento	Descripción
ACK	Es un mensaje que se envía para confirmar que una trama, ha llegado correctamente.
Envió de trama	Se envía la Trama, cada trama envía: 1. Inicio de trama (Un primer byte) 2. Otro byte con la posición del paquete 3. otro byte más con la posición pero en complemento a 1 4. Datos (128 bytes) 5. Un byte de Checksum
Guardar Paquete	Envía el paquete a la cola FIFO para ser guardada.
Paquete guardado	Es el mensaje que llega a “comunicación” indicando que el paquete ha sido guardado con éxito
Cola llena	Mensaje que indica, que la cola FIFO está llena y no puede guardar paquetes.
Paquete no guardado	Es el mensaje que llega a “comunicación” indicando que el paquete no ha sido guardado.
NACK	Es el mensaje que llega a la USART, e indica que se encontró algún error en el paquete, y se debe reenviar la trama

Fuente: Elaboración propia

Como ejemplo de aplicación de uso de máquinas de estado se procede a la implementación del protocolo de comunicaciones “Xmódem”.

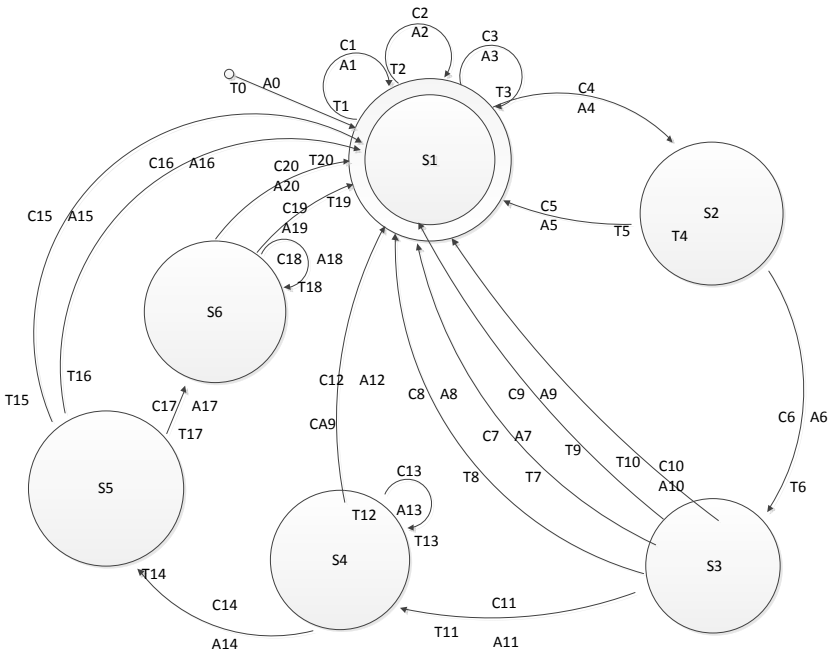

Figura 19. Máquina de estados Xmodem

Como se muestra en la figura 19, la máquina de estados del receptor del protocolo Xmodem está compuesta por seis estados diferentes, los primeros cinco son cado uno de los campos que componen la trama, por ejemplo S1 (estado uno) es el inicio de la trama, S2 corresponde al número de paquete, S3 el número de paquete en complemento uno, S4 los datos y S5 Checksum, terminando con este, los diferentes campos que componen la trama. Sin embrago se ve necesario el uso de otro estado, S6, el cual corresponde a la cola FIFO.

En la tabla 4, se puede observar dependiendo del número de transición, las condiciones y acciones que se presentan en cada una, dando así una descripción más clara de la máquina de estados mostrada en la figura 19.

Tabla 4. Descripción de la máquina de estados

Transición	Condición	Acción
T0	Condiciones iniciales	Enviar ACK, cambia estado a S1, se resetea el tiempo, el número de la trama es igual a 1, el contador de datos es igual a 0.
T1	Dato diferente a SOH y a EOT	Se resetea el tiempo.
T2	Se cumple el tiempo y no llegan datos.	Enviar ACK, se resetea el tiempo.
T3	Dato es igual a EOH	Enviar ACK, cambia estado a S1, se resetea el tiempo, el número de la trama es igual a 1, el contador de datos es igual a 0.
T4	Dato es igual a SOH	Cambia estado a S2, se resetea el tiempo.
T5	Se cumple el tiempo y no llegan datos.	Enviar NACK, cambia estado a S1, se resetea el tiempo.
T6	Hay dato	cambia estado a S3, se resetea el tiempo y el número de paquete es igual a dato
T7	El complemento del número de paquete es diferente a dato	Enviar NACK, cambia estado a S1, se resetea el tiempo.
T8	Se cumple el tiempo y no llegan datos.	Enviar NACK, cambia estado a S1, se resetea el tiempo.
T9	El número de paquete es menor al de la trama, hay datos, el complemento del número de paquete es igual al dato.	Enviar ACK, se resetea el tiempo, cambia el estado a S1.
T10	El número de paquete es mayor al de la trama, hay datos, el complemento del número de paquete es igual al dato.	Enviar CAN, cambia estado a S1, se resetea el tiempo, el número de la trama es igual a 1, el contador de datos es igual a 0.
T11	El número de paquete es igual al de la trama, hay datos, el complemento del número de paquete es igual al dato.	Cambia estado a S4, se resetea el tiempo.
T12	Se cumple el tiempo y no llegan datos.	Enviar NACK, cambia estado a S1, se resetea el tiempo, contador de datos igual a cero.
T13	Hay dato y contador de dato es menor a 128	Se resetea el tiempo, a contador de datos se le suma uno, se guarda el dato, y cambia el estado a S4.
T14	Hay dato, contador de datos es mayor o igual a 128	Se resetea el tiempo, se calcula checksum, y cambia el estado a S5.
T15	Se cumple el tiempo y no llegan datos.	Enviar NACK, cambia estado a S1, se resetea el tiempo, contador de datos es igual a cero.
T16	Hay dato, dato es diferente a checksum	Enviar NACK, cambia estado a S1, se resetea el tiempo, contador de datos es igual a cero.
T17	Hay dato, datos igual a checksum	Se resetea el tiempo, y el estado cambia a S6.
T18	No guarda paquete en la cola	Se mantiene el mismo estado.
T19	No guarda el paquete en la cola, y se cumple el tiempo.	Enviar CAN, cambia estado a S1, se resetea el tiempo, el número de la trama es igual a 1, el contador de datos es igual a 0.
T20	Se guardó el paquete	Se resetea el tiempo, se envía ACK, al número de la trama se le suma 1, y cambia el estado a S1.

Fuente: Elaboración propia

La figura 20 muestra la implementación del peor caso de la máquina de estados, el cual en este ejemplo es el estado tres. S3 se considera como el peor caso, debido a que es el estado que más condiciones tiene, conteniendo cinco transiciones (T6,T7,T8,T9 y T10) con condiciones y acciones diferentes.

```
oid comunicaciones (void)

switch(Var_estado){
 case ESTADO1 :
  ...
  break;
 case ESTADO2 :
  ...
  break;
 case ESTADO3 :
  if(USART&&(~(NumPaq)==DATOUSART)&&(NumPaq==Num_Tra)){//C11
   TIMEOUT=RESET;
   Var_estado=ESTADO4;
   break;
   }//C11
  if(UART&&(~(NumPaq)!=DATOUART)){//C6
     TIMEOUT=RESET;
   TXNACK;
   Var_estado=ESTADO1;
   break;
   }//C6
  if(!USART&&TIMEOUT){//C7
     TIMEOUT=RESET;
   TXNACK;
   Var_estado=ESTADO1;
   break;
   }//C7
  if(UART&&(~(NumPaq)==DATOUSART)&&(NumPaq<Num_Tra)){//C8
     TIMEOUT=RESET;
   TXACK;
   Var_estado=ESTADO1;
   break;
   }//C8
  if(USART&&(~(NumPaq)==DATOUART)&&(NumPaq>Num_Tra)){//C9
     TIMEOUT=RESET;
   TXCAN;
   inicalizar_comunicacion();
   Var_estado=ESTADO1;
   break;
   }//C9
 case ESTADO4 :
  ...
  break;
 case ESTADO5 :
  ...
  break;
 case ESTADO6 :
  ...
  break;
}//switch(Var_estado)
'/void comunicaciones (void)
```

Figura 20. Implementación del peor caso (S3)

Para la implantación es necesario definir las macro definiciones, que son las variables que se usaran en todos los casos, estas se reconocen porque son declaradas en mayúscula, en la figura 20, se pueden ver macro definiciones como: USART, DATOUSART, TIMEOUT, TXNACK, en la tabla 5 se muestra que indica cada macro definición.

Tabla 5. Macro definiciones

Macro definición	Función
USART	Indica si hay dato en la USART
DATOUSART	El dato que llega a la USART
TIMEOUT	Indica el vencimiento de un tiempo de espera

TXNACK	Envió de dato NACK por la USART

Fuente: Elaboración propia

Conclusiones

El método propuesto define claramente el procedimiento para el diseño de software para sistemas embebidos, identificando el rol e interacción de cada uno de los elementos del diagrama de estructura compuesta separándolos en dos grupos principales, recursos y tareas, utilizando técnicas de modelo basado en comportamiento como las propuestas por SysML y UMLRT, haciendo énfasis en los diagramas de secuencia RT.

El modelo establece la guía para la implementación del software para sistemas embebidos sin la necesidad de utilizar un sistema operativo para la administración de los recursos del sistema.

El modelo hace énfasis en uso eficiente de la CPU, identificando la estructura del Polling Loop, la secuencia de consulta de cada una de las tareas y los eventos que disparan su ejecución.

El modelo permite el análisis de los tiempos de ejecución de cada tarea, tiempos de latencia, y porcentaje de uso de la CPU, lo cual permite evaluar el cumplimiento de los tiempos de respuesta para las entradas del sistema, factor fundamental en la implementación de sistemas críticos.

La construcción del diagrama de estructura compuesta, es fundamental para identificar los actores en el diagrama de secuencia RT, lo cual permite contemplar todos los casos que se pueden presentar en la operación del sistema.

El uso de máquinas de estado Mealy es ideal para el diseño de tareas que deben realizar esperas por múltiples eventos de diferentes tipos que disparan su ejecución parcial, permitiendo que otras tareas puedan ser ejecutadas al mismo tiempo que se espera por los eventos, el modelo describe detalladamente el proceso de diseño de estas máquinas de estados y su implementación en lenguaje C para sistemas microprocesados.

El caso de estudio seleccionado, receptor Xmodem, es una ejemplo que reúne una gran cantidad de condiciones que elevan la complejidad de la tarea, debido a las esperas que debe realizar por diferentes tipos de eventos como datos en la USART, cumplimiento de time out e interacción con otras tareas, lo cual requiere de un exhaustivo modelamiento, primero en diagramas de secuencia y luego en máquinas de estado, que guían la implementación de la tarea haciendo buen uso de la CPU sin perder ningún evento.

Trabajo futuro

Como trabajo futuro se propone ampliar el modelo de desarrollo de software de sistemas embebidos incorporándole, una matriz de decisión que indique los parámetros y sus respectivas ponderaciones, para seleccionar cuando un software embebido debe ser implementado utilizando la técnica de Polling Loop o un sistema operativo en tiempo real

Bibliografía

CoFluent Design, “The MCSE Methodology overview.” pp. 1-11, 2009.

J. P. Calvez, EMBEDDED REAL-TIME SYSTEMS a Specification and Design Methodology. New York: John Wiley & Sons, Inc., 1993.

Joseph Yiu, 2007, “The Definitive Guide to the ARM Cortex-M3”, ELSEVIER, ISBN 978-1-85617-963-4

Wayne Lyons, 2005, “Meeting the Embedded Design Needs of Automotive Applications”, IEEE Computer Society, ISBN 0-7695-2288-2

Bayha, A., Franziska, G., & Shats, B. (s.f.). Model-Based Software In-the-Loop-Test of Autonomous Systems:. Germany: fortiss GmbH,.

Fennibay, D., Yurdakuly, A., & Seny, A. (s.f.). Introducing Hardware-in-Loop Concept to the Hardware/Software Co-design of Real-time Embedded Systems. Department of Computer Engineering, Bogazici University, Istanbul, Turkey.

Kramer, V., Mishra , R., Brauneis, P., & Schmidt, K. (2012). Utilization of a hardware-in-the-loop-system for controlling the speed of an eddy current brake. Germany: IOPscience.

Chou, P., & Borriello, G. (1997). Software Architecture Synthesis for Retargetable Real-time Embedded Systems. Washington: IEEE Computer Society Washington, DC, USA ©1997.

Ellis, Carla Schlatter , & Duke Univ. (1999). The case for higher-level power management., (págs. 162 - 167). Durham, NC, USA.

Gajski, D. D. (2010). Embedded System Design. London: Springer.

Kamal Hyder, B. P. (2005). Embedded Systems Design Using the Rabbit 3000 Microprocessor. USA: Elsevier.

Lauder, p., & Kay, J. (1988). A FAIR SHARE SCHEDULER. Panel Editor.

Lee, E. A., & Vincentelli, A. S. (1996). A PRELIMINARY VERSION OF A DENOTATIONAL FRAMEWORK FOR. BERKELEY: DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE.

Mowry, J. (s.f.). Modeling and Simulation in Embedded. DISTek. Integration, Inc.

O'REILLY. (2009). Making Embedded Systems. Beijing, Cambridge, Farnham, Köln, Sebastopol, Tokyo.

Pont, M. J. (2002). Embedded C. Britain: British Library.

Ryssel, U., Ploennigs, J., Kabitzsch, K., & Folie, M. (s.f.). Generative Design of Hardware-in-the-Loop Models. Dresden, Germany: Department of Computer Science Dresden University of Technology.

Selic, B., & Rumbaugh, J. (1998). Using UML for Modeling Complex Real-Time Systems.

Sharan, S. (2012). Products of Mealy-type rough finite state machines. Computing and Communication Systems (NCCCS), 2012 National Conference on, (págs. 1-5). Dhanbad, India.

Swaetz, B. G. (1980). Polling in a Loop System. New Jersey.

Autorización y Renuncia

Los autores autorizan a LACCEI para publicar el artículo en las actas de congresos. Ni los editores ni LACCEI no son responsables ni por el contenido ni por las implicaciones de lo que se expresa en el documento.